Procedia Computer Science

Volume 80, 2016, Pages 289–300

ICCS 2016. The International Conference on Computational Science

Assessing metaheuristics by means of random benchmarks

Pablo Rabanal, Ismael Rodríguez, Fernando Rubio*

Facultad Informática. Universidad Complutense de Madrid. 28040 Madrid, Spain
prabanal@fdi.ucm.es,isrodrig@sip.ucm.es,fernando@sip.ucm.es

Abstract

Typically, the performance of swarm and evolutionary methods is assessed by comparing their results when applied to some known finite benchmarks. In general, these metaheuristics depend on many parameter values and many possible exchangeable sub-steps, which yields a huge number of possible algorithm configurations. In this paper we argue that this high setup versatility lets developers expressively tune the method, in an ad-hoc way, to the target inputs to be solved, and hence to those in the benchmark under consideration. However, this does not imply properly solving any other input not considered in the benchmark. Several subtle ways to support that tuning (which can be consciously noticed by the developer, but can also be unconscious) are presented and discussed in the paper. Besides, as a possible alternative to using known finite benchmarks, we discuss the pros and cons of using random input generators, and we illustrate how to use such generators in a specific problem, MAX-3SAT. A general protocol to support the fair development of comparisons of metaheuristics based on random input generators is presented.

Keywords: Metaheuristics, benchmarks generation, over-tuning

1 Introduction

In many computational science problems, it is not only difficult to compute optimal solutions, but also to find good approximations. In these cases, metaheuristics are good choices to find reasonable solutions. In *swarm* and *evolutionary* optimization metaheuristics, simple entities iteratively construct a solution by interacting with each other according to some simple rules and local data [6, 7, 5, 13]. The standard method in the literature to assess the quality of metaheuristics consists in observing their performance when solving some known sets of problem instances (*benchmarks*). This way, the community can compare new proposed methods with old methods in terms of solutions quality, time performance, or stability for the *same* sets of library instances. Although fully checking a scientific hypothesis may trivially require infinite tests (and even this might not be enough [16]), method comparisons based on known benchmarks are assumed to be robust and fair. Unfortunately, as we will argue in this paper, evolutionary

*Work partially supported by projects TIN2012-39391-C04-04, TIN2015-67522-C3-3-R, and S2013/ICE-2731.

Selection and peer-review under responsibility of the Scientific Programme Committee of ICCS 2016

doi:10.1016/j.procs.2016.05.321

and swarm methods are easily customizable, so much as to let developers make them (almost) perfectly fit into *any* finite set of inputs. This problem dramatically affects the utility of known finite benchmarks to compare metaheuristics with each other, as a good performance for some benchmark does not imply, by itself, a good algorithm performance for other instances, even for *similar* instances. We will discuss how easily metaheuristics can be customized to a particular benchmark, we will propose an alternative comparison method based on random input generation, and we will study its pros and cons compared to known finite benchmarks.

If we consider the original definition of some evolutionary or swarm metaheuristic, and also the subsequent variations of the same method proposed in the literature, then we observe that it has a handful of parameters and steps where many alternatives can be used. Let us suppose that we are setting up a genetic algorithm. We can set parameter values such as the mutation probability; we can choose among dozens of crossover methods proposed in the literature; or we can even choose our particular way to represent candidate solutions into chromosomes, among many other possibilities. Moreover, possibilities are not constrained to variations proposed in the literature: a researcher can use a new crossover method invented by him, just because he observes that it is good for the problem he is solving. Consequently, algorithm developers have an *infinite* set of alternative sub-methods and parameter values to select and adjust.

By properly combining these abundant choices, any generic algorithm can be converted into a completely ad-hoc solution to handle a *specific* set of finite inputs. This does not necessarily imply a fraudulent intention: it can just be the undesired result of the legitimate purpose of improving the algorithm when focusing *too much* on a particular set of inputs. A similar issue appeared in other areas, though it was properly tackled long ago. For instance, in machine learning, instances used during the training process are separated from the instances used to asses the quality of the results. On the contrary, in the optimization field, using the same benchmark for both steps is customary. It is known that this choice increases the risk of over-fitting [9, 17, 15] to a very specific input. Hence, a metaheuristic can be over specialized for a specific set of instances by over-tuning it [3].

We propose discarding comparisons based on known finite set of inputs, and replacing them by comparisons based on instances created by *random input generators* for the problem under consideration. Using a random input generator has got some drawbacks as well. As we will argue later, random input generators are *necessarily* biased to some extent. Moreover, comparing the performance of two metaheuristics might be less straightforward if a random input generator is used. Some examples and a simple experiment will illustrate these problems. Despite these drawbacks, we argue that random input generators are probably a better choice to provide a fair assessment of evolutionary and swarm metaheuristics. In addition, a simple protocol for guaranteing the fair usage of comparisons based on dynamic benchmark generators is proposed (basically, it is based on making random generators record and publicly show all benchmarks that have been served, and to whom).

The rest of the paper is structured as follows. Next, we introduce our arguments against using finite fixed benchmarks, and we identify several methods to tune up an algorithm for a particular input or finite set of inputs. Then, in Section 3 we discuss the advantages and disadvantages of using random input generators, we show how to use them in a specific example (MAX-3SAT), and we report experimental results. Our conclusions are presented in Section 4.

2 Tuning Parameters to fit into a given finite benchmark

Several researchers have warned against the risks of over-tuning, in particular to highlight the necessity of assessing the performance of algorithms with sets of instances different to those