Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence

Online Search Algorithm Configuration

Tadhg Fitzgerald, Yuri Malitsky, and **Barry O'Sullivan**
Insight Centre for Data Analytics
University College Cork, Ireland
tadhg.fitzgerald@insight-centre.org

Kevin Tierney
DS&OR
University of Paderborn
Paderborn, Germany

Introduction

Search algorithms are often highly configurable, with a wide variety of settings that control various aspects of their behaviour. These parameters can influence how they operate in every way from the search strategy they use to the number of times backtracking is done. Finding the correct settings for these parameters can greatly improve an algorithm's performance, sometimes by orders-of-magnitude (Kadioglu et al. 2010; Hutter, Hoos, and Leyton-Brown 2012; Malitsky et al. 2013). The task of choosing good settings is known as algorithm configuration.

In the past, many solvers had the majority of their parameter values fixed by the developers, which greatly limited their potential. Since problems have a vast array of different features and characteristics, different strategies are required to solve each of them. This means that there is no single best set of parameters for all instances. For this reason, solvers should allow the user to select parameter values in order to achieve the best performance (Hoos 2012).

Traditionally when settings have been tuned, this process was performed by hand, however, this task is extremely time consuming and requires expert knowledge of the domain. Due to the large number of possible settings it is impossible to test even a small fraction of the configuration space systematically. This in turn has caused recent research to focus on the topic of automatic algorithm configuration.

Offline training on a set of representative instances is the technique generally used in automatic algorithm configuration. There have been a number of approaches taken to this problem. F-Race (Birattari 2002) and its successor, Iterated F-Race (Birattari et al. 2010), run a small set of parameters against each other, removing those that are found to be under performing. ParamILS (Hutter et al. 2009) conducts a focused iterated local search, followed by an improvement phase using a one-exchange neighborhood in an iterative first improvement search. Once a local optimum is encountered, the search is repeated from a new starting point. ParamILS gathers statistics on which parameters are important, and focuses on assigning them first. GGA (Ansotegui, Sellmann, and Tierney 2009) is a black-box tuner that uses a genetic algorithm approach to finding good parameter configurations. GGA uses a population split into two genders in order to balance exploitation and exploration of the search space. Sequential Model-based Algorithm Configuration (SMAC) (Hutter, Hoos, and Leyton-Brown 2011) generates a model over a solver's parameters to predict the likely performance if the parameters were to be used. The model is used to identify aspects of the parameter space such as which parameters are most important. Possible configurations are then generated according to the model and compete against the current incumbent, with the best configuration continuing to the next iteration.

While the above algorithm configuration techniques can lead to orders-of-magnitude improvement over the default parameters, they do suffer from a number of drawbacks. Firstly, a representative set of training instances must be available in order to learn good parameter sets. Secondly, the discovered settings do not adapt if the instance types change. Finally, all of the above techniques require a training period, usually a number of days, if not weeks.

Online Algorithm Configuration

Our research focuses on solutions to many of the problems that are inherent in current parameter tuning techniques in the form of online algorithm configuration. Online algorithm configuration is the task of learning new parameters as a stream of incoming problem instances is processed. There is no offline learning step and the parameters we use change if the types of instances we encounter change. Specifically, this tackles scenarios like those faced by a growing business that needs to solve a constant stream of problems e.g. container storage or machine allocation. To the best of our knowledge there has been no other research in this area.

Our current line of work explores what is possible in the area of real-time parameter tuning using the power of modern multi-core CPUs. The system tests multiple candidate parameterizations in parallel. A tournament-type process is used to decide the good settings. Multiple solvers are started on the same instance but with different parameter settings. The parameters which deliver a result in the fastest time (or in the case that all time-out, the parameterization with the highest objective value) are considered the winner.

We track how many times each set of parameters wins and then use this to decide when one parameterization is dominating another. Weaker settings are then replaced with a new set chosen at random. Currently we decide when a parameter is being dominated using a ratio method, e.g. given a ratio of 2, if parameterization A has 10 wins and parameterization B has 20, then parameterization A is considered the weaker. By including the best known parameter set (e.g. the default parameter set) among the starting set of parameters, we can guarantee a certain level of performance. This setting will not be removed until we find something which performs better than it. Furthermore, by tracking winning parameterizations as we are solving new instances, we are able to adapt to any changes in the problem instances as they occur.

Experiments

Combinatorial auctions involve users placing bids on a collection of goods, with the objective of finding the set of bids which can be satisfied and produce the maximum profit (Leyton-Brown, Pearson, and Shoham 2000). For our current experiments we use 2000 combinatorial auction instances sorted by increasing number of goods. This dataset provides a good representation of a growing company whose problem type is changing over time. We generated these instances using the Combinatorial Auction Test Suite (CATS).[1] Any trivial instances (solvable in under 30 seconds using CPLEX defaults) and any very difficult instances (taking more than 900 seconds) were removed. This is done so that we do not spend time tuning on parameters that can easily be solved using a presolver or instances that are too difficult to solve.

SMAC (Hutter, Hoos, and Leyton-Brown 2011) is the current state-of-the-art offline parameter tuning approach. We simulate the case where initially no training instances exist and the default parameters must be used to solve the first 200 instances. SMAC then uses the first 200 instances as its training set and is tuned over two days, after which each new instance is solved with the tuned parameters. We follow the typical practice of tuning multiple versions with SMAC and selecting the best based on training performance. In our case we present both the result of the best found SMAC parameters (SMAC-VB), and the average performance of the three parameterizations we trained (SMAC-Avg).

In our results we compare two versions of our system. The first has no information about the solver beforehand, and thus all six initial parameterizations are generated at random. We call this case Online-Cold. We also include Online-Warm, where one of the initial parameterizations contains the CPLEX default parameters and the other five are randomly generated. We run each version of our system three times, and present the average of all three runs in the plot.

Figure 1 shows the cumulative average solving time for CPLEX Default, SMAC virtual best, SMAC average, Online-Cold started and Online-Warm started. Notice that as the difficulty increases the average time for the default parameter set increases rapidly while our system increases at a much slower pace because we are able to adapt to the changes in the dataset. The performance of our approach almost matches that of SMAC despite the fact that our approach required no pre-training while SMAC spent 48 hours

[1] http://www.cs.ubc.ca/~kevinlb/CATS/

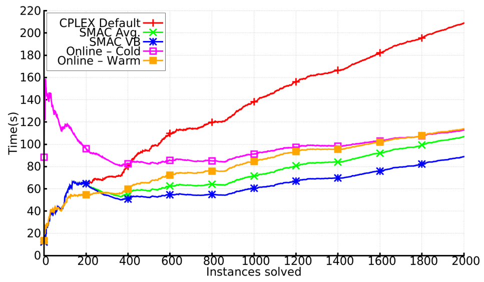

Figure 1: Cumulative average runtime for the dataset. The x-axis specifies the total number of instances observed, while y-axis specifies the time in seconds.

training offline.

Future Work

Our system currently chooses the next parameter set to try at random, but we are already able to achieve marked improvements over the default parameters. This method of selection is an obvious area for improvement. Choosing the next parameter using some sort of intelligent method should greatly improve our results. The way in which weak parameters are chosen is another area worthy of investigation. It is possible that some statistical technique will improve upon what our current ratio-based approach can achieve. This research is the first step in the field of online algorithm configuration.

Acknowledgments. This work has emanated from research conducted with the financial support of Science Foundation Ireland (SFI) under Grant SFI/12/RC/2289.

References

Ansotegui, C.; Sellmann, M.; and Tierney, K. 2009. A gender-based genetic algorithm for the automatic configuration of algorithms. In *Proceedings of CP*, 142–157.

Birattari, M.; Yuan, Z.; Balaprakash, P.; and Stützle, T. 2010. F-Race and Iterated F-Race: An Overview. In *Experimental Methods for the Analysis of Optimization Algorithms*. 311–336.

Birattari, M. 2002. A racing algorithm for configuring metaheuristics. In *Proceedings of GECCO*, 11–18.

Hoos, H. 2012. Programming by optimization. *Communications of the ACM* 55(2):70–80.

Hutter, F.; Hoos, H.; Leyton-Brown, K.; and Stuetzle, T. 2009. ParamILS: An automatic algorithm configuration framework. *JAIR* 36:267–306.

Hutter, F.; Hoos, H.; and Leyton-Brown, K. 2011. Sequential model-based optimization for general algorithm configuration. In *Proceedings of LION*, 507–523.

Hutter, F.; Hoos, H.; and Leyton-Brown, K. 2012. Parallel algorithm configuration. In *Proceedings of LION*, 55–70.

Kadioglu, S.; Malitsky, Y.; Sellmann, M.; and Tierney, K. 2010. ISAC - Instance-Specific Algorithm Configuration. In *Proceedings of ECAI*, 751–756.

Leyton-Brown, K.; Pearson, M.; and Shoham, Y. 2000. Towards a universal test suite for combinatorial auction algorithms. In *Proceedings of ACM-EC*, 66–76.

Malitsky, Y.; Mehta, D.; O'Sullivan, B.; and Simonis, H. 2013. Tuning parameters of large neighborhood search for the machine reassignment problem. In *Proceedings of CPAIOR*, 176–192.