Adaptive Range Parameter Control

Aldeida Aleti
Swinburne University of Technology
Melbourne, Australia
Email: aaleti@swin.edu.au

Irene Moser
Swinburne University of Technology
Melbourne, Australia
Email: imoser@swin.edu.au

Sanaz Mostaghim
Karlsruhe Institute of Technology
Karlsruhe, Germany
Email: sanaz.mostaghim@kit.edu

***Abstract*—All existing stochastic optimisers such as Evolutionary Algorithms require parameterisation which has a significant influence on the algorithm's performance. In most cases, practitioners assign static values to variables after an initial tuning phase. This parameter tuning method requires experience the practitioner may not have and, when done conscientiously, is rather time-consuming. Also, the use of parameter values that remain constant over the optimisation process has been observed to achieve suboptimal results. This work presents a parameter control method which redefines variables repeatedly based on a separate optimisation process which receives its feedback from the primary optimisation algorithm. The feedback is used for a projection of the value performing well in the future. The parameter values are sampled from intervals which are adapted dynamically, a method which has proved particularly effective and outperforms all existing adaptive parameter controls significantly.**

I. Introduction

All known stochastic optimisation methods such as Simulated Annealing (SA), Evolutionary Algorithms (EA) and Estimation of Distribution Algorithms (EDA) have a range of adjustable parameters like learning rates, crossover probabilities and weighting factors. Poor algorithm parameterisation hinders the discovery of good solutions. Yet inexperienced practitioners often apply stochastic methods with parameter values chosen on the basis of few tuning iterations. The interactions between the different parameters used by an algorithm have been investigated for a considerable number of years [6], [13], [30], but these studies are largely ignored by practitioners outside of the AI field.

The parameter values needed for optimal algorithm performance are known to be problem-specific [22], often even specific to the problem instance at hand [2], [28], [27], [17]. Moreover, the interactions between parameters have been found to be problem-specific [16]. Practitioners tend to choose parameter values based on a small number of preliminary experiments, a practice known as parameter tuning. Depending on the number of parameters and their plausible value ranges, investigative trials for parameter optimisations can themselves be attempts to solve a combinatorially complex problem [5], [31], [4], [24]. Moreover, it has also been established that some of the parameter values ought to vary during the search process for best algorithm performance [2], [28], [27], [17].

Acknowledging these facts, many researchers have shifted their focus to parameter control methods, where parameter values are optimised based on algorithm performance. Deterministic parameter control can be regarded as a variation of parameter tuning, in which several parameter settings are chosen based on preliminary experiments [21], to alleviate the performance problems of parameters that are invariate throughout the optimisation process. Self-adaptive parameter control integrates the search for optimal parameters into the optimisation process itself - usually by encoding parameter settings into the genotype of the solution to evolve [3], [11], [7]. Extending the solution size to include the parameter space obviously increases the search space and makes the search process more time-consuming [9].

Adaptive parameter control describes the application of separate meta-optimisation methods which use feedback from the optimisation process itself to evaluate the effect of parameter value choices and adjust the parameter values over the iterations. The approaches within this category (e.g. [29], [12]) optimise parameter values by choosing from predefined values or ranges.

II. Adaptive Parameter Control Strategies

The area of adaptive parameter control has been researched more actively in recent times [14], [12], [29], [15]. Parameter values are assessed based on recent performance and subsequently adapted for the next iteration of the algorithm. The most successful methods representative of adaptive strategies are Probability Matching (PM) [29], Adaptive Pursuit (AP) [29], Dynamic Multi-Armed Bandit (DMAB) [12] and Predictive Parameter Control (PPC) [1].

A. Probability Matching

Probability Matching (PM) [29] uses reinforcement learning to project the probability of good performance of a parameter value based on the previous performance of an algorithm using this value. The probability of a value providing good quality results at the next time step is based on a running average of past rewards. Rewards are allocated on the basis of the outcome of the optimisation process the parameter value was used in. A minimum probability is enforced on values which do not receive rewards in order to maintain a non-zero probability. The motivation of a minimum value is the assumption that parameter values which do not perform well at present might be optimal in the future.

PM has been criticised for the fact that the probability values resulting from the reward allocations poorly reflect the relative differences in algorithm performance when using the values. Values with vastly superior performance may only be

differentiated by a marginal increase of the probability of being chosen in the next step.

B. Adaptive Pursuit

Adaptive Pursuit (AP) [29] was conceived with the goal of improving the performance of PM by ensuring an appropriate difference in probabilities depending on experienced performance. After an iteration of the optimisation process, AP establishes the respective rewards for the parameter values used, but only applies the maximum award to the value of the best-performing algorithm instance. All other values have their probabilities of future use diminished. A nonzero probability is enforced as a minimum probability.

C. Dynamic Multi-Armed Bandit

Dynamic Multi-Armed Bandit (DMAB) [12] also employs a performance-based rewards approach, but here the probability of re-using a certain value is based on a tradeoff between the number of times the parameter value was used and the reward gained from its previous performance. Rather than using a rewards-based weighted adjustment of the probabilities, the DMAB completely recalculates the probabilities when a change in the rewards distribution is detected. DMAB uses the Page-Hinkley test to detect a change in the rewards distribution.

D. Predictive Parameter Control

PPC derives the probabilities for parameter values to choose for the next iteration based on algorithm performance in the previous iteration quite like PM, AP and DMAB. The reward - or credit assignment - strategy counts the number of times a parameter value was used and the number of times the algorithms using this value were successful. Success is defined as producing solutions with fitness values above a certain threshold. In the existing implementations, success has been defined as producing a population with above-average fitness.

The ratio of the times a parameter value's usage has been successful and the number of times this value was used is recorded after each iteration. It can be regarded as the probability of success given the use of value v at time t. Based on these historic probabilities, PPC uses least squares regression to derive the appropriate probabilities of success with the use of value v at time $t + 1$.

Rather than using discretised parameter value choices, PPC uses predefined ranges to sample the continuous values from. The ranges or bins are of equal size and remain the same throughout the optimisation process. The probabilities derived from the success rates of values are attributed to the range the value was sampled from rather than the value itself.

III. Adaptive Range Parameter Control

Predictive Parameter Control (PPC) [1] optimises the choice of parameter assignment using static predefined ranges. The quality feedback and therefore the probability of use in the next iteration is allocated to these ranges, not the actual sampled values. As the ranges are fixed, they are not optimised by the process. Defining narrow ranges leads to more accuracy but increased combinatorial complexity, leaving ranges wider entails a sampling inaccuracy as the actually sampled value may be far from the value whose success the range's usage probability is attributable to. Ideally, the ranges should be optimised by the parameter control process.

Adaptive Range Parameter Control (ARPC) remedies this problem by adjusting the range sizes as the optimisation process progresses. After each iteration, the best-performing range is halved, whereas the worst-performing is merged with the worse-performing of its neighbours.

This technique was first conceived for the context of parallel computing [23] but has never been used to the dynamic adjustment of parameter ranges.

The method is illustrated with the help of a single parameter V_i. Figure 1 shows how the parameter values are initially divided into two ranges: $V_{i,1}$ defined by its minimum and maximum values $[lowerbound(V_{i,1}), upperbound(V_{i,1})]$, and $V_{i,2}$ which is the set of values that lie within $[lowerbound(V_{i,2}), upperbound(V_{i,2})]$. At the beginning of the search, both intervals have equal success rates, denoted as the conditional probabilities $P(e = e^+|V_i = V_{i1})$ and $P(e = e^+|V_i = V_{i2})$, with e denoting the expectation and e^+ denoting a successful outcome given the usage of a value from range V_{ij} for the parameter V_i. This value is calculated as the ratio of the number of times the usage of the value V_{ij} was successful and the number of times it was used, denoted $\frac{u^s_{ij}}{u_{ij}}$ in the algorithmic listing 2. In the illustration, an equal height of two ranges represents the equality of the probabilities of both ranges to be selected for use in the next iteration.

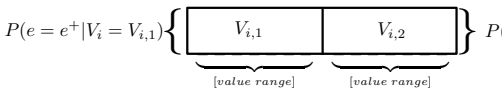


Fig. 1: Parameter V_i divided into two ranges which have had equal success rates in the previous iteration.

After applying the parameter values sampled from the ranges for the optimisation process, the conditional probabilities of each interval are recalculated based on their usage and performance in the latest iteration. Assuming that the new conditional probabilities have the proportions shown in Figure 2, the success rate of the first interval, i.e. $P(e = e^+|V_i = V_{i1})$, is greater than that of the second interval ($P(e = e^+|V_i = Vi2)$).

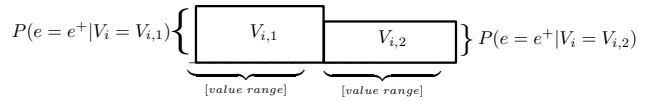


Fig. 2: The new success rates of the levels of parameter V_i based on their performance in the most recent iteration of the algorithm.

The adaptive range selection strategy divides the level with the highest success rate into two new levels, denoted as $P(e = e^+|V_i = V_{i1_1})$ and $P(e = e^+|V_i = V_{i1_2})$, which are shown in Figure 3.

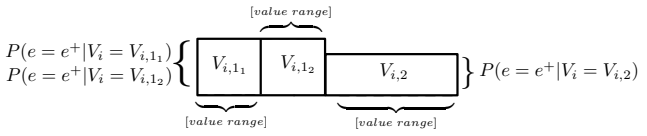


Fig. 3: The level with the highest success rate is divided into two.

The conditional probabilities of both new levels are equal to the conditional probability of the level they were created from. That is:

$$P(e = e^+|V_i = V_{i1_1}) = P(e = e^+|V_i = V_{i1})$$
$$P(e = e^+|V_i = V_{i1_2}) = P(e = e^+|V_i = V_{i1})$$

As a result, the most successful interval is refined into smaller intervals, and the selection probability of the values that lie within these ranges is increased. This increases the exploitation of the intervals and the exploration of new values within the intervals.

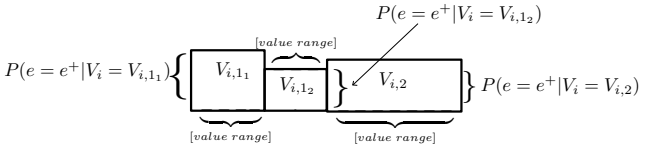


Fig. 4: The new success rates adjusting the selection probabilities of the values of parameter V_i after running the algorithm.

The adaptive range selection strategy merges worst performing range, interval $V_{i,1_2}$ in Figure 4, with the worse-performing neighbouring interval. In this case, interval $V_{i,1_2}$ has been merged with interval $V_{i,2}$ forming the new interval $V_{i,2}$ as shown in Figure 5.

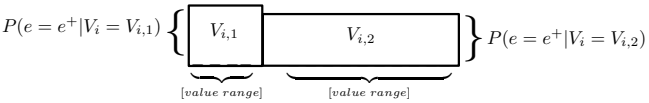


Fig. 5: The levels with the lowest success rates are merged.

The selection probability of the new interval is set equal to the higher selection probability of the two ranges.

The algorithmic listings 1 and 1 demonstrate how the adaptive parameter control applies the dynamic changes to the bin ranges.

The algorithmic listing 2 describes the steps. Every parameter V_i has a number of ranges V_{ij}. After each iteration, we investigate the parameter's range's success rate $\frac{u^s_{ij}}{u_{ij}}$, where u^s_{ij} denotes the number of times this range V_{ij} has performed above average, as a ratio of how many times this range V_{ij} was used in the last iteration, denoted as u_{ij}. The variable p_i^{best} then holds the best ratio of all ranges of parameter V_i, and V_i^{best} points to the best-performing range of this parameter. Analogously, p_i^{worst} stores the success rate of the worst-performing range of parameter V_i. The best range V_i^{best}

Algorithm 1 Adaptive Range Parameter Control

procedure ARPC(n, e, k)
  for $i \leftarrow 1, n$ **do**
    Sample n parameter values for k algorithms according to probability P_t
    Execute k algorithms
    Calculate $\overline{f}(x)$ of k trial outcomes
    for all $V_i \in n$ **do**
      for all $a \in k$ **do**
        if a used value in range j **then**
          increment u_{ij}
          if a has solutions $> \overline{f}(x)$ **then**
            increment u^s_{ij}
          end if
        end if
      end for
    end for
  end for
  for all $V_i \in n$ **do**
    for all $V_{ij} \in m$ **do**
      add $\frac{u^s_{ij}}{u_{ij}}$ to history of V_{ij}
    end for
  end for
  ADJUSTBINS(V_i)
  Calculate $P_t + 1$ using least squares regression
end procedure

Algorithm 2 Adjusting Bin Sizes

procedure ADJUSTBINS(V_i)
  for all $V_i \in n$ **do**
    for all $V_{ij} \in m$ **do**
      if $\frac{u^s_{ij}}{u_{ij}} > p_i^{best}$ **then**
        $p_i^{best} \leftarrow \frac{u^s_{ij}}{u_{ij}}$
        $V_i^{best} \leftarrow V_{ij}$
      else if $\frac{u^s_{ij}}{u_{ij}} < p_i^{worst}$ **then**
        $p_i^{worst} \leftarrow \frac{u^s_{ij}}{u_{ij}}$
        $V_i^{worst} \leftarrow V_{ij}$
      end if
    end for
  end for
  $range \leftarrow max(V_i^{best}) - min(V_i^{best})$
  $upperB(V_i^{best1}) \leftarrow lowerB(V_i^{best}) + range/2$
  $lowerB(V_i^{best2}) \leftarrow upperB(V_i^{best}) - range/2$
  $p_i^{best1} \leftarrow p_i^{best}$
  $p_i^{best2} \leftarrow p_i^{best}$
  $range \leftarrow max(V_i^{worst}) - min(V_i^{worst+1})$
  $upperB(V_i^{worst+1}) = min(V_i^{worst+1}) + range$
  $p_i^{new} \leftarrow p_i^{worst}$
end procedure

is subsequently split into V_i^{best1} and V_i^{best2}, both of which are assigned the raw probability value of p_i^{best}. Similarly, the worst range V_i^{worst} is expanded to cover the worse-performing of its neighbours $V_i^{worst+1}$, and the new range is assigned the raw probability value p_i^{worst} of the worst-performing range.

IV. Benchmark Problems

A. Quadratic Assignment

The Generalised Quadratic Assignment Problem (GQAP) is one of the most difficult combinatorial optimisation problems. The aim is to assign M utilities to N locations with minimal cost. The candidate assignments are evaluated according to equation 1.

$$\sum_{i,j} B_{ik} \cdot u_{ik} + \sum_{i,j,k,l} C_{i,j,k,l} \cdot u_{ik} \cdot u_{jl} \tag{1}$$

where

- B_{ik} is the cost of assigning utility i to location k
- $C_{i,j,k,l}$ 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

GQAP allows for multiple assignments to the same location subject to the availability of space as described by equation 2.

$$\sum_{i=1}^{M} a_{ik} \cdot u_{ik} \leq S_k (k = 1...N) \tag{2}$$

where

- a_{ik} is the space needed for utility i at location k
- S_k is the space available at location k

The GQAP 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.

B. The Royal Road Problem

Mitchell, Forrest, and Holland [18] especially devised the Royal Road (RR) problem to demonstrate that there exist problems which are easier to solve using a Genetic Algorithm than a hill climber.

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, 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.

The RR function is being used here as a benchmark to match the Genetic Algorithm, whose parameters are being optimised for the experimental results presented. It has also been used by Fialho, Schoenauer and Sebag [12], whose results are being used for the comparison.

C. Component Deployment Optimisation

One of the practical applications of stochastic optimisers is the component deployment problem in embedded systems, 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 (Papadopoulos and Grante [25]). 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 as a set of software components and a set of hardware hosts as listed below.

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:

- Component communication frequency $CF : C \times C \rightarrow \mathbb{R}$, where $CF(c_i, c_j) = 0$ if $c_i = c_j$ or there is no communication between c_i and c_j.
- Component event size $ES : C \times C \rightarrow \mathbb{N}$, where $ES(c_i, c_j) = 0$ if $c_i = c_j$ or there is no event occurring c_i and c_j.

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:

- Network bandwidth $NB : H \times H \rightarrow \mathbb{N}$, where $NB(h_i, h_j) = 0$ if $h_i = h_j$ or there is no network connection between h_i and h_j.
- Network reliability $R : H \times H \rightarrow \mathbb{R}$.
- Network delay $ND : H \times H \rightarrow \mathbb{N}$, where $ND(h_i, h_j) = 0$ if $h_i = h_j$ or there is no network connection between h_i and h_j.

The deployment problem is then defined as $D = \{d \mid d : H \rightarrow C_{sub}\}$, 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 [19]. 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. The Data Transmission Reliability (DTR) formulation we use has first been defined by Malek [19].

$$f_{DTR}(d) = \sum_{i=1}^{n} \sum_{j=1}^{n} freq(c_i, c_j) \cdot rel(d(c_i), d(c_j)) \tag{3}$$

In embedded systems with their constrained hardware resources, repeated transmissions between software components are discouraged. The Communication Overhead (CO) [20] objective attempts to enforce minimal data communication for a given set of components and system parameters. As a network- and deployment-dependent metric, the overall communication overhead of the system is used to quantify this aspect. It was first formalised by Medvidovic and Malek [20].

$$f_{CO}(d) = \sum_{i=1}^{n}\sum_{j=1}^{n} freq(c_i, c_j) \cdot nd(d(c_i), d(c_j)) + \\ + \sum_{i=1}^{n}\sum_{j=1}^{n} \frac{freq(c_i, c_j) \cdot ds(c_i, c_j)}{dr(d(c_i), d(c_j)) \cdot rel(d(c_i), d(c_j))} \tag{4}$$

The Scheduling Time (ST) objective is another objective to be minimised. It describes the average time for a hardware unit to complete the round-robin processing of all its assigned components. ST is given by

$$f_{ST}(d) = \frac{1}{m} \cdot \sum_{j=1}^{m} \left(\frac{\sum_{i \in C_{u_j}} mi_i}{ps_j} \right). \tag{5}$$

V. Experimental Design

A. Evolutionary Algorithm

The Royal Road problem is optimised using an EA with a representation using a binary string. The operators in use are bit-flip mutation and multipoint crossover.

GQAP 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 GQAP, singularity of item occurrence is not mandatory and the solutions are always valid after the operators have been applied.

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. [8].

B. Parameter Optimisation

The crossover and mutation rates are probably the most conspicuous control parameters to optimise in stochastic optimisation [10]. Hence, for the benefit of these experiments, only the crossover and mutation rates were varied. Based on preliminary exploration, a range of $0.01 - 0.7$ was adopted for the mutation rate and the interval $0.01 - 1.0$ was used for the crossover operator.

The parameter control method was invoked every time the optimising EA completed an iteration comprised of 150 function evaluations. The probabilities were calculated and new parameter values were assigned for the next iteration. This process was repeated 20 times. Consequently, each trial is allowed 3000 function evaluations. These settings apply to all benchmark optimisation trials regardless of the problem at hand.

In the case of GQAP and RR, 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 [32].

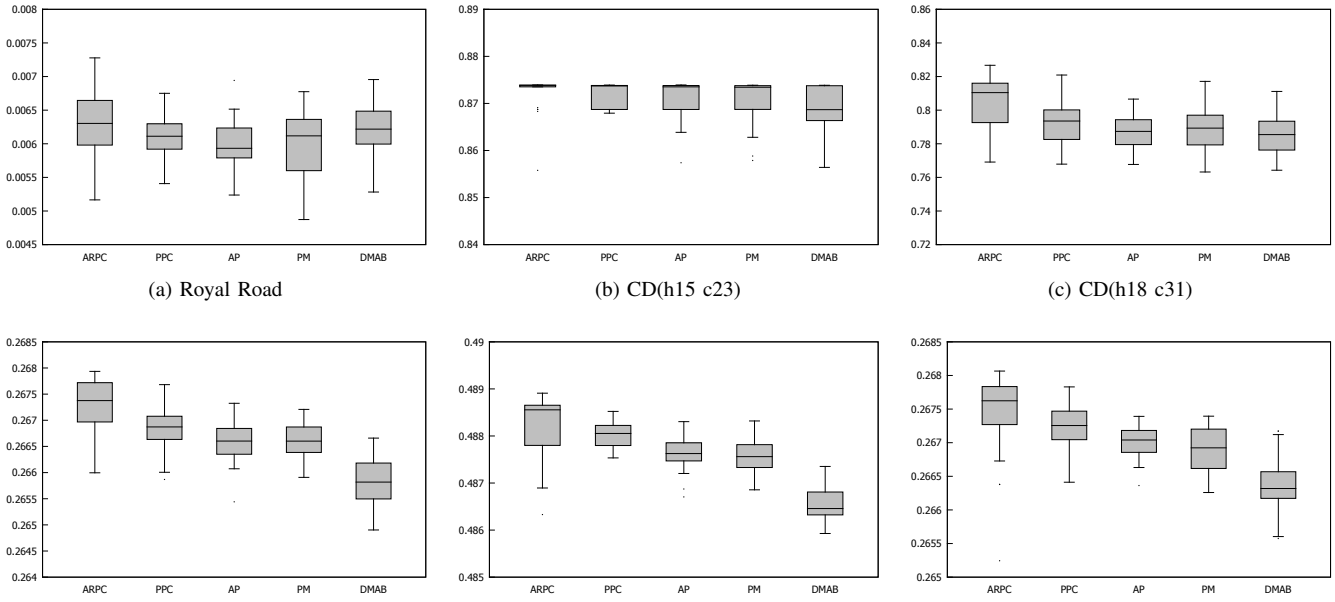
VI. Main Results

In ARPC, the change in the ranges of the intervals of crossover and mutation rates during 20 iterations is depicted in Figure 7. At the beginning of the optimisation process, all intervals are equal. The bigger the interval becomes, the smaller is the chance of the values in that interval to be selected. We can clearly see that the behaviour of adaptive range parameter control is different for different problems and different parameters.

The 30 results of the repeated trials are presented as boxplots in Figure 6. The empirical results are not normally distributed, but the mean and 25th percentile of ARPC are consistently above the respective values of the benchmark approaches. The means and standard deviations are listed in Table I, which clearly show a significant difference between the result groups of ARPC and the benchmarks. The mean performance of ARPC is consistently above the averages of the benchmark approaches. However, the standard deviation of ARPC is relatively high.

The gap between result qualities widens in favour of ARPC as the problem difficulty increases. The smaller automotive deployment problem can be assumed to be the least challenging, and there the results are not as clearly in favour of ARPC. The larger one of the automotive problems is clearly solved to better quality using ARPC, as are the more complex GQAP problems.

As our method consistently outperforms the four other optimisation schemes, to check for a statistical difference, the different parameter schemes of the optimisation methods are validated using the Kolmogorov-Smirnov (KS) nonparametric



(a) Royal Road (b) CD(h15 c23) (c) CD(h18 c31)

(d) QAP26a (e) QAP26b (f) QAP26e

Fig. 6: Boxplots of the 30 trials of the five different optimisation schemes.

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

	Mean					
	Royal Road	CD(h15 c23)	CD(h18 c31)	QAP26a	QAP26b	QAP26e
AP	5.9757E-03	0.8711	0.7863	0.2666	0.4876	0.2670
PM	5.9950E-03	0.8706	0.7893	0.2666	0.4876	0.2669
DMAB	6.1599E-03	0.8686	0.5172	0.2658	0.4865	0.2664
PPC	6.1166E-03	0.8717	0.7918	0.2668	0.4880	0.2673
ARPC	**6.2736E-03**	**0.8725**	**0.8036**	**0.2672**	**0.4882**	**0.2674**
	Standard Deviation					
	Royal Road	CD(h15 c23)	CD(h18 c31)	QAP26a	QAP26b	QAP26e
AP	3.616E-04	3.896E-03	**9.931E-03**	3.998E-04	3.673E-04	**2.491E-04**
PM	4.983E-04	4.415E-03	1.268E-02	**3.398E-04**	3.401E-04	3.189E-04
DMAB	3.713E-04	4.842E-03	1.162E-02	4.410E-04	3.564E-04	4.233E-04
PPC	**3.495E-04**	**2.538E-03**	1.266E-02	4.471E-04	**2.869E-04**	3.268E-04
ARPC	5.511E-04	3.679E-03	1.620E-02	1.032E-03	6.286E-04	6.325E-04

test [26]. The 30 hypervolume indicators of the repeated trials for each of the problem instances were submitted to the ks analysis. ARPC was compared to the other four optimisation schemes, with a null hypothesis of a significant difference between the performances (ARPC vs. PPC, ARPC vs. DMAB, ARPC vs. AP and ARPC vs. PM). The results of the tests are shown in Table II.

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 0.2414 at a 70% confidence level. Hence we conclude that the superior performance of PPC is statistically significant.

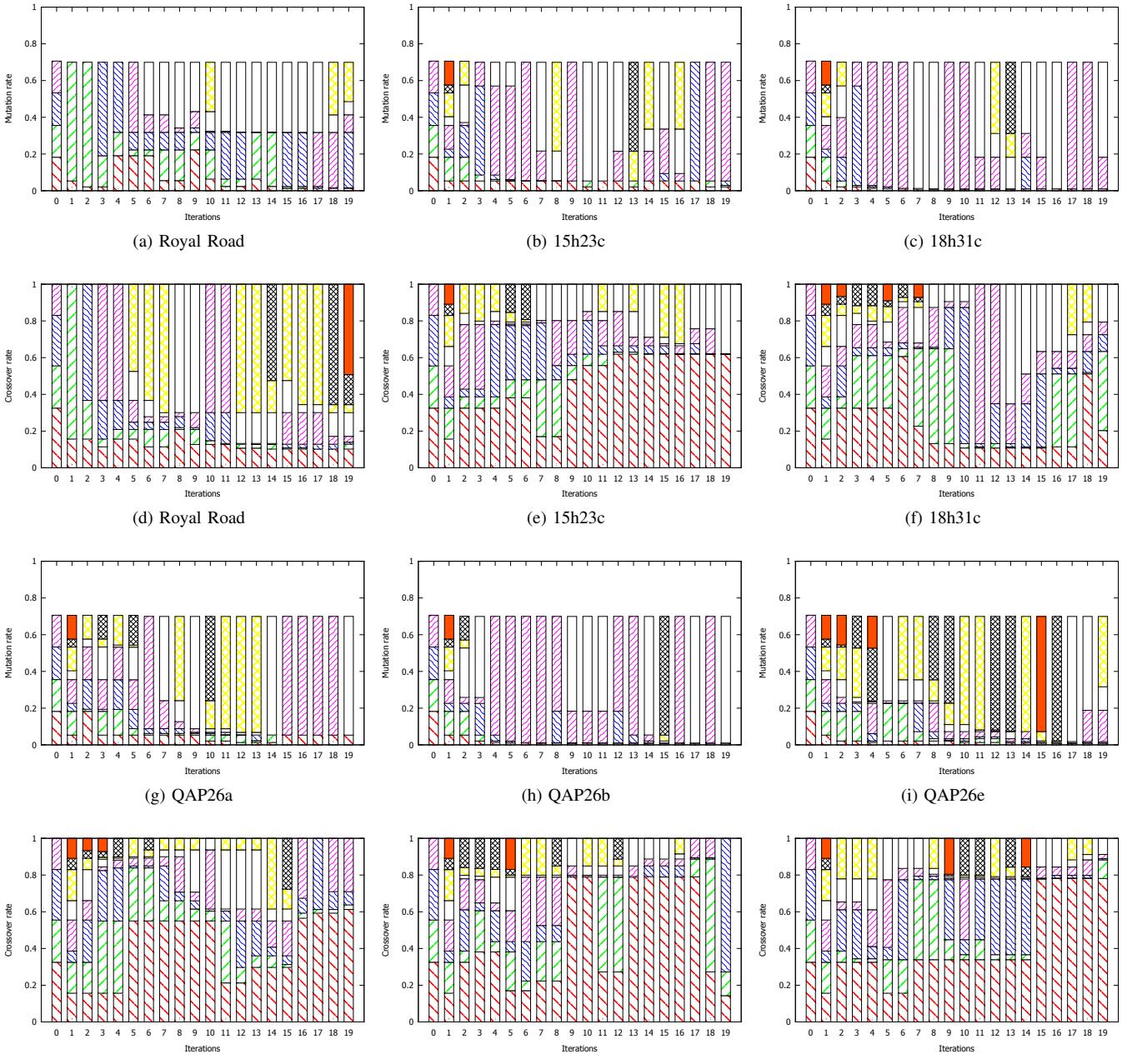
The change in the ranges of the intervals of crossover and mutation rates during 20 iterations is depicted in Figure 7. At the beginning of the optimisation process, all intervals are equal. The bigger an interval becomes, the smaller its chance of being sampled for the next iteration. Accordingly, the most successful values for each iteration are to be placed in the smallest interval.

From the bar diagrams we can see that the - relatively small - automotive problem instances are best optimised with a very small mutation rate throughout the process, whereas the RR problem seems to need slightly higher mutation rates (approx. 0.2) at the start but toward the end of the process the level ranges are not as focussed. A different observation can be made regarding the optimal mutation rates for the GQAP instances; there, the most successful mutation rates are clearly very low at the end of the optimisation process.

The levels of crossover rate develop quite differently compared to mutation rate. Higher rates are often more successful towards the end of the optimisation process which runs somewhat contrary to popular opinion that crossover rates should decrease towards the end of the optimisation process so as

TABLE II: The Kolmogorov-Smirnov test values of fitness functions for the 30 runs of each problem instance using different optimisation schemes.

	Royal Road		CD(h15 c23)		CD(h18 c31)		QAP26a		QAP26b		QAP26e	
	d	p	d	p	d	p	d	p	d	p	d	p
ARPC vs. PPC	0.2759	0.184	0.3807	0.029	0.4483	0.004	0.4828	0.001	0.5862	0.000	0.4138	0.009
ARPC vs. DMAB	0.2414	0.321	0.5172	0.000	0.5172	0.000	0.8621	0.000	0.9310	0.000	0.7586	0.000
ARPC vs. AP	0.3793	0.022	0.5172	0.000	0.6207	0.000	0.6207	0.000	0.6552	0.000	0.6552	0.000
ARPC vs. PM	0.2759	0.184	0.5517	0.000	0.5172	0.000	0.6539	0.000	0.6552	0.000	0.6552	0.000



(a) Royal Road (b) 15h23c (c) 18h31c

(d) Royal Road (e) 15h23c (f) 18h31c

(g) QAP26a (h) QAP26b (i) QAP26e

(j) QAP26a (k) QAP26b (l) QAP26e

Fig. 7: Levels over time.

not to disturb solutions with high quality. For some problems, both crossover rates from the upper third and from the lower third of the overall range seem beneficial at the same time.

VII. Conclusions

In this paper we have introduced a meta-algorithm to adjust variables needed for the parameterisation of an optimisation algorithm. According to our knowledge, the best-performing state-of-the-art approaches with the same functionality are AP, PM, DMAB and PPC. The new approach clearly outperforms the other parameter control methods. As the problem difficulty increases, so does the difference in result quality produced by ARPC compared to the benchmark approaches.

The adaptation of parameter value ranges increases the sampling accuracy, since the more successful a parameter value range is, the smaller the interval becomes, which can explain the superior performance of ARPC compared to other approaches. Furthermore, unlike other adaptive parameter control methods, ARPC does not have any hyperparameter that requires tuning before the optimisation process.

The mutation/crossover range analysis shows that high-performing ranges are sometimes absorbed (merged) into very large intervals, making it difficult for the algorithm to re-establish small, promising areas within the range. There may be a potential for further optimisation of the range adaptation in this respect.

In the future, we will also investigate the use of information about possible correlations of the values of different parameters as well as the potential integration of such information into the meta-optimiser.

VIII. 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).

References

[1] A. Aleti and I. Moser. Predictive parameter control. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, GECCO '11, pages 561–568, New York, NY, USA, 2011. ACM.

[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. 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.

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

[5] 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.

[6] K. Deb and S. Agrawal. Understanding interactions among genetic algorithm parameters. In *in Foundations of Genetic Algorithms 5*, pages 265–286. Morgan Kaufmann, 1999.

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

[10] G. 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.

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

[12] A. 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*, GECCO '09, pages 779–786. ACM, 2009.

[13] D. E. Goldberg, K. Deb, and J. H. Clark. Genetic algorithms, noise, and the sizing of populations. *COMPLEX SYSTEMS*, 6:333–362, 1991.

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

[15] Z.-F. Hao, R.-C. Cai, and H. Huang. An adaptive parameter control strategy for aco. In *Machine Learning and Cybernetics, 2006 International Conference on*, pages 203 –206, aug. 2006.

[16] W. E. Hart and R. K. Belew. Optimizing an arbitrary function is hard for the genetic algorithm. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 190–195. Morgan Kaufmann, 1991.

[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] 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.

[20] 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.

[21] E. Mezura-Montes and A. Palomeque-Ortiz. Self-adaptive and deterministic parameter control in differential evolution for constrained optimization. In E. Mezura-Montes, editor, *Constraint-Handling in Evolutionary Optimization*, volume 198 of *Studies in Computational Intelligence*, pages 95–120. Springer Berlin / Heidelberg, 2009.

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

[23] S. Mostaghim, F. Pfeiffer, and H. Schmeck. Self-organized invasive parallel optimization. In *Proceedings of the International Workshop on Bio-inspired Approaches for Distributed Computing*, pages 49 – 56, 2011.

[24] 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.

[25] 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.

[26] 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.

[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] A. S. Wu, R. K. Lindsay, and R. L. Riolo. Empirical observations on the roles of crossover and mutation. In *In Back [Bac97*, pages 362–369. Morgan Kaufmann, 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, 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.