RAIRO-Inf. Theor. Appl. **40** (2006) 353-369
DOI: 10.1051/ita:2006009

TOWARDS A THEORY OF PRACTICE IN METAHEURISTICS DESIGN: A MACHINE LEARNING PERSPECTIVE [*]

Mauro Birattari[1], Mark Zlochin[1] and Marco Dorigo[1]

Abstract. A number of methodological papers published during the last years testify that a need for a thorough revision of the research methodology is felt by the operations research community – see, for example, [Barr *et al., J. Heuristics* **1** (1995) 9–32; Eiben and Jelasity, *Proceedings of the 2002 Congress on Evolutionary Computation (CEC'2002)* 582–587; Hooker, *J. Heuristics* **1** (1995) 33–42; Rardin and Uzsoy, *J. Heuristics* **7** (2001) 261–304]. In particular, the performance evaluation of nondeterministic methods, including widely studied metaheuristics such as evolutionary computation and ant colony optimization, requires the definition of new experimental protocols. A careful and thorough analysis of the problem of evaluating metaheuristics reveals strong similarities between this problem and the problem of evaluating learning methods in the machine learning field. In this paper, we show that several conceptual tools commonly used in machine learning – such as, for example, the probabilistic notion of class of instances and the separation between the training and the testing datasets – fit naturally in the context of metaheuristics evaluation. Accordingly, we propose and discuss some principles inspired by the experimental practice in machine learning for guiding the performance evaluation of optimization algorithms. Among these principles, a clear separation between the instances that are used for tuning algorithms and those that are used in the actual evaluation is particularly important for a proper assessment.

Mathematics Subject Classification. 68T05, 68T20, 68W20, 68W40, 90C27.

[*] *This research has been supported by the ANTS project, an* Action de Recherche Concertée *funded by the Scientific Research Directorate of the French Community of Belgium. Mauro Birattari acknowledges the support of a fellowship granted by the* Metaheuristics Network, *a Research Training Network funded by the Improving Human Potential programme of the Commission of the European Community, grant HPRN-CT-1999-00106. Mark Zlochin acknowledges the support of a Training Site fellowship funded by the Improving Human Potential (IHP) programme of the Commission of the European Community, grant HPMT-CT-2000-00032. Marco Dorigo acknowledges support from the Belgian FNRS, of which he is a Research Director.*

[1] IRIDIA, Université Libre de Bruxelles, Brussels, Belgium; mbiro@ulb.ac.be; mzlochin@ulb.ac.be; mdorigo@ulb.ac.be

Introduction

At the current stage of development of the fields of operations research and combinatorial optimization, a thorough revision of the research methodology is required. This need is apparently shared by many members of the research community as it is testified by the interest raised during the last years by a number of methodological works such as those proposed in [1,10,18,30]. With this paper, we intend to complement the above mentioned works with a conceptual analysis of some fundamental issues that have remained so far under-explored. In particular, we propose an original point of view on the research methodology in combinatorial optimization that is inspired by the machine learning field.

What is wrong with the currently employed methods? The problem is best illustrated by an old physicists' joke:

> *One chap, who was very fond of horses and horse races, decided to be scientific about it and asked his physicist friend to make a scientific prediction of the outcome of the next race. An hour before the race he came to his friend, but the latter replied that he was still working on the problem. The same thing happened half an hour before the race... and then five minutes before... A week later the physicist, looking very proud of himself, came back with a pile of papers, featuring a lot of diagrams and equations.*
>
> *Our race-fond chap glanced at the diagrams and the equations: "What does all this mean?" – he asked. "Well, so far I have managed to solve the problem for spherical horses in vacuum..."*

It is our strong belief that a similar joke could spread throughout our own community: a non negligible part of operations research in the last two decades was concerned, at least in some sense, with spherical horses in vacuum. This claim can, perhaps, be clarified by considering the largely overlooked issue of *modeling* within operations research.

Obviously, the main source of interest in operations research stems (or, at least, stemmed in the early days of the field) from the practical applications of its methods. Moreover, it is apparent, but possibly under emphasized, that the problems traditionally considered in operations research are *abstractions* of problems actually encountered in practice. In other words, they are *mathematical models* of real-life problems. Indeed, traditional operations research models are better understood as a *hierarchy of abstractions*:

(1) At the lowest level of the hierarchy, we find straightforward *mathematical models* of some well defined practical problems. This first level is the one usually adopted by industrial researchers and practitioners: their goal is to solve a specific problem instance rather than generalize their results and understand the properties of a class of instances.

(2) At a second level of the hierarchy, we encounter classical problems such as the traveling salesman problem (TSP), the quadratic assignment problem (QAP), and so on. Each of these problems is defined as a class of specific

problem instances (as introduced at the previous level) that share similar characteristics, as for example the kind of constraints imposed on feasible solutions.

(3) Ascending further, we finally reach highly abstract problems, such as NK landscapes [20] or deceptive problems [15].

These three levels should not be seen as an exhaustive description of the hierarchy of abstraction but just as the three main steps of the ladder. Indeed, a finer-grained analysis would reveal that other levels of the hierarchy should be considered which lie in between. As an example, let us consider the case of the dynamic routing problem, which is often cast into a simpler and better understood static TSP. Such a model problem lies indeed between the first and the second level of the hierarchy. In the rest of the paper, it will be sufficient to restrict our attention to the three levels described above.

At this point, it seems beneficial to recall what were the reasons for this simplification and for the introduction of the various levels of the above described hierarchy of abstractions. The first reason is apparently the obvious aspiration to reduce the implementation effort in scientific research. A more scientifically sound reason is to try to isolate problem characteristics that have a significant impact on the performance of algorithms. This, in fact, was the main motivation for introducing the highly abstract problems mentioned above. For a more general discussion of the role and of the importance of modeling in optimization, we refer the reader to [24, 27].

Unfortunately, with time, the understanding that all these problems are but models has receded. This in turn resulted in the shift of the research focus. Research often became concentrated on the model *per se*, leading to the development of complexity theory, NP-completeness theory in particular [12]. Although complexity theory represented a major breakthrough in the field of operations research, it should be emphasized that this theory is concerned only with the worst-case difficulty of the whole class of model problems, rather than the particular problem instances actually encountered in practice (or even simplified models thereof). For example, the fact that the TSP is NP-hard only means that, unless $NP = P$, there is no polynomial algorithm, which finds the optimal solution for all the possible TSPs. This, however, does not imply anything about the difficulty of a particular class of real-life routing problems, or even of the corresponding subclass of TSPs used to model these routing problems.

There is however a positive side to this development. The NP-completeness theory played an important role in making clear that, without restricting the class of problems considered in the scientific research, it is impossible to predict the performance of algorithms on real-life problems. This idea was further reinforced by the non-dominance result known as *No Free Lunch Theorem* [35] that states that, under certain assumption, no optimization algorithm is better than any other, if the performance is averaged over all possible optimization problems. In particular, this means that no algorithm can be *uniformly* better than any other one. As a consequence, the combinatorial optimization field faced a significant increase in the amount of experimental work. This concerned in particular the so called *metaheuristics*, including simulated annealing [21], tabu search [13, 14], iterated local search [23], evolutionary algorithms [11, 15, 17, 31], and ant colony optimization [5, 7–9], for which the theoretical analysis seems particularly difficult. However, partly because the research on the models *per se* became a dominant approach, the experimental standards have little bearing to the actual design of algorithms in practice. This is not to say that the experimental research using abstract model problems has nothing to offer as far as practice is concerned. Insofar as the model manages to capture the essential characteristics of the practical problem, the experimental results with this abstract model can be very illuminating, provided that the experimental design is modeling the setting in which the algorithms are used in practice. In fact, as we argument in the following, a major problem with current research in metaheuristics is not the lack of adequate test problems, but rather their inadequate use in empirical studies.

The remainder of this paper is dedicated to the development of a formal framework for the design of metaheuristics and for their experimental analysis that captures the essential characteristics of practical operations research: the urgently needed *theory of practice*.

1. Motivations and goals

The need for a formal framework for the design and empirical analysis of metaheuristics became evident to us during our involvement in the *Metaheuristics Network*, a Research and Training Network funded by the Improving Human Potential program of the Commission of the European Community. The activity of the network included a large-scale empirical analysis and comparison of a number of metaheuristics, including evolutionary computation, ant colony optimization, iterated local search, tabu search, and simulated annealing. This analysis necessitated an in-depth study of the existing methodological literature – including articles discussing experiments with algorithms in general, such as [19, 26], and articles specifically concerned with metaheuristics, such as [1, 30]. In the process, it became clear that, while including many valuable observations and recommendations regarding the experimental design and the statistical analysis of the results, the existing works are often limited to general discussion and references to general-purpose statistical methods. Moreover, most of the existing works ignore the link between the combinatorial optimization research and real-life optimization, and do not question some common, though problematic, elements of the experimental practice.

Accordingly, this paper is orthogonal to the existing literature and our main concerns are related to the following two issues:

- the definition of a meaningful class-dependent measure of performance for the tested algorithms;
- the definition of an experimental methodology that guarantees obtaining a meaningful estimate of the chosen measure of performance.

These issues are complementary to those already addressed in the existing methodological literature. More specifically, we observe that the commonly adopted experimental practice renders the results of the performance evaluations both statistically and practically meaningless, partly because the performance criteria are not well-defined, and partly because statistically unsound tuning procedures are employed.

In the following, the required formal framework for the experimental evaluation of metaheuristics is derived from first principles. We start with an analysis of the real-life setting in which optimization algorithms are used. This analysis reveals a considerable similarity between the formal structure of the evaluation of metaheuristics and the typical machine learning problems. Accordingly, we observe that several conceptual tools commonly used in the statistical machine learning field, such as the probabilistic notion of class of instances or the separation between the training and the testing datasets, fit naturally in the context of metaheuristics evaluation. We address some possible criticisms against our approach, present a critical analysis of several existing experimental techniques commonly used in today's metaheuristics research, and conclude with an outline of future research directions.

2. The real-life setting

The main justification to the very existence of the combinatorial optimization field comes from the (sometimes forgotten) fact that traditional combinatorial optimization problems are abstract models of real-life problems encountered in practice. Consequently, it is commonly believed that the performance of an algorithm on these model problems is indicative of its performance on real-life problems. However, what is typically ignored is that, in order for the studies conducted on models to be of any practical relevance, it is absolutely necessary to mimic not only the characteristics of problems, but also the whole life-cycle that optimization algorithms follow in practice. We borrow here the concept of product *life-cycle* commonly considered in manufacturing. Given the immaterial nature of optimization algorithms, this concept assumes here the same connotations it assumes in the field of software engineering. For an organic presentation of this concept, we refer the reader to any manual of software engineering, for example [33]. In the real world, the life-cycle of optimization algorithms might be quite diverse. Different life-cycle models can be appropriate: linear (also known as cascade or waterfall), spiral, and so on. In this paper, we will implicitly consider a simple linear life-cycle model in which the algorithm undergoes a *design/tuning* phase and is then employed in *production*. Beside being the simplest life-cycle model, the linear model is also the building block composing more complex models such as the spiral which, in this context, can be described as a sequence of interleaved *design/tuning* phases and *production* phases: At the end of each production phase, sufficient information is gathered which can be employed in the following *design/tuning* phase for improving the algorithm. From this description of the spiral model, it is apparent

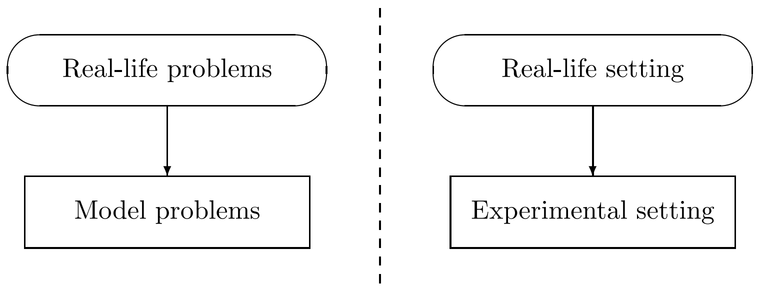

FIGURE 1. Model problems are abstractions of real-life problems. Similarly, the experimental setting should mimic the real-life situations in which optimization algorithms are developed and practically used.

that the conclusion we draw referring to the linear model immediately extend to the general case.

Obviously, there is no need to exactly mimic all the details of a life-cycle. Similarly to the problem modeling, some sort of abstraction is needed, as suggested in Figure 1. In the following, we examine the setting in which optimization algorithms are employed in real-life applications and we extract several high-level characteristics that we consider essential.

As a general rule, real-life optimization does not involve a single problem, but rather a class thereof: consider for example a scheduling problem that needs to be solved on a daily basis. It should be noted that the single-problem setting is simply a degenerate case, where the problem class has only one member. Even in this case, the results on a particular problem are interesting only as far as they suggest some more general trend.

As already mentioned, when a linear life-cycle model is assumed, an algorithm undergoes first a *development* phase and then the actual *production* phase. In the latter, the problems to be solved can be considered to be roughly homogeneous. An important observation is that the problems to be solved during the production phase are not available during the development phase. This implies the following rule that should be observed in any serious empirical study:

First experimental principle: *The problems used for assessing the performance of an algorithm cannot be used in the development of the algorithm itself.*

The need for introducing such a principle is best illustrated by the following *reductio ad absurdum* argumentation, which applies to the vast and practically relevant class of combinatorial optimization problems for which the solution space is finite, such as, for example, traveling salesman, quadratic assignment, timetabling, scheduling, etc. Let us consider the following trivial algorithm that takes as a parameter a natural number and, for any given problem instance, returns a fixed solution indexed by the parameter. Let us further assume that a single problem instance is used both for tuning the parameter of the algorithm and then for assessing the performance of the tuned algorithm. In this case, since by hypothesis the number of solutions of an instance is finite, provided we spend enough time in tuning the parameter – that is, in actually trying to solve the instance – the tuned algorithm will be typically able to return instantaneously a high quality solution of the instance under analysis. Moreover, if the size of the instance is not prohibitively large, the tuning of the parameter – that is, the solution of the instance – can be performed with an exact algorithm. This will guarantee that the tuned algorithm will be able to solve instantaneously to optimality the instance under analysis. Clearly, this hardly suggests that our trivial algorithm is of any practical value.

While these observations are obvious and might even be considered trivial by our reader, still the first experimental principle is routinely violated in the current literature, as we discuss in the following.

It should be emphasized that this principle does not imply that domain-specific knowledge cannot be used in the development of algorithms. Such knowledge can influence the development process in two ways. First, in case explicit knowledge is available (either based on domain theory or on the previous experience of the algorithm designer), it can clearly affect the design of the algorithm. Second, in many cases, problems similar to the ones used in production are available and pilot runs on these problems can be used to fine-tune the algorithm. These observations lead us to a second principle, which is complementary to the first:

Second experimental principle: *The designer can take into account any available domain-specific knowledge as well as make use of pilot studies on similar problems.*

Finally, in real-life applications, the time available for either the development or the production phases is limited, hence the corresponding *fairness principle*, also advocated in [30], should be observed in any comparative experiment:

Third experimental principle: *When comparing several algorithms, all the algorithms should make use of the available domain-specific knowledge, and equal computational effort should be invested in all the pilot studies. Similarly, in the test phase, all the algorithms should be compared on an equal computing time basis.*

Now, given the three principles described above, we are in a position to present a formal description of the experimental methodology.

3. The configuration problem

In this paper, we focus our attention on metaheuristics. Nonetheless, most of the methodological recommendations apply also to experimental studies of approximation or even exact algorithms.

As already pointed out in several methodologically-minded papers such as, for example [19, 26], any experimental study can only assess the performance of a *particular implementation* rather than a *general abstract algorithm.* It should be noted, however, that an implementation is defined not only by structural decisions such as problem representation, data-structures, and so on, but also by a *particular configuration*, that is, a certain set of values of the parameters of the algorithm as, for example, the evaporation rate and the sample size in ant colony optimization [6, 7], the tabu-list size in tabu search [13], or the mutation rate and the cross-over operator in evolutionary computation [15][1]. This leads to the following observation, which is often ignored in the current research:

> *Whenever the free parameters of the algorithms to be compared are fixed through some unspecified procedures, performance evaluation and comparison are only meaningful with respect to the particular configurations considered.*

This means that a conclusion like:

> 1) *Algorithm Alg_A is better than algorithm Alg_B,*

cannot be drawn from experiments, unless specific reference is made to the procedure used to select the configurations to be tested. In the general case, the only claim that can be formulated on the basis of an empirical comparison sounds like:

> 2) *The tested configuration of Alg_A performs better than the tested configuration of Alg_B.*

In order to compare Alg_A and Alg_B without restricting oneself to particular configurations, the appropriate statement to be tested should be

> 1') *The pair $Alg_A + Conf_1$ performs better than $Alg_B + Conf_2$.*

Where $Conf_1$ and $Conf_2$ are two configuration procedures. Whenever it is possible, one should adopt the *same configuration procedure* for both algorithms. This will contribute to isolate as much as possible the effects of the algorithmic ideas on the overall performance. In this case, the relevant statement to be tested is:

> 1") *The pair $Alg_A + Conf$ performs better than $Alg_B + Conf$.*

Also here, the superiority of one algorithm over the other cannot be assessed in absolute terms, but only with respect to the selected configuration procedure.

Unfortunately, this is not a common practice in the current literature and, in many papers involving comparisons of a newly proposed method against some previously published approach, a better performance of the new algorithm is often merely the result of a more careful tuning, that is, of a more sophisticated configuration procedure, rather than an indication of the superiority of the proposed algorithmic ideas. This has been shown, for example, in a thorough study concerning the performance of several metaheuristics on the MAX-SAT problem: when an equal tuning effort was invested in all the compared algorithms, the performance was virtually identical [38].

[1]It should be emphasized at this point that, in this paper, we are only concerned with static parameters, whose value is set by the algorithm designer. In case some parameters are set on the fly by the program itself, they are not considered to be part of the configuration. In the evolutionary computation literature, examples of such *on-the-fly tuning* can be found already in [31, 32] and, more recently, in [22, 34].

In light of this observation, the configuration problem becomes one of the central issues in the evaluation and comparison of metaheuristics. In fact, one can go as far as claiming that, unless the experimenter is only interested in a particular configuration, the configuration procedure becomes an inseparable part of the algorithm.

In the following, we present a formal framework for the description of the configuration problem. We observe that, when rigorously formulated, the configuration problem is extremely similar to the parameter selection problem in machine learning. Consequently, we show that several important conceptual tools from the statistical machine learning field, such as the probabilistic characterization of a class of problems and the parameter setting methodology, can be naturally translated into the optimization setting.

3.1. Notation

In order to give a formal definition of the general problem of configuring a metaheuristic, the following notations will be used:

- Θ is the set of candidate configurations and $\theta \in \Theta$ is a particular configuration.
- I is the set of possible instances and $i \in I$ is a particular instance.
- t is the available computation time, in abstract units, for the algorithm.
- $c(\theta, i, t)$ is the cost of the best solution found by running configuration θ on instance i for a time t. In the following, for the sake of a lighter notation, the dependency of the cost c on the time t will be often omitted. Since we wish to consider both deterministic and randomized algorithms, we assume that the cost is a random variable distributed according to some probability measure P_C, where C is the range of values possibly assumed by c. In the deterministic case, the probability measure P_C is degenerate, with the whole probability mass concentrated in a single point.

Now, in designing an optimization algorithm, one wishes to use a configuration, which is, informally speaking, "the best" one for the particular class of problems being solved. In order to be able to formally specify the configuration problem, a precise definition of *the best configuration* is needed. Consequently, one needs a formal definition of the concept of *class of problems* and some precisely defined class-dependent performance measure $\mathcal{C}(\theta)$. These two issues are addressed in the following sections.

3.2. What is a problem subclass?

For some years now, the metaheuristic research community reached the understanding that uniform dominance of one algorithm over another is highly unlikely, even for particular combinatorial problems. As a result, more and more studies attempt to compare algorithms on *problem subclasses.* However, up to now, the notion of problem subclass remained extremely vague, and the term is often used simply to refer to particular sets of benchmark problems. Clearly, this use of the term does not capture all of the properties that are intuitively expected from a problem subclass[2]. For example, in order to be of any practical interest, a problem subclass should be large – perhaps, even infinite. Moreover, there is typically no well-defined boundary delimiting the subclass in question. Finally, some sort of homogeneity is expected from the problems belonging to a same subclass.

Following the statistical machine learning field, where a similar notion of *class* is considered, a problem subclass in combinatorial optimization can be defined in a probabilistic way [2, 3]. In the following, we identify a subclass with a *probability measure P_I* defined over the set of possible instances. We will use the two concepts of *problem subclass* and of *probability measure P_I* interchangeably. For simplicity of presentation, in this paper the instances to be solved will be assumed to be generated from P_I in an independent way, although it is straightforward to generalize the analysis to a less restrictive setting such as, for example, a Markov chain generating mechanism – see [2] for a discussion of the issue.

A number of clarifications are needed here in order to prevent possible misunderstandings:

- While the probabilistic definition of a subclass is not explicitly used in the current research, it is nonetheless implicitly assumed in any experiment in which results are averaged over a number of test problems or in which some statistical test of significance is employed.
- The probabilistic definition of the problem subclass allows for the application of a precise meaning to the common belief that the benchmark problems are *representative* in some sense of the wider problem class. The precise meaning of *representative* in this case is that the benchmark problems are sampled from the probability distribution associated with the class.
- One should not confuse the high-level notion of a definition in probabilistic terms of a problem subclass with the practical implementation of a specific probabilistic benchmark generator. In fact, adopting a probabilistic definition of a problem subclass does not imply that the generating distribution is known to the algorithm designer. The instance-generating mechanism is only assumed to have a certain *qualitative form* – namely, independent and identically distributed (IID) sampling – without assuming anything about the particular *quantitative form* of this mechanism. The assumption of IID sampling is clearly an *abstraction* of the real-life instance generation, but it should be remembered that so are, for example, the model-problem instances typically considered in combinatorial optimization.

[2]Eiben and Jelasity [10] discuss the properties that a problem subclass should have for serving the purpose of testing and comparing evolutionary computation algorithms. In their discussion, Eiben and Jelasity do not need to provide a precise definition of the concept of problem subclass and therefore they use this concept in a somehow *informal* way. The definition of the concept that we propose in the following is motivated by their same concerns, but it has the merit of being formal and fully rigorous. It remains nevertheless perfectly consistent with the analysis proposed in [10].

3.3. Performance evaluation

Having given a precise meaning to the notion of problem subclass, we are now in the position to discuss the issues connected to the evaluation of the performance of a configuration of an algorithm with respect to a certain problem subclass.

First, let us consider a single run of the algorithm on a fixed problem instance. Let us assume that c is the cost of the best solution found within this single run. The observed cost, by itself, does not typically provide a meaningful evaluation of the quality of a solution. It is customary therefore to somehow refer the observed cost to the cost of the optimal solution[3]. Consequently, one is typically interested in the value of the *approximation error* with respect to the cost of the optimal solution $c_{opt}(i)$. The *absolute error* $|c - c_{opt}(i)|$ has many obvious shortcomings. For example, it is not invariant under a simple linear scaling of the cost function.

However, as noted in the seminal work by Zemel [36], also the more commonly used *relative approximation error*, $|c-c_{opt}(i)|/c_{opt}(i)$, is not invariant with respect to some trivial transformation of the problem. For example, in the TSP, an affine transformation of the weights leaves the problem essentially the same but changes the value of the relative error of all solutions. Accordingly, several alternatives were considered in the literature. For example, in [36], it was shown that:

$$e(c,i) = \frac{c - c_{opt}(i)}{c_{worst}(i) - c_{opt}(i)}, \tag{1}$$

where $c_{worst}(i)$ is the cost of the worst solution for instance i, is invariant under several trivial transformations of the problem. However, the problem with this measure of performance, referred to in the literature as *differential approximation measure* [4] or *z-approximation* [16], is that it requires calculation of the cost of the worst possible solution, which is often as difficult as finding $c_{opt}(i)$. To overcome this problem, an alternative error measure was proposed in [38]:

$$e(c,i) = \frac{c - c_{opt}(i)}{E_{unif}\, c(i) - c_{opt}(i)}, \tag{2}$$

where E_{unif} denotes expectation with respect to the uniform distribution of the solutions. While remaining invariant under trivial problem transformations, the error measure (2) has two important advantages over (1). First, $E_{unif}\, c(i)$ can be computed efficiently for many problems. Second, under (2), the expected error of a random solution is equal to 1, hence the proposed measure indicates how well a considered algorithm performs in comparison to the most trivial algorithm – a random generator of solutions. An additional useful consequence is that the error is automatically normalized across different problems.

[3]In order to simplify the presentation, in this paper, we assume that the cost of the optimal solution is known. See [30] for an excellent overview of different approaches for dealing with the situations in which the optimal solution is not known.

Having defined the error measure for a given solution of a particular instance, we may now consider a more general case of evaluation of the performance of a configuration with respect to a problem subclass. Recall that we have two sources of variability: the randomness of the algorithm, as captured by the distribution P_C, and the variability of the instances, described by the measure P_I. The question is how to define a *single* value of performance $\mathcal{C}(\theta) = \mathcal{C}(\theta|\Theta, I, P_I, P_C, t)$ on the basis of these two distributions[4]. One natural definition of a class-dependent performance measure is the expected value of the error $e(c, i)$ [2, 3]:

$$\mathcal{C}(\theta) = E_{I,C}\Big[e\big(c(\theta, i), i\big)\Big] = \int_I\int_C e\big(c(\theta, i), i\big)\,\mathrm{d}P_C(c|\theta, i)\,\mathrm{d}P_I(i). \tag{3}$$

Instead of the expectation, one can also consider other functionals of the error distribution, such as the median, the third inter-quartile or other quantiles, the probability of having an error below some threshold, and so on. The value $\mathcal{C}(\theta)$ has a double role. On the one hand, by definition, $\mathcal{C}(\theta)$ is used to measure the performance of the algorithm during the test phase (production). On the other hand, if $\mathcal{C}(\theta)$ were known for each θ, it could also be used during the design phase for selecting the best configuration. However, as a rule, the measures P_I and P_C are not explicitly available and hence the analytical computation of $\mathcal{C}(\theta)$ is not possible.

For the performance evaluation, it is a common practice to measure the performance of the algorithm by the average error over the test runs on the test instances. This measure is, in fact, nothing else than an unbiased finite-sample estimate of (3). It is tempting to apply the same approach also for the second use of $\mathcal{C}(\theta)$, namely the configuration problem. However, in light of the first experimental principle discussed in Section 2, the test instances cannot be used during the development phase. Still, according to the second experimental principle, one may use during the development stage other instances from the same problem class (when available). Consequently, one may approximate the criterion (3) by an empirical mean over the set of these training instances. Hence, similarly to machine learning, we propose an experimental methodology in which the algorithm's performance is measured using the *test error* and during the design phase one strives to choose a configuration with as low a *training error* as possible[5].

Now, the most straightforward way to practically carry out the training would be the *brute-force approach*, which tries every possible configuration and chooses the best one, that is, the one with the smallest training error. However, this approach can be infeasible in practice, due to the typically large size of the configuration space. Moreover, in many cases, the parameters are continuous and, accordingly, the configuration space is infinite. In order to deal with this problem,

[4]One may also consider a multi-objective optimization setting, with several relevant measures. This extension is beyond the scope of this paper.

[5]This methodology is also advocated in [10].

one may use the variety of parameter selection methods employed in machine learning. For example, in [2,3] it was shown that the so called *racing algorithms* [25,28] can be successfully adapted to the metaheuristics configuration task.

4. The proposed methodology and the alternatives

Let us summarize the general outline of the experimental design to be used in metaheuristics evaluation and comparisons:

- For every algorithm specify the configuration space, that is, the space of all the allowed combinations of parameters. If some domain knowledge is used to narrow down the configuration space, it should be equally applied to all the considered algorithms.
- Choose the configurations of the algorithms by using the **same configuration procedure**, possibly based on training instances generated from the same problem class, as the problems on which the algorithms are to be tested[6].
- Run the resulting configurations, with the **same computational resources** (*e.g.*, computation time[7], memory, etc.), on the available test instances and calculate the empirical approximation of the measure of performance defined by equation (3).

Hence, in the proposed methodology the training stage corresponds to the development phase in the real-life setting, while the testing stage corresponds to the production phase. Strictly speaking, the training stage is preceded by the actual implementation, but since the tuning is typically much more time-consuming, one may safely ignore the code-writing *per se*.

In the second step, the configuration procedure depends on the exact context of the experiment. For example, one may use some (possibly, approximate) training error minimization procedure, as discussed in the previous section. It could be argued that in some cases the training instances required for such a procedure are not available. In such cases, the alternatives implicitly considered in the literature are either to use the test instances themselves for choosing the tested configuration or to choose the configuration based solely on the prior experience with the algorithm. Let us examine these two alternatives in more details.

The first approach, namely the use of the test instances, is commonly used in the current literature. In its explicit form, some sort of average test error minimization procedure, typically based on some pilot studies, is used to choose the parameter values. In the second form, the configuration problem is not addressed explicitly,

[6]In order to guarantee that one is evaluating metaheuristics rather than a particular configuration method, one should use, for all the algorithms, the same configuration method, with the same computational resources and the same training instances. However, it might be also interesting to study the performance of *different configuration methods* on a *same metaheuristic*.

[7]Whenever a comparison involves an algorithm with a well-defined stopping condition (such as constructive algorithms or local search), an iterated version with the same overall computation time should be studied [30].

but rather the results of the best performing configuration are reported for every instance. However, both explicit and implicit forms of this approach violate the first experimental principle from Section 2, which is based on the observation that *the problems to be solved during the production phase are not available during the development phase.* Consequently, the results obtained with this approach do not produce any well-defined performance evaluation.

Another alternative is not to apply any configuration procedure, but rather to use some configuration, which was previously used in the literature or which is known to be "good" from the previous personal experience of the researcher. This approach can be criticized on two grounds. First, typically the configurations reported in the literature are chosen following some pilot experimentation, that is, one is simply using some empirical error minimization procedure implicitly. If the pilot studies were conducted using the same benchmark problem set, then one uses (perhaps, without even being aware of it) the test-instances-based approach, which we have just shown to be invalid. If, on the other hand, a different problem set was used for these pilot studies (or the experimental setting was different), then the employed configuration may be suboptimal for the problem at hand. In particular, this approach may easily introduce a bias towards one of the tested algorithms, due to uneven tuning effort. Clearly, the more recent algorithms are often more carefully tuned in comparison to the older ones, whose developers did not have access to today's computing power.

To summarize, most of the configuration procedures used today are based, either explicitly or implicitly, on the empirical error minimization. However, unlike the systematic methodology advocated in this paper, the existing approaches either involve some sort of "cheating", in the form of using the test instances during the development phase, or, alternatively, do not guarantee that all the algorithms are configured equally well.

Zlochin and Dorigo [38] adopted the methodology discussed here for comparing, on standard MAX-SAT benchmark problems, the performance of a number of modern metaheuristics belonging to the so-called *Model Based Search* approach [37]. Although the superiority of each of the considered approaches over the others had been claimed in previously published works, the results proposed in [38] have shown that, by providing equal tuning resources to all the compared methods, their performance became virtually identical.

5. Conclusions and future research

In this paper, we have presented a conceptual analysis of the real-life setting in which metaheuristics are used. Following this analysis, we have presented an experimental methodology based on a precisely defined notion of problem class and a related class-dependent measure of performance. We observe that the formal structure of the metaheuristic evaluation/comparison procedure is extremely similar to the one of machine learning. Accordingly, we propose an experimental methodology similar to the one used in statistical machine learning. We have also shown that existing alternative approaches all contain some fundamental flaws.

Clearly, the proposed experimental principles still leave much freedom to the researcher to specify an exact experimental design depending on the context and questions one wishes to address in the experiment. In this sense, as already mentioned earlier, this study is complementary to existing methodological literature. Even for the configuration problem, which was the main focus of this paper, many possible realizations of the proposed methodology are possible. For example, when studying the influence of some parameter on the performance of an algorithm, one can either fix the remaining parameters or, alternatively, one can optimize them for every considered value of the parameter of interest. Also, when one wishes to study the robustness of the algorithm with respect to the parameters' setting, obvious changes need to be made to the experimental design.

Naturally, in order to make the best use of the proposed methodology, efficient configuration procedures are needed. One approach, already mentioned above, is to make use of the racing algorithms [2, 3]. Other alternatives include using related methods developed in the simulation field (such as [29] and references therein) or applying general purpose stochastic optimization methods, such as ant colony optimization or genetic algorithms. These issues are the subject of ongoing research.

Finally, performance is not the only criterion for choosing a particular algorithm in practice. Clearly, the attractiveness of a particular algorithm is also determined by subjective factors such as simplicity of the algorithmic ideas and ease of implementation. Moreover, the human factor can, in principle, also affect the efficiency of the algorithm's implementation, hence influencing the measured performance. Still, while not discarding these *subjective* factors, this paper sheds some light on the *objective* problems plaguing the current research in metaheuristics and suggests a systematic methodology for overcoming these problems.

References

[1] R.S Barr, B.L. Golden, J.P. Kelly, M.G.C. Resende and W.R. Stewart, Designing and reporting computational experiments with heuristic methods. *J. Heuristics* **1** (1995) 9–32.

[2] M. Birattari, *The Problem of Tuning Metaheuristics, as Seen from a Machine Learning Perspective*. Ph.D. thesis, Université Libre de Bruxelles, Brussels, Belgium (2004).

[3] M. Birattari, T. Stützle, L. Paquete and K. Varrentrapp, A racing algorithm for configuring metaheuristics, in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002)*, edited by W.B. Langdon, *et al.* Morgan Kaufmann Publishers, San Francisco, CA (2002) 11–18.

[4] M. Demenage, P. Grisoni and V.Th. Paschos, Differential approximation algorithms for some combinatorial optimization problems. *Theoret. Comput. Sci.* **209** (1998) 107–122.

[5] M. Dorigo and C. Blum, Ant colony optimization theory: A survey. *Theoret. Comput. Sci.* **344** (2005) 243–278.

[6] M. Dorigo and G. Di Caro, The Ant Colony Optimization meta-heuristic, in *New Ideas in Optimization*, edited by D. Corne, M. Dorigo and F. Glover. McGraw Hill, London, UK (1999) 11–32.

[7] M. Dorigo, G. Di Caro and L. M. Gambardella, Ant algorithms for discrete optimization. *Artificial Life* **5** (1999) 137–172.

[8] M. Dorigo, V. Maniezzo and A. Colorni, Ant System: Optimization by a colony of cooperating agents. *IEEE Trans. Systems, Man, and Cybernetics – Part B* **26** (1996) 29–41.

[9] M. Dorigo and T. Stützle, *Ant Colony Optimization.* MIT Press, Cambridge, MA (2004).

[10] A.E. Eiben and M. Jelasity, A Critical Note on Experimental Research Methodology in EC, in *Proceedings of the 2002 Congress on Evolutionary Computation (CEC'2002)*, Piscataway, NJ, IEEE Press (2002) 582–587.

[11] L.J. Fogel, A.J. Owens and M.J. Walsh, *Artificial Intelligence Through Simulated Evolution.* John Wiley & Sons, New York, NY (1966).

[12] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of $\mathcal{NP}$-Completeness.* Freeman, San Francisco, CA (1979).

[13] F. Glover, Tabu search – part I. *ORSA J. Comput.* **1** (1989) 190–206.

[14] F. Glover, Tabu search – part II. *ORSA J. Comput.* **2** (1990) 4–32.

[15] D.E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, Reading, MA (1989).

[16] R. Hassin and S. Khuller, Z-approximations. *J. Algorithms* **41** (2001) 429–442.

[17] J. Holland, *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, MI (1975).

[18] J.N. Hooker, Testing heuristics: we have it all wrong. *J. Heuristics* **1** (1995) 33–42.

[19] D.S. Johnson, A theoretician's guide to the experimental analysis of algorithms, in *Data structures, near neighbor searches, and methodology: 5th and 6th DIMACS implementation challenges.* American Mathematical Society, Providence, RI (2002) 215–250.

[20] S.A. Kauffman, *The Origins of Order. Self-Organization and Selection in Evolution.* Oxford University Press, Oxford, UK (1993).

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

[22] K. Liang, X. Yao and C. Newton, Adapting self-adaptive parameters in evolutionary algorithms. *Appl. Intell.* **15** (2001) 171–180.

[23] H.R. Lourenço, O. Martin and T. Stützle, Iterated local search, in *Handbook of Metaheuristics. International Series in Operations Research & Management Science*, edited by F. Glover and G. Kochenberger. Kluwer Academic Publishers, Norwell, MA **57** (2002) 321–353.

[24] D.G. Luenberger, *Introduction to Linear and Nonlinear Programming.* Addison-Wesley Publishing Company, Reading, MA (1973).

[25] O. Maron and A.W. Moore, Hoeffding races: Accelerating model selection search for classification and function approximation, in *Advances in Neural Information Processing Systems*, edited by J.D. Cowan, G. Tesauro and J. Alspector. Morgan Kaufmann Publishers, San Francisco, CA **6** (1994) 59–66.

[26] C.C. McGeogh, Toward an experimental method for algorithm simulation. *INFORMS J. Comput.* **2** (1996) 1–15.

[27] Z. Michalewicz and D.B. Fogel, *How to Solve it: Modern Heuristics.* Springer-Verlag, Berlin, Germany (2000).

[28] A.W. Moore and M.S. Lee, Efficient algorithms for minimizing cross validation error, in *International Conference on Machine Learning.* Morgan Kaufmann Publishers, San Francisco, CA (1994) 190–198.

[29] B.L. Nelson, J. Swann, D. Goldsman and W. Song, Simple procedures for selecting the best simulated system when the number of alternatives is large. *Oper. Res.* **49** (2001) 950–963.

[30] R.R. Rardin and R. Uzsoy, Experimental evaluation of heuristic optimization algorithms: A tutorial. *J. Heuristics* **7** (2001) 261–304.

[31] I. Rechenberg, *Evolutionsstrategie – Optimierung technischer Systeme nach Prinzipien der biologischen Information.* Fromman Verlag, Freiburg, Germany (1973).

[32] H.-P. Schwefel, *Numerical Optimization of Computer Models.* John Wiley & Sons, Chichester, UK (1981).

[33] I. Sommerville, *Software Engineering.* Addison Wesley, Harlow, UK, sixth edition (2001).

[34] M. Toussaint, Self-adaptive exploration in evolutionary search. Technical Report IRINI-2001-05, Institut für Neuroinformatik, Ruhr-Universität Bochum, Bochum, Germany (2001).

[35] D.H. Wolpert and W.G. Macready, No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation* **1** (1997) 67–82.

[36] E. Zemel, Measuring the quality of approximate solutions to zero-one programming problems. *Math. Oper. Res.* **6** (1981) 319–332.

[37] M. Zlochin, M. Birattari, N. Meuleau and M. Dorigo, Model-based search for combinatorial optimization: A critical survey. *Ann. Oper. Res.* **131** (2004) 375–395.

[38] M. Zlochin and M. Dorigo, Model based search for combinatorial optimization: A comparative study, in *Parallel Problem Solving from Nature – PPSN VII*, edited by M. Guervós, J.J. *et al.* Springer Verlag, Berlin, Germany (2002) 651–661.