Entropy-based Adaptive Range Parameter Control for Evolutionary Algorithms

Aldeida Aleti
Faculty of Information Technology
Monash University, Australia
aldeida.aleti@monash.edu

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

ABSTRACT

Evolutionary Algorithms are equipped with a range of adjustable parameters, such as crossover and mutation rates which significantly influence the performance of the algorithm. Practitioners usually do not have the knowledge and time to investigate the ideal parameter values before the optimisation process. Furthermore, different parameter values may be optimal for different problems, and even problem instances. In this work, we present a parameter control method which adjusts parameter values during the optimisation process using the algorithm's performance as feedback. The approach is particularly effective with continuous parameter intervals, which are adapted dynamically. Successful parameter ranges are identified using an entropy-based clusterer, a method which outperforms state-of-the-art parameter control algorithms.

Categories and Subject Descriptors

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

General Terms

Algorithms

Keywords

Evolutionary Algorithms, optimisation, adaptive parameter control

1. INTRODUCTION

Evolutionary Algorithms (EAs) have successfully been employed to solve a range of hard optimisation problems [1, 2, 30, 17, 23, 28]. In recent years, it has been acknowledged that the success of these algorithms depends on the numerous parameters that these algorithms have [19, 14, 32, 6], which make the optimisation procedure flexible and efficient for any kind of problem, regardless of the search space difficulty.

Unfortunately, the settings of the parameter values are known to be problem-specific [31], often even specific to the problem instance at hand [5, 39, 38], and greatly affect the performance of the algorithm [32, 6, 25, 15]. In cases where the number of parameters and their plausible value ranges are high, investigating all possible combinations of parameter values can itself be an attempt to solve a combinatorially complex problem [8, 44, 7, 33]. It has also been empirically and theoretically demonstrated that different parameter settings may be required for different optimisation stages of a problem instance [6, 5, 39, 38, 10, 40], and therefore should vary during the search process for best algorithm performance [5, 39, 38].

To address the problem of parameterising Evolutionary Algorithms, adaptive parameter control [41, 16, 11, 12, 20, 21, 22, 26] emerged as a new line of thought. Instead of exploring parameter settings before using an EA, in adaptive parameter control parameters are adjusted during the optimisation process. More specifically, properties of an EA run (such as the quality of the solutions produced) are monitored and the change in the properties is used as a signal to change parameter values.

One of the difficult issues faced in adaptive parameter control methods is the choice for real-valued parameter assignments, such as the mutation rate and the crossover rate. Usually, adaptive parameter control methods [43, 10, 21, 16, 42, 41, 3] discretise the choices for parameter assignments and sample parameter values from the preselected ranges. Insight from existing research [5] shows that initial discretisation can be problematic. As the ranges are fixed, they remain sub-optimal throughout the optimisation process.

Confirming this view, Adaptive Range Parameter Selection (ARPS) [4] was introduced, which uses adaptive value ranges that change dynamically during the optimisation process. ARPS outperforms state-of-the-art adaptive parameter control methods, such as Probability Matching (PM) [41], Adaptive Pursuit (AP) [41], Dynamic Multi-Armed Bandit (DMAB) [16] and Predictive Parameter Control (PPC) [3]. However, it was observed that high-performing ranges were sometimes absorbed into very large intervals, making it difficult for the algorithm to re-establish small, promising areas within the range.

With the goal of handling real-valued parameter assignments effectively, we introduce a new adaptive parameter control approach which uses an entropy-based measure to discretise parameter value ranges. At every iteration, feedback from the search is used to cluster parameter values into successful and unsuccessful groups. The outcome from the clustering process is employed to cut the parameter range into two sub-ranges such that the class information entropy is minimised. Results from the experimental validation of the approach show that the Entropy-based Adaptive Range Parameter Control (EARPC) outperforms state-of-the-art methods.

2. BACKGROUND

2.1 Evolutionary Algorithms

Generally speaking, Evolutionary Algorithms maintain a population of solutions that evolves by means of the mutation operator, the crossover operator, the selection procedure, and the replacement procedure. The objective is to search the solution space in order to optimise some quality function(s). The optimisation process starts with a set of solutions as initial population, which can be randomly generated, created by applying heuristic rules (e.g. greedy algorithm), or provided by an expert. After the initialisation, EA evolves the population using the crossover, mutation and selection operators. The crossover and mutation operators are applied according to predefined crossover and mutation rates. These operators are applied to specific solutions, which are selected according to the selection procedures. The new individuals (offspring) created by the genetic operators are added to the population. The replacement procedure selects the solutions that will survive in the next generation and removes as many individuals as required to maintain the prescribed population size.

Before using an EA for optimisation, a number of algorithm parameters have to be adjusted, such as the population size, offspring number, selection procedure, mutation rate and crossover rate. A comprehensive overview of the research in various parameters of Evolutionary Algorithms is presented by Eiben and Schut [14].

2.2 Adaptive Parameter Control

Formally, given a set $\{\upsilon_1, ..., \upsilon_n\}$ of n algorithm parameters, where each parameter υ_i has $\{\upsilon_{i1}, ..., \upsilon_{im}\}$ values that can be discrete numbers or intervals of continuous numbers, parameter control has the task of deriving the optimal next value υ_{ij} to optimise the influence of υ_i on the performance of the algorithm. As an example, when the mutation rate υ_1 is dynamically adjusted by considering 4 intervals ($m = 4$), υ_{12} stands for a mutation rate sampled from the second interval. In the discrete case of optimising the type of mutation operator υ_2, υ_{22} stands for the second operator.

The majority of the adaptive parameter control methods found in the literature [11, 12, 20, 21, 22, 26] belong to the class of probability matching techniques, in which the probability of applying a parameter value is proportional to the quality of that parameter value.

The earliest approaches [21, 20] tended to ‘lose’ value ranges if the feedback from the algorithm was not in their favour in the initial phases of the process. In later work, a minimum selection probability p_{min} is introduced [21], to ensure that under-performing parameter values did not disappear during the optimisation, in case they were beneficial in the later stages of the search.

These probability matching techniques [11, 12, 20, 21, 22, 26, 18] were based on the notion of selecting parameter values in proportion to their previous performances. One of the more recent and mature examples of probability matching is the work by Igel and Kreutz [21]. Their Equation 1 for calculating the selection probability for each parameter value incorporates the maintenance of a minimum probability of selection.

$$p'_t(\upsilon_{ij}) = p_{min} + (1 - m p_{min}) \frac{p_t(\upsilon_{ij})}{\sum_{r=1}^{m} p_t(\upsilon_{ir})} \qquad (1)$$

where m is the number of possible values for parameter υ_i. Probability matching has been criticised for the loose correlation between the reward allocations and the differences in performance with vastly superior values receiving a marginal increase in selection probability.

Adaptive Pursuit (AP) [41] was conceived as an attempt to address this issue, ensuring that clearly superior values are rewarded with a maximum probability of choice. Even though every parameter value is selected from time to time, in practice the Adaptive Pursuit algorithm spends a number of iterations before responding to a change of best parameter value.

Dynamic Multi-Armed Bandit (DMAB) [16] addresses the problem by completely recalculating the probabilities when a change in the effects distribution is detected by using a change detection test, in this case the statistical Page-Hinkley (PH) test. The PH test checks whether the quality of the parameter values has changed. When a change is detected, the algorithm is restarted. As a result, DMAB can quickly identify the new best parameter value without being slowed down by old information.

Adaptive Range Parameter Selection (ARPS) [4] is the first attempt at adapting real-valued parameter ranges. The ARPS algorithm discretises continuous-valued parameters by partitioning its ranges into two equal intervals. At every iteration, the best-performing interval is subdivided by splitting it in the middle. At the same time, the worst-performing interval is merged with the worse-performing of its neighbours. The selection probabilities of the split intervals are initially the same, the merged interval maintains the neighbour’s probability of selection.

Refining the most successful parameter areas into narrower intervals, the probability of choosing particular values increases. Merging badly performing parameters decreases their probability of selection. An analysis of the mutation/crossover ranges [4] revealed that ARPS sometimes absorbs high-performing intervals into very large ranges as a result of short-term underperformance. The merged ranges are usually very large and it takes many iterations for an absorbed sub-range to re-establish itself. This behaviour almost certainly affects the performance of the search. Therefore, an improved approach to maintaining parameter value intervals is the subject of this investigation.

3. ENTROPY-BASED ADAPTIVE RANGE PARAMETER CONTROL

At every iteration of an EA run, the chosen parameter values are employed to create different solutions, which are evaluated using the fitness function(s). The output from the evaluation process provides valuable information for the adaptive parameter control, since it can be used to assess the effect of the parameter values on the performance of the optimisation algorithm. The feedback from the search is used to approximate the cause of the change in the quality of the solutions and assign a quality to parameter values, denoted as $q(\upsilon_{ij})$. The quality of the parameter values is used to select the next parameter assignments, which are employed to create the next generation of solution(s).

In this framework, the feedback from the algorithm tends to fluctuate, and very good parameter ranges occasionally receive poor feedback, leading to the problem of the loss of a range as a result of merging when ARPS is used. Entropy-based Adaptive Range Parameter Control (EARPC) clusters the values of a parameter by performance and subdivides the intervals according to the clustering. Since the process is repeated at every iteration, disadvantaging a well-performing interval has no effect beyond the subsequent iteration.

The EARPC algorithm consists of two major steps; in the first step, it uses K-means clustering to determine how best to subdivide the values into two groups. K-means clustering starts with an initial random centre point for each group. All other points are allocated to the group whose centre is closer to them.

As we are clustering the parameter values based on their performance in the algorithm $q(\upsilon_{ij})$, the performance feedback used for controlling the parameters is used. Once all points have been allocated to one of the clusters, the mean of the values of each group is calculated. The point which is closest to this group mean becomes the new centre, and all other points are reallocated according to their distances to the new means. The algorithm stops when the centres no longer change.

The second step uses these clusters as a basis to define the range boundaries for the groups. The k-means method expects the k to be determined from the outset. Therefore, in this work, we assume $k = 2$ intervals. At the end of the k-means clustering algorithm, we obtain 2 clusters.

3.1 Adaptation of Parameter Interval Boundaries

From k-means clustering, we obtain two clusters of parameter values, grouped according to their distances to the clusters' centre points. When we align the parameter values along the axis of the actual parameter values, the members of the groups are likely interspersed. If c_1 denotes a member of cluster 1 and c_2 a member of cluster 2, an ordering according to the parameter values may present itself as '$c_1, c_1, c_2, c_1, c_2, c_1, c_1, c_2, c_2, c_1, c_2$'. The algorithm uses this sequence to determine the interval boundaries. The ideal boundary is located where the class entropy, calculated by Eq. 2, is minimal.

Let parameter value k partition range of parameter υ_i representing values between $[min, max)$ into two intervals: υ_{i1} representing values in the range $[min, k]$ and υ_{i2} denoting values in the range $[k, max)$. To obtain the minimal entropy, we calculate all entropy values arising from placing k between each of the adjacent pairs in the sequence $c_1, c_1, c_2, c_1, c_2, ...$

$$e(\upsilon_{i1}) = -\frac{|c_1(\upsilon_{i1})|}{|c_x(\upsilon_{i1})|} log_2 \frac{|c_1(\upsilon_{i1})|}{|c_x(\upsilon_{i1})|} - \frac{|c_2(\upsilon_{i1})|}{|c_x(\upsilon_{i1})|} log_2 \frac{|c_2(\upsilon_{i1})|}{|c_x(\upsilon_{i1})|}$$

The calculation of the class entropy has to be repeated for each interval υ_{ij}. In each case, it calculates the number of 'matching' cluster members (c_1 for interval υ_{i1}) in one interval as a proportion of all samples located in that interval ($c_x(\upsilon_{i1})$) and subsequently the proportion of non-matching cluster members (c_2 for interval υ_{i1}) of all samples in that cluster. The overall class information entropy when a range υ_{ij} is partitioned at the cut-point k into two new ranges υ_{i1} and υ_{i2} is calculated as the weighted average of the class entropies of the sub-ranges. This constitutes the *class information entropy* and is computed as follows:

Algorithm 1 Entropy-based Adaptive Range Parameter Control.

```
procedure EARPC($\upsilon_i$, $clusters$)
    CLUSTER($\upsilon_i$, $clusters$)
    $minEntropy$ = MAX(Integer)
    for all parameters $\upsilon_i,\ i \in n$ do
        for all parameter values $\upsilon_{ij},\ j \in m$ do
            $\mathcal{E}(j|\upsilon_i)$ =INFOENTROPY($\upsilon_{i1}, \upsilon_{i2}, j$)
            if $\mathcal{E}(j|\upsilon_i) < minEntropy$ then
                $minEntropy = \mathcal{E}(j|\upsilon_i)$
                $cutPoint = j$
            end if
        end for
        ADJUSTRANGE($cutPoint, \upsilon_i$)
    end for
end procedure

procedure CLUSTER($\upsilon_i, clusters$)
    while True do
        $oldC = newC$
        for all parameter values $\upsilon_{ij},\ j \in m$ do
            for all cluster center $newC[k]$ do
                if ED($\upsilon_{ij}, newC[k]$) $< mD$) then
                    $cl[j] = k$
                end if
            end for
        end for
        for $s = 0; s < clusters; s++$ do
            for $r = 0; r < m; r++$ do
                if $cl[r] = s$ then $vec[s] = \upsilon_{ir}$
                end if
            end for
            $newC[j]$= CALCULATECENTER($vec$)
        end for
        if $newC = oldC$ then return
        end if
    end while
end procedure

procedure INFOENTROPY($\upsilon_{i1}, \upsilon_{i2}, k$)
    $e(\upsilon_{i1}) = -\frac{|c_1(\upsilon_{i1})|}{|c_x(\upsilon_{i1})|} log_2 \frac{|c_1(\upsilon_{i1})|}{|c_x(\upsilon_{i1})|} - \frac{|c_2(\upsilon_{i1})|}{|c_x(\upsilon_{i1})|} log_2 \frac{|c_2(\upsilon_{i1})|}{|c_x(\upsilon_{i1})|}$
    $e(\upsilon_{i2}) = -\frac{|c_1(\upsilon_{i2})|}{|c_x(\upsilon_{i2})|} log_2 \frac{|c_1(\upsilon_{i2})|}{|c_x(\upsilon_{i2})|} - \frac{|c_2(\upsilon_{i2})|}{|c_x(\upsilon_{i2})|} log_2 \frac{|c_2(\upsilon_{i2})|}{|c_x(\upsilon_{i2})|}$
    $\mathcal{E}(k|\upsilon_i) = \frac{|c_x(\upsilon_{i1})|}{|c_x(\upsilon_i)|} e(\upsilon_{i1}) + \frac{|c_x(\upsilon_{i2})|}{|c_x(\upsilon_i)|} e(\upsilon_{i2})$
    return $\mathcal{E}(k|\upsilon_i)$
end procedure

procedure ADJUSTRANGE($cutPoint, \upsilon_i$)
    $\upsilon_{i1}(min) = \upsilon_i(min)$
    $\upsilon_{i1}(max) = cutPoint$
    $\upsilon_{i2}(min) = cutPoint+$ MINFLOAT( )
    $\upsilon_{i2}(max) = \upsilon_i(max)$
end procedure
```

$$\mathcal{E}(k|\upsilon_i) = \frac{|c_x(\upsilon_{i1})|}{|c_x(\upsilon_i)|} e(\upsilon_{i1}) + \frac{|c_x(\upsilon_{i2})|}{|c_x(\upsilon_i)|} e(\upsilon_{i2}) \tag{2}$$

The best cut-point k is the one that minimises the *class information entropy* $\mathcal{E}(k|\upsilon_i)$. The method employs the cut-point k to partition the parameter range into two sub-ranges. These two intervals are employed to select the parameter values for the next iteration. The selection of parameter values is proportional to the average quality of the parameter values in each range. Let $p(\upsilon_{ij})$ denote the selection probability of parameter range υ_{ij} and $q(\upsilon_{ij})$ the combined quality of the members of the interval. The mean quality of the intervals is calculated according to Eq. 3, regardless of cluster membership.

$$q(\upsilon_{ij}) = \frac{1}{|c_x(\upsilon_{ij})|} \sum_{c_x \in \upsilon_{ij}} q(c_x) \tag{3}$$

The probability of sampling from a range $p(\upsilon_{ij})$ is derived from the combined quality of its members $q(\upsilon_{ij})$, normalised by the summed qualities of all intervals. The main steps of EARPC are given in Algorithm 1.

4. EXPERIMENTS

Evolutionary 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 [36]. Therefore, for the current comparison, all algorithms trials were repeated 30 times for each optimisation scheme, for 30 000 function evaluations. These values were decided after running the algorithm once for every problem and choosing the value where the quality of the solutions seemed to not improve any further. Nevertheless there are indications that all algorithms still make small but steady improvements after these numbers of evaluations.

4.1 Benchmark Problems

According to Lin et al. [37], differences in performance among approximate algorithms are more likely to be detected statistically if all algorithmic approaches solve the same problem instances. Along this line, different instances of two problems were chosen: the generally accepted Quadratic Assessment Problem (QAP), which was especially designed for testing EAs, and the component deployment problem, which is a real world problem in the Software Engineering domain.

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. Both problems are NP-hard, which makes them difficult to be solved by an exact algorithm and justifies the use of Evolutionary Algorithms to optimise them.

4.1.1 Quadratic Assignment Problem

The Quadratic Assignment Problem (QAP) was first introduced by Koopmans and Beckmann [24], who used this mathematical model to solve the problem of assigning economical activities to resources. In QAP, n facilities have to be allocated to n locations, such that the total cost is minimised and every resource has only one economical activity.

The total cost is calculated as the flow between the facilities multiplied by the costs for placing the facilities at their respective locations. QAP is considered to be a very challenging combinatorial optimisation problem. More formally, the problem is modelled with two $n \times n$, representing the cost and the flow. The aim is to assign n utilities to n locations with minimal cost. The candidate assignments are evaluated according to equation 4.

$$C = \sum_{ij}^{n} B_{ij} \cdot u_{ij} + \sum_{ij,k,l} C_{ij,k,l} \cdot u_{ik} \cdot u_{jl} \tag{4}$$

where

- n is the number of facilities and locations.
- B_{ik} is the cost of assigning utility i to location k
- $C_{ij,k,l}$ is the cost of the flow between neighbouring utilities (given utility i is assigned to location k and utility j is assigned to location l
- u_{ik} is 1 if utility i is assigned to location k, 0 otherwise

QAP does not allow for multiple assignments to the same location, hence solutions are subject to the following constraints:

$$\sum_{j=1}^{n} u_{ij} = 1, \sum_{i=1}^{n} u_{ij} = 1, u_{ij} \in \{0,1\}, i,j = \{1,2,...,n\}$$

QAP is a well-known problem and instances of considerable difficulty have been made available as benchmarks. It is a single-objective problem with one objective function that lends itself as quality feedback for the performance assessment of the parameter values.

4.1.2 Component Deployment Problem

One of the practical applications of Evolutionary Algorithms is the component deployment problem in embedded systems [1], relevant e.g. for the automotive industry. An existing hardware topology is used to run software components which form the basis of the increasingly sophisticated functionality of contemporary cars. The quality of the overall system depends on the choice of hardware unit to host a particular software component [34].

The quality of the system is commonly measured in terms of non-functional attributes such as safety, reliability and performance. We model the embedded system as a set of software components and a set of hardware hosts. Formally, let $C = \{c_1, c_2, ..., c_n\}$, where $n \in N$, denote the set of software components. The parameters for the software architecture are as follows:

- Communication frequency $f : C \times C \to \mathbb{R}$.
- Data size $ds : C \times C \to \mathbb{N}$.
- Estimated time for a single execution $ext : C \to \mathbb{N}$.

Let $H = \{h_1, h_2, ..., h_m\}$, where $m \in M$, denote the set of hardware resources. The parameters for the hardware architecture are as follows:

- Data rate $dr : H \times H \to \mathbb{N}$.
- Network reliability $r : H \times H \to \mathbb{R}$.
- Network delay $nd : H \times H \to \mathbb{N}$.
- Processing speed $ps : H \to \mathbb{R}$.

The deployment problem is then defined as $D = \{d \mid d : H \to C_{sub}\}$, where $C_{sub} = \{c_0, c_1, ..., c_j\} \subset C$, and D is the set of all functions assigning hardware resources to components. Note that, since C and H are finite, D is also finite, and represents the set of all deployment candidates.

The deployment problem has many quality-related aspects and is therefore always modelled as a multi-objective problem. Data Transmission Reliability (DTR) follows the definition of Malek [27]. Reliability of the data transmission is a crucial quality attribute in a real-time embedded system, where important decisions are taken based on the data transmitted through the communication links.

$$f_{DTR}(d) = \sum_{i=1}^{n}\sum_{j=1}^{n} f(c_i, c_j) \cdot r(d(c_i), d(c_j)) \tag{5}$$

In embedded systems with their constrained hardware resources, repeated transmissions between software components are discouraged. The Communication Overhead (CO) [29] objective attempts to enforce minimal data communication for a given set of components and system parameters, calculated as follows:

$$f_{CO}(d) = \sum_{i=1}^{n}\sum_{j=1}^{n} f(c_i, c_j) \cdot nd(d(c_i), d(c_j)) + \\ + \sum_{i=1}^{n}\sum_{j=1}^{n} \frac{f(c_i, c_j) \cdot ds(c_i, c_j)}{dr(d(c_i), d(c_j)) \cdot r(d(c_i), d(c_j))} \tag{6}$$

The scheduling length (SL) describes the average time for a hardware unit (i.e. ECU) to complete the round-robin processing of all its assigned components. ST is given by

$$f_{SL}(d) = \frac{1}{m} \cdot \sum_{j=1}^{m} \left(\frac{\sum_{c \in C_{h_j}} ext(c)}{ps(h_j)} \right). \tag{7}$$

where m is the number of components deployed in h_j and C_{h_j} is the set of components deployed to host h_j.

4.2 Experimental Settings

QAP as an assignment problem with constraints maps n tasks to n resources. Hence the solution representation is a simple array which describes the numbered locations and the values of the array represent the items. Multipoint crossover swaps the assignments between solutions. The mutation operator changes the assigned item of a single location. As we are solving the QAP, singularity of item occurrence is mandatory the genetic operators are specialised to create valid solutions.

A variety of instances with descriptions of properties are available in QAPLIB collection [9]. The instances Taixxxc, like Tai256c, occur in the generation of grey patterns. The distances of Skoxxxf problems are rectangular and the entries in flow matrices are pseudorandom numbers. Similarly, the distance matrices of Wilxxx and Thoxxx problems, such as Wil100 and Tho150 are rectangular.

The industrial problem of software deployment uses an EA with a specialised solution encoding which maps each hardware unit to one or more software components. The crossover operator combines the allocation lists of two solutions' locations (hardware units) and subsequently divides them again into two child solutions avoiding duplication. The mutation operator exchanges the host allocations of two randomly chosen components. The problem definition does not allow for duplication, and a repair operation follows the crossover/mutation move. Also, 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 NSGA-II, devised by Deb et al. [13]. The component deployment problem (CDP) instance used for the benefit of these experiments is the H36C51, composed of 36 hardware units and 51 software components.

The crossover and mutation rates, used for the benefit of these experiments, are probably the most conspicuous control parameters to optimise in stochastic optimisation [14]. Based on preliminary exploration, a range of $0.01 - 0.5$ was adopted for the mutation rate and the interval $0.6 - 1.0$ was used for the crossover operator.

In the case of QAP, the quality feedback for the parameter values is based on the fitness values returned by the objective function. The multiobjective nature of the deployment problem necessitates the combination of the fitness values from all three objective functions into a single unary measure of solution quality. The most conspicuous choice for this feedback is the hypervolume indicator, which has been described as the most reliable unary measure of solution quality in multiobjective space [45].

5. RESULTS

The means and standard deviations of the 30 trials from the different optimisation schemes listed in Table 1 clearly indicate a significant difference between the result groups of EARPC and the benchmarks. The mean performance of EARPC is consistently above the averages of the benchmark approaches. The standard deviation of EARPC is relatively high in the Quadratic Assignment Problem, but lower than the respective values of the other parameter control strategies in the component deployment problem.

As EARPC consistently outperforms the four other parameter control schemes, we employ the Kolmogorov-Smirnov (KS) non-parametric test [35] to check for a statistical difference. The 30 results of the repeated trials for each of the problem instances were submitted to the KS analysis. EARPC was compared to the other four adaptive parameter control schemes, with a null hypothesis of an insignificant difference between the performances (EARPC vs. ARPS, EARPC vs. DMAB, EARPC vs. AP and EARPC vs. PM). The results of the tests are shown in Table 2.

All KS tests, used for establishing that there is no difference between independent datasets under the assumption that they are not normally distributed, result in a rejection of the null hypothesis with a minimum d-value of 0.36 at a 97% confidence level. Hence we conclude that the superior performance of EARPC is statistically significant.

The gap between result qualities widens in favour of the EARPC as the problem size increases. The biggest Quadratic Assignment Problem of dimension n=256 (TAI256c) is clearly solved to better quality using EARPC, as it is more complex than the smaller QAP instances.

In EARPC, the intervals of crossover and mutation rates over 230 iterations are depicted in Figures 1 and 2. The problem optimised for the benefit of this experiment is the QAP problem instance TAI256c. At the beginning of the optimisation process, all intervals are equal. The selection of the parameter values is based on the assigned probabilities

Table 1: The means and standard deviations of fitness functions for the 30 runs of each problem instance using different parameter control schemes.

		Mean				
Problem	**Instance**	**PM**	**AP**	**DMAB**	**ARPS**	**EARPC**
QAP	Tho150	9154000	9157000	9197000	9141000	**9053000**
	Wil100	287100	286800	287600	286100	**285500**
	Sko100	164716	164875	164990	164681	**164139**
	Tai256c	47450000	47490000	47750000	47410000	**46820000**
CDP	H36C51	12239	12147	12405	12128	**11854**
		Standard Deviation				
Problem	**Instance**	**PM**	**AP**	**DMAB**	**ARPS**	**EARPC**
QAP	Tho150	36173.41	33358.02	41541.24	29024.92	47453.43
	Wil100	784.2973	634.0923	745.9012	523.1157	686.7064
	Sko100f	619.2552	613.1455	686.2719	872.4846	573.6736
	Tai256c	261879.1	314044	293074.2	269008.5	166171.3
CDP	H36C51	447.5069	344.8195	432.7907	423.8407	289.714

Table 2: The Kolmogorov-Smirnov test values of fitness functions for the 30 runs of each problem instance using different parameter control schemes.

		Kolmogorov-Smirnov test							
Problem	**Instance**	**EARPC vs. PM**		**EARPC vs. AP**		**EARPC vs. DMAB**		**EARPC vs. ARPS**	
		d	p	d	p	d	p	d	p
QAP	Tho150	0.8065	< 0.01	0.8387	< 0.01	0.9032	< 0.01	0.7742	< 0.01
	Wil100	0.8065	< 0.01	0.8065	< 0.01	0.9032	< 0.01	0.9355	< 0.01
	Sko100f	0.4667	< 0.01	0.5333	< 0.01	0.6	< 0.01	0.3667	0.03
	Tai256c	0.871	< 0.01	0.9355	< 0.01	0.9677	< 0.01	0.9677	< 0.01
CDP	H36C51	0.4	0.01	0.4333	< 0.01	0.5	< 0.01	0.3667	0.03

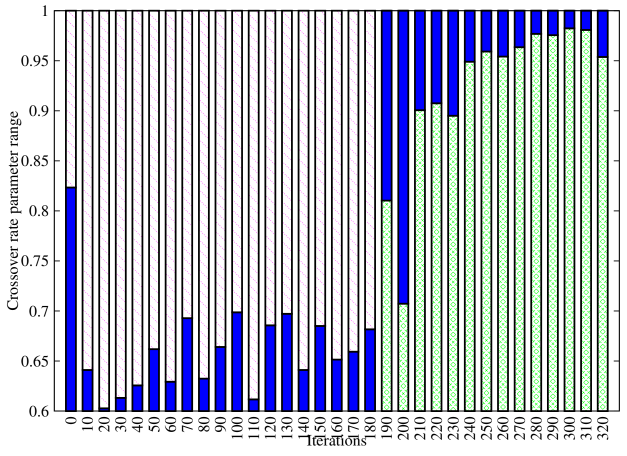

Figure 1: Adaptation of the range of crossover rate.

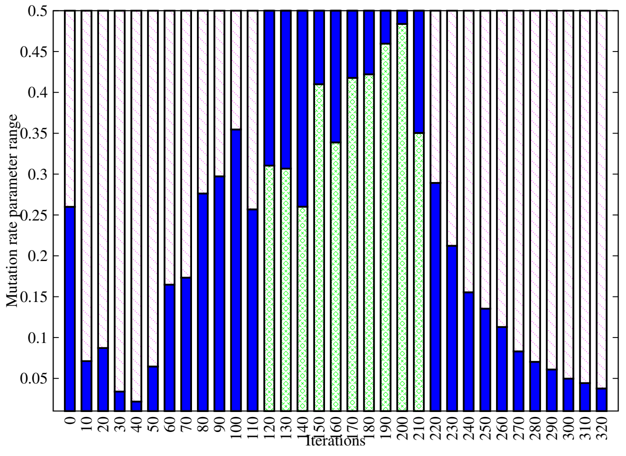

Figure 2: Adaptation of the range of mutation rate.

calculated using Equation 3. The probability of sampling from a range $p(\upsilon_{ij})$ is derived from the combined quality of its members $q(\upsilon_{ij})$, normalised by the summed qualities of all intervals. The higher the quality of the values in an interval, the more samples are taken from that interval.

In Figures 1 and 2, the most successful intervals are depicted with darker shade. We can clearly see that the behaviour of the entropy-based adaptive range parameter value selection is different for different parameters. The intervals of crossover rate develop quite differently compared to mutation rate. Higher crossover rates are more successful towards the end which runs somewhat contrary to popular opinion that crossover rates should decrease with iterations so as not to disturb solutions with high quality. On the other hand, the algorithm favours high mutation rates only in the middle of the optimisation process, and low rates at the beginning and at the end.

It can further be observed that high quality parameter values, both in mutation and crossover rates, are usually refined into narrower intervals compared to badly performing parameter values. This indicates that well performing values are usually diffuse into small regions of the parameter range and will be selected more frequently.

One might argue that although parameter configuration seems to be a very important factor of any algorithm, the parameter space is considerably smaller than the search space of all relevant NP-hard problems and the number of parameters is relatively small. So contrary to NP-hard optimisation problems, where 'solving by hand' is entirely inappropriate, 'choosing parameters by hand' is possible.

However, the use of parameter values that remain constant over the optimisation process has been observed in previous studies to achieve suboptimal results. This was also demonstrated by the analysis of parameter ranges during the optimisation process. Parameters of an EA can have different optimal values at different stages of the search. Thus, when an optimisation algorithm is tuned prior to the optimisation process, the selected parameters at the end of this process are not necessarily optimal.

This suggests that controlling parameters dynamically during the optimisation process is likely to lead to better outcomes. Hence, the proposed method of adapting parameters during the optimisation process is a more suitable option for the parameter configuration process. Furthermore, by reducing the number of parameters that need to be set in Evolutionary Algorithms, the transfer of these algorithms into industrial settings, where practitioners do not have any knowledge about EA parameters is facilitated.

6. CONCLUSION

The adaptive parameter control method introduced in this paper configures Evolutionary Algorithms during the optimisation process for optimal performance. According to our knowledge, the best-performing approaches with the same functionality are AP, PM, DMAB and ARPS. The new approach clearly outperforms the other parameter control methods.

The adaptation of parameter value ranges increases the sampling accuracy, which can explain the superior performance of EARPC compared to other approaches. Through using EARPC, we observed that different parameter intervals were optimal for different stages of the optimisation process. Furthermore, the ranges of the crossover rate developed quite differently compared to the mutation rate. Higher crossover rates were more successful towards the end of the optimisation process, whereas the opposite was observed for mutation rates.

To conclude the adaptive parameter control strategy presented in this paper, within its acknowledged limitations, is an effective method for adjusting parameters of Evolutionary Algorithms. In the future we intend to investigate algorithms for multi-interval partitioning, which we believe may improve the accuracy of interval boundaries and produce better algorithm performance.

7. REFERENCES

[1] A. Aleti, S. Björnander, L. Grunske, and I. Meedeniya. ArcheOpterix: An extendable tool for architecture optimization ofAADL models. In *Model-based Methodologies for Pervasive and Embedded Software (MOMPES)*, pages 61–71. ACM and IEEE Digital Libraries, 2009.

[2] 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.

[3] A. Aleti and I. Moser. Predictive parameter control. In *13th Annual Genetic and Evolutionary Computation Conference, GECCO 2011, Proceedings, Dublin, Ireland, July 12-16, 2011*, pages 561–568, 2011.

[4] A. Aleti, I. Moser, and S. Mostaghim. Adaptive range parameter control. In *Proceedings of the 2012 IEEE Congress on Evolutionary Computation*, pages 2405–2412, 2012.

[5] 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.

[6] T. Bäck, A. E. Eiben, and N. A. L. van der Vaart. An empirical study on gas without parameters. In *Parallel Problem Solving from Nature – PPSN VI (6th PPSN'2000)*, volume 1917 of *Lecture Notes in Computer Science (LNCS)*, pages 315–324. Springer-Verlag (New York), 2000.

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

[8] 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.

[9] R. E. Burkard, S. Karisch, and F. Rendl. Qaplib-a quadratic assignment problem library. *European Journal of Operational Research*, 55(1):115–119, 1991.

[10] J. Cervantes and C. R. Stephens. Limitations of existing mutation rate heuristics and how a rank GA overcomes them. *IEEE Trans. Evolutionary Computation*, 13(2):369–397, 2009.

[11] D. Corne, M. J. Oates, and D. B. Kell. On fitness distributions and expected fitness gain of mutation rates in parallel evolutionary algorithms. In *Parallel Problem Solving from Nature – PPSN VII (7th PPSN'02)*, volume 2439 of *Lecture Notes in Computer Science (LNCS)*, pages 132–141. Springer-Verlag, 2002.

[12] L. Davis. Adapting operator probabilities in genetic algorithms. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 70–79. Morgan Kaufman, 1989.

[13] 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.

[14] A. E. Eiben and M. C. Schut. New ways to calibrate evolutionary algorithms. In P. Siarry and Z. Michalewicz, editors, *Advances in Metaheuristics for Hard Optimization*, Natural Computing Series, pages 153–177. Springer, 2008.

[15] A. E. Eiben and S. K. Smit. Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm and Evolutionary Computation*, 1(1):19–31, 2011.

[16] Á. Fialho, M. Schoenauer, and M. Sebag. Analysis of adaptive operator selection techniques on the royal road and long k-path problems. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 779–786. ACM, 2009.

[17] J. Fredriksson, K. Sandström, and MikaelÅkerholm. Optimizing resource usage in component-based real-time systems. In *Component-Based Software Engineering*, volume 3489 of *LNCS*, pages 49–65. Springer, 2005.

[18] M. Giger, D. Keller, and P. Ermanni. Aorcea - an adaptive operator rate controlled evolutionary algorithm. *Computers and Structures*, 85(19-20):1547 – 1561, 2007.

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

[20] T.-P. Hong, H.-S. Wang, and W.-C. Chen. Simultaneously applying multiple mutation operators in genetic algorithms. *Journal of Heuristics*, 6(4):439–455, 2000.

[21] C. Igel and M. Kreutz. Operator adaptation in evolutionary computation and its application to structure optimization of neural networks. *Neurocomputing*, 55(1-2):347–361, 2003.

[22] B. A. Julstrom. What have you done for me lately? Adapting operator probabilities in a steady-state genetic algorithm. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 81–87, San Francisco, CA, 1995. Morgan Kaufmann.

[23] T. Kichkaylo and V. Karamcheti. Optimal resource-aware deployment planning for component-based distributed applications. In *HPDC:High Performance Distributed Computing*, pages 150–159. IEEE Computer Society, 2004.

[24] T. C. Koopmans and M. J. Beckmann. Assignment problems and the location of economic activities. *Econometrica*, 25:53–76, 1957.

[25] F. G. Lobo. Idealized dynamic population sizing for uniformly scaled problems. In *13th Annual Genetic and Evolutionary Computation Conference, Proceedings*, pages 917–924. ACM, 2011.

[26] F. G. Lobo and D. E. Goldberg. Decision making in a hybrid genetic algorithm. In *Proceedings of the Congress on Evolutionary Computation*, pages 121–125. IEEE Press, 1997.

[27] S. Malek. *A User-Centric Approach for Improving a Distributed Software System's Deployment Architecture*. PhD thesis, Faculty of the graduate schools, University of Southern California, 2007.

[28] S. Malek, M. Mikic-Rakic, and N. Medvidovic. A decentralized redeployment algorithm for improving the availability of distributed systems. In *Comp. Dep.*, volume 3798 of *LNCS*, pages 99–114. Springer, 2005.

[29] N. Medvidovic and S. Malek. Software deployment architecture and quality-of-service in pervasive environments. In *Workshop on the Engineering of Software Services for Pervasive Environements, ESSPE*, pages 47–51. ACM, 2007.

[30] I. Meedeniya, B. Buhnova, A. Aleti, and L. Grunske. Reliability-Driven Deployment Optimization for Embedded Systems. *Journal of Systems and Software*, 2011.

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

[32] Z. Michalewicz and M. Schmidt. Parameter control in practice. In F. G. Lobo, C. F. Lima, and Z. Michalewicz, editors, *Parameter Setting in Evolutionary Algorithms*, volume 54 of *Studies in Computational Intelligence*, pages 277–294. Springer, 2007.

[33] 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*, pages 975–980, 2007.

[34] 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.

[35] 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.

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

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

[38] 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.

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

[40] D. Thierens. Adaptive mutation rate control schemes in genetic algorithms. In *Proceedings of the 2002 Congress on Evolutionary Computation CEC2002*, pages 980–985. IEEE Press, 2002.

[41] 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.

[42] D. Thierens. Adaptive strategies for operator allocation. In *Parameter Setting in Evolutionary Algorithms*, volume 54 of *Studies in Computational Intelligence*, pages 77–90. Springer, 2007.

[43] Y.-Y. Wong, K.-H. Lee, K.-S. Leung, and C.-W. Ho. A novel approach in parameter adaptation and diversity maintenance for genetic algorithms. *Soft Computing*, 7(8):506–515, 2003.

[44] 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.

[45] 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.