Algorithm Configuration: Learning policies for the quick termination of poor performers

Daniel Karapetyan[1], Andrew J. Parkes[2], and Thomas Stützle[3]

[1] Institute for Analytics and Data Science, University of Essex, UK
[2] ASAP Research Group, School of Computer Science, University of Nottingham, UK
[3] IRIDIA, CoDE, Universit Libre de Bruxelles (ULB), Brussels, Belgium

Abstract. One way to speed up the algorithm configuration task is to use short runs instead of long runs as much as possible, but without discarding the configurations that eventually do well on the long runs. We consider the problem of selecting the top performing configurations of the Conditional Markov Chain Search (CMCS), a general algorithm schema that includes, for examples, VNS. We investigate how the structure of performance on short tests links with those on long tests, showing that significant differences arise between test domains. We propose a "performance envelope" method to exploit the links; that learns when runs should be terminated, but that automatically adapts to the domain.

1 Introduction

Careful configuration of algorithms can lead to a significant improvement in performance [1, and many others]. This is usually done by searching in the space of configurations and evaluating each configuration on a set of target instances. However, such instances are often large and will require long runs, so direct and complete usage of such intended instances problems will be overly time-consuming. A natural desire is that, in a justified fashion, we should be able to reduce the run times by exploiting the results of "short runs" in order to configure for "long runs": There is a need to learn how to extrapolate from "short" to "long". This suggests that machine learning methods should be applied to collections of such "short-run data", to analyse patterns, and so produce predictions for the performance in the long runs. This view suggests that for algorithm configuration at least 3 different 'generic' spaces are relevant:

- C "Configuration space" – the direct parameters of the algorithms.
- S "Short-run space" – the space of (detailed) results using short runs.
- L "Long-run space" – results from long runs.

A common procedure would be to do a (local) search in C-space, using fitnesses obtained from L-space. In this context, the usage of machine learning might be to develop a mapping from the C-space to the L-space, e.g. see SMAC [1]. In this paper, we instead study the potential for learning the mappings from S-space to L-space – aiming to exploit how short runs are able to predict longer term behaviours. The goal is to use such information to optimise the policies for when a trial of a particular configuration should be terminated – because there is high confidence it will not lead to a good final solution.

2 Experimental Setup

We used CMCS, a recent framework that defines the behaviour of a multi-component optimisation algorithm with a set of numeric parameters [2, 3]. We used three problem domains: the Simple Plant Location Problem (SPLP) [2], the Far From Most String Problem (FFMSP) (the details of our components, testbed, etc. are not yet published) and the Bipartite Boolean Quadratic Problem (BBQP) [3]. We generated all 'meaningful' 3-component configurations with deterministic control mechanism, thus ending up with a finite number of configurations. We do not include details of exactly what these mean (see [3, 2]), as for the purposes of this paper, these can simply be regarded as a categorical set of potential options, and defining the "C-space". The goal of the work is to find configurations that are the best performers, with respect to the long-run L-space, but exploiting their properties with respect to the short-run S-space in order to reduce the overall time budget.

The full time budget for each 'long' run was selected as 1024ms. Ten random instances were generated for each domain, with the size chosen to make them hard enough for this time budget. We then generated performance data[4] for each domain, by solving each of the ten instances by each of the configurations, and recording the solution quality at 1ms, 2ms, 4ms, ..., 1024ms. Objective values were scaled to $[0, 1]$ for each instance, and then averaged over the data instances. Hence, quality 0 (resp. 1) means that, for each instance, the configuration yielded solution as good (resp. bad) as any other configuration within the full 1024ms time budget.

In results for SPLP in Figure 1(a), the solid line shows the Performance Profile (PP) of the top configuration, i.e. how the solution quality of the best-in-L-space configuration improves over time. The other two lines, which we call *cutoff lines*, are from aggregating PPs of a set of configurations. To obtain an $x\%$ cutoff line, we select the top-$x\%$ of the configurations in the S-space and then produce the cutoff line as a combined worst-case PP for all those configurations. (All three lines are monotonic, as CMCS records the best-so-far solutions.) As hoped, there is a strong correlation between the short- and long-run performance of the top configurations; specifically, the cutoff lines drop relatively quickly, suggesting that there is a potential for a significant speed up by terminating configurations that perform poorly in the 'long' runs. To evaluate this, we plotted the ranks corresponding to the lines in Figure 1(b). The lowness of the solid line indicates the top configuration was among the top performers throughout the run. The drop in the cutoff lines indicates that the runs longer than $\sim$ 20ms are likely to be useful in early prediction of the 'long' performance. E.g., at 32ms, the 1% cutoff line potentially allows us to rule out 89% of all the configurations. Short-run performances do link to long-runs; hence, we can use this to quickly terminate configurations that are not likely to have good L-space performance. However, Figures 1(c,d), showing the results for the FFMSP domain, demonstrate that in other domains one may need longer short-runs to predict 'long' performance.

[4] A URL will be provided

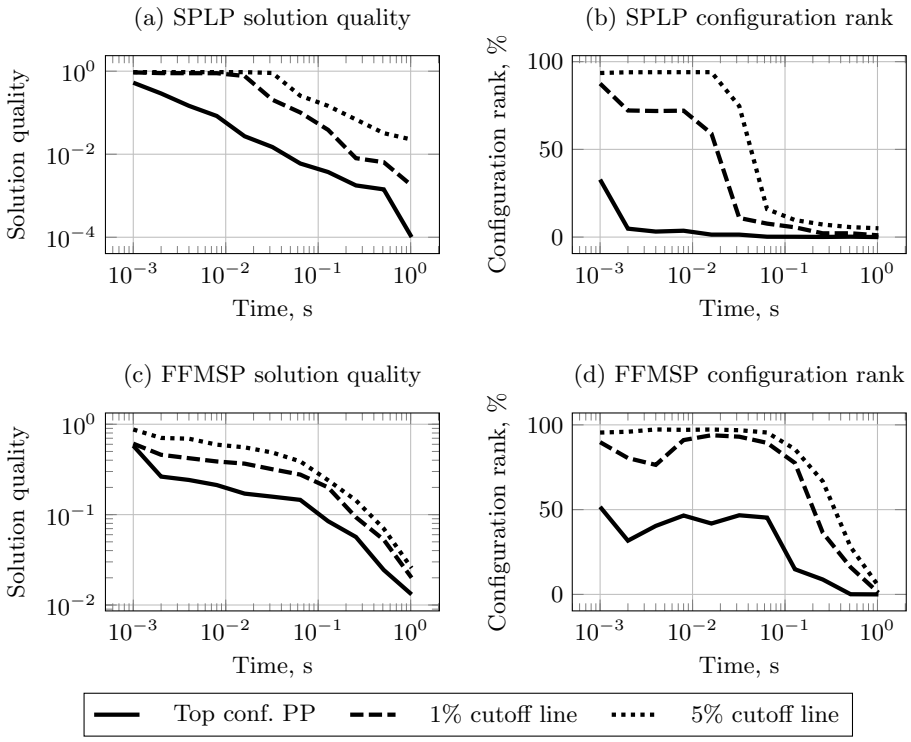

Fig. 1. Solution quality and configuration rank as they change throughout the run.

Finding the exact cutoff lines in advance is impractical, as it requires long runs for all the configurations. The heuristic below obtains and exploits a reasonably reliable 1% cutoff line much quicker. Specific parameter values used are given, but are changeable.

1. Randomly select 1% out of all the configurations and place them into a pool P and run full-time tests for each $c \in P$. Generate the cutoff line as the combined worst case PP for all $c \in P$.
2. First pass: For each previously untested configuration c, start a full-time test and gradually build its PP. If at any of 1ms, 2ms, 4ms, ..., 512ms its PP rises above the cutoff line by more than 20%, terminate the run. However, if c survives early termination, add c to P and remove the worst-performing (in the L-space) configuration from P; update the cutoff line.
3. Second pass: repeat Step 2, but scanning only through the configurations that were terminated early. This pass helps recovery from the potential bias at the beginning of the first pass if the initial cutoffs were too tight.
4. Return P as an approximation of the top 1% of all the configurations.

Table 1 gives experimental results. (We did not report the results for BBQP in Figure 1 due to lack of space). The 'Speed up' column tells how much quicker the heuristic is compared to evaluating all the configurations in S-space. The

Domain	#conf.	Speed up	Overlap (top conf.)	Overlap (1%)
SPLP	26,608	9.4	100%	99.3%
FFBSP	8,064	2.1	100%	99.9%
BBQP	9,860	9.2	100%	98.7%

Table 1. Accuracy of the cutoff approximation algorithm.

'Overlap (top conf.)' tells how often the heuristic finds the best performing configuration (in 100 experiments), and the 'Overlap (1%)' column tells the average overlap between the true top 1% configurations and the ones found by our heuristic. Note the speed up factor significantly depends on the domain; the SPLP and BBQP domains gave more than 9x speed up, though in FFMSP, the gain is much more modest. In all the domains, our heuristic procedure successfully found the best performing configuration every time, and was 99% accurate in finding the top-1% of configurations.

3 Conclusions and Future Work

The PPs arising from CMCS were shown to have sufficient structure in their behaviours such that a "performance envelope/cut-off" could be constructed to effectively determine when terminating a test run was safe, in that most of the good configurations would be found. Behaviours of the PPs differed between domains, suggesting that dynamic adaptive methods are needed. We gave a heuristic method for this, that automatically strengthens the cut-offs as more PPs are collected, and so reduces overall runtime.

The work here only used a PP of a simple linear aggregate over a set of different test instances. However, future extensions should consider "Performance Trajectories"; using the entire, time-dependent vector of the performances over the set of test instances. Initial explorations have taken such performance vectors at a given "short time point" and then considered them as feature vectors with labels given by the ultimate aggregate quality with a long runtime. For SPLP, standard classification methods did identify the regions in the S-vector space that lead to longer term good performance. This suggests that information in performance trajectories is also available that can be extracted by machine learning in order to classify (or cluster) behaviours and so potentially be used to optimise policies for when tests on configurations can be terminated.

References

1. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. LNCS 6683, 507–523 (2011), Proc. of LION 5.
2. Karapetyan, D., Goldengorin, B.: Conditional Markov Chain Search for the simple plant location problem improves upper bounds on twelve Korkel-Ghosh instances. Preprint in arXiv 1604.05636 (2018)
3. Karapetyan, D., Punnen, A.P., Parkes, A.J.: Markov chain methods for the bipartite boolean quadratic programming problem. European Journal of Operational Research 260(2), 494–506 (2017)