Predictive Parameter Control

Aldeida Aleti
Faculty of Information and Communication Technologies
Swinburne University of Technology
Hawthorn, VIC 3122, Australia
aaleti@swin.edu.au

Irene Moser
Faculty of Information and Communication Technologies
Swinburne University of Technology
Hawthorn, VIC 3122, Australia
imoser@swin.edu.au

ABSTRACT

In stochastic optimisation, all currently employed algorithms have to be parameterised to perform effectively. Users have to rely on approximate guidelines or, alternatively, undertake extensive prior tuning. This study introduces a novel method of parameter control, i.e. the dynamic and automated variation of values for parameters used in approximate algorithms. The method uses an evaluation of the recent performance of previously applied parameter values and predicts how likely each of the parameter values is to produce optimal outcomes in the next cycle of the algorithm. The resulting probability distribution is used to determine the parameter values for the following cycle. The results of our experiments show a consistently superior performance of two very different EA algorithms when they are parameterised using the predictive parameter control method.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Heuristic methods*

General Terms

Algorithms, Performance

Keywords

Component deployment, evolutionary algorithms, multiobjective problem, optimisation, parameter control, time series forecasting

1. INTRODUCTION

Due to their general applicability, stochastic optimisers such as Evolutionary Algorithms (EAs) are popular among scientists and engineers facing difficult combinatorial problems. These practitioners usually have little expertise in the AI field. They require guidance in the application of stochastic methods to their particular problems. However, even after many years of research into EAs and other stochastic approaches, there are no straightforward parameterisation guidelines for interdisciplinary users of these methods.

Formerly, EAs were seen as robust algorithms that exhibit approximately similiar performance over a wide range of problems [15]. The contemporary view on EAs however acknowledges that specific problems require specific parameter settings to achieve best possible performance [19]. It has been shown in the areas of search, optimisation and machine learning that there is no single algorithm setting that will outperform all other settings on every problem [30].

This problem has been addressed by many researchers [11, 15, 13, 27]. The existing approaches for setting the parameters of approximate optimisation methods can be divided in two groups: those who set the parameter values in advance of the search process, also called parameter tuning and those who change them during evolution, referred to as parameter control.

It has been empirically and theoretically demonstrated that different parameter settings are required not only for each different problem but also for each problem instance [2, 28, 27, 17]. Thus, when an optimisation algorithm is tuned prior to the optimisation stage, the selected parameters at the end of this process are not necessarily optimal. This suggests that adapting parameters dynamically during the optimisation process is likely to lead to better outcomes. Hence, parameter control can be seen as a more suitable option for the parameter optimisation process.

All parameter control methods we are currently aware of use a learning mechanism to derive parameter quality from recent performance, such as result quality in preceding iteration(s) (time $t-1$). One might argue that these approaches are 'one step behind', in that they represent the parameter value which is optimal for the previous iteration. Ideally, we would use a setting optimised for the beginning iteration (time t). It follows that an ideal method would attempt to predict successful parameter values for time t based on previous performance. Hence, our parameter control method uses time series forecasting.

2. BACKGROUND

All commonly used stochastic algorithms include parameters which have a significant effect on the performance of the algorithm. Recommendations for common values of these parameters usually exist in the literature. However, differ-

GECCO'11, July 12–16, 2011, Dublin, Ireland.

ent parameter settings are ideal for different problems or even different instances of the same problem. It is common practice in the AI community to parameterise an algorithm with values found during preliminary trials. This tuning practice, however, can be time-consuming and error-prone. It presents an even bigger challenge to interdisciplinary users of AI techniques. Alternatively, practitioners can choose to optimise their parameter settings at the same time as the problem is being optimised, which is known as parameter control.

2.1 Parameter Tuning

In optimisation, approximate methods such as EAs are often used because of a lack of knowledge about the fitness landscapes of the problem at hand. However, successful applications are dependent on the usage of suitable parameter values. In such situations, no guidelines regarding the parameter settings are usually available. As a consequence, practitioners tend to choose parameter values based on few trials in which various parameter settings are explored in an attempt to fine-tune an EA to a particular problem. As the number of training instances is usually limited, parameter quality estimates can be orders of magnitude too optimistic, which may lead to strongly impaired performance.

In more rigorous approaches, ANOVA or Design of Experiments have been used for parameter tuning [6, 31, 5, 20]. This can lead to a combinatorial explosion of parameter value combinations and require large amounts of computations.

Since exploring EA parameters is generally very time consuming and computationally expensive, it was suggested [10] that these parameter explorations should be carried out in a more general offline setting. Their goal is to find EA parameter configurations that are optimal for a class of problems, and then use these configurations in practice whenever such a problem arises [10].

In brief, parameter tuning has been criticised to be inappropriate for the following reasons [11]:

- Parameters are not independent and must be explored systematically in combination. This is practically impossible due to the large number of combinations.
- Parameter tuning is time-consuming and it provides no guarantees of optimality.
- It has been demonstrated that different values of parameters can be optimal at different stages of the search [2, 28, 27, 17]

2.2 Parameter Control

Parameter control addresses the need to change EA parameters as the search proceeds from a more diffuse global search to a more focussed converging local search. More specifically, it describes a process where trials start with initially suboptimal parameter values which are improved during the run-time of the algorithm. Parameter control methods can be classified into three categories [11]:

2.2.1 Deterministic parameter control

The parameter values are changed according to deterministic rules which are set a priori in a fashion similar to those used in simulating annealing. No feedback from the search is used. The rules are specified by the user. Usually the rules are based on time, i.e. a new rule is used every predefined number of iterations.

Deterministic parameter control is difficult to accomplish since it is not obvious to predict the number of generations the EA will take to converge, and set a parameter control schedule accordingly. It also faces similar difficulties as parameter tuning, as the parameter adaptation mechanism has to be defined a priori and does not take any notion of the actual progress of the search. One might argue that predefining intervals in the optimisation process and preassigning different parameter settings to each interval is likely to lead to suboptimal values for some problems or instances. For example, parameter reassignment intervals will ideally be shorter when optimising smaller problems, as the search progress will be faster when the problem complexity is lower.

2.2.2 Self-adaptive parameter control

In self-adaptive approaches [4, 12, 7], the parameters to be adapted are encoded in the individuals and are evolved in line with the traits of the solution to the problems. Individuals that have a high fitness will survive and propagate these successful parameter values which are assumed to be the reason for the good performance. Bäck [3] provides an insightful review of self-adaptive methods.

Self-adaptive parameter control is acknowledged as one of the most effective approaches to adapting the parameters of EAs, especially when performing continuous parameter optimisation [9]. However, these methods significantly increase the size of the search space and the complexity of the optimisation problem, since the EA has to find parameter values which facilitate the effective transmission of the genome information in addition to searching for good solutions to the problem [14]. Another drawback of self-adaptive control in EAs is the relative slowness of adaptation. It takes time before the optimal value is found [11]. Rudolph [26] explored the theoretical underpinnings of the self-adaptation of the mutation distribution finding that this method gets caught by nonglobal optima with positive probability even under an infinite time horizon.

2.2.3 Adaptive parameter control

A more successful approach is to monitor particular properties of an EA run and use changes in the properties as a signal to change parameter values. Although the rules are based on information from the search, the update mechanism to control the parameter values is decided by the user, rather than being part of the evolutionary cycle.

The 1/5 rule of Rechenberg [25] constitutes a first historical example for adaptive mutation step size control. Since then, the field of parameter control has been researched widely [16, 14, 29]. An insightful overview is given by Eiben et al. [11]. An empirical analysis of different adaptive parameter control methods is presented by Fialho et al. [14], concluding that the Dynamic Multi-Armed Bandit shows a good general performance, is stable w.r.t. the hyper-parameters and has less variance compared to other methods. Although this adaptive method involves many hyper-parameters that need to be tuned, the authors argue that the number of hyper-parameters does not depend on the number of the algorithm parameters that need to be controlled.

3. PREDICTIVE PARAMETER CONTROL

3.1 Approach

Given the inherent problems with deriving parameter values based on very recent, but indeed past, performance, we use a method of Predictive Parameter Control (PPC) that combines a measure of past performance with time series prediction to adjust the parameter values dynamically during the run.

Given a set V_i, $i = (1, ..., n)$ of algorithm parameters with values V_{ij}, $j = (1, ..., m)$, where m can be a discrete number or an interval of continuous numbers, the adaptive method has the task of deriving the optimal next value V_{ij} to optimise the influence of V_i on the performance of the algorithm.

As an example, when the mutation rate V_1 is dynamically adjusted, 4 intervals ($m = 4$) have been found to produce good results. V_{12} stands for a mutation rate sampled from the second interval. In the discrete case of optimising the type of mutation operator V_1, V_{12} stands for the second operator.

The probability of choosing values from a particular interval depends on the previous performance of values sampled from this interval. Values from intervals which have not been used have a small initial probability of being chosen. When a value is used but does not produce any above-average outcome, its probability of being chosen reverts to the initial value.

To explore the parameter space, k independent algorithm instances are started in parallel. Each optimisation process optimises the same problem instance. The processes set their adaptive parameters for an iteration at a time, choosing the respective value probabilistically from the distribution obtained from the performance-based prediction.

At the end of the iteration consisting of e function evaluations each, the current level of fitness achieved by the algorithm is evaluated. The probabilities of success of each parameter value are calculated based on their performance in the past iterations. The success rates are added to the history list of each parameter value. The previous success rates are used to infer a trend which helps determine the probability of each parameter value of being chosen for the next iteration.

3.2 Forecasting the Next Distribution

At the end of an iteration, the average solution fitness among all instances is established. Each value interval keeps a history list of success rates from all iterations. Algorithm instances which produced above-average population fitness increase are considered successful. For each parameter value or interval, the number of times the value has been used u_{ij} and the number of times the value led to a success u^s_{ij} (above-average increase in population fitness) are established. The success rate is calculated using eq. 1.

$$P_t(V_{ij} \rightarrow success) = \frac{u^s_{ij}}{u_{ij}} \tag{1}$$

Instead of applying the success rates of the last iteration directly as the probability of using the parameter value in the next iteration, we apply time series prediction as a method of forecasting the probabilities to be applied in the stochastic choice of next parameter values.

The least squares regression method assumes linearity of values. For the performance of the parameter values, our observation is that the development is piecewise linear, as can be observed in Figure 1.

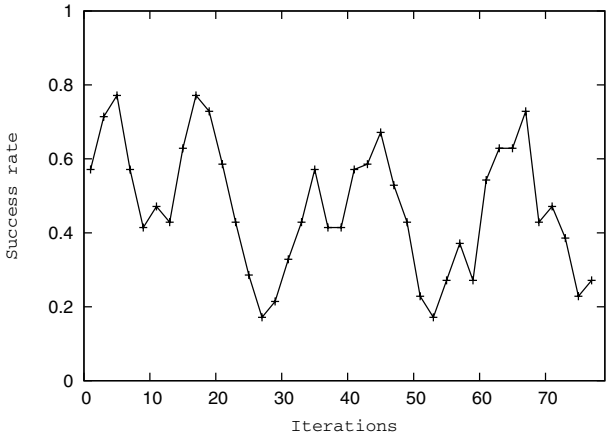


Figure 1: Success rates in 80 iterations for the last interval of crossover.

Fitting the last values of the history to a regression line makes it possible to derive the next value from an average history development using eq. 2.

$$P_{t+1}(V_{ij} \rightarrow success) = a + b * (t + 1) \tag{2}$$

The independent variable in this regression is t, the iteration counter. a is the intercept and b represents the slope.

3.3 Algorithm

Algorithm 1: Predictive parameter control

```
for n iterations of e function evaluations each do
    choose parameter values for k algorithms according
    to probabilities P_t(V_ij → success);
    execute k instances of the algorithm;
    calculate average fitness increase f̄(x) of k trial
    outcomes;
    foreach parameter V_i, i ∈ n do
        foreach algorithm instance in k do
            if instance used parameter in range j then
                increment u_ij ;
                if fitness increase of instance population
                f(x) > f̄(x) then
                    increment u^s_ij;
    foreach parameter V_i, i ∈ n do
        foreach parameter value V_ij, j ∈ m do
            add u^s_ij / u_ij to probability history;
    submit probability history to least squares
    regression;
    calculate P_{t+1}(V_ij → success) = a + b * (t + 1) ;
    set probabilities P_{t+1}(V_ij → success) for the next
    iteration;
```

4. EXPERIMENTAL SETTING

For the experimental validation we have chosen two problems, the generally accepted Royal Road problem which was especially designed for testing EAs, and a relevant practical problem faced by the automotive industry, the component deployment problem. The problems were chosen due to their dissimilarity, which enables a more informed judgement as to the portability of the approach when applied to an EA.

4.1 The Royal Road Problem

The Royal Road functions were introduced by Mitchell, Forrest, and Holland [18]. They were designed as functions that would be simple for an EA to optimise, but difficult for a hillclimber.

The function of the form $F : \{0,1\}^l \rightarrow \mathbb{R}$ is used to define a search task in which one wants to locate strings that produce high fitness values. The string is composed of 2^k non-overlapping contiguous sections each of length $b + g$, where b is known as the block and g is known as the gap. In the fitness calculation, only the bits in the block part are considered, whereas the gaps make no contribution.

Higher order schemata are formed from sets of the base level blocks, referred to in terms of levels, where the base level containing the initial blocks is level 0. The fitness calculation proceeds in two steps, the part calculation and the bonus calculation. The overall fitness assigned to the string is the sum of these two calculations.

4.1.1 Fitness calculation

The *part calculation* considers each block individually, assigning a fitness score. The scores are later summed to produce the total part contribution to the overall fitness. The fitness of each block is based entirely on the number of 1 bits it contains. The aim is to reward bits equal to 1 up to m consecutive bits, which adds to the block's fitness by v. For example, a block with three $1's$ would have fitness $3 * v$. If a block contains more than m 1's, but less than b 1's, it receives $-v$ for each 1 over the limit. With the default settings $m = 4, v = 0.02$, a block with six 1's is assigned a fitness of: $(6-4)*-0.02 = -0.04$. Finally, if a block consists entirely of 1's it is considered complete. This block receives nothing from the part calculation and will be rewarded in the bonus calculation.

The aim of the *bonus calculation* is to reward complete blocks and some combinations of complete blocks. Holland gives rewards for attaining a certain level. At the lowest level, rewards are given for complete blocks. If such a block exists, it receives a fitness equal to u^*. Any additional complete blocks receive a fitness of u. With Holland's defaults, $k = 4, b = 8, g = 7$, there are 16 regions of length 15, giving an overall string length of 240.

4.2 The component deployment problem

Component deployment in embedded systems [1] refers to the allocation of software components to hardware resources and the assignment of inter-component communication links. It can be viewed as a constrained generalised assignment problem (GAP). The decisions regarding the deployment architecture of an embedded system are very important, since they influence not only the functional attributes, but also the quality of the resulting system, which, as pointed out in the literature, e.g. by Papadopoulos and Grante [21], is at least as important as its functionality. The quality of the system is commonly measured in terms of non-functional attributes such as safety, reliability, performance and maintainability. We model the embedded system using inputs from our collaboration with the Cooperative Research Centre for Advanced Automotive Technology.

4.2.1 Software architecture

Let $C = \{c_1, c_2, ..., c_n\}$, where $n \in N$, denote the set of software components, and let the parameters of the software architecture be given as follows:

- Communication frequency, $cf : C \times C \rightarrow \mathbb{R}$.
- Event size, $es : C \times C \rightarrow \mathbb{N}$.
- Workload, $wl : C \rightarrow \mathbb{N}$

4.2.2 Hardware architecture

Let $H = \{h_1, h_2, ..., h_m\}$, where $m \in M$, denote the set of hardware resources and let the parameters of the hardware architecture be given as follows:

- Processing speed, $ps : H \rightarrow \mathbb{N}$
- Network bandwidth, $nb : H \times H \rightarrow \mathbb{N}$.
- Network reliability $r : H \times H \rightarrow \mathbb{R}$.
- Network delay $nd : H \times H \rightarrow \mathbb{N}$.

4.2.3 Deployment

The deployment problem is then defined as $D = \{d \mid d : C \rightarrow H\}$, where D is the set of all functions assigning components to hosts. A single deployment alternative is $d_i = \{(c_1, h_{i_1}), (c_2, h_{i_2}), ..., (c_n, h_{i_n})\} \in D$, but for the sake of readability, we write it as $d_i = [h_{i_1}, h_{i_2}, ..., h_{i_n}]$.

4.2.4 Quality attributes

The goal of the optimisation process is to find the possible best deployment architecture alternatives which make a trade-off among three important conflicting quality attributes: data transmission reliability (dtr), communication overhead (co), and scheduling length (sl).

Data transmission reliability is defined as:

$$dtr(d_k) = \sum_{i=1}^{n} \sum_{j=1}^{n} cf(c_i, c_j) \cdot r(d_k(c_i), d_k(c_j))$$

where $cf(c_i, c_j)$ is the communication frequency between components c_i and c_j, and $r(d_k(c_i), d_k(c_j))$ is the reliability of the communication link between the hardware resources where c_i and c_j are deployed.

Communication overhead is equal to:

$$co(d_k) = \sum_{i=1}^{n} \sum_{j=1}^{n} cf(c_i, c_j) \cdot nd(d_k(c_i), d_k(c_j)) + \sum_{i=1}^{n} \sum_{j=1}^{n} \frac{cf(c_i, c_j) \cdot es(c_i, c_j)}{nb(d_k(c_i), d_k(c_j)) \cdot r(d(c_i), d(c_j))}$$

where $nd(d_k(c_i), d_k(c_j))$ is the network delay of the communication link between the hardware resources where c_i and c_j are deployed, $nb(d_k(c_i), d_k(c_j))$ is the network bandwidth, and $es(c_i, c_j)$ is the event size (message size) of the communication between components c_i and c_j.

Scheduling length is calculated as:

$$sl(d_k) = \sum_{j=0}^{m} \sum_{c \in d^{-1}(h_j)} \frac{wl(c)}{ps(h_j)}$$

where $wl(c)$ is the workload of component c and $ps(h_j)$ is the processing speed of the hardware resource h_j which contains c.

4.2.5 Fitness calculation

Different quality metrics for measuring the fitness of multiobjective optimisation results exist. Zitzler et al. [33] provide a comprehensive review, finding that many commonly used quality metrics do not reliably reflect the fitness of the optimisation results. One of the few recommended metrics is the hypervolume indicator, which measures the hypervolume between the nondominated solutions and a fixed reference point in the result space. For a detailed description of hypervolume indicator see [32].

4.3 Experimental Conditions

Approximate algorithms are not expected to deliver exact and repeatable results, but to provide good approximate solutions where exact approaches cannot be devised. Hence, results concerning the performance of approximate algorithms such as EAs, are usually reported as mean values over repeated trials. To obtain a fair comparison, the generally accepted approach is to allow the same number of function evaluations for each trial [23]. Therefore, for the current comparison, all algorithms were granted 100 function evaluations per trial with Royal Road experiments and 35 000 evaluations with the component deployment problem, repeating the trials 30 times for each optimisation scheme. Nevertheless there are indications that all algorithms still make small but steady improvements after these numbers of evaluations.

4.3.1 Problem instances

According to Lin et al. [24], differences in performance among approximate algorithms are more likely to be detected statistically if all algorithmic approaches solve the same problem instances. Thus, one of the trials used the Royal Road problem with the settings specified in Holland [18].

We also created two instances of the component deployment problem, one with 60 hardware resources and 120 software components (h60 c120), the other with 60 hardware hosts and 220 software components (h60 c220). Due to the multiobjective nature of the component deployment problem, all 30 trial outcomes were reported in terms of the hypervolume indicator. For all algorithms, the hypervolume indicator was the basis of parameter values' performance assessment when optimising the component deployment problem.

4.3.2 Evolutionary Algorithm

The Royal Road problem is optimised using an EA with string-based representation and multipoint crossover. The component deployment problem is multiobjective in nature and requires a more specialised approach. One of the state-of-the-art multiobjective EA implementations is Deb's [8]. Its distinctive feature is nondominated sorting, which filters the population into layers of nondominated fronts and ranks the solutions according to the level of front they are a member of.

These EA implementations use customised crossover and mutation operators with their respective probabilities of being applied. The crossover and mutation rates are probably the most conspicuous control parameters to optimise in stochastic optimisation. Hence this seminal work explores parameter control of the two parameters.

4.3.3 Predictive parameter control

The only hyper-parameter for the PPC is the learning rate e (the number of function evaluation before each update of the parameters). However, this parameter does not seem to depend on the type of problem or problem size. As a learning rate we used $e = 200$ function evaluation in all 3 problem instances.

For both the crossover and the mutation rates we use the same value intervals to sample from according to the probabilities produced by the time series. Preliminary trials have shown that a cardinality of four intervals or bins with ranges of $\{[0.2-0.4), [0.4-0.6), [0.6-0.8), [0.8-1.0)\}$ produced the best results among several cardinalities with even spreads between 0.2 and 1.

4.3.4 Benchmarks

Our method was compared to three other state of the art adaptive methods: Probability Matching (PM) [29], Adaptive Pursuit (AP) [29] and Dynamic Multi-Armed Bandit (DMAB) [14]. A comprehensive description and comparison of the three adaptive methods is given by Fialho et al. [14].

Table 1: Hyper-parameters (hp) of the three adaptive methods: Dynamic Multi-Armed Bandit (DMAB), Adaptive Pursuit (AP) and Probability Matching (PM)

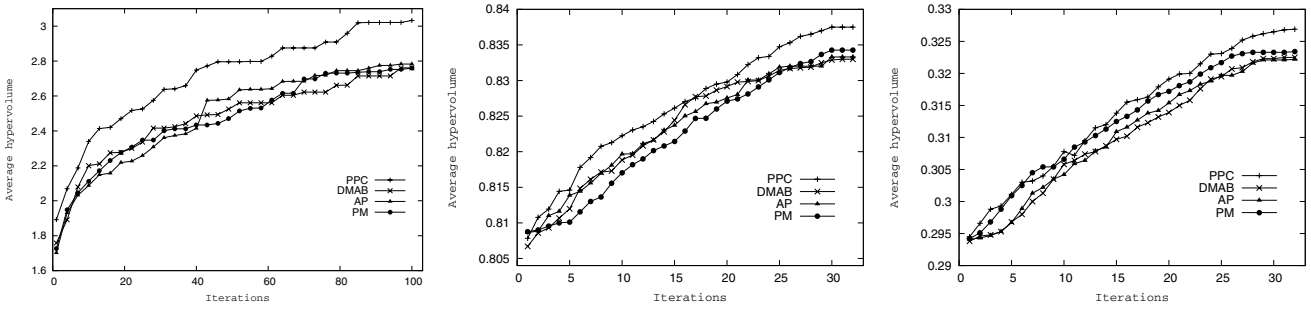
Method	hp	Value	Description
DMAB	ς	0.5	scaling factor
DMAB	γ	100	PH threshold
AP,PM	p_{min}	0.1	minimum selected probability
AP,PM	α	0.8	adaptation rate
AP,PM	β	0.8	adaptation rate

All adaptive algorithms involve hyper-parameters, which need to be tuned depending on the optimisation problem at hand. This defines another optimisation problem, which can become quite computationally expensive if we attempt an exhaustive exploration of the search space. We used recommendations from Thierens [29] and Fialho et al. [14] when setting the values of hyper-parameters for the adaptive algorithms, which are depicted in Table 1.

5. RESULTS AND DISCUSSION

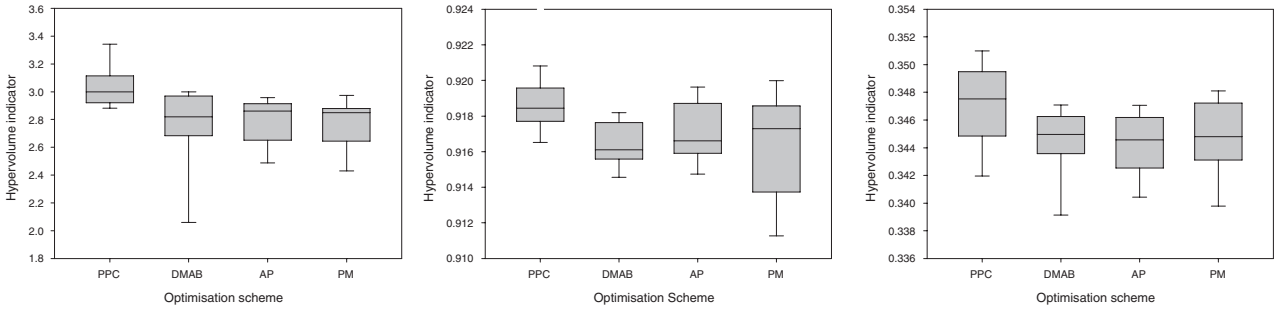
The average hypervolume growth in Figure 2, which was recorded to check the quality development of the results during the optimisation cycles, clearly shows the superior performance of PPC compared to the benchmark methods. The difference in performance seems more pronounced in the trials using the Royal Road problem. The least benefit PPC provides for the smaller of the component deployment instances. This is an 'easier' instance to solve, hence the algorithm performance can be expected to be more robust to parameter settings. Nonetheless, the Kolmogorov-Smirnov test finds a significantly superior performance of PPC compared to the benchmark algorithms on all three problems.

There are also indications that the difference in performance between the algorithms grows as the search progresses. With regards to the component deployment instances, the



(a) Royal Road (b) CD(h60 c120) (c) CD(h60 c120)

Figure 2: Average hypervolume indicator growth of the 30 trials of the four different optimisation schemes.



(a) Royal Road (b) CD(h60 c120) (c) CD(h60 c220)

Figure 3: Boxplots of the 30 trials of the four different optimisation schemes.

benchmark algorithms seem to maintain the same level of quality, while the PPC implementation continues to improve. Trials with more function evaluations will be needed to explore the capabilities of the algorithm until stagnation.

The 30 hypervolume indicators of the repeated trials are presented as boxplots in Figure 3. The empirical results are not normally distributed, but the mean and 25th percentile of PPC are consistently above the respective values of the benchmark approaches. The means and standard deviations are listed in Table 2. Especially in the case of the component deployment problem, the differences in the means are minimal. This is an inherent feature of this triobjective problem. For example the difference in the data transmission reliabilities is often apparent only in the third or fourth precision, which has a scaling effect on the hypervolume. For this reason, we include the results of a statistical analysis which clearly show a significant difference between the result groups of PPC and the benchmarks.

As our method consistently outperforms the three other optimisation schemes, to check for a statistical difference, the different parameter schemes of the optimisation methods are validated using the Kolmogorov-Smirnov nonparametric test (ks) [22]. The 30 hypervolume indicators of the repeated trials for each of the problem instances were submitted to the ks analysis. PPC was compared to the other three optimisation schemes, with a null hypothesis of a significant difference between the performances (PPC vs. DMAB, PPC vs. AP and PPC vs. PM). The results of the tests are shown in Table 3.

All ks tests, used for establishing differences between independent datasets under the assumption that they are not normally distributed, result in a confirmation of the null hypothesis with a minimum d-value of 0.5 at a 100% confidence level. Hence we conclude that the superior performance of PPC is statistically significant.

6. CONCLUSION

The current study presents a new predictive method of adapting parameters of approximate algorithms dynamically. In the experimental studies we have used this PPC method to adjust the crossover and mutation rates throughout the optimisation process carried out using two very different EA implementations. The trials have demonstrated that it outperforms the adaptive parameter control methods currently considered most successful.

Given these encouraging results, there is a need to explore the stagnation behaviour of the method, as the results produced by the PPC method still seem to improve even at the end of the trials, unlike those of the benchmark algorithms.

The application of the PPC method to other parameters of EA and other stochastic algorithms is a priority. We are particularly interested in using PPC for the on-the-fly optimisation of a cooling schedule for an implementation of

Table 2: The means of standard deviations of fitness functions for the 30 runs of each problem instance using different optimisation schemes.

	Mean			Standard Deviation		
	Royal Road	CD(h60 c120)	CD(h60 c220)	Royal Road	CD(h60 c120)	CD(h60 c220)
AP	2.783	0.9171	0.3443	0.164	1.906E-03	2.976E-03
PM	2.760	0.9163	0.3447	0.192	2.885E-03	2.799E-03
DMAB	2.706	0.9165	0.3444	0.349	1.437E-03	2.759E-03
PPC	3.033	0.9187	0.3468	0.159	1.263E-03	3.328E-03

Table 3: The t-test and ks-test values of fitness functions for the 30 runs of each problem instance using different optimisation schemes.

	Royal Road ks-test		CD(h60 c120) ks-test		CD(h60 c220) ks-test	
	d	p	d	p	d	p
PPC vs. DMAB	0.5000	0.008	0.5786	0.000	0.5000	0.008
PPC vs. AP	0.5500	0.003	0.4932	0.003	0.4974	0.010
PPC vs. PM	0.7000	0.000	0.4226	0.017	0.4136	0.039

Simulated Annealing, which is always a challenging problem for practitioners, as this parameter cannot rely on a predefined static value.

7. ACKNOWLEDGEMENTS

This original research was proudly supported by the Commonwealth of Australia, through the Cooperative Research Centre for Advanced Automotive Technology (C4-509: Dependability optimization at an architectural level of system design).

8. REFERENCES

[1] A. Aleti, L. Grunske, I. Meedeniya, and I. Moser. Let the ants deploy your software - an ACO based deployment optimisation strategy. In *ASE*, pages 505–509. IEEE Computer Society, 2009.

[2] T. Bäck. The interaction of mutation rate, selection, and self-adaptation within a genetic algorithm. In *Parallel Problem Solving from Nature 2,PPSN-II*, pages 87–96. Elsevier, 1992.

[3] T. Bäck. Introduction to the special issue: Self-adaptation. *Evolutionary Computation*, 9(2):iii–iv, 2001.

[4] T. Baeck, M. Schuetz, and S. Khuri. A comparative study of a penalty function, a repair heuristic and stochastic operators with the set-covering problem. *Lecture Notes in Computer Science*, 1063:320–332, 1996.

[5] T. Bartz-Beielstein, C. Lasarczyk, and M. Preuss. Sequential parameter optimization. In *IEEE Congress on Evolutionary Computation*, pages 773–780. IEEE, 2005.

[6] M. Birattari, T. Stützle, L. Paquete, and K. Varrentrapp. A racing algorithm for configuring metaheuristics. In *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 11–18. Morgan Kaufmann Publishers, 2002.

[7] K. Deb and H.-G. Beyer. Self-adaptive genetic algorithms with simulated binary crossover. *Evolutionary Computation*, 9(2):197–221, 2001.

[8] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm:NSGA-II. *IEEE-Evolutionary Computation*, 6:182–197, 2002.

[9] K. DeJong. Parameter setting in EAs: a 30 year perspective. In F. G. Lobo, C. F. Lima, and Z. Michalewicz, editors, *Parameter Setting in Evolutionary Algorithms*, volume 54 of *Studies in Computational Intelligence*, pages 1–18. Springer, 2007.

[10] K. A. DeJong. *Analysis of Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, The University of Michigan, 1975.

[11] A. E. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transations on Evolutionary Computation*, 3(2):124–141, 2007.

[12] R. Farmani and J. A. Wright. Self-adaptive fitness formulation for constrained optimization. *IEEE Trans. Evolutionary Computation*, 7(5):445–455, 2003.

[13] Á. Fialho, L. D. Costa, M. Schoenauer, and M. Sebag. Analyzing bandit-based adaptive operator selection mechanisms. *Annals of Mathematics and Artificial Intelligence - Special Issue on Learning and Intelligent Optimization*, 2010.

[14] Á. Fialho, M. Schoenauer, and M. Sebag. Analysis of adaptive operator selection techniques on the royal road and long K-path problems. 2009.

[15] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

[16] N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

[17] J. Hesser and R. Manner. Towards an optimal mutation probability for genetic algorithms. *Lecture Notes in Computer Science*, 496:23–32, 1991.

[18] T. Jones. A description of holland's royal road function. *Evolutionary Computation*, 2(4):409–415, 1994.

[19] Z. Michalewicz and B. F. Fogel. *How to solve it: modern heuristics*. Springer-Verlag, 2004.

[20] V. Nannen and A. E. Eiben. Relevance estimation and value calibration of evolutionary algorithm parameters. In M. M. Veloso, editor, *IJCAI'07, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pages 975–980, 2007.

[21] Y. Papadopoulos and C. Grante. Evolving car designs using model-based automated safety analysis and optimisation techniques. *The Journal of Systems and Software*, 76(1):77–89, 2005.

[22] A. N. Pettitt and M. A. Stephens. The kolmogorov-smirnov goodness-of-fit statistic with discrete and grouped data. *Technometrics*, 19(2):205–210, 1977.

[23] R. L. Rardin and R. Uzsoy. Experimental evaluation of heuristic optimization algorithms:A tutorial. *Journal of Heuristics*, 7(3):261–304, 2001.

[24] R. L. RardinBenjamin W. Lin. A short convergence proof for a class of ant colony optimization algorithms. *Management Science*, 25:1258–1271, 1980.

[25] I. Rechenberg. *Evolutionsstrategie: optimierung technischer systeme nach prinzipien der biologischen evolution*. Frommann-Holzboog, 1973.

[26] G. Rudolph. Self-adaptive mutations may lead to premature convergence. *IEEE Trans. Evolutionary Computation*, 5(4):410–414, 2001.

[27] J. Smith and T. C. Fogarty. Self adaptation of mutation rates in a steady state genetic algorithm. In *International Conference on Evolutionary Computation*, pages 318–323, 1996.

[28] C. R. Stephens, I. G. Olmedo, J. M. Vargas, and H. Waelbroeck. Self-adaptation in evolving systems. *Artificial Life*, 4(2):183–201, 1998.

[29] D. Thierens. An adaptive pursuit strategy for allocating operator probabilities. In H.-G. Beyer and U.-M. O'Reilly, editors, *Genetic and Evolutionary Computation Conference, GECCO 2005*, pages 1539–1546. ACM, 2005.

[30] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

[31] B. Yuan and M. Gallagher. Statistical racing techniques for improved empirical evaluation of evolutionary algorithms. In *Parallel Problem Solving from Nature - PPSN VIII*, volume 3242 of *LNCS*, pages 172–181. Springer-Verlag, 2004.

[32] E. Zitzler, D. Brockhoff, and L. Thiele. The Hypervolume Indicator Revisited:On the Design of Pareto-compliant Indicator Via Weighted Integration. In *Evolutionary Multi-Criterion Optimization, EMO'07*, volume 4403 of *LNCS*, pages 862–876. Springer, 2007.

[33] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and da V. G. Fonseca. Performance assessment of multiobjective optimizers: an analysis and review. *IEEE Tran. on Evolutionary Comp.*, 7:117–132, 2003.