A Model for Characterising the Collective Dynamic Behaviour of Evolutionary Algorithms

Mikdam Turkey [1] and **Riccardo Poli** [2]

Abstract. Exploration and exploitation are considered essential notions in evolutionary algorithms. However, a precise interpretation of what constitutes exploration or exploitation is clearly lacking and so are specific measures for characterising such notions. In this paper, we start addressing this issue by presenting new measures that can be used as indicators of the exploitation behaviour of an algorithm. These work by characterising the extent to which available information guides the search. More precisely, they quantify the dependency of a population's activity on the observed fitness values and genetic material, utilising an empirical model that uses a coarse-grained representation of population dynamics and records information about it. The model uses the k-means clustering algorithm to identify the population's "basins of activity". The exploitation behaviour is then captured by an entropy-based measure based on the model that quantifies the strength of the association between a population's activity distribution and the observed fitness landscape information.

In experiments, we analysed the effects of the search operators and their parameter settings on the collective dynamic behaviour of populations. We also analysed the effect of using different problems on algorithm behaviours. We define a *behavioural landscape* for each problem to identify the appropriate behaviour to achieve good results and point out possible applications for the proposed model.

1 Introduction

Population-based search algorithms solve problems using a set of interaction mechanisms which control the generation of new individuals in a population. Based on the information obtained via previous individuals, an algorithm directs the search and changes the density of the population in different regions. This bias in directing the movement of the population is *hoped* to guide the algorithm towards promising areas in the search space and exploring them with more intensity than less promising areas. As the interaction mechanisms operate on moving, creating and/or eliminating individuals, the algorithm produces an emergent "large-scale" behaviour in the dynamics of the population.

Population dynamics has been a central issue in many theoretical investigations. For example, within evolutionary computation theory, approaches include: the schema theory [8, 22, 23], Markov chain formulations [27], Walsh-function-based analyses [1], statistical-mechanical formulations [21] and also some computational complexity approaches [29, 9, 10]. The study of the dynamics/behaviour of populations with respect to specific problems has also been the subject of a number of studies where empirical measures were defined and studied. For example, semi-empirical measures of problem difficulty, such as the fitness-distance correlation [12] and the negative slope coefficient [17], have been proposed to characterise what makes a problem easy or hard for evolutionary algorithms.

All of these approaches have seen some successes at mathematically or empirically modelling evolutionary algorithms. However, very often researchers describe the behaviour of search algorithms in terms of exploration and exploitation, not of run times, success probabilities, or Walsh coefficients. Exploration refers to behaviour resulting in the discovery of new good regions in the search space, while exploitation refers to the behaviour of exploring previously discovered good regions [8, 3]. Researchers have found that to achieve good performance it is often important to control the trade-off between explorative and exploitative behaviour. This is done by either tuning the parameters of search algorithms prior to a run or by changing them dynamically throughout the run based on features of the population such as fitness or diversity. However, *there is no precise definition in the literature of the notions of exploration and exploitation, no precise characterisation of the distinction between them, and no numerical quantification of them.*

In our previous work [25], we presented a model to analyse the collective dynamic behaviour of population-based algorithms. Based on that model, we defined many measures to extract emergent features attributed to that behaviour. These features can be used to *qualitatively* describe the exploitation/exploration behaviour of an algorithm. We used self-organising maps (SOMs) [13] to implement the model. The SOMs track the population's movement as an algorithm operates on solving a problem and this allowed us to gather information about its activities in each of the regions currently being explored. In [26], we have taken the work further by performing more experiments and presenting a new set of measures to characterise the dependency of different collective behaviour features on observed fitness values. We called these measures *fitness dependency* measures and used them as indicators of the exploitation behaviour of a population. Building on that work, here we have used the model presented in [26, 25] to define a new measure, the *genetic dependency* that we have used along with fitness dependency to analyse the influence of the observed fitness values and genetic material in guiding the search activities. Our current model is implemented using the k-means clustering method [4] to identify regions of activity within the population over a run. Based on this model we define entropy-based measures [18] to quantify the dependency of population activities on fitness values and on the diversity of the available genetic material. The proposed measures are used as *indicators of the exploitation behaviour* of algorithm as they characterise the extent by which a population

[1] School of Computer Science & Electronic Engineering, University of Essex, Wivenhoe Park, Colchester, Essex, CO4 3SQ, email: {mturkey, rpoli}@essex.ac.uk

[2] School of Computer Science & Electronic Engineering, University of Essex, Wivenhoe Park, Colchester, Essex, CO4 3SQ, email: {mturkey, rpoli}@essex.ac.uk

exploits available information to guide search activities.

The proposed model is an effort to contribute to our understanding of the exploration/exploitation phenomena by trying to define *what constitutes exploitation* and measuring quantities that assess the degree of that behaviour in an algorithm. An algorithm produces different behaviours when it operates on different problems. In accordance with the No Free Launch (NFL) theorem [30], only certain types of collective dynamic behaviour will lead the algorithm to perform well on a problem. In this work, we show that, for a given problem, an algorithm's performance can be modelled as a function of the algorithm's collective behaviour. In the future, this performance model could be used to implement an algorithm selection mechanism to help identify algorithms with a collective behaviour that likely to achieve better results on a problem. It could also be used to tune an algorithm's parameters so as to exhibit such behaviour or to guide the design of new algorithms.

The rest of the paper is organised as follows. In Section 2, we present the proposed model and define the dependency measures that will be used to characterise the exploitation behaviour. Section 3 outlines the details of our experiments and presents our results of different genetic operators, their settings and interaction on the collective exploitation behaviour of an algorithm. We consider the possible ways in which our model and indicators can be used in Section 4 and we finish with some conclusions in Section 5.

2 The Proposed Model

We will first describe the proposed model (Section 2.1) and then present the dependency measures that we use to characterise a population's exploitation behaviour (Section 2.2).

2.1 Formulation

Our model is similar, but not identical, to the one proposed in [25]. The proposed model uses a set of nodes to capture the distribution of population individuals at certain points in time. Each node has a *centroid* representing the centre of a group of individuals in a localised area of the search space. In addition, nodes store information describing the activities of population in the corresponding area. To track the dynamic behaviour of a population over a run, the model is instantiated every so many time steps to represent the motion of the population and record information about its activities as the algorithm progresses in solving a problem.

Formally, a set of nodes, C^t, represents a snapshot of the population distribution at time t. Let C^0 be the set of nodes that represents the distribution of the initial individuals in the population, $P_{init} = \{(x_1, y_1), \ldots, (x_\eta, y_\eta)\}$, where η is the population size, $x_i \in \mathbb{R}^D$ represents the position of individual i in the search space (D being its dimension) and $y_i \in \mathbb{R}$ is the associated fitness value. Also, let the time-ordered set $P = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ represent a sequence of all the individuals created by an algorithm over a period of a run from the initial population, N being the total number of individuals created by the algorithm (i.e., the total number of calls to the fitness function).

The sets of nodes $C^1, C^2, \ldots$ are created every τ individuals produced by the algorithm (i.e., τ represents the sampling period). For example, C^1 captures the distribution of the sequence $P_1 = \{(x_1, y_1), \ldots, (x_\tau, y_\tau)\}$. Generally speaking, C^t is created by tracking and recording information about individuals in the sequence $P_t = \{(x_{k+1}, y_{k+1}), \ldots, (x_{k+\tau}, y_{k+\tau})\}$, where $k = \tau \times (t-1)$.

More precisely the set C^t consists of n_t nodes, where each node, C^t_r, is represented by the following tuple

$$C^t_r = \langle m^{C^t_r}, h^{C^t_r}, d^{C^t_r}, f^{C^t_r} \rangle \qquad (1)$$

where in C^t_r, t is time and $r \in \{1 \ldots n_t\}$ is the position in the set.

The elements of P_t are partitioned into non-overlapping subsets, $P^{C^t_r}$, the individuals of which are associated with the node C^t_r based on their distance from the r-th centroid (as explained later in this section). Let $x^{C^t_r}$ and $y^{C^t_r}$ represent sets of positions and fitness values of the individuals associated with C^t_r, respectively. Then the elements of the tuple C^t_r are computed as follows:

$m^{C^t_r} \in \mathbb{R}^D$ is the node *centroid*. The node centroid represents a region of a population activity. As the algorithm works on redistributing the population around activity regions, node centroids are recalculated so as to represent newly created individuals. As we will see later in this section, we utilised the k-means clustering method to identify population's activity regions, divide the individuals among the nodes and calculate the centroid of each one;

$h^{C^t_r}$ (*hit counter*) is the number of individuals associated with the node C^t_r. Formally, $h^{C^t_r} = |P^{C^t_r}|$. It represents the *amount of activity* that an algorithm allocates for a particular area of the search space.

$d^{C^t_r}$ (*hit distance*) is the mean distances between a node's centroid, $m^{C^t_r}$, and the positions of the individuals, $x^{C^t_r}$, associated with C^t_r. Formally, $d^{C^t_r} = \frac{1}{h^{C^t_r}} \sum_{a \in x^{C^t_r}} \|a - m^{C^t_r}\|$. This feature describes the *local genotypic diversity* of an activity region.

$f^{C^t_r}$ is the *mean fitness* of the individuals associated with C^t_r. Formally, $f^{C^t_r} = \frac{1}{h^{C^t_r}} \sum_{a \in y^{C^t_r}} a$. This feature gives insights on the quality of the region of the search space tracked by this node.

As mentioned earlier, unlike [26, 25], here we use the k-means clustering method [4] to partition a sequence of individuals P_t into groups (we call them nodes C^t) based on distance and to calculate the centroid of each group (i.e., $m^{C^t_r}$). This makes our model much simpler. In the previous implementation, SOMs were used to preserve the topological distribution of the individuals in a population and keep track of the changes of that distribution (by observing the changes to a SOM's centroids). These two factors are essential to calculate some of the emergent features presented in the our previous work. However, in this work, we are only interested in capturing a coarse-grained representation of population distribution in order to extract certain features and produce probability distributions for them. For this reason, here we can use the simpler and more efficient k-means clustering algorithm instead of SOMs.

The most common k-means algorithm uses an iterative refinement technique to assign observations (individuals) to clusters (nodes) and to update the centroid of each cluster. The process starts by choosing the initial values for cluster centroids. That can be done by selecting random individuals as the initial centroids or by using a seeding technique. Then, each individual is assigned to the nearest cluster and the centroid of each cluster is computed as $m^{C^t_r} = \frac{1}{|P^{C^t_r}|} \sum_{a \in x^{C^t_r}} a$. The process of assigning individuals to clusters and updating the centroids continues until we reach a point where no further improvements in the value of the within-cluster sum of squares can be obtained. This is defined as $WCSS = \sum_{S \in C^t} \sum_{a \in x^S} \|a - m^S\|^2$. The k-means algorithm can result in empty clusters, in which case we remove them from C^t. Hence, the number of nodes, n_t, changes over time. In our implementation, we choose the initial value of the number of clusters as 25, therefore, $n_t \leq 25$.

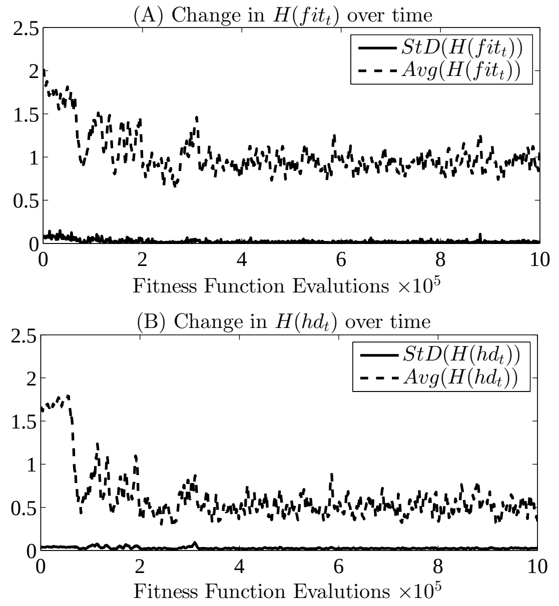

Figure 1. Mean and standard deviation of the entropy of fit_t (A) and hd_t (B) over 10 independent runs of the k-means algorithm (see text). The tiny standard deviations indicate that the sensitivity of k-means to initial conditions adds minimal noise to our entropy measures.

2.2 Characterising the Exploitation Behaviour

Population activities may be guided (i.e. the search may be biased) by various sources of information that an algorithm can use. In addition to the observed fitness values, for example, distance or similarity between individuals can be used, as in fitness-sharing techniques. In this section, we develop a measure to quantify the level of search dependency on certain sources of information and use it as an indicator of a population's exploitation behaviour. In order to characterise the dependency of population behaviour on available information, we have chosen to use a *measure of the amount of certainty this information brings to the activities of a population*. That is to say, we want to assess how much the activity distribution of populations is dependent on the distributions of features such as fitness average or hit distance.

To quantify this association, we used *an entropy-based measure*, which is a normalised variant of mutual information called *uncertainty coefficient* [18]. This is defined as follows:

$$U(Y|X) = \frac{H(Y) - H(Y|X)}{H(Y)} = \frac{I(X,Y)}{H(Y)} \qquad (2)$$

where

$$I(X,Y) = \sum_{x\in X}\sum_{y\in Y} p(x,y)\log\Big(\frac{p(x,y)}{p(x)p(y)}\Big) \qquad (3)$$

is the mutual information between two random variables X and Y and

$$H(X) = -\sum_{x\in X} p(x)\log(x) \qquad (4)$$

is the entropy of a random variable X.

In information theory, entropy is a measure of the uncertainty or randomness associated with a random variable [20], while mutual information is a quantity that measures the mutual dependence of the two random variables. From Equation 2, we can see that mutual information can be expressed as $I(X,Y) = H(Y) - H(Y|X)$, where $H(Y|X)$ is the conditional entropy of Y given X. This quantifies what is left of the uncertainty about Y after knowing X. Based on the definition of mutual information, *the dependency between X and Y is expressed by how much uncertainty about Y has been removed by knowing X*. The value of $U(Y|X)$ lies between zero and one, where 0 means that X and Y have no association, while 1 means that by knowing X one can totally predict Y.

To use the uncertainty coefficient to assess the dependency of a population's behaviour on fitness and hit distance (local genotypic diversity), we first need to compute the probability distribution of the features: hit counter (activity), mean fitness and mean hit distance over all areas of population activity. In our model the probability distribution of a feature is approximated via a histogram of that feature across the nodes in the model. The histogram has m equal-size bins or classes (in this work, we choose $m = 10$) whose width is simply the difference between the maximum and minimum values of a feature over all nodes divided by m. The probability distribution of each class is then computed as the proportion of activity (as indicated by the hit counter) the nodes of each class received. More specifically, if $L_i^{\phi_1}$ for $i = 1\dots m$ are the bins (classes) obtained by dividing up nodes according to the values of the feature ϕ_1, then the probability distribution of that feature is approximated as:

$$p(L_i^{\phi_1}) = \Big(\sum_{X\in L_i^{\phi_1}} h^X\Big)/\tau, \quad \text{for } i = 1\dots m \qquad (5)$$

where τ is the total number of hits (created individuals) between two sampling points and h^X is the hit counter of node X.

We also need to compute the joint probability of two classes of features. Let $L_i^{\phi_2}$ for $i = 1\dots m$, be another sequence of bins for the feature ϕ_2, then the joint probability of ϕ_1 and ϕ_2 is approximated as follows:

$$p(L_i^{\phi_1}, L_j^{\phi_2}) = \Big(\sum_{X\in L_i^{\phi_1}\cap L_j^{\phi_2}} h^X\Big)/\tau, \quad \text{for } i,j = 1\dots m \qquad (6)$$

With these approximations of the probability and joint probability distributions of features in hand, the entropy and mutual information (Equations 4 and 3, respectively) and then the uncertainty coefficient (Equation 2) can be calculated. In particular, we have applied these definitions to analyse the dependency of hit counter (activity) on mean fitness and mean hit distance. For simplicity, we will denote the corresponding values of the uncertainty coefficient as $fd_{a,f}^t$ and $gd_{a,f}^t$, respectively. They can be defined as follows. Let $act_t = \{h^X, \forall X \in C^t\}$, $fit_t = \{f^X, \forall X \in C^t\}$ and $hd_t = \{d^X, \forall X \in C^t\}$, then $fd_{a,f}^t = U(act_t|fit_t)$ and $gd_{a,f}^t = U(act_t|hd_t)$.

The quantity $fd_{a,f}^t$ assesses the dependency of activity on fitness values, therefore, we will refer to it as *fitness dependency*. The quantity $gd_{a,f}^t$, instead, assesses the dependency of activity on genetic material and we will refer to it as *genetic dependency*. The two measures are used to assess the *exploitation behaviour* of algorithm a as it operates on problem f at point of time t.

The average of fitness dependency over one algorithm run is defined as $fd_{a,f} = \frac{1}{M}\sum_{t=1}^{M} fd_{a,f}^t$, where, $M = N/\tau$ represents the total number of sampling points, N is the total number of fitness function evaluations in a run and τ is the sampling period. $gd_{a,f}$ is defined in a similar way. We propose to use these two exploitation measures to represent the dynamic collective behaviour of an algorithm with respect to certain problem *over a run*.

The k-means algorithm is sensitive to the initial values of the cluster centroids, meaning that the clustering method may end up clustering a population's individuals in a different way depending on initial conditions. This might have a potential impact on our measures as it can effect the probability distribution of the emergent features. To examine the effects of clustering noise, we tested the extent to which different initial conditions can effect the entropy of fit_t and hd_t. In this experiment, at each sampling point t, the individuals P_t were clustered 10 times by k-means (using different random seeds). Then, we computed the entropy of fit_t and hd_t, $H(fit_t)$ and $H(hd_t)$, for each of the 10 clustering results. Finally, we compute the average and the standard deviation for these 10 values for each feature. Results indicated that initial conditions for k-means minimally affect our measures, as in all cases the entropy values had very low standard deviations. Figure 1 shows an example of typical run of `BLX-0.5-0.025-2` on $F9$. Similarly we also found that results are not particularly sensitive to the choice of k and that using $k = 25$ in the k-means algorithm provides a good balance between accuracy and computation time. For this reason, we used this value.

3 Experimental Results

In this section we will look at our experimental setup and experiments.

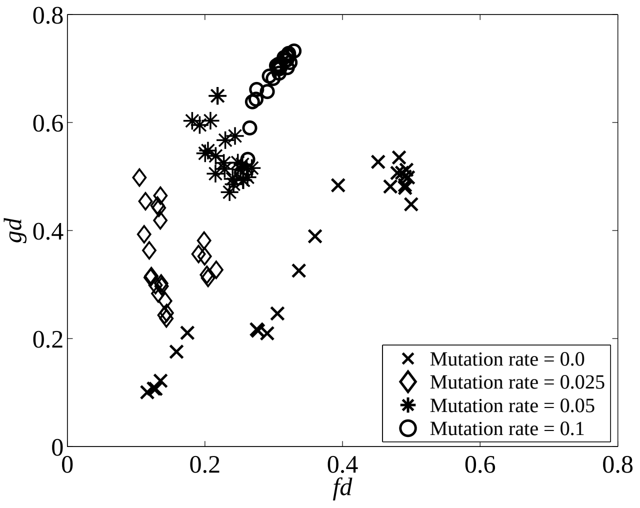

Figure 3. The effects of using different mutation rates on a set of algorithms operating on problem $F4$

3.1 Experimental Setup

The collective dynamic behaviour of an evolutionary algorithm is the result of combining the biases of its search operators and their interaction with the problem landscape. The bias of each operator depends on the nature of the operator itself and the values of its control parameters. Thus, although the exploitation behaviour of an algorithm is a collective phenomenon, it depends on the operator biases, parameter settings and the way operators interact with each other, in addition to the bias generated by the fitness landscape. To examine these effects and interactions in relation to the exploitation behaviour of an algorithm, we applied our dependency measures to evolutionary algorithms using three different crossover methods, different levels of selection pressure and different mutation rates. We used generational evolutionary algorithms with fixed population size (100) and constant crossover rate (0.7). Non-uniform mutation has been used with different mutation rates. To refer to our algorithms, we used a naming convention that summarises details about the algorithm in its name. The details of this naming convention are reported in Appendix A.

An algorithm's dynamic collective behaviour is greatly influenced by the nature of the problem fitness landscape. To study the impact of the problem on the algorithm collective behaviour and performance, we carried out experiments on a number of 10-dimensional real-coded benchmark problems defined in [24]. Each algorithm (e.g., `Arth-0.4-0.025-2` or `BLX-0.25-0.05-3`, see Apendex A) has been tested in 100 independent runs with each problems. Each run lasted for 100,000 fitness function evaluations.

The crossover methods chosen for our tests are: *arithmetic crossover, blending crossover (BLX-α), and heuristic crossover* (their details and settings are explained in Appendix B). These crossovers were chosen as they handle the parental genetic material in very different manners: *deterministic*, *stochastic*, and *fitness-biased*, respectively. They exhibit different levels of exploration/exploitation because they make use of the information available to them in very different ways. Also, their behaviour changes significantly depending on their parameter settings.

More specifically, arithmetic crossover (Arth-λ) exhibits an exploration behaviour and increases the population diversity for $\lambda > 1$ or $\lambda < 0$, and otherwise reduces it [6]. BLX-α widens the distribution of genetic materials when $\alpha > (\sqrt{3} - 1)/2$, and otherwise reduces it [15]. Heuristic crossover, on the other hand, creates offspring close to the better parent, which makes this operator more exploitative (it does not only make use of the provided genetic material, but also uses the fitness values to bias the search). To obtain a full spectrum of behaviours, in our experiments, we used four control parameter settings for each of the three crossover methods. These made the bias of Arth-λ and BLX-α range from exploitative to explorative and the bias of heuristic crossover from more exploitative to less exploitative.

To examine the effect of mutation on the exploitation behaviour of algorithms, *non-uniform mutation* [14] has also been used with a range of mutation rates. Non-uniform mutation lowers the mutation step over a run to allow more exploration at the initial generations and fine tuning at the late stages of the search. This mutation is considered very appropriate for real-coded evolutionary algorithms [7]. More details are presented in Appendix B

Obviously, we cannot assess the effect of genetic operators on the exploitation behaviour of an algorithm based only on the way they combine/alter parental genetic material to produce new individuals. This is because the quality of the operators' output depends on the quality of the material provided in input, which is chosen from the population by the selection mechanism. Therefore, we used tournament selection with different tournament sizes, so as to vary the selection pressure in the algorithm. Larger tournaments induce a higher selection pressure and lead to a higher exploitation behaviour whereas smaller tournaments lead to a lower pressure and a higher exploration.

3.2 Algorithm Settings and Operators Effects

The collective dynamic behaviour of an algorithm is the result of applying the algorithm on a certain problem. Both the algorithm and the problem have an effect on the population dynamics. In this section, we apply our dependency measures on different algorithms operating on problems with different properties. Figure 2(A) compares the fitness dependency and genetic dependency of two sets of algorithms, `Heur-0.2-0.025-`x and `Heur-0.4-0.025-`x for $x = 2, \ldots, 8$, run on problem $F9$. The two sets of algorithms

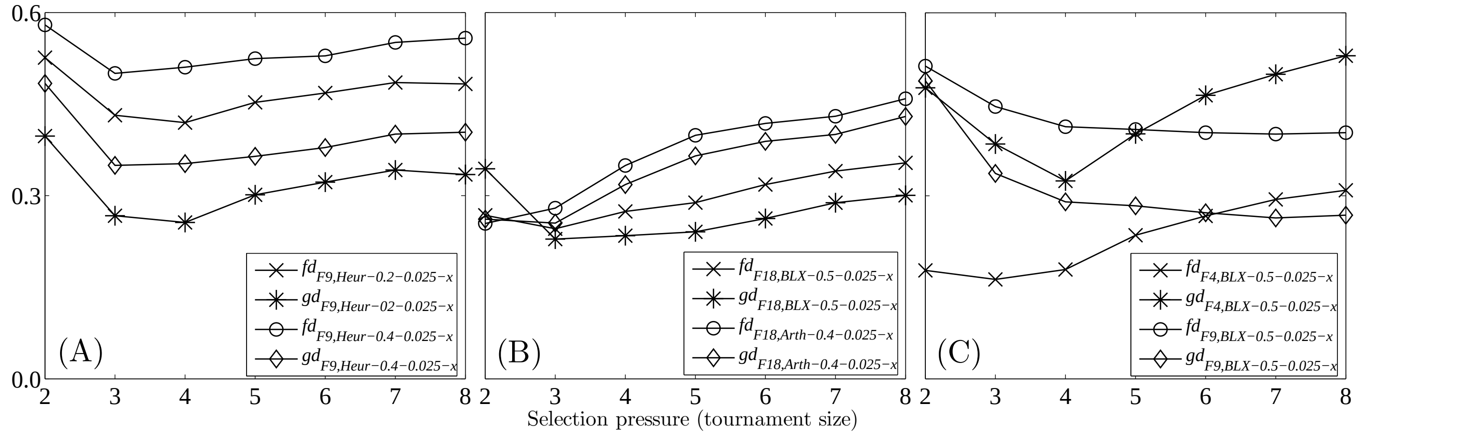

Figure 2. The effects of using different crossover parameter settings, using different crossover methods and applying an algorithm on different problems on the exploitation behaviour of an algorithm, for different values of the selection pressure (tournament size).

use the heuristic crossover method with different parameter settings (0.2 and 0.4) over differ selection pressures (tournament sizes). From the figure we see that Heur-0.2 algorithms exhibit lower fitness-dependencies and genetic-dependencies compared to Heur-0.4 algorithms. This is due to the fact that Heur-0.2 algorithms, especially with high selection pressure levels, tend to narrow the population distribution and loose the ability to exploit. On the contrary, Heur-0.4 algorithms move more slowly toward better solutions, maintaining a higher population diversity for relatively longer periods. This allows an algorithm to exploit good information for longer resulting in higher average fitness and genetic dependencies. We should also note that fitness dependency is greater than genetic dependency for both sets of algorithms. This is because there are more operators in these algorithms that are biased by fitness values (crossover in addition to selection).[3]

In Figure 2(B), we compare the collective dynamic behaviour of two algorithms with different crossover methods, BLX-0.5 and Arth-0.4, run on problem $F18$. The figure shows that using different crossover methods leads to different behaviours. The exploitation behaviour of Arth-0.4 algorithms is higher than that of BLX-0.5 algorithms. The stochastic nature of BLX crossover contributes to reduce the exploitation behaviour in comparison to arithmetic crossover which creates new solutions in a deterministic fashion.

To demonstrate the impact of the problem fitness landscape on the algorithm collective dynamic behaviour, we run the same set of algorithms on two different problems: $F4$, a noisy unimodal fitness function, and $F9$, a multimodal function with high number of local optima. We can see that although we have used the same set of algorithms, namely `BLX-0.5-0.025-`x, for $x = 2, \ldots, 8$, the collective behaviour is totally different. The fitness dependency of algorithms operating on $F4$ is always less than the genetic dependency. This is likely due to the unimodal nature of this fitness landscape. On the contrary, the genetic dependency is less than the fitness dependency in algorithms run on $F9$. We will explore more the effects of the fitness landscape on the collective behaviour of algorithms later on in the article.

[3] Of course, we have to keep in mind that the collective dynamic behaviour of these algorithms is also dependent on the problem they are operating on and different behaviours can result with different fitness landscapes (more about the effect of problem on algorithm's behaviour in this and next sections).

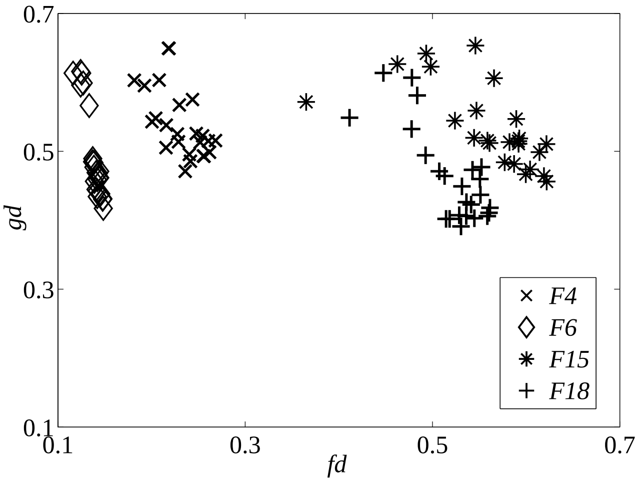

Figure 4. The distribution of algorithms in behavioural space for four different problems

Many researches have analysed the impact of mutation rate on algorithm performance and behaviour (e.g. [16, 28]). In this work, we exam the effect of changing mutation rate on the collective behaviour of algorithms. As we mentioned earlier, non-uniform mutation has been used with different mutation rates. Figure 3 depicts the change in the collective dynamic behaviour of algorithms in response to changing the mutation rate. We chose a set of 24 algorithms, `Arth-`$\{$`0.2,0.4`$\}$`-`μ`-`x, `BLX-`$\{$`0.25,0.5`$\}$`-`μ`-`x and `Heur-`$\{$`0.2,0.4`$\}$`-`μ`-`x, for $x = 2, \ldots, 5$, run on problem $F4$. We used four mutation rates, $\mu = 0.0, 0.025, 0.05, 0.1$. The figure presents the collective dynamic behaviour of an algorithm in terms of its fitness dependency and genetic dependency. Each algorithm is represented as a point in fitness- and genetic-dependencies space, which we will refer to as the *algorithm behavioural space*. From the figure we can see that changing mutation rate actually shifts the algorithm behaviour in the behavioural space. Note that how changing the mutation rate has led the algorithms to form a different niche in the behavioural space.

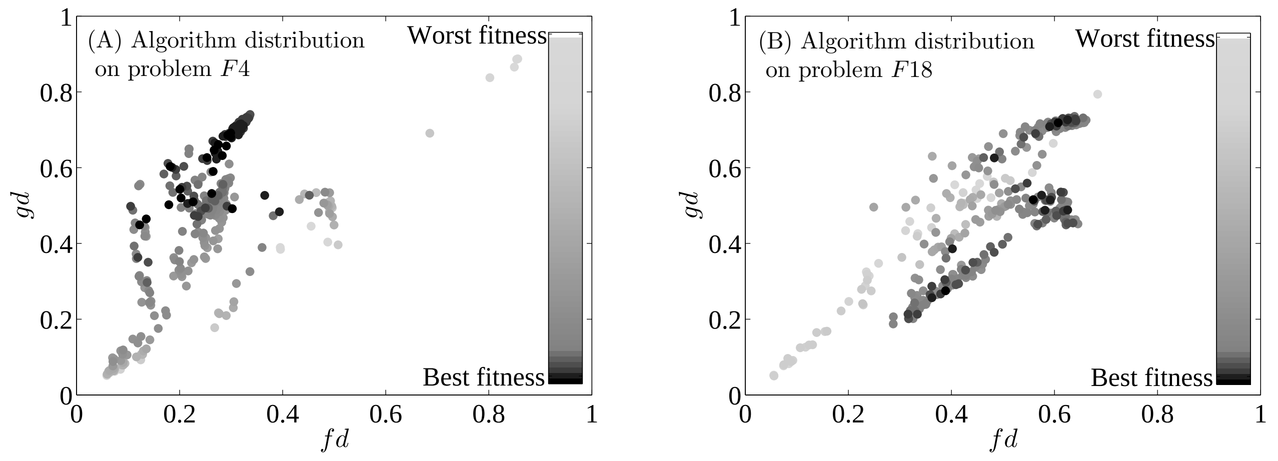

Figure 5. The distribution of algorithms and their best fitness values in the behavioural space of problem $F4$ and $F18$

3.3 Fitness Landscape Effect

As mentioned before, the nature of the fitness landscape has a great impact on how an algorithm conducts the search and distributes the population in the search space. Our measures of population collective dynamic behaviour, and in fact every performance measure, characterise the association between algorithm and problem. An algorithm exhibits different behaviours depending on the problem it operates on, which is consistent with the NFL theorem. To demonstrate this, we ran a set of 24 algorithms on four different problems. The algorithms are `Arth-`$\{$`0.2,0.4`$\}$`-0.05-`x, `BLX-`$\{$`0.25,0.5`$\}$`-0.05-`x and `Heur-`$\{$`0.2,0.4`$\}$`-0.05-`x, for $x = 2, \ldots, 5$. The four problems are $F4$, $F6$, $F15$ and $F18$. Figure 4 shows the distributions of the 24 algorithms with respect to the four problems. It is obvious from the figure that the same set of algorithms has different distribution in the behavioural space.

4 Prospects for Applications of the Proposed Model

In section 3.3, we showed that the collective dynamic behaviour of an algorithm changes according to the problem faced by the algorithm. This means that every time we use an algorithm, a, on a problem, f, the collective dynamic behaviour of a, represented in terms of our dependency measures as a point $(fd_{a,f}, gd_{a,f})$ in the behavioural space, will be different from the behaviour of another algorithm, $(fd_{a',f}, gd_{a',f})$, or the behaviour of the same algorithm applied to different problem, $(fd_{a,f'}, gd_{a,f'})$, unless that the two algorithms, a and a', are qualitatively similar or the two problems, f and f', have similar features, respectively.

We analysed the distribution of the collective dynamic behaviours of a large set of algorithms in the behavioural space and noticed that algorithms with good performance values (e.g., best fitness achieved) tend to form niches in the behavioural space. In addition, algorithm performance seems to be a continuous function of behaviour in such a space.

Figure 5 depicts the distribution of 336 algorithms on two different problems, $F4$ and $F18$. The tested algorithms are `Arth-`$\{$`0.2, 0.4, 1.2, 1.5`$\}$-μ-x, `BLX-`$\{$`0.0, 0.25, 0.5, 0.75`$\}$-μ-x and `Heur-`$\{$`0.1, 0.2, 0.3, 0.4`$\}$-μ-x, where $x = 2, \ldots, 8$ and $\mu = 0.0,\ 0.025,\ 0.05,\ 0.1$. We can see that certain areas in behavioural space have algorithms with better performance. The continuous change in algorithm performances in the behavioural space demonstrates that the relationship between the collective dynamic behaviour, $(fd_{a,f}, gd_{a,f})$, and algorithm's performance, $p_{a,f}$, can be modelled as a smooth function, $p_{a,f} = \mathrm{pbl}(fd_{a,f}, gd_{a,f})$, where we term the function pbl a *behavioural landscape* function of problem f.

The behavioural landscape of problems $F4$ and $F18$ can be modelled by using a regression method to define the relationship between algorithm's behaviour and performance. We analysed the data reported in Figure 5 using a local regression method (LOESS) [5]. The resulting behavioural landscapes are shown in Figure 6. Simple inspection of such landscapes reveals that they may help us to identify the desired collective behaviour to achieve good results on a problem.

For this reason, the proposed model has the potential to present a solution to the algorithm-selection [19] or parameter-tuning problems [2]. Both problems can be viewed as the problem of searching the behavioural landscape to find the right algorithm/parameter settings to produce the desired collective dynamic behaviour and consequently achieve the best results on the problem in hand. Obviously in any real-world application, the shape of the behavioural landscape is unknown and, so, an algorithm selection/parameter tuning mechanism would need to sample such a landscape and direct further the exploration toward areas of algorithms with high performance. This can be done by adjusting the parameters of an algorithm, or by selecting another algorithm, to produce a collective behaviour in a certain region in the behavioural landscape containing algorithms more likely to have better performance.

The behavioural landscape represents a *unified approach to characterise a problem* and to provide useful knowledge about it to guide algorithm design and implementation. It also gives us a different perspective on what makes a problem hard to solve by certain algorithms.

5 Conclusions and Future Work

In this paper, we have presented an empirical model to represent and analyse the collective dynamic behaviour of population-based algo-

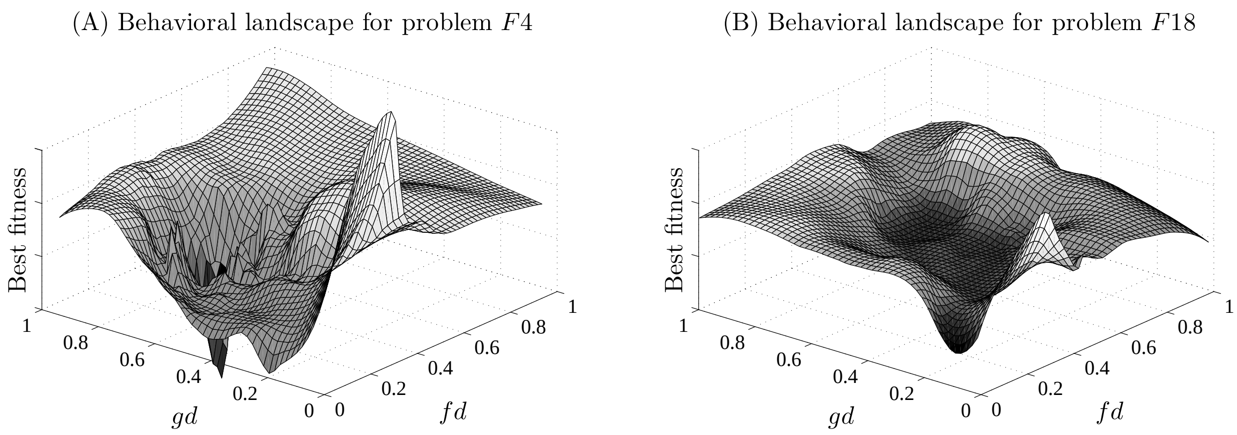

Figure 6. The behavioural landscapes of problems $F4$ and $F18$ show the association between algorithms behaviour and performance. Such landscapes may help us to identify the desired collective behaviour to achieve good results on a problem.

rithms. The model uses a coarse-grained representation of populations to identify areas of search activity and record information describing genotypic and phenotypic aspects of this activity. Our aim was to characterise the exploitation behaviour of algorithms by quantifying how much the search is guided by the information available in a population. In other words, we measured the extent by which a population makes use of (exploits) observed fitness values and genetic material to direct and intensify population activities.

The k-means clustering method was used to identify activity regions, while the dependency of activity on genetic material and fitness values was assessed by an entropy-based measure, namely the uncertainty coefficient.

In order to analyse the effects of different search operators and their parameter settings on the exploitation behaviour of EAs, we applied the proposed model to algorithms with three different crossover methods, different levels of selection pressure and different mutation rates. The chosen crossover methods combine individual solutions and utilise information presented to them in very different ways. Furthermore, each of them has a control parameter that can be used to tune the exploration/exploitation behaviour of the operator.

The mutation rate is considered a key control parameter of EAs. Our results confirmed this. Changing mutation rate can affect dramatically the collective behaviour of populations. We showed that the mutation rate can affect the bias of other operators and change the dependency on fitness and on genetic material.

The effect of using different problems on the collective dynamic behaviour has been analysed. A range of benchmark problems with a variety of properties have been used. We showed that for a problem, a behavioural landscape of the algorithm collective dynamic behaviours can be defined. We explained that this behavioural landscape can be utilised to implement algorithm-selection or parameter-tuning mechanisms. We also showed that this landscape can be used to characterise a problem and provide guidelines to practitioners to design and implement algorithms for a certain problem.

Although the proposed model has been used to analyse EAs, different kinds of population-based algorithms could be analysed, as the model is totally algorithm-independent and only observes population individuals as they are created. This gives the proposed model an advantage over many analysis tools that have been tailored to analyse or model a certain kind of algorithms or single operators. In addition, the model can be implemented using clustering techniques other than k-means. Possible candidate tools are self-organising maps [13] or principal component analysis (PCA) [11].

A key contribution of the proposed model is that it helps to identify what constitutes exploitation (or exploration) behaviour and to understand the effects that operators with different biases have on the dynamic behaviour of populations. The model also provides a practical tool to analyse a population's collective behaviour. This tool can be of use by EA practitioners for analysing the effects of operators and for tuning their control parameters. The model also allows the comparison of different algorithms at a higher level than just performance.

In future work, the notion of behavioural landscape that we have defined in this work could be utilised in implementing a performance prediction model. The correlation between algorithm performance and the fitness or genetic dependency can be analysed and possibly used to define a new kind of behavioural landscape.

ACKNOWLEDGEMENTS

This work has been supported by EPSRC via a Doctoral Training Account (DTA) research studentship.

REFERENCES

[1] A.D. Bethke, *Genetic algorithms as function optimizers*, Doctoral dissertation, Unversity of Michigan, 1981.
[2] Kenneth De Jong, 'Parameter setting in eas: a 30 year perspective', in *Parameter Setting in Evolutionary Algorithms*, eds., FernandoG. Lobo, CludioF. Lima, and Zbigniew Michalewicz, volume 54 of *Studies in Computational Intelligence*, 1–18, Springer Berlin Heidelberg, (2007).
[3] A.E. Eiben and CA Schippers, 'On evolutionary exploration and exploitation', *Fundamenta Informaticae*, **35**(1), 35–50, (1998).
[4] B.S. Everitt, S. Landau, M. Leese, and D. Stahl, *Cluster Analysis*, Wiley series in probability and statistics, Wiley, 2011.
[5] Wolfgang Hardle, *Applied nonparametric regression*, volume 5, Cambridge University Press, 1990.
[6] F. Herrera, M. Lozano, and AM Sánchez, 'A taxonomy for the crossover operator for real-coded genetic algorithms: An experimental study', *International Journal of Intelligent Systems*, **18**(3), 309–338, (2003).
[7] Francisco Herrera, Manuel Lozano, and Jose L. Verdegay, 'Tackling real-coded genetic algorithms: Operators and tools for behavioural analysis', *Artificial intelligence review*, **12**(4), 265–319, (1998).
[8] John Henry Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*, The MIT press, 1992.

[9] Thomas Jansen, Kenneth A. De Jong, and Ingo Wegener, 'On the choice of the offspring population size in evolutionary algorithms', *Evolutionary Computation*, **13**(4), 413–440, (2005).

[10] Thomas Jansen and Ingo Wegener, 'The analysis of evolutionary algorithms - a proof that crossover really can help', *Algorithmica*, **34**(1), 47–66, (2002).

[11] I. Jolliffe, *Principal component analysis*, Wiley Online Library, 2005.

[12] Terry Jones and Stephanie Forrest, 'Fitness distance correlation as a measure of problem difficulty for genetic algorithms', in *Proceedings of the 6th International Conference on Genetic Algorithms*, pp. 184–192, San Francisco, CA, USA, (1995). Morgan Kaufmann Publishers Inc.

[13] T. Kohonen, *Self-organizing maps*, Springer series in information sciences, Springer, 2001.

[14] Zbigniew Michalewicz, *Genetic algorithms+ data structures= evolution programs*, springer, 1996.

[15] Tatsuya Nomura and Katsunori Shimohara, 'An analysis of two-parent recombinations for real-valued chromosomes in an infinite population', *Evolutionary Computation*, **9**(3), 283–308, (2001).

[16] G. Ochoa, I. Harvey, and H. Buxton, 'Error thresholds and their relation to optimal mutation rates', in *Proceedings of the 5th European Conference on Advances in Artificial Life (ECAL-99)*, eds., Dario Floreano, Jean-Daniel Nicoud, and Francesco Mondada, volume 1674 of *LNAI*, pp. 54–63, Berlin, (September 13–17 1999). Springer.

[17] R. Poli and L. Vanneschi, 'Fitness-proportional negative slope coefficient as a hardness measure for genetic algorithms', in *Proceedings of the 9th annual conference on Genetic and evolutionary computation GECCO'07*, pp. 1335–1342. ACM, (2007).

[18] William H. Press, Saul A. Teukolsky, William T. Vetterling, Brian P. Flannery, and Brian P. Flannery, *Numerical Recipes: The Art of Scientific Computing*, Cambridge University Press, 2007.

[19] John R Rice, 'The algorithm selection problem', *Advances in Computers*, **15**, 65–118, (1976).

[20] Claude E. Shannon, 'A mathematical theory of communications', *The Bell Systems Technical Journal*, **27**, 379–423, 623–656, (1948).

[21] J. Shapiro, A. Prügel-Bennett, and M. Rattray, 'A statistical mechanical formulation of the dynamics of genetic algorithms', in *Selected Papers from AISB Workshop on Evolutionary Computing*, pp. 17–27. Springer-Verlag, (1994).

[22] C.R. Stephens and R. Poli, 'Coarse graining in an evolutionary algorithm with recombination, duplication and inversion', in *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 2, pp. 1683–1690. IEEE, (2005).

[23] C.R. Stephens and R. Poli, 'Coarse-grained dynamics for generalized recombination', *Evolutionary Computation, IEEE Transactions on*, **11**, 541–557, (2007).

[24] Ponnuthurai N Suganthan, Nikolaus Hansen, Jing J Liang, Kalyanmoy Deb, YP Chen, Anne Auger, and S Tiwari, 'Problem definitions and evaluation criteria for the cec 2005 special session on real-parameter optimization', Technical report, KanGAL Report, (2005).

[25] Mikdam Turkey and Riccardo Poli, 'An empirical tool for analysing the collective behaviour of population-based algorithms', in *Proceedings of the 2012 European conference on Applications of Evolutionary Computation*, EvoApplications'12, pp. 103–113. Springer-Verlag, (2012).

[26] Mikdam Turkey and Riccardo Poli, 'A model for analysing the collective dynamic behaviour and characterising the exploitation of population-based algorithms', *Evolutionary computation*, **22**(1), 159–188, (2014).

[27] M.D. Vose and G.E. Liepins, 'Punctuated equilibria in genetic search', *Complex Systems*, **5**, 31–44, (1991).

[28] Michael D Vose, 'A closer look at mutation in genetic algorithms', *Annals of Mathematics and Artificial Intelligence*, **10**(4), 423–434, (1994).

[29] Ingo Wegener, 'On the expected runtime and the success probability of evolutionary algorithms', in *Proceedings of the 26th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2000)*, eds., Ulrik Brandes and Dorothea Wagner, volume 1928 of *Lecture Notes in Computer Science*, pp. 1–10, Konstanz, Germany, (June 15-17, 2000). Springer.

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

A Algorithms Naming Convention

We used a naming convention for EAs to capture four facts about an algorithm: crossover method, crossover control parameter, mutation rate and tournament size (selection pressure). The algorithm name has the format `CCCC-c-m-t`. Here `CCCC` refers to the crossover method and it can be `Arth` for arithmetic crossover, `BLX` for blending crossover and `Heur` for heuristic crossover (see Appendix B); **c** is a real number representing the control parameter of the crossover method used; `m` is the per-locus mutation rate and `t` is an integer representing the tournament size. For example, `Arth-0.4-0.10-4` is an EA using arithmetic crossover with control parameter 0.4, mutation rate 0.10 and tournament size 4.

B Crossover and Mutation Operators

Three crossover methods have been used in this paper. They all return two offspring and all have a control parameter that can be used to tune their exploration/exploitation behaviour. These methods are:

- **Arithmetic crossover** (Arth-λ): Two offspring individuals, C_1 and C_2, are produced from two parents, P_1 and P_2, as follows: $C_1 = P_1 \times \lambda + P_2 \times (1 - \lambda)$ and $C_2 = P_1 \times (1 - \lambda) + P_2 \times \lambda$.
- **Blending crossover** (BLX-α): Two offspring individuals are produced by randomly (uniformly) generating values for their genes within an interval that depends on the corresponding parental genes. Suppose that p_1^i and p_2^i are the i^{th} parameter of parents P_1 and P_2, respectively, the corresponding parameter, c_k^i, of offspring C_k is randomly chosen from the interval $[p_{min} - I \times \alpha, p_{max} + I \times \alpha]$, where $p_{min} = \min\{p_1^i, p_2^i\}$, $p_{max} = \max\{p_1^i, p_2^i\}$ and $I = p_{max} - p_{min}$.
- **Heuristic crossover** (Heur-λ): This method creates one offspring individual around the parent with the highest fitness. Here we modified it slightly so as to produce two offspring. Suppose that we have two parent individuals, P_1 and P_2, and that P_1 is the one with higher fitness, then the two offspring individuals, C_1 and C_2, are created as: $C_1 = P_1 - \lambda \times (P_2 - P_1)$ and $C_2 = P_1 + \lambda \times (P_2 - P_1)$.

Non-uniform mutation has been used in this work. This mutation has the ability to adapt over run to allow different degrees of exploration. It is defined as following,

Let $C = (c_1, \ldots, c_D)$ represents an individual undergoing mutation, where D represents the problem dimension, and it is set to 10, and $c_i \in [a_i, b_i]$ is a gene to be mutated and a_i and b_i are the lower and upper limits of possible values range, then the mutated gene c_i' is calculated as follows

$$c_i' = \begin{cases} c_i + \Delta(t, b_i - c_i) & \text{if } q = 1 \\ c_i - \Delta(t, c_i - a_i) & \text{if } q = 0 \end{cases}$$

with q is a random number which may have the value of either one or zero, and $\Delta(t, y) = y(1 - r^{(1-\frac{t}{g_{max}})^b})$, where t is the number of current generation, g_{max} is the maximum number of generations, r is a random number from the interval $[0, 1]$ and b is a parameter determines the degree of dependency on the number of generation (t). We chose $b = 5$.