Designing a Portfolio of Parameter Configurations for Online Algorithm Selection

Aldy Gunawan and **Hoong Chuin Lau** and **Mustafa Mısır**
Singapore Management University
80 Stamford Road
Singapore 178902
{aldygunawan, hclau, mustafamisir}@smu.edu.sg

Abstract

Algorithm portfolios seek to determine an effective set of algorithms that can be used within an algorithm selection framework to solve problems. A limited number of these portfolio studies focus on generating different versions of a target algorithm using different parameter configurations. In this paper, we employ a Design of Experiments (DOE) approach to determine a promising range of values for each parameter of an algorithm. These ranges are further processed to determine a portfolio of parameter configurations, which would be used within two online Algorithm Selection approaches for solving different instances of a given combinatorial optimization problem effectively. We apply our approach on a Simulated Annealing-Tabu Search (SA-TS) hybrid algorithm for solving the Quadratic Assignment Problem (QAP) as well as an Iterated Local Search (ILS) on the Travelling Salesman Problem (TSP). We also generate a portfolio of parameter configurations using best-of-breed parameter tuning approaches directly for the comparison purpose. Experimental results show that our approach lead to improvements over best-of-breed parameter tuning approaches.

Introduction

Algorithm Selection (Rice 1976) deals with the problem of choosing the best algorithm among a set of algorithms for a given problem instance. The key idea is to determine a model that provides a mapping between instance features and performance of a group of algorithms on a set of instances. The resulting model is used to make performance predictions for new problem instances. In relation to algorithm selection, i.e. Algorithm Portfolios (e.g. (Gomes and Selman 2001)) primarily focus on determining a set of algorithms for an algorithm selection process. The goal is to choose these algorithms in a way that their strengths complement each other or provide algorithmic diversity that hedge against heterogeneity in problem instances in pretty much the same spirit as investment portfolios to reduce risks in economics and finance (Huberman, Lukose, and Hogg 1997). Meta-learning (Smith-Miles 2008) has also been proposed as a unified framework for considering the algorithm selection problem as a machine learning problem.

The idea of algorithm selection has also been investigated in the context of parameter tuning or configuration (Hutter et al. 2009). The goal is to determine a configuration for a target algorithm that will (hopefully) work well for given instances. Hyper-parameter tuning (Bergstra and Bengio 2012) has the same tuning objective but only for machine learning algorithms. In addition, the evolutionary algorithm and meta-heuristic community also study how to set the parameters of an algorithm. They categorise the methods for calibrating parameters as parameter tuning and parameter control (Eiben et al. 2007). Parameter tuning refers to the process taking place offline, while parameter control is concerned with the strategies for adapting the parameter values in an online manner via some rules or learning algorithms.

SATZilla (Xu et al. 2008) is a successful example of applying algorithm portfolios to solve the SAT problem, which has consistently ranked top in the various SAT competitions. Its success lies in its ability to derive an accurate runtime prediction model makes effective use of the problem-specific features of SAT. Hydra (Xu, Hoos, and Leyton-Brown 2010) is another portfolio-based algorithm selection method that combines with automatic configuration to solve combinatorial problems such as SAT effectively.

In terms of solving optimization problems, several algorithm configurators have been recently proposed. CALIBRA (Adenso-Diaz and Laguna 2006) combines Taguchi fractional experimental design and local search. ParamILS (Hutter et al. 2009), as explained above, applies iterated local search to find a single parameter configuration. Racing algorithms like F-Race (Birattari et al. 2002) look for effective parameter values by performing a race between different configurations. Instance-Specific Algorithm Configuration (ISAC) (Kadioglu et al. 2010) incorporates a G-means clustering algorithm for clustering instances with respect to the features with an existing parameter tuning method, i.e. GGA. GGA is used to configure an algorithm for each instance cluster and works like other case-based reasoning related algorithm selection approaches. (Lau, Xiao, and Halim 2009) proposed Randomized Convex Search (RCS) with the underlying assumption that the parameter configurations (points) lie inside the convex hull of a certain number of the best points. FocusedILS, derived from ParamILS, has been used to provide a number of different parameter configurations for a given single algorithm. It has also been applied

for designing multiple parameter configurations for a planner called Fast Downward, in (Seipp et al. 2012). The results are used for seven portfolio generation methods to build sequential portfolios. (Hutter, Hoos, and Leyton-Brown 2011) proposed a model-based approach, namely SMAC, that can be used to handle categorical parameters.

This paper seeks to extend the literature on automatic algorithm configuration. The experiments were conducted for combinatorial optimization problems. Rather than providing a single parameter configuration that works well in general or a pre-set schedule of algorithms, we work in the space of online algorithm selection and feeds this process with a portfolio of parameter configurations derived from Design of Experiments (DOE). Our aim is to develop a generic approach for designing algorithm portfolio of parameter configurations for use within an online algorithm selection process to solve combinatorial optimization problems. Our contributions are listed as follows:

- We apply DOE to build algorithm portfolios consisting of parameter configurations for a given target algorithm. Unlike configurators like ParamILS, F-Race or CALIBRA that provide single value for each parameter, DOE provides a subregion of values for each parameter (that are statistically important compared to other regions).
- We propose a random sampling approach to determine a portfolio of parameter configurations from the subregions. Even though methods like ISAC (Kadioglu et al. 2010) and Hydra (Xu, Hoos, and Leyton-Brown 2010) already deliver portfolios of configurations, these techniques run a tuner for multiple times, resulting in huge computational overheads. In our case, DOE and sampling is done once, which reduces computational overheads tremendously.
- We employ two online algorithm selection methods, namely simple random and learning automata. Again, the aforementioned portfolio-based methods that make use of parameter configurations are usually performed offline without any solution sharing, while our approach combines the strengths of multiple configurations by selecting them online and operating them on the same solution, which is different from standard algorithm configuration scenarios. Although dynamic portfolio methods (Kadioglu et al. 2011) perform online selection, they also ignore solution sharing. The empirical results on two NP-hard problems, namely the Quadratic Assignment Problem (QAP) and Travelling Salesman Problem (TSP), show the advantage of using multiple configurations and solution sharing in algorithm selection.

Algorithm Configuration Problem

The algorithm configuration problem (ACP) (Hutter, Hoos, and Stutzle 2007) is about determining a configuration for a given algorithm to perform well on a set of problem instances. An algorithm TA has k parameters to set, $P = \{pr_1, \ldots, pr_k\}$. Each of these parameters can be set from a particular range of values, $pr_i \in D_i$. The configuration space involves $C = D_1 \times \ldots \times D_k$ many possible configurations. The objective is to determine which configuration among such a usually large set can provide the best performance on an instance set, I. In this respect, the ACP can be considered an optimisation problem where a solution is a configuration c_i of the algorithm TA on I. One issue with this idea is related to solution evaluation. For assessing the quality of a c_i, TA with c_i should run on I. Although the required computational time for this task varies w.r.t. TA, I and the problem domain of I, it is computationally expensive in general. Heuristic-based search and optimisation techniques such as GGA (Ansótegui, Sellmann, and Tierney 2009) and ParamILS (Hutter et al. 2009) have been employed in order to overcome this issue.

Such tuning methods are eligible to deliver an effective configuration for a given algorithm. The idea of algorithm portfolios (Gomes and Selman 2001) have been used to take advantage of such techniques for building strong algorithm sets including algorithms with different configurations. Existing tuning based portfolio approaches like ISAC (Kadioglu et al. 2010) and Hydra (Xu, Hoos, and Leyton-Brown 2010) were designed to address the offline algorithm selection problem. They pursue to the goal of specifying the best single algorithm configuration for solving a particular problem instance. These systems require a set of features representing instances to select algorithms after delivering a set of configurations derived from a computationally expensive training phase. For instance, Hydra mentions that it took 70 CPU days to construct a portfolio of configurations. A similar tool used for a SAT solver, i.e. SATenstein (KhudaBukhsh et al. 2009), spent 240 CPU days.

Unlike these cases, the aim of this study is to build a portfolio of configurations that can be used in an online setting. The online nature of our approach can allow changing configurations while a selected configuration is fixed for the offline ones. Our system performs like a parameter tuning tool where any domain specific features are not needed. Besides that, the tuning process is faster since the tuning operation is performed once while the tuners used in the aforementioned portfolio approaches run for multiple times. Although our approach is not directly comparable with these offline portfolio methods due to its distinct design, a state-of-the art parameter tuning approach, i.e. ParamILS which is also used in Hydra, is experimented for comparison.

Background: Design of Experiments (DOE)

DOE has been widely used in scientific/engineering decision-making to select and determine the key parameters of a particular process (Montgomery 2005). One typical application of DOE includes selection of design parameters. A DOE-based framework was proposed in (Gunawan, Lau, and Lindawati 2011) to find ranges of parameters values which serve as input for configurators such as ParamILS (Hutter et al. 2009). In this paper, we utilize the first two phases of the framework, namely screening and exploitation, in order to provide promising sub-regions of configurations.

Suppose we have k parameters of the target algorithm TA to be tuned, where each parameter pr_i (discrete or continuous) lies within a numeric interval. In the screening phase, a complete 2^k factorial design is applied to identify m parameters ($m \leq k$) which have significant effects to the perfor-

mance of the target algorithm (the "important" parameters). This requires $n \times 2^k$ observations where n represents the number of replicates. Experiments are replicated to help identifying the sources of variability and to better estimate the true effects of treatments. The importance of a parameter pr_i can be defined by conducting the test of significance on the main effect of the parameter with a significance level, e.g. α = 10%. Furthermore, the ranking of the critical parameters is determined by the absolute values of the main effects of those parameters. Then, a reduced value range for each important parameter is returned while the remaining parameters are considered to be set to a constant value.

In the exploration phase, we treat the m important parameters determined from the screening phase, with the aim to find a promising range for them. The target algorithm is run with respect to the parameter configuration space θ which contains (2^m+1) possible parameter configurations with an additional setting defined by the centre points of all important parameters. By building a linear model of the response surface and applying the steepest descent, we obtain the promising range for each important parameter. In our paper, the run time of DOE depends on the number of configurations multiplied by the number of iterations required for TA (Gunawan, Lau, and Lindawati 2011).

The complete factorial design is expected to be computationally expensive if the target algorithm has a large number of parameters like CPLEX. Parameter search space reduction becomes essential for such cases (Montgomery 2005).

Solution Approach

Algorithm portfolios are often used with offline algorithm selection strategies. Our approach is basically in two parts - portfolio generation and online selection. We observe first that even though we work with a single target algorithm, the portfolio of parameter configurations give rise to different versions of the same algorithm, which enable us to generate a portfolio of low-level heuristics for algorithm selection. Our proposed approach is outlined in Algorithm 1.

Algorithm 1: Online Algorithm Portfolio Selection

TA: target algorithm with k parameters, θ: configuration space defined by the initial ranges of each parameter pr_i ($\forall i = 1, 2, \ldots, k$), I: instances, z: portfolio size

Portfolio Design

1 Run DOE to obtain a promising range $[l_i, u_i]$ for each parameter pr_i

2 Generate a portfolio of z parameter configurations from the promising ranges

Online Algorithm Selection

3 Apply a selection method (SR or LA)

Portfolio Design

Recall that DOE provides a promising range for each parameter pr_i based on steepest descent of a linear response surface. More precisely, we have the promising interval $[l_i, u_i]$ for each parameter pr_i. Given these intervals, we propose two different methods to generate a portfolio of parameter configurations. The first is to simply use a constant step size, namely, $[l_i, l_i + \delta, l_i + 2\delta, \ldots, u_i]$ where δ is a constant step size, which is arbitrarily set.

The second method is to perform intensification on the promising configuration space via random sampling. First, we generate n random samples of parameter configurations from the promising space. A contour plot is then generated where all sampled points (parameter configurations) having the same response are connected to produce the contour lines of the surface. This contour plot provides an approximate fitness landscape from which we can sample z points randomly, with a probability that decreases from one contour line to the next. More precisely, if the contour plot is divided into y contour lines, then we draw z_i samples from the region bounded by contour lines i and $i + 1$, where $z = z_1 + z_2 + \ldots + z_y$ and $z_i > z_{i+1}$ for all $1 \leq i \leq y - 1$.

Online Algorithm Selection

Even though the No Free Lunch theorem (Wolpert and Macready 1997) states that all algorithms perform similar on all possible problem instances that are closed under permutation, the search spaces of target problem instances usually do not have the property of 'closure under permutation'. Thus, performing (online) algorithm selection has potential to deliver better performance than each algorithm's separate execution. In (He, He, and Dong 2012), this idea is supported by a theoretical study indicating the advantage of using multiple mutation operators in evolutionary algorithms. A similar approach is followed to prove the benefit of accommodating multiple algorithms for hyper-heuristics in (Lehre and Özcan 2013). As a consequence, both experimental and theoretical studies suggest that online algorithm selection is effective.

We utilize two approaches to perform online algorithm selection, while it is possible to find others in hyper-heuristic (Burke et al. 2013), adaptive operator selection (Da Costa et al. 2008) and low-knowledge algorithm control (Carchrae and Beck 2005) studies. First, Simple Random (SR) (Cowling, Kendall, and Soubeiga 2001) randomly chooses a parameter configuration at each iteration. Although this is very naive approach, it can effectively manage a small-sized algorithm set. Second, Learning Automata (LA), a.k.a. stateless reinforcement learning, has been used to perform heuristic selection in (Mısır et al. 2010) due to its convergence property to a Nash equilibrium. Formally, a LA is described by a quadruple $\{A, \beta, p, U\}$. $A = \{a_1, \ldots, a_n\}$ is the set of actions available. p maintains the probabilities of choosing each of these actions. $\beta(t)$ is a random variable between 0 and 1 for the environmental response. U is a learning scheme used to update p (Thathachar and Sastry 2004).

A learning automaton operates iteratively by evaluating the feedback provided as the result of a selected action. Each action refers to an algorithm/configuration in our setting. The feedback from the environment is stated as the environmental response ($\beta(t)$) referring whether a selected action is favorable ($\beta(t) = 1$) or unfavorable ($\beta(t) = 0$). This feedback is then used to update the corresponding action proba-

bilities via general LA equations, i.e. Equation 1 and 2. The λ_1 and λ_2 values are the learning rates used to update the selection probabilities. The linear reward-inaction (L_{R-I}) update scheme is accommodated to change the probabilities. In this scheme, λ_2 is set to zero, meaning that no penalty occurs in case of an unfavorable action. Moreover, two feedback rules are defined, which are finding a new best solution and delivering an improved solution respectively. This means that two values are chosen for λ_1 depending on which of two feedbacks are received.

$$\begin{aligned} p_i(t+1) &= p_i(t) \quad +\lambda_1\ \beta(t)(1-p_i(t)) \\ &\quad\ -\lambda_2(1-\beta(t))p_i(t) \\ &\quad\ \text{if } a_i \text{ is the action taken at time step } t \end{aligned} \tag{1}$$

$$\begin{aligned} p_j(t+1) &= p_j(t) \quad -\lambda_1\ \beta(t)p_j(t) \\ &\quad\ +\lambda_2(1-\beta(t))[(r-1)^{-1}-p_j(t)] \\ &\quad\ \text{if } a_j \neq a_i \end{aligned} \tag{2}$$

where r is the number of actions in A.

Experimental Results

We perform experiments on two classical combinatorial optimization problems - the Quadratic Assignment Problem (QAP) and Traveling Salesman Problem (TSP). For each experiment, we perform four different selection methods: 1) simple random with constant step size (SR-Constant), 2) simple random with random sampling from the contour plot (SR-Contour), 3) learning automata with constant step-size (LA-Constant), and 4) learning automata with random sampling from the contour plot (LA-Contour). The learning rate (λ_1) of LA is set to 0.1 and 0.01 respectively for finding a new best solution or improving the current solution.

Quadratic Assignment Problem

QAP is the problem of finding a minimum cost allocation of facilities to locations, taking the costs as the sum of all distance-flow products. The target algorithm to solve QAP is the hybridization of Simulated Annealing and Tabu Search (SA-TS) (Ng, Gunawan, and Poh 2008) with four parameters to be tuned. The parameter configurations (Gunawan, Lau, and Lindawati 2011) used are detailed in Table 1.

Benchmark instances are taken from QAPLIB (Burkard, Karisch, and Rendl 1997). The instances are considered in four classes as in (Taillard 1995): unstructured (Group I), grid-based distance matrix (Group II), real-life instances (Group III) and real-life-like instances (Group IV). Due to the limitation of the target algorithm that can only handle symmetrical distance matrix, we focus on instances from the first three classes. By referring to (Gunawan, Lau, and Lindawati 2011) for classifying instances into training and testing instances, we conduct the experiments for two different set of instances: 1) testing instances and 2) all instances. Instance classes consist of 11, 24, 14 training and 5, 11, 7 testing instances, respectively.

The application of DOE screening phase yields the following result, e.g. for Group I. It reveals that two parameters ($Temp$ and $Alpha$) are statistically significant (with p-value $\leq 5\%$), while the effect of other parameters: $TabuLngth$ and $Limit$ are insignificant. Based on the coefficient value obtained, we determine the constant value for each insignificant parameter, e.g. the effect of parameter $Limit$ is 0.137, so the value of this parameter is set to its lower bound value, which is 0.01 (Table 1). Using this information in the DOE exploration phase, we find the promising planar region for both parameters ($Temp$ and $Alpha$). The final ranges for both parameters are summarized in Table 1. The last two columns show details on step sizes and number of random samples used to generate the portfolio.

The contour plots generated from 100 random samples, which is based on the DOE range, are shown in Figure 1. From the plots, we pick three and two different parameter configurations randomly from two promising regions, I and II, respectively.

In order to compare the performance of our proposed approach, we also run the target algorithm with constant parameter values generated by RCS and ParamILS. Both configurators also use the same inputs from DOE range (Table 1). The parameter values are obtained from (Gunawan, Lau, and Lindawati 2011). Take note that in order to ensure the fairness between RCS and ParamILS, we use the same number of iterations, e.g. 100 iterations. Based on our experiments, RCS takes longer computation time compared against ParamILS. The performance metric optimized by both configurators is the objective function value.

For each instance, we perform 10 runs and compare the percentage deviations of the average objective function values of the solutions obtained and the best objective function values against the best known/optimal solutions. The results are summarized in Table 2.

We also run ParamILS five times in order to generate five parameter configurations. Both selections methods, simple random and learning automata, are used to compare against points generated from the contour plot (SR-Contour and LA-Contour). The results are indicated as SR-ParamILS and LA-ParamILS. The purpose of this comparison is to show how generating a portfolio of algorithms with different parameter values generated from the contour plot outperforms a portfolio of algorithms using constant step-size and best-of-breed parameter tuning approaches (e.g. ParamILS).

All the selection methods listed in Table 2 have different computational expenses. The constant step-size based approaches do not have extra cost while the contour plot strategies have an additional sampling step requiring additional computational resources. The superior performance of the contour plot based methods show that sampling works. The online selection scenarios using ParamILS to generate portfolios, require more time for $s > 1$ in comparison to the contour plot based idea. The reason behind this computational performance difference is that ParamILS should be called s times unlike the contour plot idea which requires only one run of sampling. Thus, the contour plot based idea has clear computational advantage over ISAC, Hydra like configurator based portfolio approaches.

Wilcoxon Signed Rank Test is further conducted to test all pairwise differences between each algorithm selection

Table 1: The parameter space of the QAP

Parameters	Initial range	DOE range		Constant Step size	Contour Plot
Init. temperature ($Temp$)	[100, 7000]	Group I	[4378, 6378]	250	5 selected configurations
		Group II	[4238, 6238]	250	
		Group III	[4000, 6000]	250	
Cooling factor ($Alpha$)	[0.5, 0.95]	Group I	[0.935, 0.945]	0.005	
		Group II	[0.935, 0.945]	0.005	
		Group III	[0.85, 0.95]	0.05	
Tabu list length ($TabuLngth$)	[5, 10]	Group I	5	-	-
		Group II	6	-	-
		Group III	6	-	-
Diversification factor ($Limit$)	[0.01, 0.1]	Group I	0.01	-	-
		Group II	0.1	-	-
		Group III	0.1	-	-

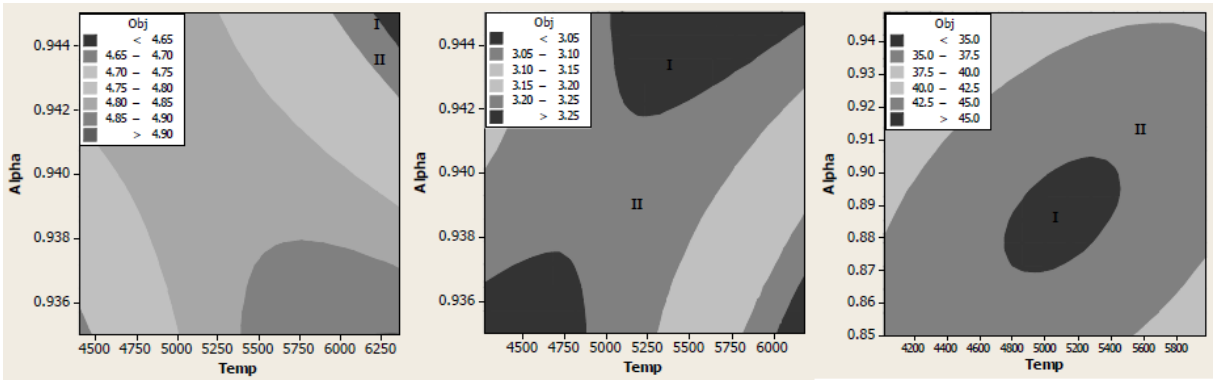

Figure 1: Contour Plot (Groups I, II and III, respectively)

approach. We summarize the ranks of each approach below. LA-Contour ≈ SR-Contour ≻ LA-Constant ≻ LA-ParamILS ≈ SR-Constant ≻ SR-ParamILS ≻ RCS ≈ ParamILS (Group I), LA-Constant ≻ LA-ParamILS ≻ SR-Contour ≈ LA-Contour ≻ SR-Constant ≻ SR-ParamILS ≻ RCS ≈ ParamILS (Group II) and SR-Contour ≻ LA-ParamILS ≻ SR-Constant ≈ LA-Constant ≻ LA-Contour ≻ SR-ParamILS ≻ ParamILS ≈ RCS (Group III).

In general, we obtain better results by generating a portfolio of algorithms with different parameter configurations, either by applying Simple Random or Learning Automata, compared against constant parameter values (RCS or ParamILS). In both Groups I and III, Learning Automata with random sampling (LA-Contour) outperforms others. On the other hand, Learning Automata with a constant step-size (LA-Constant) performs the best. Using the contour plot to generate a portfolio of promising parameter values also outperforms other approaches.

We additionally compare our approach against five single parameter configurations, from the generated portfolio. The portfolio usually provides better percentage deviations from the best known solutions, especially for the Group I-I instances. The percentage deviations for testing obtained by both SR-Contour and LA-Contour are only 0.149% and 0.151% respectively. The five single parameter configurations' distance to the best known solutions, however, range from 0.209% to 0.451% when each runs in isolation.

Lastly, we provide a glimpse of the effectiveness of the generated portfolio by examining the frequency distribution of selection, as shown in Figure 2. The cumulative frequency of choosing each parameter configuration vary over iterations, suggesting that different configurations are effectively used throughout the online selection process. And considering that LA outperformed both ParamILS and SR, we can conclude that the LA's learning process pays off.

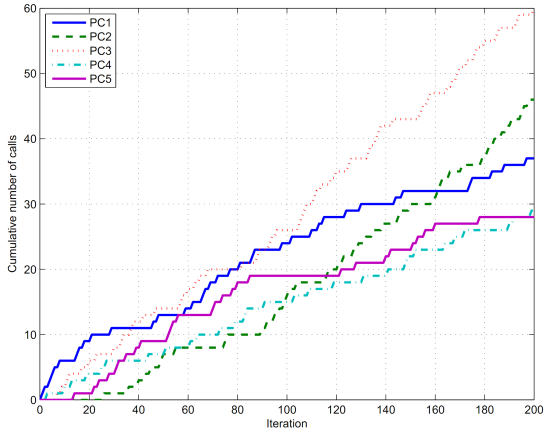

Figure 2: The effect of learning automata on parameter configuration selection while solving a QAP instance, tho150

Table 2: The performance of the tested approaches on the QAP instances with respect to the best known solutions (Param is ParamILS, Const is Constant, Cont is Contour)

Instances	Metric	Methods							
		RCS	**Param**	**SR-Param**	**LA-Param**	**SR-Const**	**LA-Const**	**SR-Cont**	**LA-Cont**
Group I	% Dev Avg (Test)	0.606	0.692	0.779	0.509	0.535	0.492	0.473	**0.471**
	% Dev Best (Test)	0.314	0.345	0.416	0.350	0.378	0.340	**0.301**	0.325
	% Dev Avg (All)	0.880	0.973	1.011	0.756	0.737	0.734	0.716	**0.700**
	% Dev Best (All)	0.505	0.581	0.618	0.470	0.449	**0.444**	**0.444**	0.476
Group II	% Dev Avg (Test)	0.214	0.210	0.394	0.139	0.168	**0.134**	0.149	0.151
	% Dev Best (Test)	0.030	0.024	0.061	0.025	0.022	0.018	**0.012**	0.032
	% Dev Avg (All)	0.262	0.247	0.417	**0.183**	0.192	0.189	0.189	**0.183**
	% Dev Best (All)	0.068	0.060	0.103	0.038	0.045	0.031	**0.030**	0.031
Group III	% Dev Avg (Test)	1.231	1.196	1.990	0.667	0.744	0.767	**0.636**	0.935
	% Dev Best (Test)	**0.000**	0.191	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**
	% Dev Avg (All)	2.921	2.848	3.414	**1.620**	2.563	1.824	2.516	1.700
	% Dev Best (All)	1.114	0.873	0.278	0.252	1.170	0.241	0.266	**0.231**

Table 3: The parameter space of the TSP

Parameters	Initial range	DOE Range	Constant Step size	Contour Plot
Maximum # iterations ($Iter_{max}$)	[100, 900]	[400, 600]	50	5 selected configurations
Perturbation strength (P_s)	[1, 10]	[1, 3]	1	
Non-improving moves tolerance (Tl_{nip})	[1, 10]	[4, 6]	1	
Perturbation choice (P_c)	[3, 4]	3	-	

Table 4: The performance of the tested approaches on the TSP instances with respect to the best known solutions (Param is ParamILS, Const is Constant, Cont is Contour)

Metric	Methods						
	Param	**SR-Param**	**LA-Param**	**SR-Const**	**SR-Cont**	**LA-Const**	**LA-Cont**
% Dev Avg (Test)	1.742	1.331	1.321	1.325	1.377	1.295	**1.291**
% Dev Best (Test)	0.852	0.752	0.787	0.792	0.768	0.704	**0.664**
% Dev Avg (All)	1.671	1.259	1.211	1.262	1.304	1.272	**1.207**
% Dev Best (All)	0.838	0.736	0.702	0.815	0.770	0.717	**0.684**

Travelling Salesman Problem

Travelling Salesman problem (TSP) is the problem of finding a tour that visits all cities exactly once that minimises the total distance travelled. In our experiment, Iterated Local Search (ILS) with a 4-Opt perturbation is used as the target algorithm (Halim, Yap, and Lau 2007).

Table 3 summarizes the list of the parameters to be tuned with their initial and final, after DOE, ranges. Similar to QAP, the last two columns provide how we generate the portfolio of configurations. 47 TSP instances out of 70 from TSPLIB are used for training while the rest (23 instances) are treated as testing instances. Five parameter configurations are also produced using ParamILS to compare them against the five points generated from the contour plot, indicated as SR-Param and LA-Param.

From Table 4, the algorithm selection methods are ranked as follows: LA-Contour $\approx$ LA-Constant $\succ$ LA-Param $\approx$ SR-Constant $\approx$ SR-Param $\approx$ SR-Contour $\succ$ ParamILS. We observe that our approach works well compared to existing configurators. In particular, the selection method using LA-Contour outperforms others. Besides that, our approach using the contour plot is superior than the portfolio of configurations generated by ParamILS.

Conclusion

This paper shows that DOE coupled with random sampling can automatically generate effective portfolios of parameter configurations that can be used by online selection. The computational results on two classical combinatorial optimisation problems showed the strength of our proposed method compared to state-of-the-art configurators such as ParamILS. We show that our proposed approach leads to improvements to two combinatorial optimization problems, QAP and TSP, when compared against single configurations.

In order to make further improvements, a new sampling strategy will be developed, particularly to speed-up the post-DOE stage. The current DOE approach will be modified such that larger parameter sets including conditional parameters can be handled. For taking advantage of the existing algorithm portfolio selection technologies, problem instance features will be used to derive better portfolios and support online algorithm selection. The test domains will be extended to satisfaction for comparing our system to the state-of-the-art configuration based portfolio techniques.

Acknowledgements

This research is supported by Singapore National Research Foundation under its International Research Centre @ Singapore Funding Initiative and administered by the IDM Programme Office, Media Development Authority (MDA).

References

Adenso-Diaz, B., and Laguna, M. 2006. Fine-tuning of algorithms using fractional experimental designs and local search. *OR* 54(1):99–114.

Ansótegui, C.; Sellmann, M.; and Tierney, K. 2009. A gender-based genetic algorithm for the automatic configuration of algorithms. *CP'09* 142–157.

Bergstra, J., and Bengio, Y. 2012. Random search for hyper-parameter optimization. *JMLR* 13:281–305.

Birattari, M.; Stützle, T.; Paquete, L.; and Varrentrapp, K. 2002. A racing algorithm for configuring metaheuristics. In *GECCO'02*, 11–18.

Burkard, R.; Karisch, S.; and Rendl, F. 1997. QAPLIB–a quadratic assignment problem library. *Journal of Global Optimization* 10(4):391–403.

Burke, E. K.; Gendreau, M.; Hyde, M.; Kendall, G.; Ochoa, G.; Ozcan, E.; and Qu, R. 2013. Hyper-heuristics: A survey of the state of the art. *JORS* 64:1695–1724.

Carchrae, T., and Beck, J. 2005. Applying machine learning to low-knowledge control of optimization algorithms. *Computational Intelligence* 21(4):372–387.

Cowling, P.; Kendall, G.; and Soubeiga, E. 2001. A hyper-heuristic approach to scheduling a sales summit. In *PATAT '00: Selected papers*, 176–190. Springer.

Da Costa, L.; Fialho, A.; Schoenauer, M.; and Sebag, M. 2008. Adaptive operator selection with dynamic multi-armed bandits. In *GECCO'08*, 913–920.

Eiben, A.; Michalewicz, Z.; Schoenauer, M.; and Smith, J. 2007. Parameter Control in Evolutionary Algorithms. *Parameter Setting in Evolutionary Algorithms* 19–46.

Gomes, C., and Selman, B. 2001. Algorithm portfolios. *AI* 126(1):43–62.

Gunawan, A.; Lau, H. C.; and Lindawati. 2011. Fine-tuning algorithm parameters using the design of experiments approach. In Coello Coello, C., ed., *LION'11*, LNCS, 278–292. Springer.

Halim, S.; Yap, R.; and Lau, H. 2007. An integrated white+black box approach for designing and tuning stochastic local search. In *CP'07*, volume 4741 of *LNCS*. Springer. 332–347.

He, J.; He, F.; and Dong, H. 2012. Pure strategy or mixed strategy? - an initial comparison of their asymptotic convergence rate and asymptotic hitting time. In Hao, J.-K., and Middendorf, M., eds., *EvoCOP'12*, volume 7245 of *LNCS*, 218–229.

Huberman, B.; Lukose, R.; and Hogg, T. 1997. An economics approach to hard computational problems. *Science* 275(5296):51.

Hutter, F.; Hoos, H. H.; Leyton-Brown, K.; and Stützle, T. 2009. ParamILS: an automatic algorithm configuration framework. *JAIR* 36(1):267–306.

Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2011. Sequential model-based optimization for general algorithm configuration. In *LION'11*, volume 6683 of *LNCS*. 507–523.

Hutter, F.; Hoos, H. H.; and Stutzle, T. 2007. Automatic algorithm configuration based on local search. In *AAAI'07*, volume 22, 1152.

Kadioglu, S.; Malitsky, Y.; Sellmann, M.; and Tierney, K. 2010. ISAC–instance-specific algorithm configuration. In *ECAI'10*, 751–756.

Kadioglu, S.; Malitsky, Y.; Sabharwal, A.; Samulowitz, H.; and Sellmann, M. 2011. Algorithm selection and scheduling. In *CP'11*, volume 6876 of *LNSC*, 454–469. Springer.

KhudaBukhsh, A. R.; Xu, L.; Hoos, H. H.; and Leyton-Brown, K. 2009. SATenstein: Automatically building local search sat solvers from components. In *IJCAI'09*, 517–524.

Lau, H. C.; Xiao, F.; and Halim, S. 2009. A framework for automated parameter tuning in heuristic design. In *MIC'09*.

Lehre, P. K., and Özcan, E. 2013. A runtime analysis of simple hyper-heuristics: To mix or not to mix operators. In *FOGA'13*.

Mısır, M.; Wauters, T.; Verbeeck, K.; and Vanden Berghe, G. 2010. A new learning hyper-heuristic for the traveling tournament problem. In Caserta, M., and Voss, S., eds., *Metaheuristics: Intelligent Problem Solving - MIC'09*. Springer.

Montgomery, D. 2005. *Design and Analysis of Expeirments*. John Wiley and Sons Inc, 6th Edition.

Ng, K. M.; Gunawan, A.; and Poh, K. L. 2008. A hybrid algorithm for the quadratic assignment problem. In *CSC'08*.

Rice, J. 1976. The algorithm selection problem. *Advances in computers* 15:65–118.

Seipp, J.; Braun, M.; Garimort, J.; and Helmert, M. 2012. Learning portfolios of automatically tuned planners. In *ICAPS'12*.

Smith-Miles, K. 2008. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys* 41(1):1–25.

Taillard, E. 1995. Comparison of iterative searches for the quadratic assignment problem. *Location Science* 3(2):87–105.

Thathachar, M., and Sastry, P. 2004. *Networks of Learning Automata: Techniques for Online Stochastic Optimization*. Kluwer Academic Publishers.

Wolpert, D., and Macready, W. 1997. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation* 1:67–82.

Xu, L.; Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2008. SATzilla: portfolio-based algorithm selection for SAT. *JAIR* 32(1):565–606.

Xu, L.; Hoos, H. H.; and Leyton-Brown, K. 2010. Hydra: Automatically configuring algorithms for portfolio-based selection. In *AAAI'10*, 210–216.