Computers & Operations Research 51 (2014) 190–199

Contents lists available at ScienceDirect

Computers & Operations Research

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

Grammar-based generation of stochastic local search heuristics through automatic algorithm configuration tools[☆]

Franco Mascia[*], Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Thomas Stützle

IRIDIA CoDE, Université libre de Bruxelles (ULB), 1050 Brussels, Belgium

ARTICLE INFO

Available online 2 June 2014

Keywords:
Heuristics
Grammatical evolution
Automatic algorithm configuration
Bin packing
Flowshop scheduling

ABSTRACT

Several grammar-based genetic programming algorithms have been proposed in the literature to automatically generate heuristics for hard optimization problems. These approaches specify the algorithmic building blocks and the way in which they can be combined in a grammar; the best heuristic for the problem being tackled is found by an evolutionary algorithm that searches in the algorithm design space defined by the grammar.

In this work, we propose a novel representation of the grammar by a sequence of categorical, integer, and real-valued parameters. We then use a tool for automatic algorithm configuration to search for the best algorithm for the problem at hand. Our experimental evaluation on the one-dimensional bin packing problem and the permutation flowshop problem with weighted tardiness objective shows that the proposed approach produces better algorithms than grammatical evolution, a well-established variant of grammar-based genetic programming. The reasons behind such improvement lie both in the representation proposed and in the method used to search the algorithm design space.

1. Introduction

Recent advances in the development of methods for the automatic configuration of optimization algorithms (also known as offline configuration) have shown the benefits of configuring optimization algorithms to specific problems [1–9], and several works in the literature use such methods to generate new optimization algorithms [10,11]. We call these latter approaches *top-down approaches* for automatic algorithm design, since they use a parametrized algorithmic framework to produce specific algorithms. Such frameworks are normally designed starting from a general (and usually complex) procedure and integrating alternative high-level algorithm components as fully-functioning blocks. In such a top-down approach, the parameter space is easily defined according to these alternative components. However, the flexibility of the framework is determined by the complexity of the general procedure.

A different, *bottom-up* approach for automatic algorithm design is used by grammar-based genetic programming [12–17]. In this approach, the algorithm design space is described by a set of production rules, and valid algorithms are instantiated by repeated applications of these rules. The benefit of a bottom-up approach is an increased flexibility when defining valid combinations of algorithmic components. Moreover, thanks to this flexibility, algorithmic components in bottom-up approaches are often more fine-grained than in top-down approaches. So far, the search for the best instantiation of the grammar has been done by genetic programming and other evolutionary methods.

In this paper, we investigate whether automatic algorithm configuration methods can be applied in a bottom-up approach. We answer this question in two steps. First, we replace the evolutionary algorithm in grammatical evolution (GE) [18], a type of grammar-based genetic programming, by an automatic configuration method, irace [6], using the same representation of the grammar as in the evolutionary algorithm. Second, we propose a method to generate a parameter space from a grammar, such that instantiations of the grammar can be represented by parameter configurations, which are more natural for automatic configuration methods. We compare these proposals with a pure GE method recently tested for the one-dimensional bin packing problem (1BPP) [17].

Experimental results show that irace using the parameter space generated by our method finds better algorithms than the GE method. We confirm these results by extending our analysis to the permutation flowshop with weighted tardiness problem (PFSP-WT).

This paper is structured as follows. First, we examine related works and we describe the GE method. Second, we explain our proposal for

[☆]This research has received funding from the COMEX project (Nr: P7156) within the Interuniversity Attraction Poles Programme of the Belgian Science Policy Office and the Meta-X project, (Nr: AUWB-08/13-ULB2) funded by the Scientific Research Directorate of the French Community of Belgium, and the MIBISOC network, an Initial Training Network funded by the European Commission, grant PITN-GA-2009-238819.

[*] Corresponding author.
E-mail addresses: fmascia@ulb.ac.be (F. Mascia), manuel.lopez-ibanez@ulb.ac.be (M. López-Ibáñez), jeremie.dubois-lacoste@ulb.ac.be (J. Dubois-Lacoste), stuetzle@ulb.ac.be (T. Stützle).

http://dx.doi.org/10.1016/j.cor.2014.05.020
0305-0548/© 2014 Elsevier Ltd. All rights reserved.

generating parameter spaces from a grammar. Third, we apply our proposal to the 1BPP, compare it to the GE method from [17] and test it on a new, more challenging case study on the PFSP-WT. Finally, we present our conclusions and discuss the new research directions opened by this paper.

2. Top-down vs. bottom-up approaches for automatic algorithm design

2.1. Top-down approaches

Automatic algorithm configuration methods were conceived for tuning the parameters of stochastic optimization algorithms given a set of training instances representative of the problem of interest. These methods allow algorithm designers to test many more parameter configurations than what is typically feasible when algorithms are tuned by hand in an ad-hoc manner. Moreover, automatic algorithm configuration methods avoid inherent biases of human designers when selecting which parameters to tune and which experiments to carry out.

When designing an optimization algorithm for a specific problem, algorithm designers are likely to implement alternative design choices (either new or coming from the literature) for further testing. It is only a small step to implement these design choices as parameters within an algorithm [19]. Such an algorithm quickly becomes an algorithm framework, with components that represent alternative design choices exposed as parameters of the framework. A particular instantiation of the parameters of such a framework leads to the selection of specific design choices and, hence, a specific algorithm. By applying automatic configuration to algorithm frameworks, it is therefore possible to automatically design algorithms for specific problems. We call this method a *top-down* approach, and we can find various examples in the literature.

KhudaBukhsh et al. [10] built a parametrized algorithmic framework for the satisfiability (SAT) problem from components of algorithms that had shown good results in previous editions of the SAT competition. By setting the parameters of the framework to specific values, they could instantiate various successful SAT solvers and generate new variants. They used ParamILS [3] to find the best variant to tackle specific types of SAT instances. The multi-objective ant colony optimization (MOACO) framework [11] follows a similar idea. Unique algorithmic components of various MOACO algorithms from the literature have been identified and incorporated into a common algorithmic framework, where alternative components may be selected by means of parameters. This framework was able to instantiate most of the MOACO algorithms from the literature and to generate hundreds of new algorithm designs. Using irace [6], it was possible to find a configuration of the framework that outperformed the MOACO algorithms from the literature for the bi-objective travelling salesman problem.

```
1: $x$ := randomized_first_fit()
2: for $i = 1$ to $100$ iterations do
3:     $x^*$ := ig_step($x$)
4:     if fitness($x^*$) < fitness($x$) then
5:         $x := x^*$
6:     end if
7: end for
8: return $x$
```

Fig. 1. Algorithmic scheme of the IG for the 1BPP.

2.2. Bottom-up approaches

A *bottom-up* approach combines algorithmic components, which can range from a single operator to fully-functioning procedures, to form valid expressions in a language, which can be either a pseudo-code or a specific programming language. In contrast to top-down approaches, in a bottom-up approach there is no need for a general algorithm framework, where many higher-level alternative design choices co-exist. This provides a greater flexibility when defining the space of valid algorithms, but it complicates the representation of valid algorithms and the search for the best one.

In bottom-up approaches, the space of valid algorithms is often given as a context-free grammar, that is, a set of production rules that describe how terminal and non-terminal symbols can be combined to produce valid sentences in the language. Fig. 2 shows an example grammar expressed in Backus–Naur Form (BNF), where each production rule is of the form <non−terminal>::=<expression>. Each rule describes how the non-terminal symbol on the left-hand side can be replaced by the expression on the right-hand side. Expressions are strings of terminal and/or non-terminal symbols. If there are alternative strings of symbols for the replacement of the non-terminal on the left-hand side, the alternative strings are separated by the symbol "|".

How to search for the best algorithm in the design space defined by the grammar and how to represent the sequence of derivation rules that represent an algorithm is the object of different methodologies in genetic programming (GP) [15,20]. In the context of generating stochastic optimization algorithms, Caseau et al. [12] design hybrid large neighborhood search algorithms for vehicle routing problems by using a genetic algorithm that applies crossover and mutation to a list of algebraic terms extracted from the grammar. Fukunaga [13,14] uses a strongly typed genetic programming algorithm to evolve Lisp-like S-expressions that represent local search heuristics for SAT. Three recent works [16,17,21] use *grammatical evolution* (GE) [18], which is a variant of GP that represents an instantiation of the grammar as a sequence of integers. Given its simplicity and its recent popularity for the automatic bottom-up design of algorithms, we explain this latter approach in more detail in the following section. In a hyper-heuristics context, the bottom-up approaches

```
      <start> ::= <select_bins> remove_items_from_bins() <repack>
<select_bins> ::= <type> | <type> <select_bins>
       <type> ::= highest_filled(<num>, <ignore>, <remove>)
                | lowest_filled(<num>, <ignore>, <remove>)
                | random_bins(<num>, <ignore>, <remove>)
                | gap_lessthan(<num>, <threshold>, <ignore>, <remove>)
                | num_of_items(<num>, <numitems>, <ignore>, <remove>)
        <num> ::= 2 | 5 | 10 | 20 | 50
  <threshold> ::= average | minimum | maximum
   <numitems> ::= 1 | 2 | 3 | 4 | 5 | 6
     <ignore> ::= 0.995 | 0.997 | 0.999 | 1.0 | 1.1
     <remove> ::= ALL | ONE
     <repack> ::= best-fit-decreasing | worst-fit-decreasing
                | first-fit-decreasing
```

Fig. 2. Grammar for generating ig_step in Fig. 1 for the 1BPP [17].