Invasive Parameter Control

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

S. Mostaghim
Karlsruhe University
Karlsruhe, Germany
Email: sanaz.mostaghim@kit.edu

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

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], [12], [29], 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 [21], often even specific to the problem instance at hand [2], [27], [26], [16]. Moreover, the interactions between parameters have been found to be problem-specific [15]. 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], [30], [4], [23]. Moreover, it has also been established that some of the parameter values ought to vary during the search process for best algorithm performance [2], [27], [26], [16].

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 [20], 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], [10], [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. [28], [11]) 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 [13], [11], [28], [14]. 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) [28], Adaptive Pursuit (AP) [28], Dynamic Multi-Armed Bandit (DMAB) [11] and Predictive Parameter Control (PPC) [1].

A. Probability Matching

PM [28] 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

AP [28] 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

DMAB [11] 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 \( t + 1 \).

Aldo ▶ aren’t discretised parameter value choices the same thing as predefined ranges ◀ 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. INVASIVE 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.

IPC 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 range is merged with the worse-performing of its neighbours.

This technique was first conceived for the context of parallel computing [22] 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 \( P(e = e^+|V_i = V_{i,1}) \) and \( P(e = e^+|V_i = V_{i,2}) \), 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 \( \frac{u_{ij}}{u_{ij} + d_{ij}} \) 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.

\[
P(e = e^+|V_i = V_{i,1}) = \frac{V_{i,1}}{V_{i,2}} \cdot \frac{P(e = e^+|V_i = V_{i,1})}{P(e = e^+|V_i = V_{i,2})}
\]

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. \( P(e = e^+|V_i = V_{i,1}) \), is greater than that of the second interval \( P(e = e^+|V_i = V_{i,2}) \).

\[
P(e = e^+|V_i = V_{i,1}) = \frac{V_{i,1}}{V_{i,2}} \cdot \frac{P(e = e^+|V_i = V_{i,1})}{P(e = e^+|V_i = V_{i,2})}
\]

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 invasive selection strategy divides the level with the highest success rate into two new levels, denoted as \( P(e = e^+|V_i = V_{i,1}) \) and \( P(e = e^+|V_i = V_{i,2}) \), which are shown in Figure 3.
Algorithm 1 Invasive Parameter Control

1: procedure IPC(n, e, k)
2:  for i ← 1, n do
3:     Sample n parameter values for k algorithms according to probability P_{ij}
4:  end for
5:  Execute k algorithms
6:  Calculate \( \mathcal{F}(x) \) of k trial outcomes
7:  for all \( V_i \in n \) do
8:      for all \( a \in k \) do
9:          if a used value in range \( j \) then
10:             increment \( u_{ij} \)
11:          end if
12:      end if
13:  end for
14:  end for
15: end procedure

Algorithm 2 Adjusting Bin Sizes

procedure ADJUSTBINS(\( V_i \))
2:  for all \( V_i \in n \) do
3:      for all \( V_{ij} \in m \) do
4:          if \( \frac{u_{ij}}{u_{ij}} > p_{ij}^{\text{best}} \) then
5:              \( p_{ij}^{\text{best}} \leftarrow p_{ij} \)
6:              \( V_i^{\text{best}} \leftarrow V_{ij} \)
7:          else if \( \frac{u_{ij}}{u_{ij}} < p_{ij}^{\text{worst}} \) then
8:              \( p_{ij}^{\text{worst}} \leftarrow p_{ij} \)
9:              \( V_i^{\text{worst}} \leftarrow V_{ij} \)
10:         end if
11:      end if
12:  end for
13: end procedure
is subsequently split into $V_{i}^{\text{best}1}$ and $V_{i}^{\text{best}2}$, both of which are assigned the raw probability value of $p_{i}^{\text{best}}$. Similarly, the worst range $V_{i}^{\text{worst}}$ is expanded to cover the worse-performing of its neighbours $V_{i}^{\text{worst}+1}$, and the new range is assigned the raw probability value $p_{i}^{\text{worst}}$ 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.

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

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.

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

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 [17] 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 [11], 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 [24]). 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 $CF : \mathbb{C} \times \mathbb{C} \rightarrow \mathbb{R}$, 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 $ES : \mathbb{C} \times \mathbb{C} \rightarrow \mathbb{N}$, 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 $NB : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$, 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 $R : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{R}$.
- Network delay $ND : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$, 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 $D = \{d \mid d : H \rightarrow C_{\text{sub}}\}$, where $C_{\text{sub}} = \{c_{0}, c_{1}, ..., c_{i}\} \subset C$, 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 [18]. 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 [18].

$$
f_{\text{DTR}}(d) = \sum_{i=1}^{n} \sum_{j=1}^{n} \text{freq}(c_{i}, c_{j}) \cdot \text{rel}(d(c_{i}), d(c_{j}))
$$
In embedded systems with their constrained hardware resources, repeated transmissions between software components are discouraged. The Communication Overhead (CO) [19] 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 [19].

\[
f_{CO}(d) = \sum_{i=1}^{n} \sum_{j=1}^{n} f_{req}(c_i, c_j) \cdot nd(d(c_i), d(c_j)) + \sum_{i=1}^{n} \sum_{j=1}^{n} f_{req}(c_i, c_j) \cdot ds(c_i, c_j)
\]

(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

\[
f_{ST}(d) = \frac{1}{m} \cdot \sum_{j=1}^{m} \left( \frac{\sum_{i \in C_{s_j}} n_{i,j}}{p_{s_j}} \right).
\]

(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 component. 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

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 [31].

VI. MAIN RESULTS

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 invasive 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 IPC 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 IPC and the benchmarks. The mean performance of IPC is consistently above the averages of the benchmark approaches. However, the standard deviation of IPC is relatively high.

The gap between result qualities widens in favour of IPC 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 IPC. The larger one of the automotive problems is clearly solved to better quality using IPC, 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 test [25]. The 30 hypervolume indicators of the repeated trials for each of the problem instances were submitted to the ks analysis. IPC was compared to the other four optimisation schemes, with a null hypothesis of a significant difference.
between the performances (IPC vs. PPC, IPC vs. DMAB, IPC vs. AP and IPC 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 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 IPC compared to the benchmark approaches.

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.

REFERENCES

Fig. 7: Levels over time.


