Knowledge-Based Systems 60 (2014) 73–81

Contents lists available at ScienceDirect

Knowledge-Based Systems

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

A filtering method for algorithm configuration based on consistency techniques

Ignacio Araya *, María-Cristina Riff

Depto. Informática, Universidad Técnica Federico Santa María, Av. España 1680, V región, Chile

ARTICLE INFO

Article history:
Received 9 January 2013
Received in revised form 2 January 2014
Accepted 6 January 2014
Available online 11 January 2014

Keywords:
Parameter tuning
Constraint satisfaction problems
Consistency techniques
Algorithm configuration
Algorithm design

ABSTRACT

Heuristic based algorithms are typically constructed following an iterative process in which the designer gradually introduces or modifies components or strategies whose performance is then tested by empirical evaluation on one or more sets of benchmark problems. This process often starts with some generic or broadly applicable problem solving method (e.g., metaheuristics, backtracking search), a new algorithmic idea or even an algorithm suggested by theoretical considerations. Then, through an iterative process, various combinations of components, methods and strategies are implemented/improved and tested. Even experienced designers often have to spend substantial amounts of time exploring and experimenting with different alternatives before obtaining an effective algorithm for a given problem.

In this work, we are interested in assisting the designer in this task. Considering that components, methods and strategies are generally associated with parameters and parameter values, we propose a method able to detect, through a fine-tuning process, ineffective and redundant components/strategies of an algorithm. The approach is a model-free method and applies simple consistency techniques in order to discard values from the domain of the parameters. We validate our approach with two algorithms for solving SAT and MIP problems.

1. Introduction

When we design an algorithm for problem solving, generally we address several important decisions related to its components, such as which variable-ordering heuristic to use, which transformation algorithm to use, and whether to include some specific methods. These decisions can be handled by algorithm parameters. Then, we can apply a fine-tuning process to determine which components are crucial for the algorithm performance.

There exist several automated procedures to find the optimal *instantiation of the algorithm parameters* (called *configuration* here) of a given algorithm. The problem of finding the optimal configuration is known as the *algorithm configuration problem* (AC) [17,3,9,16]. Automated procedures for solving this problem are called *configuration algorithms* or *configurators*, while algorithms whose parameters are tuned are called *target algorithms*.

Usually, finding near-optimal configurations allows good performance on large data sets. However, this does not directly assist with the design phase, specifically, with discarding, simplifying or just better understanding the components of the target algorithm. Several configuration algorithms exist that supply information about the parameters/components of the target algorithm in addition to finding near-optimal configurations.

Sampling methods (e.g., Latin-Square [21] and Taguchi Orthogonal Arrays [26]) take a representative sample of the configuration space using a full factorial design. The different configurations are analyzed to predict which parameter values work best and which are the most robust. CALIBRA [2] and *meta-GA* [21] reduce the search area from which new configurations are sampled at each iteration. Thus, they can be only used to find good configurations and are not well suited to analyzing the parameters.

Model-based methods construct a model based on a reduced set of configurations. This model predicts the performances of new configurations. A common approach is to use a regression method to predict the utility of a configuration [12,22,14]. Based on the model, it is possible to identify some features of the parameters. Sequential model-based methods allow for a better exploration of the configuration space. Coy et al. [11] proposes a procedure consisting of a model-based method followed by a local search procedure to optimize the parameter values. SPO [5,6] goes further. In each iteration, it generates a new set of configurations and predicts their utilities using the current model. The vectors with the highest predicted performance are used to update the model for the next iteration. Finally, the procedure returns an accurate model of the most promising areas. Model-based methods commonly use

* Corresponding author. Tel.: +56 322654962.
E-mail addresses: iaraya@inf.utfsm.cl (I. Araya), mcriff@inf.utfsm.cl (M.-C. Riff).

0950-7051/$ - see front matter © 2014 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.knosys.2014.01.005

Gaussian process models [23], thus they are limited to continuous parameters. SMAC [16] uses a new model class, based on the Hamming distance and random forests [10], to support categorical parameters. SMAC is also capable of handling multiple instances by integrating information about their features into the response surface model. Thus, the final model predicts the algorithm runtime for the configurations and instance features.

The objective of racing methods [20,8,7] is to identify the best configurations from a large set, performing a minimum number of tests. In addition to finding the best configuration (with a certain confidence level) these methods provide information to estimate the robustness to changes in parameter values. Iterative F-RACE [4,9] combines racing with model-based methods. It starts by using a reduced population that represents the whole space of configurations. Using F-RACE [8,9], it reduces this population until a certain condition is met. Then, a multi-variate normal distribution fit on the surviving vectors is used as a probability density function to sample points for a new population. The entire procedure is repeated until a termination criteria is met. Like sequential model-based methods, it finds good configurations and valuable information about the parameters.

Evolutionary algorithms also have been used to find configurations with high utility. The ParamILS framework [18,17] starts with a default parameter configuration and iteratively improves it by searching in a neighborhood defined as the variation of the value of only one parameter. It requires that the procedure configuration θ is better than configuration θ' be defined. A basic implementation of the procedure consists of comparing the average utilities over N runs, however, FocusedILS [17], an extension of the framework, replaces the procedure with racing.

Evolutionary algorithms are very good at finding high quality vectors; however, they do not provide any indication of the robustness of the target algorithm. REVAC [24] is a specific type of EA for configuration where the population approximates the density function of the most promising areas, similar to Iterative F-RACE. The function is decomposed by coordinates (i.e., blind for parameter interactions), but can be used to analyze the sensitivity and relevance of the different parameters. An extension of REVAC [13] finds values that can obtain good results across a large range of different instances.

Multi-objective methods allow us to take into account multiple problems or performance indicators. M-FETA [25] models the problem as a multi-objective optimization problem. It creates a Parameter Pareto Front that can be used to evaluate robustness to changes in problem definition, as well as performance using multiple performance criteria. An important added value of multi-objective methods with respect to other approaches lies in the insights regarding the applicability and *fallibility* (i.e., the relative difference between good and bad configuration performances [13]) of the target algorithm.

1.1. Our approach

In this paper we introduce NODOM-C (NOn-DOMinated Consistency algorithm), a model-free algorithm based on sampling. The goal of NODOM-C is to detect useful components and to help the algorithm designer to identify those that are ineffective.

NODOM-C may work with target algorithms containing a very large number of parameters. To our knowledge, only FocusedILS, GGA [3] and SMAC (a model-based approach) work with this type of target algorithm. Of the approaches that work with a large number of parameters, only SMAC may provide information about the parameters of the target algorithm. Iterative F-RACE also provides valuable information. Both of them, however, are sophisticated model-based approaches.

NODOM-C is based on sampling. However, unlike other sampling methods, which use sampling to detect promising parameter regions, our method uses sampling to remove and filter the configuration space. We thus refer to our approach as a *filtering method for algorithm configuration*. The main procedure iterates over all the parameters of the target algorithm. For each parameter, all its possible values are compared by using a sample where the rest of the parameters are set randomly from the parameter domains. Special care is taken to perform a *fair comparison*: two parameter values are compared using the same values for the rest of the parameters, the same instance set and the same seed. The values of the parameters, instances and seed change from one comparison to the other. A number m of comparisons is performed for every pair of parameters values. Then, a parameter value is eliminated from the domain if it is *dominated* by some other value (i.e., if in every comparison, it is worse than or equal to some other value) in the domain. If, after iterating over all the parameters, the configuration space has been reduced, then the whole procedure may be repeated to obtain further reductions. Finally, NODOM-C returns a reduced space of configurations and sets of comparable values for the parameters. From the results, the algorithm designer can easily deduce some interesting information: which components to discard and which components to use to perform a determined task (without impacting performance),

It is important to emphasize that, unlike other configuration algorithms, our goal is not to find the best configuration for the target algorithm for a given set of instances. (A best configuration works *relatively well* for each instance of the given set, however, it may omit some important components of the target algorithm which may be effective in a few instances of the set.) In fact, our goal is to detect *ineffective components* to reduce the complexity of the target algorithm while maintaining performance. Ineffective components mean components which are useless in *every single instance in the set*. The algorithm is inspired by regression models, where we can reduce the complexity of a model by reducing the number of variables involved without decreasing the quality of the models estimates.

In Section 2, we provide a formal definition of the algorithm configuration problem. Section 3 describes some concepts and definitions used in the filtering process. In Section 4, we detail NODOM-C, our filtering method for algorithm configuration. Experiments summarizing different aspects of the approach are shown in Section 5. Conclusions are given in Section 6.

2. The algorithm configuration problem

The Algorithm Configuration problem (AC) can be stated as follows: given a target algorithm, a set of parameters for the algorithm and a set of input data, find parameter values under which the algorithm achieves the best performance on the input data.

First, let us introduce some definitions. Let $p_1, \ldots, p_k$ be the parameters of the target algorithm. The domain of possible values for each parameter p_i is denoted by Θ_i. $\mathbf{\Theta} = \Theta_1 \times \cdots \times \Theta_d$ denotes the space of all feasible configurations, and $\theta = (\theta_1, \ldots, \theta_d) \in \mathbf{\Theta}$ corresponds to a parameter instantiation or configuration. We distinguish two main types of parameters:

- *Categorical parameters*: Those parameters which correspond to a choice inside the target algorithm. Components, methods and strategies are included in this category.
- *Numerical parameters*: Those parameters with real or integer domains. Some examples of numerical parameters are the population size and the mutation rate in a genetic algorithm, the temperature in a simulated annealing, and the required precision of an iterative method.