Computers and Operations Research 104 (2019) 191–206

Contents lists available at ScienceDirect

Computers and Operations Research

journal homepage: www.elsevier.com/locate/cor

Revisiting simulated annealing: A component-based analysis

Alberto Franzin*, Thomas Stützle

IRIDIA, Université Libre de Bruxelles (ULB), Belgium

ARTICLE INFO

Article history:
Received 4 June 2018
Revised 7 November 2018
Accepted 16 December 2018
Available online 19 December 2018

Keywords:
Simulated annealing
Metaheuristics
Stochastic local search
Automatic algorithm design
Experimental analysis

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. Here, 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. Our experiments consider 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 (Gendreau and Potvin, 2010; Hoos and Stützle, 2005). 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 (Fogel et al., 1966; Holland, 1975; Rechenberg, 1973; Schwefel, 1977) or ideas related to search intensification and diversification (Glover, 1977) that later led to tabu search or scatter search. Despite the successes, a critical review of the history of the field given in Sörensen et al. (2018) argues that "It is not an exaggeration to claim that the field of (meta)heuristics [...] has yet to reach a mature state". One reason can be summarized as a focus on *competition* rather than on *knowledge*, with unfortunate side-effects such as the proliferation of dubiously novel methods, generally based on natural metaphors (Sörensen, 2015) and "high" or "promising" performance claims sometimes backed with poor scientific practices (see e.g. Weyland, 2010; Weyland, 2015) and lacking insights on how and why such methods work.

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 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.

To make these ideas concrete, 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 6000 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 (Aarts et al., 2005; Nikolaev and Jacobson, 2010). Over the years, authors have proposed many variants of SA for different problems, offering by now a large number of implementation choices to an algorithm designer who would like to use SA. We build this framework by taking a component-wise per-

* Corresponding author.
E-mail addresses: afranzin@ulb.ac.be (A. Franzin), stuetzle@ulb.ac.be (T. Stützle).

https://doi.org/10.1016/j.cor.2018.12.015
0305-0548/

spective on the design of SA algorithms. We extract the algorithmic components (including alternative algorithm options and numerical parameters) from proposed variants of SA algorithms and classify them according to their use, to offer alternative choices for each main class of components.

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; 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 us to exploit the recent developments in automatic algorithm configuration techniques, which, given a set of training instances of the problem to be solved, search without manual algorithm designer interaction for the best parameter settings using computer experiments (Hoos 2012). This task of automated algorithm configuration is supported by recent tools such as ParamILS (Hutter et al., 2009), SMAC (Hutter et al., 2011), or irace (López-Ibáñez et al., 2016). In this perspective, our work follows other proposals for the generation of automatically configurable algorithm frameworks such as Satenstein, a framework for local search algorithms for the satisfiability problem in propositional logic (KhudaBukhsh et al., 2009; 2016), frameworks for multi-objective ACO algorithms (López-Ibáñez and Stützle, 2012), ACO algorithms for continuous optimization (Liao et al., 2014), or multi-objective evolutionary algorithms (Bezerra et al., 2016).

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 algorithm developer. Second, in the framework typically many more algorithm components are made available than even an experienced developer of metaheuristic algorithms may be aware of due to the vast literature. 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 from these data. We experimentally show the advantages of our approach studying SA algorithms for three well-known combinatorial optimization problems, the quadratic assignment problem (QAP) and the permutation flow-shop scheduling problem (PFSP) under the makespan and total completion time objectives, respectively.

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 is given in Section 7 and we conclude in Section 8. Supplementary material for this work is available at Franzin and Stützle (2018).

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. (1953), 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 rather free to move, and the structure is subject to substantial changes. The temperature decreases over time, and so does the probability for a particle to move, until the system reaches a state of lowest energy, its ground state. Kirkpatrick et al. (1983), and independently Černý (1985), 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, the ground state corresponds to the globally optimal solution, 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 candidate 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 $f(s^*) \leq f(s) \quad \forall s \in S$. With $\mathcal{N}(s)$ we denote the neighbourhood of s. $\Delta(s', s)$ is the objective function difference of two candidate solutions s and s'; we will also refer with $\Delta_{i,j}$ to the difference $f(s_i) - f(s_j)$ 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 (Kirkpatrick et al., 1983; Metropolis et al., 1953), 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 on both the amount of worsening and T. 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 (Hajek, 1988; Lundy and Mees, 1986; Mitra et al., 1985; Nikolaev and Jacobson, 2010). Unfortunately, these analyses 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 Nikolaev and Jacobson (2010) and Aarts et al. (2005) and cited works therein.

There are several reasons that make SA ideal 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 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 making the right design choices and choosing the right values of numerical parameters is often tedious and error-prone; hence, 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 (Jerrum, 1992), Gen-

eralized Hill Climbing (Johnson and Jacobson, 2002), Static Simulated Annealing (Orosz and Jacobson, 2002), or simply fixed temperature schemes (Cohn and Fielding, 1999; Fielding, 2000). 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 (Dueck and Scheuer, 1990; Moscato and Fontanari, 1990), Great Deluge Algorithm and Record-to-Record Travel (Dueck, 1993), or the more recent Late Acceptance Hill Climbing (Burke and Bykov, 2012; 2017). All these variants are described in the next section. A discussion about the similarities and differences with other metaheuristics can be found in Henderson et al. (2003) and Nikolaev and Jacobson (2010). 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 (Andrieu et al., 2003).

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 (Henderson et al., 2003), 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, which chooses 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 Osman and Potts (1989). We will describe these problem-dependent algorithmic components only if they are used in our experiments.

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 it is 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.

Algorithm 1: Component-based formulation of SA. The components we have identified for our analysis are written in Smallcaps.

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          if ŝ improves over s* then
9              s* := ŝ;
10         end
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*;
```

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, from which some statistics are computed. Some of the following schemes also propose a final temperature value related to the initial one, but these are rather proposals than mandatory rules. A variation that we apply here is to include a multiplicative scaling user-defined constant k to make the methods more flexible.

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.

Fixed value. The simplest option **IT1** is to choose a fixed initial temperature $T_0 = k$. Another option **IT2** is to set an initial temperature proportional to the objective function value of the initial candidate solution $T_0 = k \times f(s_0)$ as in Hussin and Stützle (2014).

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 $f(s_0)$, $f(s_1)$, $f(s_2)$, ..., $f(s_l)$ 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 Burkard and Rendl (1984):

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

$$T_f = k \times \min_{1 \le i \le N, >0} |\Delta_{i,i+1}|. \tag{2}$$

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

$$T_0 = k/N \times \sum_{i=1}^{N} |\Delta_{i,i+1}| \tag{3}$$

or a more elaborated scheme, as for example in Connolly (1990) (**IT5**):

$$T_f = k \times \min_{1 \le i \le N, >0} min \Delta_{i,i+1} \tag{4}$$

$$T_0 = T_f + k \times (\max_{1 \le i \le N} \Delta_{i,i+1} - T_f)/10, \tag{5}$$

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 Johnson et al. (1989, 1991) and Tam (1992). This component (**IT6**) performs a random walk in the search space and computes the value $T_0 = |(k \times \Delta_{avg})/\log p_0|$, 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 **IT6** is derived from the Metropolis acceptance condition.

Misevičius (2003) proposed an initial temperature scheme (**IT7**) for QAP, again based on a random walk in the search space. It extends **IT5** by taking into account also the average gap in the random walk. **IT7** is defined as

$$T_0 = k \times ((1 - \lambda_1 - \lambda_1')\Delta_{min} + \lambda_1 \Delta_{avg} + \lambda_1' \Delta_{max}) \tag{6}$$

$$T_f = k \times ((1 - \lambda_2 - \lambda_2')\Delta_{min} + \lambda_2 \Delta_{avg} + \lambda_2' \Delta_{max}), \tag{7}$$

where the real-valued weights $\lambda_1, \lambda_1', \lambda_2, \lambda_2' \in [0, 1]$ are chosen to satisfy the conditions $\lambda_1 + \lambda_1' \le 1$, $\lambda_2 + \lambda_2' \le 1$. 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.

Problem-dependent schemes. While some of the previous schemes were introduced for some specific problems, they are general. In Osman and Potts (1989), the authors propose an initial temperature for an SA algorithm for the Permutation Flowshop Scheduling Problem (PFSP), which 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

$$T_0 = k \times \sum_{i=1}^{n} \sum_{j=1}^{m} p_{ij}/(m \times n), \tag{8}$$

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 parameter k is set to 1/5 in the original work.

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, the actual outcome of the search, or considerations when continuing the search is deemed too expensive or unlikely find further improvements.

Fixed termination. One possible choice for termination (**SC1**) is a fixed maximum amount of time (Hussin and Stützle, 2014; Tam, 1992). Another possible choice (**SC2**) is a fixed number of candidate moves (Connolly, 1990). The search can be terminated also when a certain minimum temperature value has been reached (**SC3**) (Osman and Potts, 1989); 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**) (Jajodia et al., 1992), or a maximum number of temperature restarts (**SC5**) (Burkard and Rendl, 1984).

Adaptive termination. To provide more flexibility, criteria based on the observation of the actual algorithm execution have been implemented. One such criterion (**SC6**) is to stop the execution after a fixed number of candidate 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 candidate moves is below a given threshold (Johnson et al., 1989; 1991). Finally, criterion **SC9** stops the algorithm when none of the last k candidate moves found a new best solution. The number of candidate moves to be considered by the adaptive criteria can be expressed either in terms of an absolute value or proportional to the neighborhood size.

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 Exploration Criterion is to choose one candidate solution to evaluate from the neighbourhood $\mathcal{N}(s)$. The first SA algorithm (Kirkpatrick et al., 1983) explores the neighborhood randomly, that is, it generates and evaluates a randomly generated neighbour at every step (**NE1**). This kind of default behaviour of SA is used in the vast majority of SA implementations.

Connolly (1990) 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**), which guarantees at least to identify a solution as a local optimum in case no worsening move is accepted after scanning the full neighbourhood.

Ishibuchi et al. (1995) propose two other schemes to select a solution in the neighbourhood. In the first one (**NE3**), k solutions are randomly generated in the neighbourhood, and the best of the k is then compared with the current incumbent. This scheme is modified to (**NE4**), which follows NE3 but stops the process of generating neighboring candidate solutions as soon as one candidate solution is found that improves on the current incumbent, which is then immediately accepted.

3.4. Acceptance criterion (line 6)

This is the component that determines whether the solution $s' \in \mathcal{N}(s)$ generated by the Exploration Criterion is accepted. The

traditional Metropolis criterion (Kirkpatrick et al., 1983; Metropolis et al., 1953) is the best known example for this component. Almost all criteria described here follow one simple pattern: always accept improving or same quality 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**) (Metropolis et al., 1953) is the criterion proposed in the original SA formulation (Kirkpatrick et al., 1983). It always accepts moves to improving and same quality solutions, and accepts worsening moves with a probability that depends on the increase $\Delta(s', s)$ of the objective function value and the temperature T:

$$p_{Metropolis} = \begin{cases} 1 & \text{if} \quad \Delta(s', s) \leq 0 \\ \exp\left(-\Delta(s', s)/T\right) & \text{otherwise.} \end{cases} \tag{9}$$

As the exponential function is relatively expensive from a computational point of view, Johnson *et al.* **AC2** (Johnson et al., 1989) 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 $\sim$0.0067. Worsening moves that entail probabilities lower than this latter value are immediately discarded. 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 (2014) 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}:

$$p = \begin{cases} 1 & \text{if} \quad \Delta(s', s) \leq 0 \\ \exp\left(-\Delta(s', s)/T\right) & \text{if} \quad f(s) < f(s') \leq f(s) \times \phi_{BM} \\ 0 & \text{if} \quad f(s') > f(s) \times \phi_{BM}. \end{cases} \tag{10}$$

Another acceptance criterion was proposed in Bohachevsky et al. (1986) (**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

$$p_{GSA} = \begin{cases} 1 & \text{if} \quad \Delta(s', s) \leq 0 \\ \exp\left(-\beta f(s')^g \Delta(s', s)\right) & \text{otherwise,} \end{cases} \tag{11}$$

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

Geometric criterion. A criterion proposed in Ogbu and Smith (1990) (**AC5**) always accepts an improving solution, and accepts a worsening solution with a probability that decreases in a geometric way[1]:

$$p^k_{Geom} = \begin{cases} 1 & \text{if} \quad \Delta(s', s) \leq 0 \\ p_0 \times r^{k-1} & \text{otherwise,} \end{cases} \tag{12}$$

where p_0 is the initial acceptance probability, $r < 1$ is the reducing factor, and k is the number of update steps. This criterion, thus, does not use the temperature in the evaluation of a solution. In fact, it is rather related to randomized iterative improvement as defined in Hoos and Stützle (2005).

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

Deterministic criteria. While the probabilistic acceptance of worsening moves is the distinctive feature of SA, some authors have questioned whether the stochasticity introduced by the acceptance criterion is necessary to obtain good results (Dueck and Scheuer, 1990; Hajek and Sasaki, 1989; Moscato, 1989; Moscato and Fontanari, 1990). In these works, the authors have proposed a deterministic version of the Metropolis criterion, called Threshold Accepting (TA, **AC6**) by Dueck and Scheuer (1990), which accepts every worsening solution whose difference in objective function value is lower than a threshold $\bar{\phi}$:

$$p_{TA} = \begin{cases} 1 & \text{if} \quad \Delta(s', s) \leq \bar{\phi} \\ 0 & \text{otherwise.} \end{cases} \tag{13}$$

$\bar{\phi}$ is a parameter whose value decreases during the search process, just as the temperature in SA. In Dueck and Scheuer (1990) the authors do not provide any guidance in how to initialize and update $\bar{\phi}$, while the authors of Moscato and Fontanari (1990) explicitly use SA terminology such as "temperature" and "cooling".

Starting from Threshold Accepting, Dueck proposed two alternative deterministic acceptance criteria under two different metaphors (Dueck, 1993). 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:

$$p^k_{GDA} = \begin{cases} 1 & \text{if} \quad f(s) \leq \bar{\phi}^k \\ 0 & \text{otherwise,} \end{cases} \tag{14}$$

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):

$$p_{RTR} = \begin{cases} 1 & \text{if} \quad \phi \leq f(s^*) + \gamma \\ 0 & \text{otherwise,} \end{cases} \tag{15}$$

where γ is a fixed parameter.

A more recent work by Burke and Bykov (2012, 2017) proposes 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 that was already accepted. The acceptance of a solution s is therefore controlled according to the formula

$$p^l_{LAHC} = \begin{cases} 1 & \text{if} \quad f(s_l) \leq \max\{f(s_{l-1}), f(s_{l-\kappa})\} \\ 0 & \text{otherwise,} \end{cases} \tag{16}$$

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 to accept only improving or same quality solutions (**AC10**), discarding worsening ones:

$$p_{det} = \begin{cases} 1 & \text{if} \quad \Delta(s', s) \leq 0 \\ 0 & \text{otherwise.} \end{cases} \tag{17}$$

The effect of this choice is turning SA into a Hill-Climbing search (Appleby et al., 1961), 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 proposed 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 (Kirkpatrick et al., 1983), the authors propose two decreasing geometric schemes, $T_{i+1} = \alpha \times \beta^{T_i}$ (**CS1**) and $T_{i+1} = \alpha \times T_i$ (**CS2**), 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 Geman and Geman (1984), the authors use a logarithmic cooling scheme (**CS3**) $T_{i+1} = a/\log(b+i)$ (**CS3**) with $b = 1$ in the original work. We also implement the scheme $T_{i+1} = a/(b + \log i)$ (**CS4**) (Strenski and Kirkpatrick, 1991).

Lundy-Mees and variants. A popular cooling scheme is the one proposed by Lundy and Mees (1986) (**CS5**), in which $T_{i+1} = T_i/(a + b \times T_i)$. a and b have to be chosen as $a + b \times T_i > T_i$ to guarantee final convergence.

Connolly (1990) 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 **CS5**), until too many consecutive candidate 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 Szu and Hartley (1987) and uses $T_{i+1} = a/(1 + b \times T_i)$ with $a = b = 1$.

Quadratic schemes. Andersen, Vidal and Iversen (Andersen et al., 1993) developed a quadratic cooling scheme (**CS8**) for a network design problem. The formula proposed is

$$T_{i+1} = a \times K^2 + b \times K + c, \text{ where} \tag{18}$$

$$a = \frac{T_0 - T_f}{I^2}, \quad b = 2 \times \frac{T_f - T_0}{I}, \quad c = T_0,$$

and 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 Dueck (1993) uses an arithmetic decrease of the temperature value (in this case, of the threshold) $T_{i+1} = T_i - a$ 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 $T_{i+1} = \max\{T_i - a, 0\}$.

Non-decreasing schemes. Since the inception of SA, various authors have studied variants that keep the same temperature values throughout the whole search (Cohn and Fielding, 1999; Fielding, 2000; Hajek and Sasaki, 1989; Jerrum, 1992; Jerrum and Sinclair, 1996; Kirkpatrick, 1984; Mitra et al., 1985; Orosz and Jacobson, 2002) 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 $T_{i+1} = T_i = T_0 \; \forall i$ (**CS10**). Here, T_0 is a supposedly *optimal* temperature value that can obtain superior results with respect to cooling schemes that reduce the temperature parameter. The *Q8-7* cooling scheme **CS6** of Connolly (1990) may also be seen as a scheme where a fixed temperature value is discovered by the algorithm at runtime.

As non-decreasing schemes are dependent on good initial settings, a more robust scheme (**CS11**) sets a temperature band $[T_0, a \times T_0]$, $a > 1$, and, at each update, randomly chooses a value from it.

Another scheme is the Old Bachelor Acceptance (OBA) (Hu et al., 1995). 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 (**CS12**) adjusts the temperature "symmetrically" according to the formula

$$T_{i+1} = \begin{cases} T_i + ((\text{age}/a)^b - 1) \times \Delta \times (1 - i/M)^c & \text{if } s_i \text{ is discarded} \\ T_i - ((\text{age}/a)^b - 1) \times \Delta \times (1 - i/M)^c & \text{if } s_i \text{ is accepted,} \end{cases} \tag{19}$$

where a, b, c are control parameters, M is the total number of candidate moves, Δ is the granularity of the update and *age* is the number of consecutively rejected moves. OBA2 (**CS13**) instead uses a "steepest descent, mildest ascent" strategy (Hansen and Jaumard, 1990) that makes acceptance of a solution more likely in the first d proposed moves after an accepted move:

$$T_{i+1} = \begin{cases} T_i + (\Delta/d) \times (1 - i/M) & \text{if } s_i \text{ is discarded} \\ T_i - \text{count} \times \Delta \times (1 - i/M) & \text{if } s_i \text{ is accepted,} \end{cases} \tag{20}$$

where d and *count* are control parameters, which are updated by incrementing *count* by one if age is smaller than d; in the other case, age is set to 1.

3.6. Temperature length (line 12)

This component controls the number of candidate 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 candidate moves $L = k$ (**TL1**) with a value of $L = 1$ meaning that the temperature is updated after every move. The second is to update after a number of candidate moves proportional to the size of the neighbourhood $\mathcal{N}(s)$, $L = k \times |\mathcal{N}(s)|$ (**TL2**), or proportional to the square of neighbourhood size $L = k \times |\mathcal{N}(s)|^2$ (**TL3**) (Jajodia et al., 1992). Also the size n of the problem instance is used $L = k \times n$ (**TL4**) e.g. in Hussin and Stützle (2014), or its square $L = k \times n^2$ (**TL5**) (Connolly 1990).

Adaptive temperature length. Instead of fixed temperature lengths, it is possible to set these depending on the search progress. Abramson (1991) updates the temperature after a certain

number of accepted moves (**TL6**). In Jajodia et al. (1992) the authors combine this approach with a maximum number of total candidate moves at a given temperature (that might be fixed or proportional to $|\mathcal{N}(s)|$), to avoid spending too many candidate moves at a certain temperature (**TL7**).

Variable temperature length. Other proposals (Rose et al., 1990; Souilah, 1995) 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 ($L_{i+1} = L_i + k$, **TL8**), geometric ($L_{i+1} = k \times L_i$, **TL9**), logarithmic ($L_{i+1} = k/L_i$, **TL10**) and exponential ($L_{i+1} = L_i^{1/\alpha}$, **TL11**) updates for temperature lengths, respectively, 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 behaviour. Therefore, authors have proposed to reset the temperature to the initial or another level once it reaches a critically low value (Kirkpatrick, 1984), allowing in this way effective escapes from local optima. The reset of the temperature to its initial value is called *temperature restart*, while setting it to a possibly different value is called *reheating*. Of course, one may also 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 proposed 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 candidate 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 $T_{i+1} = T_i/k,\ 0 < k < 1$. We can perform this operation once the acceptance rate falls below a given threshold, overall (**TR9**) or in the last k candidate moves (**TR10**), or the search has not accepted any new solution in the last k candidate moves (**TR11**). We also consider reheating after a certain number of candidate 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**) (Abramson et al., 1999) 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 candidate 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 (Pagnozzi and Stützle 2018), 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, which we distinguish between three levels:

Level 1: instantiate known SA algorithms for algorithm comparisons (no configuration);
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. At 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 (Hoos, 2012; Stützle, 2009). 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 behaviour, which tries to obtain as good solutions as early as possible during the search (Zilberstein, 1996). To do so, we follow the methodology proposed in López-Ibáñez and Stützle (2014).

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 (Breiman, 2001).

4.1. Experimental setup

As mentioned above, we implemented the SA components within the EMILI framework (Pagnozzi and Stützle, 2018). As the automatic algorithm configuration tool we use irace (López-Ibáñez et al., 2016; 2011), which is an R implementation of the Iterated Racing algorithm (Balaprakash et al., 2007; Birattari et al., 2010). 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 (Wright and Ziegler, 2017).

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 60 000 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 using common random seeds. 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 2.4GB 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 (Burkard et al., 1998; Koopmans and Beckmann, 1957). 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

$$\sum_{i=1}^{n}\sum_{j=1}^{n} f_{ij} d_{\pi(i)\pi(j)}. \tag{21}$$

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

$$\mathcal{N}(\pi)=\{\pi' | \pi'(j)=\pi(h) \wedge \pi'(h)=\pi(j) \wedge \forall l \notin \{j,h\} : \pi'(l)=\pi(l)\} \tag{22}$$

for a solution π. The size of this neighbourhood is $n(n-1)/2$. The objective function value of a solution s' can be computed from the objective function value of a neighbouring solution s in constant time. However, in practice, such advantage is beneficial only when the full neighbourhood is always explored, bringing down the computational complexity from a total of $O(n^3)$ to $O(n^2)$. It does not make too much difference when a single solution is evaluated in the neighbourhood, and the overall time needed to converge to a local optimum is comparable.

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 (Fischetti et al., 2012), 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 (Hussin and Stützle, 2014), and one where the matrices are generated randomly according to an Euclidean structure, closer to real-life instances (Taillard, 1995). 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 in the configuration phase and 50 instances as the independent test set for the evaluation of the configured algorithms. The anytime behaviour is evaluated also on larger random and structured instances of size 500 from the same benchmark (Hussin and Stützle, 2014). Unless otherwise specified, the runtime limit for QAP is 10 seconds; experiments with 30 and 100 seconds of runtime are reported in the supplementary material.

4.3. PFSP setup

The PFSP is a scheduling problem that arises in various industrial environments (Framiñán et al., 2014; Garey et al., 1976; Pinedo, 2012). 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) (Fernandez-Viagas et al., 2017) and under the *Total Completion Time* objective (PFSP-TCT) (Pan and Ruiz, 2012; 2013). 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 $\sum_{i=1}^{n} C_{\pi(i),m}$.

For both objectives, the initial solution is generated using the NEH heuristic (Nawaz et al., 1983). 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 $n(n-1)$, and each solution is evaluated in $O(n)$ time.

The training set is composed by 40 randomly generated instances with sizes between 50 jobs and 20 machines to 250 jobs and 50 machines (Mascia et al., 2013). The test set is the popular Taillard benchmark (Taillard 1993), consisting of 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 Vallada et al. (2015). The runtime limit is based on the instance size and is computed as $(n \times m \times 0.015)/2$ seconds. Experiments with $10\times$ runtime are reported in the supplementary material.

5. SA algorithms for the QAP

Attempts to solve the QAP with Simulated Annealing can be traced back at least to 1984 (Burkard and Rendl, 1984). 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 (Battiti and Tecchiolli, 1994; Çela, 1998; Paul, 2010; Paulli, 1993). In what follows we list SA implementations proposed for the QAP or closely related problems.

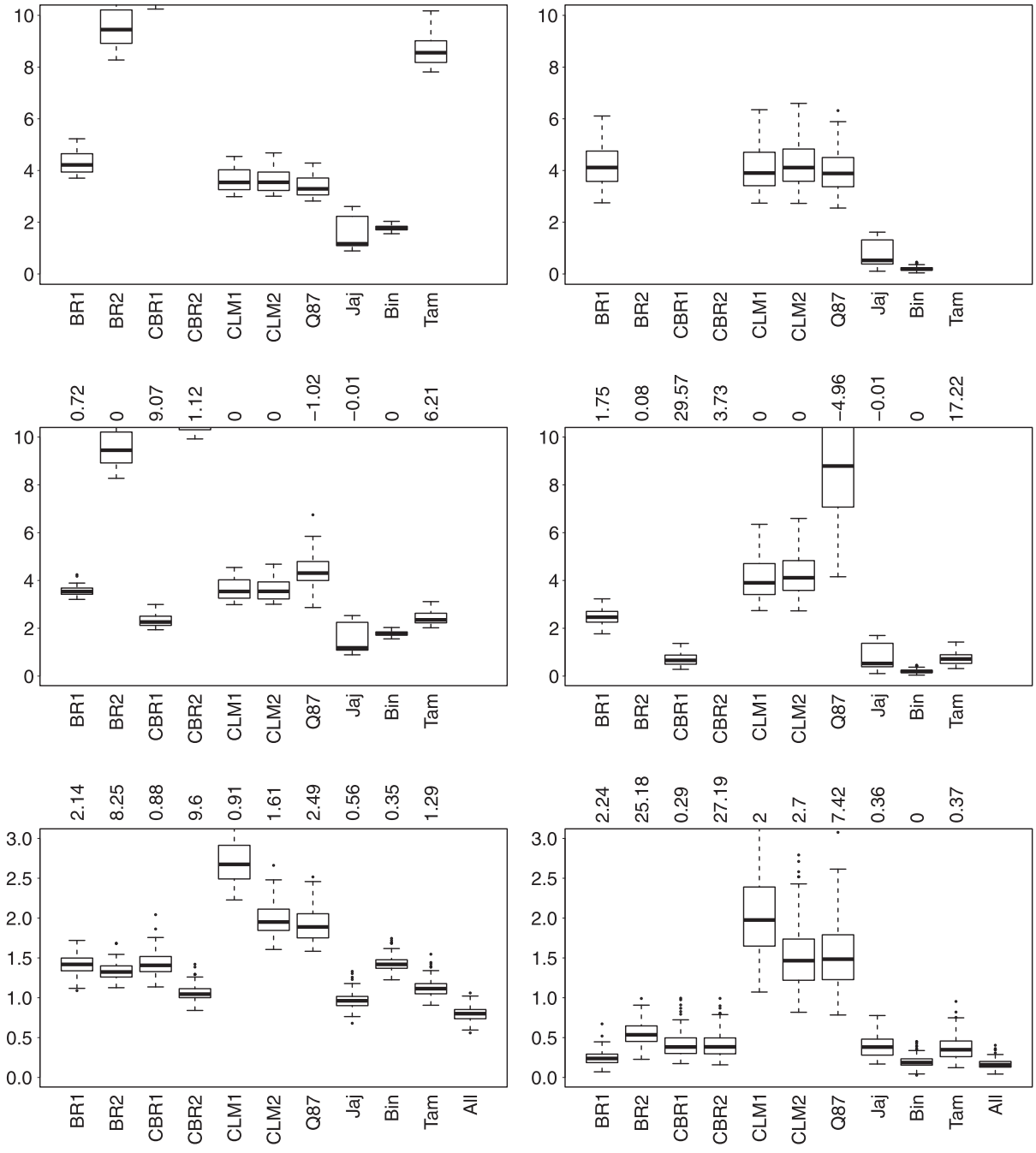

Fig. 1. Average Relative Percentage Deviation (ARPD) from the best known solutions obtained by the algorithms on random (first row, left plot) and structured instances (right plot). In the top row the algorithms are in their default settings (Level 1, step 1), in the middle row the results with ten seconds on runtime (Level 1, step 2), and in the bottom row the results obtained after tuning their numerical parameters, including the results obtained by the algorithms generated when tuning the whole framework (`All`, Level 2 and 3). Above the boxplots in the middle and bottom row is given the difference in terms of ARPD from the row above; negative differences correspond to a worsening of the ARPD.

The first two schemes we implement, BR1 and BR2, are proposed in 1984 by Burkard and Rendl (1984). We consider also the SA of Burkard and Rendl as described by Connolly (1990) for comparing the results of his experiments (CBR1 and CBR2). Connolly (1990) proposes several versions of SA for QAP: two of them employ the cooling scheme of Lundy and Mees (CLM1 and CLM2), while the third version uses the Q8-7 scheme (Q87). Two other SA algorithms for the QAP were proposed by Jajodia et al. (1992) (`Jaj`) and Tam (1992) (`Tam`) in 1992. The last SA for the QAP that we evaluate is by Hussin and Stützle (2014) (`Bin`). A synopsis of how these implementations can be described in terms of their components is given in the supplementary material. Fig. 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.

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. 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

Table 1
Results of the Friedman rank sum test for the algorithms for the QAP in their default settings (top block, Level 1 first step), with ten seconds of runtime (middle, Level 1, second step) and after tuning their numerical parameters on the QAP instances, including the algorithms generated automatically (`All`, bottom, Level 2 and 3). 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)
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)
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)

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 hill-climbing approach, which 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,[2] and have been tuned manually for, by current standards, very small instances and very short computation times; apparently, the parameter settings do not scale well to other settings.

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. 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 probably due to a poor specification of the threshold of consecutive rejected moves that triggers the fixed temperature phase in the *Q8-7* cooling scheme. This value is originally set according to the total number of solutions evaluated. Consequently, we scaled this parameter up by estimating the number of moves that can be generated in the given time. Apparently, the choice is not robust to a much larger number of candidate moves.

The results for increased computation time indicate that a reconfiguration 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. All algorithms now improve their performance; only for `Bin` on the structured instances no statistically significant difference is observed, probably as it was already automatically tuned originally; the differences on the random instances, may be due to the different tuning setup with respect to the original work.

Almost all 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 those using 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 neighbourhood 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 Fig. 1 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 1. On both instance classes the algorithms `All` (the rightmost boxplots) outperform the tuned existing algorithms. On the random instances the improvement is more substantial, while on the structured instances it is less strong but still statistically significant, probably because the margin for further improvement on structured instances is minor.

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 (Osman and Potts, 1989), 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 (in-

[2] 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 49 360 MIPS at 2.7GHz, (with a ratio of 5673). Source: https://en.wikipedia.org/wiki/Instructions_per_second#Timeline_of_instructions_per_second.

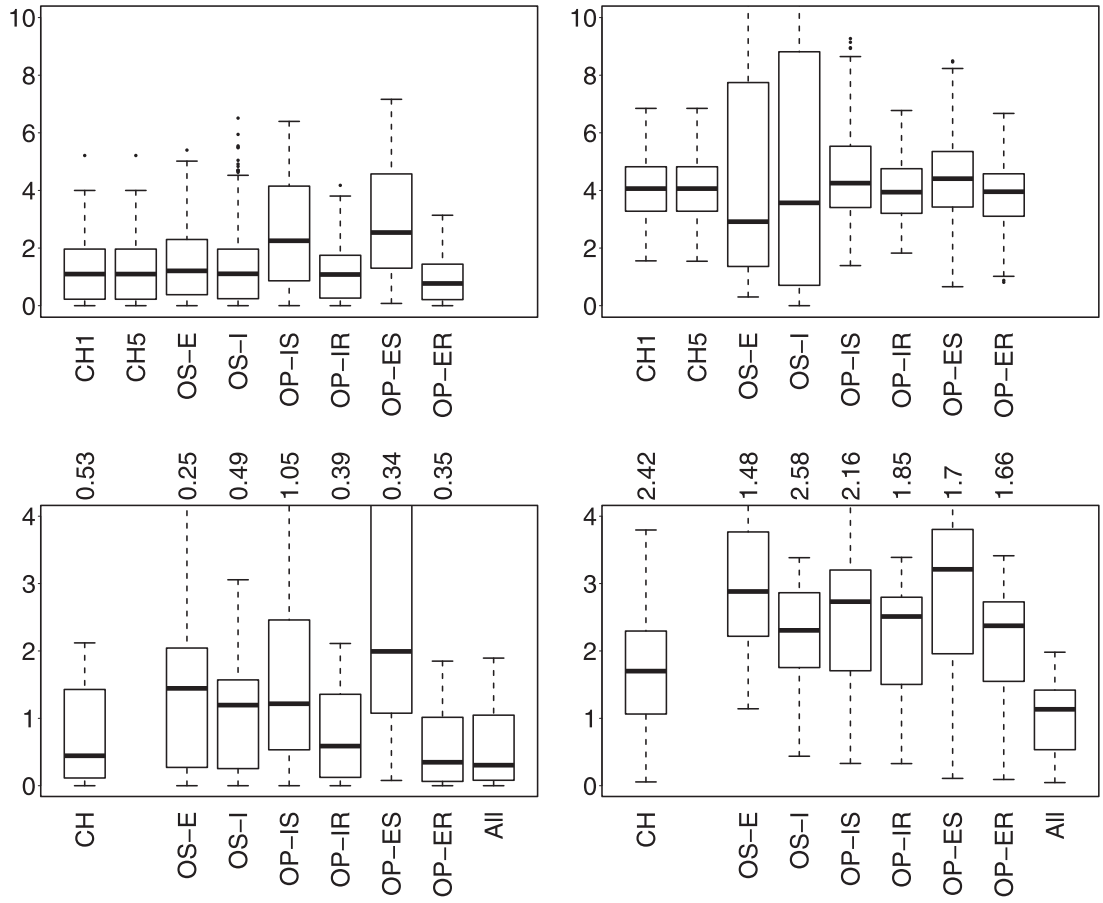

Fig. 2. Average Relative Percentage Deviation (ARPD) from the best known solutions obtained by the algorithms on the whole Taillard benchmark under the MS and TCT objectives (left and right plot respectively). In the top row the results are obtained with the algorithms in their default settings (Level 1). In the bottom row the results are obtained after tuning the algorithms, collapsing the two `CH1` and `CH5` into a single `CH` algorithm and including the results obtained by the algorithms generated when tuning the whole framework (`All`; Levels 2 and 3). Above the boxplots in the bottom row is given the improvement in terms of ARPD from the top row.

dicated by `I` or `E` in the algorithm name). The second algorithm, `OS`, is proposed by Ogbu and Smith (1990); in the original paper the authors also evaluate both the insert and the exchange neighbourhoods. Finally, in the more recent work `CH` (Chen and Hsieh, 2014) two variants use two different values for one parameter. The details of these algorithms are summarized in the supplementary material. 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. The plots with the results across the whole test set are instead reported in Fig. 2 in terms of ARPD, obtained by the algorithms in their original settings (Level 1 of the analysis, top row), and after the tuning (Level 2 of the analysis, bottom row); the improvements of the tuning with respect to the default versions is reported on top of the boxplots. As `CH1` and `CH5` differ only for the value of one numerical parameter, they appear as a single algorithm `CH` in the tuning. With the tuned algorithms we also include the results obtained when automatically designing SA algorithms. In Table 2 we report the rankings obtained for the MS and TCT objectives by the algorithm default settings, and after the tuning.

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. `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 (algorithms `OP-*`), 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, where we found the contrary behaviour. Contrarily to the `OS` algorithms, the exchange neighbourhood outperforms the insert one for both objectives and both before and after the tuning.

Table 2
Comparison of the results of the Friedman rank sum test for the algorithms in their default settings (Level 1) 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) (Level 2). Algorithms are ranked according to their results. In the tuned settings also results for **All** are included (Level 3). 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-ER` (0), `OP-IR` (155.5), `CH5` (179.5), `CH1` (189.5), `OS-I` (243.5), `OS-E` (258), `OP-IS` (478.5), `OP-ES` (627.5)
Tuned	40.26	`OP-ER` (0), `All` (33.5), `OP-IR` (213), `CH` (233.5), `OS-I` (268), `OS-E` (476), `OP-IS` (516), `OP-ES` (700)
Default	72.47	`OS-E` (0), `OP-ER` (13), `OS-I` (71), `OP-IR` (122), `OP-ES` (134), `OP-IS` (156), `CH1` (159), `CH5` (161)
Tuned	46.82	`All` (0), `CH` (173), `OP-ER` (321), `OP-IR` (378), `OS-I` (444), `OP-IS` (547), `OP-ES` (612), `OS-E` (685)

As a next step, we apply Level 3 of our analysis, that is, we automatically design SA algorithms from the whole framework. On the MS objective the results are not statistically significantly different from the ones obtained by `OP-ER`. 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.

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). On the largest instances the search takes more time to discover a good solution, but from that point on the behaviour is similar to the SAs tuned for anytime behaviour. The relative plots are given in the supplementary material.

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 with the data generated during the configuration phase for `All`. From this model, we get as a byproduct an estimate of the variable importance. Random forests Breiman (2001) 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 Genuer et al. (2010). The results of our analysis, normalized to sum one, are reported in Table 3 for the four scenarios.

Table 3
The ten most important components and numerical parameters on the four scenarios.

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%

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 neighborhood exploration, a component often neglected in the SA literature. It is interesting to observe that the historically most studied component of SA from both a theoretical and experimental perspective, the cooling scheme, is indeed important (as it influences the behaviour of the acceptance criterion), but not as important as commonly expected: 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 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 instances. 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 the 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 configuration for optimizing the final solution quality.

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. 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.

In the supplementary material we include results obtained on the testset with longer runtimes, namely 30 and 100 seconds for the QAP scenarios and with $10\times$ time for the PFSP scenarios; the tunings are the same ones of the previous experiments, performed over 10 seconds. It is interesting to observe that a longer runtime does not necessary entail better results. This happens for both the nontuned and the tuned versions of our set of algorithms. A temperature restart scheme is necessary to escape convergence and profit from additional runtime. If an algorithm not using temperature restarts has already converged to its best solution, more time will not be beneficial; in some cases (e.g. the nontuned versions of `BR1` and especially `Jaj` for the QAP) the ten seconds limit was not sufficient for reaching convergence, and a longer runtime allows that, at least up to a certain extent. The automatically designed algorithms we obtained are able to improve with longer runtimes, in particular when designed for anytime behaviour. As observed especially for the PFSP scenarios, the improvement over a much longer runtime can anyway be limited; the practitioner should therefore consider carefully whether it is actually worth to wait longer, without any guarantee about the improvement of the final solution quality. Tuning over a longer runtime is likely to be beneficial, at the cost of much higher computational requirements.

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. This clearly justifies 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.

Let us now mention also the discrepancy between our experimental results and the focus in many theoretical works about SA. As previously mentioned, SA theory has often focused 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. Johnson et al. (1989) for the former, and Fox (1993, 1995) for the latter). Recent experimental works such as Franzin and Stützle (2017) and Jackson et al. (2018) have instead studied several acceptance criteria for SA, and align with the present work both in methodology and findings.

Finally, though we stress that a comparison with other methods is neither in the scope of this work, nor entirely possible due to different experimental settings with other existing works, we discuss the efficacy of the automatically generated SA algorithms when compared to different existing methods. For the QAP, there is no single algorithm that can be considered state of the art (Benlic and Hao, 2013). The performance of different heuristics has been observed to be dependent on target quality (Paul, 2010), instance size (Hussin and Stützle, 2014) and instance characteristics (Benlic and Hao, 2013), and several algorithm selection approaches have been proposed (Beham et al., 2017; Dantas and Pozo, 2018; Pitzer et al., 2013; Smith-Miles, 2008). In the supplementary material we report additional results obtained on the commonly used QAPlib (Burkard et al., 1997) by the SA algorithms obtained for the random QAP instances scenario for different running times, where we obtain quite good solution qualities. For the PFSP scenarios we can consider the recent work of Pagnozzi and Stützle (2018) as state of the art, with automatically generated hybrid algorithms that outperformed existing alternatives. In this case, the automatically generated SAs are not able to reach the same quality level.

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 frameworks. 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 exper-

imental 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 (Dueck and Scheuer, 1990), Great Deluge (Dueck, 1993), Record-To-Record Travel (Dueck, 1993), or Late Acceptance Hill-Climbing (Burke and Bykov, 2012; 2017). 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 key important components of SA. When configuring for best reachable solution quality, the cooling scheme seems relevant as it influences the behaviour of the acceptance criterion, but to a lesser 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 extended 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 López-Ibáñez et al. (2017) and Marmion et al. (2013). This would enable us to compare automatically designed SAs against other automatically designed metaheuristics, to study the role, impact and composition of SA algorithms when combined with or used as part of other metaheuristics, and to understand when and how to move beyond the SA structure to automatically design bottom up new metaheuristics without constraining them to a predetermined form.

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.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.cor.2018.12.015.

References

Aarts, E.H.L., Korst, J.H.M., Michiels, W., 2005. Simulated annealing. In: Search Methodologies. Springer, pp. 187–210.
Abramson, D., 1991. Constructing school timetables using simulated annealing: sequential and parallel algorithms. Manage. Sci. 37 (1), 98–113.
Abramson, D., Amoorthy, M.K., Dang, H., 1999. Simulated annealing cooling schedules for the school timetabling problem. Asia-Pac. J. Oper. Res. 16 (1), 1–22.
Andersen, K., Vidal, R.V.V., Iversen, V.B., 1993. Design of a teleprocessing communication network using simulated annealing. In: Vidal, R.V.V. (Ed.), Applied Simulated Annealing. Springer, pp. 201–215.
Andrieu, C., de Freitas, N., Doucet, A., Jordan, M.I., 2003. An introduction to MCMC for machine learning. Mach. Learn 50 (1–2), 5–43.
Appleby, J., Blake, D., Newman, E., 1961. Techniques for producing school timetables on a computer and their application to other scheduling problems. Comput. J. 3 (4), 237–245.
Balaprakash, P., Birattari, M., Stützle, T., 2007. Improvement strategies for the F-race algorithm: sampling design and iterative refinement. In: Bartz-Beielstein, T., Blesa, M.J., Blum, C., Naujoks, B., Roli, A., Rudolph, G., Sampels, M. (Eds.), Hybrid Metaheuristics. In: Lecture Notes in Computer Science, Vol. 4771. Springer, Heidelberg, Germany, pp. 108–122.
Battiti, R., Tecchiolli, G., 1994. Simulated annealing and tabu search in the long run: a comparison on QAP tasks. Comput. Math. Appl. 28 (6), 1–8. doi:10.1016/0898-1221(94)00147-2.
Beham, A., Affenzeller, M., Wagner, S., 2017. Instance-based algorithm selection on quadratic assignment problem landscapes. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion. ACM, pp. 1471–1478.
Benlic, U., Hao, J.K., 2013. Breakout local search for the quadratic assignment problem. Appl. Math. Comput. 219 (9), 4800–4815.
Bezerra, L.C.T., López-Ibáñez, M., Stützle, T., 2016. Automatic component-wise design of multi-objective evolutionary algorithms. IEEE Trans. Evol. Comput. 20 (3), 403–417. doi:10.1109/TEVC.2015.2474158.
Birattari, M., Yuan, Z., Balaprakash, P., Stützle, T., 2010. F-race and iterated F-race: an overview. In: Bartz-Beielstein, T., Chiarandini, M., Paquete, L., Preuss, M. (Eds.), Experimental Methods for the Analysis of Optimization Algorithms. Springer, Berlin, Germany, pp. 311–336.
Bohachevsky, I.O., Johnson, M.E., Stein, M.L., 1986. Generalized simulated annealing for function optimization. Technometrics 28 (3), 209–217.
Breiman, L., 2001. Random forests. Mach. Learn. 45 (1), 5–32. doi:10.1023/A:1010933404324.
Burkard, R.E., Çela, E., Pardalos, P.M., Pitsoulis, L.S., 1998. The quadratic assignment problem. In: Pardalos, P.M., Du, D.Z. (Eds.), Handbook of Combinatorial Optimization, Vol. 2. Kluwer Academic Publishers, pp. 241–338.
Burkard, R.E., Karisch, S.E., Rendl, F., 1997. QAPLIB–a quadratic assignment problem library. J. Global Optim. 10 (4), 391–403.
Burkard, R.E., Rendl, F., 1984. A thermodynamically motivated simulation procedure for combinatorial optimization problems. Eur. J. Oper. Res. 17 (2), 169–174. doi:10.1016/0377-2217(84)90231-5.
Burke, E.K., Bykov, Y., 2012. The Late Acceptance Hill-Climbing Heuristic. University of Stirling Tech. Rep. CSM-192.
Burke, E.K., Bykov, Y., 2017. The late acceptance hill-climbing heuristic. Eur. J. Oper. Res. 258 (1), 70–78.
Çela, E., 1998. The Quadratic Assignment Problem: Theory and Algorithms. Kluwer Academic Publishers, Dordrecht, The Netherlands.
Chen, R.M., Hsieh, F.R., 2014. An exchange local search heuristic based scheme for permutation flow shop problems. Appl. Math. Inf. Sci. 8 (1), 209–215.
Černý, V., 1985. A thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm. J. Optim. Theory Appl. 45 (1), 41–51.
Cohn, H., Fielding, M.J., 1999. Simulated annealing: searching for an optimal temperature. SIAM J. Optim. 9 (3), 779–802.
Connolly, D.T., 1990. An improved annealing scheme for the qap. Eur. J. Oper. Res. 46 (1), 93–100.
Dantas, A.L., Pozo, A.T.R., 2018. A meta-learning algorithm selection approach for the quadratic assignment problem. In: 2018 IEEE Congress on Evolutionary Computation (CEC). IEEE, pp. 1–8.
Dueck, G., 1993. New optimization heuristics: the great deluge algorithm and the record-to-record travel. J. Comput. Phys. 104 (1), 86–92.
Dueck, G., Scheuer, T., 1990. Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing. J. Comput. Phys. 90 (1), 161–175.
Fernandez-Viagas, V., Ruiz, R., Framiñán, J.M., 2017. A new vision of approximate methods for the permutation flowshop to minimise makespan: state-of-the-art and computational evaluation. Eur. J. Oper. Res. 257 (3), 707–721.
Fielding, M.J., 2000. Simulated annealing with an optimal fixed temperature. SIAM J. Optim. 11 (2), 289–307.
Fischetti, M., Monaci, M., Salvagnin, D., 2012. Three ideas for the quadratic assignment problem. Oper. Res. 60 (4), 954–964.
Fogel, D.B., Owens, A.J., Walsh, M.J., 1966. Artificial Intelligence Through Simulated Evolution. John Wiley & Sons.
Fox, B.L., 1993. Integrating and accelerating tabu search, simulated annealing, and genetic algorithms. Ann. Oper. Res. 41 (2), 47–67.
Fox, B.L., 1995. Simulated annealing: folklore, facts, and directions. In: Monte Carlo and Quasi-Monte Carlo Methods in Scientific Computing. Springer, pp. 17–48.
Framiñán, J.M., Leisten, R., Ruiz, R., 2014. Manufacturing Scheduling Systems: An Integrated View on Models, Methods, and Tools. Springer, New York, NY.

Franzin, A., Stützle, T., 2017. Comparison of acceptance criteria in randomized local searches. In: Lutton, E., Legrand, P., Parrend, P., Monmarché, N., Schoenauer, M. (Eds.), EA 2017: Artificial Evolution. In: Lecture Notes in Computer Science, Vol. 10764. Springer, Heidelberg, Germany, pp. 16–29.

Franzin, A., Stützle, T., 2018. Revisiting simulated annealing: a component-based analysis: supplementaty material. http://iridia.ulb.ac.be/supp/IridiaSupp2018-001.

Garey, M.R., Johnson, D.S., Sethi, R., 1976. The complexity of flowshop and jobshop scheduling. Math. Oper. Res. 1, 117–129.

Geman, S., Geman, D., 1984. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. IEEE Trans. Pattern Anal. Mach. Intell. 6 (6), 721–741.

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

Genuer, R., Poggi, J.M., Tuleau-Malot, C., 2010. Variable selection using random forests. Pattern Recognit. Lett. 31 (14), 2225–2236.

Glover, F., 1977. Heuristics for integer programming using surrogate constraints. Decis. Sci. 8, 156–166.

Hajek, B., 1988. Cooling schedules for optimal annealing. Math. Oper. Res. 13 (2), 311–329.

Hajek, B., Sasaki, G., 1989. Simulated annealing–to cool or not. Syst. Control Lett. 12 (5), 443–447.

Hansen, P., Jaumard, B., 1990. Algorithms for the maximum satisfiability problem. Computing 44, 279–303.

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

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

Hoos, H.H., 2012. Programming by optimization. Commun. ACM 55 (2), 70–80. doi:10.1145/2076450.2076469.

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

Hu, T.C., Kahng, A.B., Tsao, C.W.A., 1995. Old bachelor acceptance: a new class of non-monotone threshold accepting methods. ORSA J. Comput. 7 (4), 417–425.

Hussin, M.S., Stützle, T., 2014. Tabu search vs. simulated annealing for solving large quadratic assignment instances. Comput. Oper. Res. 43, 286–291.

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

Hutter, F., Hoos, H.H., Leyton-Brown, K., Stützle, T., 2009. ParamILS: an automatic algorithm configuration framework. J. Artif. Intell. Res. 36, 267–306.

Ishibuchi, H., Misaki, S., Tanaka, H., 1995. Modified simulated annealing algorithms for the flow shop sequencing problem. Eur. J. Oper. Res. 81 (2), 388–398.

Jackson, W.G., Özcan, E., John, R.I., 2018. Move acceptance in local search metaheuristics for cross-domain search. Expert Syst. Appl. 109, 131–151.

Jajodia, S., Minis, I., Harhalakis, G., Proth, J.M., 1992. CLASS: Computerized layout solutions using simulated annealing. Int. J. Prod. Res. 30 (1), 95–108.

Jerrum, M., 1992. Large cliques elude the metropolis process. Random Struct. Algorithms 3 (4), 347–359.

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

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

Johnson, D.S., Aragon, C.R., McGeoch, L.A., Schevon, C., 1989. Optimization by simulated annealing: an experimental evaluation: part I, graph partitioning. Oper. Res. 37 (6), 865–892.

Johnson, D.S., Aragon, C.R., McGeoch, L.A., Schevon, C., 1991. Optimization by simulated annealing: an experimental evaluation: part II, graph coloring and number partitioning. Oper. Res. 39 (3), 378–406.

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

KhudaBukhsh, A.R., Xu, L., Hoos, H.H., Leyton-Brown, K., 2016. SATEnstein: automatically building local search SAT solvers from components. Artif. Intell. 232, 20–42.

Kirkpatrick, S., 1984. Optimization by simulated annealing: quantitative studies. J. Stat. Phys. 34 (5–6), 975–986.

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

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

Liao, T., Stützle, T., de Oca, M.A.M., Dorigo, M., 2014. A unified ant colony optimization algorithm for continuous optimization. Eur. J. Oper. Res. 234 (3), 597–609.

López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L.P., Stützle, T., Birattari, M., 2016. The irace package: iterated racing for automatic algorithm configuration. Oper. Res. Perspect. 3, 43–58. doi:10.1016/j.orp.2016.09.002.

López-Ibáñez, M., Dubois-Lacoste, J., Stützle, T., Birattari, M., 2011. The Irace Package, Iterated Race for Automatic Algorithm Configuration. IRIDIA, Université Libre de Bruxelles, Belgium Tech. Rep.. http://iridia.ulb.ac.be/IridiaTrSeries/IridiaTr2011-004.pdf.

López-Ibáñez, M., Kessaci, M.E., Stützle, T., 2017. Automatic Design of Hybrid Metaheuristics from Algorithmic Components. IRIDIA, Université Libre de Bruxelles, Belgium Tech. Rep.. http://iridia.ulb.ac.be/IridiaTrSeries/link/IridiaTr2017-012.pdf.

López-Ibáñez, M., Stützle, T., 2012. The automatic design of multi-objective ant colony optimization algorithms. IEEE Trans. Evol. Comput. 16 (6), 861–875. doi:10.1109/TEVC.2011.2182651.

López-Ibáñez, M., Stützle, T., 2014. Automatically improving the anytime behaviour of optimisation algorithms. Eur. J. Oper. Res. 235 (3), 569–582. doi:10.1016/j.ejor.2013.10.043.

Lundy, M., Mees, A., 1986. Convergence of an annealing algorithm. Math. Program 34 (1), 111–124.

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

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

Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A., Teller, E., 1953. Equation of state calculations by fast computing machines. J. Chem. Phys. 21, 1087–1092.

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

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

Moscato, P., 1989. On evolution, search, optimization, genetic algorithms and martial arts: towards memetic algorithms, caltech concurrent computation program. C3P Report 826, Caltech.

Moscato, P., Fontanari, J.F., 1990. Stochastic versus deterministic update in simulated annealing. Phys. Lett. A 146 (4), 204–208.

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

Nikolaev, A.G., Jacobson, S.H., 2010. Simulated annealing. In: Gendreau, Potvin (Eds.), Handbook of Metaheuristics, 2nd ed. Springer, New York, NY, pp. 1–39.

Ogbu, F.A., Smith, D.K., 1990. The application of the simulated annealing algorithm to the solution of the n/m/C max flowshop problem. Comput. Oper. Res. 17 (3), 243–253.

Orosz, J.E., Jacobson, S.H., 2002. Analysis of static simulated annealing algorithms. J. Optim. Theory Appl. 115 (1), 165–182.

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

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

Pan, Q.K., Ruiz, R., 2012. Local search methods for the flowshop scheduling problem with flowtime minimization. Eur. J. Oper. Res. 222 (1), 31–43.

Pan, Q.K., Ruiz, R., 2013. A comprehensive review and evaluation of permutation flowshop heuristics to minimize flowtime. Comput. Oper. Res. 40 (1), 117–128.

Paul, G., 2010. Comparative performance of tabu search and simulated annealing heuristics for the quadratic assignment problem. Oper. Res. Lett. 38 (6), 577–581.

Paulli, J., 1993. A computational comparison of simulated annealing and tabu search applied to the quadratic assignment problem. In: Vidal, R.V.V. (Ed.), Applied Simulated Annealing, 1993. Springer, pp. 201–215.

Pinedo, M.L., 2012. Scheduling: Theory, Algorithms, and Systems, 4th ed. Springer, New York, NY.

Pitzer, E., Beham, A., Affenzeller, M., 2013. Automatic algorithm selection for the quadratic assignment problem using fitness landscape analysis. In: European Conference on Evolutionary Computation in Combinatorial Optimization. Springer, pp. 109–120.

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

Rose, J., Klebsch, W., Wolf, J., 1990. Temperature measurement and equilibrium dynamics of simulated annealing placements. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 9 (3), 253–259.

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

Smith-Miles, K., 2008. Towards insightful algorithm selection for optimisation using meta-learning concepts. In: Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on. ieee, pp. 4118–4124.

Sörensen, K., 2015. Metaheuristics—the metaphor exposed. Int. Trans. Oper. Res. 22 (1), 3–18. doi:10.1111/itor.12001.

Sörensen, K., Sevaux, M., Glover, F., 2018. A history of metaheuristics. In: Martí, R., Pardalos, P.M., Resende, M.G.C. (Eds.), Handbook of Heuristics. Springer International Publishing, pp. 1–18.

Souilah, A., 1995. Simulated annealing for manufacturing systems layout design. Eur. J. Oper. Res. 82 (3), 592–614.

Strenski, P.N., Kirkpatrick, S., 1991. Analysis of finite length annealing schedules. Algorithmica 6 (1–6), 346–366.
Stützle, T., 2009. Some thoughts on engineering stochastic local search algorithms. In: Viana, A. (Ed.), Proceedings of the EU/MEeting 2009: Debating the Future: New Areas of Application and Innovative Approaches, pp. 47–52.
Szu, H., Hartley, R., 1987. Fast simulated annealing. Phys. Lett. A 122 (3), 157–162.
Taillard, E.D., 1993. Benchmarks for basic scheduling problems. Eur. J. Oper. Res. 64 (2), 278–285.
Taillard, E.D., 1995. Comparison of iterative searches for the quadratic assignment problem. Locat. Sci. 3 (2), 87–105.
Tam, K.Y., 1992. A simulated annealing algorithm for allocating space to manufacturing cells. Int. J. Prod. Res. 30 (1), 63–87.
Vallada, E., Ruiz, R., Framiñán, J.M., 2015. New hard benchmark for flowshop scheduling problems minimising makespan. Eur. J. Oper. Res. 240 (3), 666–677. doi:10.1016/j.ejor.2014.07.033.
Weyland, D., 2010. A rigorous analysis of the harmony search algorithm: how the research community can be misled by a “novel” methodology. Int. J. Appl. Metaheuristic Comput. 12 (2), 50–60.
Weyland, D., 2015. A critical analysis of the harmony search algorithm: how not to solve Sudoku. Oper. Res. Perspect. 2, 97–105.
Wright, M.N., Ziegler, A., 2017. Ranger: a fast implementation of random forests for high dimensional data in c++ and r. J. Stat. Softw. 77 (1), 1–17. doi:10.18637/jss.v077.i01.
Zilberstein, S., 1996. Using anytime algorithms in intelligent systems. AI Mag. 17 (3), 73–83.