A Model for Analysing the Collective Dynamic Behaviour and Characterising the Exploitation of Population-based Algorithms

Mikdam Turkey mturkey@essex.ac.uk

Riccardo Poli rpoli@essex.ac.uk

School of Computer Science, Electronic Engineering, University of Essex, Wivenhoe Park, Colchester, Essex, CO4 3SQ, United Kingdom

Abstract

Several previous studies have focused on modelling and analysing the collective dynamic behaviour of population-based algorithms. However, an empirical approach for identifying and characterising such a behaviour is surprisingly lacking. In this paper, we present a new model to capture this collective behaviour, and to extract and quantify features associated with it.

The proposed model studies the topological distribution of an algorithm's activity from both a genotypic and a phenotypic perspective, and represents population dynamics using multiple levels of abstraction. The model can have different instantiations. Here it has been implemented using a modified version of self-organising maps. These are used to represent and track the population motion in the fitness landscape as the algorithm operates on solving a problem.

Based on this model, we developed a set of features that characterise the population's collective dynamic behaviour. By analysing them and revealing their dependency on fitness distributions, we were then able to define an indicator of the exploitation behaviour of an algorithm. This is an entropy-based measure that assesses the dependency on fitness distributions of different features of population dynamics.

To test the proposed measures, evolutionary algorithms with different crossover operators, selection pressure levels and population handling techniques have been examined, which lead populations to exhibit a wide range of exploitation-exploration behaviours.

Keywords

Collective behaviour analysis, exploitation, population dynamics, population-based algorithms, evolutionary algorithms, emergent features, self-organising maps.

1 Introduction

Population-based algorithms explore a search space by sampling it to gather information and then redistributing the population within it based on such information. This biases the movement of the population in a way that is hoped to guide the algorithm to examine known good areas of the search space with more intensity and/or to try to locate new promising areas. Naturally, it is well known that whether an algorithm's bias is beneficial or not depends on the match between that bias and the fitness measure used to inform the search (Wolpert and Macready, 1997). Nonetheless, as the search mechanisms move, create and/or eliminate individuals, the population shows a collective behaviour which often changes dynamically over time.

Understanding and characterising the collective dynamic behaviour of population-based algorithms is thus very important. For example, identifying emergent collective properties related to an algorithm's dynamics allows the comparison of algorithms in terms of behaviours, not just performance (e.g., the number of fitness evaluations to reach an optimum). Therefore, this is not only a prerequisite for designing new population-based algorithms for specific applications but it may also enable practitioners to make technically sound choices of algorithms, operators, parameters, etc.

1.1 Previous Empirical and Theoretical Studies

Various previous approaches have tried to develop a better understanding of the collective behaviour of population-based algorithms. An approach is to practically study the dynamic behaviour of an algorithm with respect to a specific problem by defining and analysing suitable empirical measures (Bethke, 1981; Goldberg, 1989; Mitchell et al., 1992; Jones, 1995; Tomassini et al., 2005; Vanneschi, 2007; Graff and Poli, 2008; Poli and Vanneschi, 2007; Vanneschi et al., 2004). Typically, efforts have concentrated on trying to characterise landscape features and landscape/search-algorithm interactions with the objective of understanding what makes a problem easy or hard for evolutionary algorithms rather than on the characterisation of the behaviour of populations.

An alternative approach is to theoretically model the dynamics of search algorithms. Within evolutionary computation theory, approaches have included: Markov chain formulations (Vose and Liepins, 1991; Davis and Principe, 1993; Poli et al., 2004; Mitavskiy and Rowe, 2006), the schema theory (Holland, 1992; Stephens and Waelbroeck, 1999; Poli et al., 2004; Poli and McPhee, 2003a,b; Stephens and Poli, 2007), computational complexity techniques (Droste et al., 2002; Jansen et al., 2005; Witt, 2006; Jansen and Wegener, 2002; Storch and Wegener, 2004; Neumann and Wegener, 2007), the statistical-mechanics formulation (Prügel-Bennett and Shapiro, 1994; Shapiro et al., 1994), and the characterisation of the fundamental limitations of search (Wolpert and Macready, 1997; Whitley and Watson, 2005; Schumacher et al., 2001; Igel and Toussaint, 2003; Poli et al., 2009). All of these have seen some successes at mathematically modelling evolutionary algorithms.

The main questions that the aforementioned theoretical studies have tried to address are whether an algorithm is able to reach an optimal solution (i.e., convergence), how fast solutions this can be reached (i.e., convergence speed) and/or modelling the changes of a population's state over time (i.e., population dynamics). Here we will focus on areas of theory that investigate population dynamics, namely *schema theory*, *Markov chain model*, and *statistical mechanical model*, to position our work in relation to them.

Schema theory captures the dynamics of a population at a somewhat coarse-grained level by describing how the population redistributes itself among a set of schemata, where the schema is a fixed subset of the search space. The model describes how the number of individuals in a schema changes over time depending on the number of individuals in a related set of schemata and their fitness. Markov chain models of EAs use a different, more microscopic representation for the state of the population and describe how the probability distribution over population states changes over time. This model can make precise predictions about population dynamics, but these require considerable mathematical and computational resources. An alternative model, statistical mechanics, has been proposed to represent the dynamics of large complex systems in terms of their statistical properties. Statistical mechanics models population dynamics by predicting the changes of the population's fitness distribution over time.

Note that the first two models focus on a *genotypic representation* of the population while the third model uses *phenotypes* (fitness) to represent population dynamics, but no model uses both. Finally, we should note that in all cases *representation schemes are static*, in the sense that once one has decided what form of schema, state space representation or fitness-distribution representation to adopt, this remains constant throughout a run. As we will see later, our work departs from these traditions and proposes an dynamic genotype-phenotype representation of population dynamics.

1.2 Exploration and Exploitation

While the models presented above are of significant theoretical importance, they have little practical use in characterising realistic algorithms running on realistic problems and extracting features of the collective dynamic behaviour of populations.

Looking at what has been done in this area, we find that researchers often talk about two emergent behaviours of search algorithms: *exploration* and *exploitation*. Intuitively, exploration refers to a behaviour associated with the discovery of new good regions in the search space, while exploitation refers to the exploration previously discovered good regions (Holland, 1992; Eiben and Schippers, 1998; Macready and Wolpert, 1998). However, we should note that there is no precise definition in the literature of the notions of exploration and exploitation and no precise characterisation of the distinction between them.

Clearly, whatever definition one embraces, when discussing exploration and exploitation one often talks about both the search space and the fitness values. After all the genetic operators create new solutions by combining/altering existing (useful) information contained in the *genotypic representation* of individuals, while selection favours fitter solutions thereby using the *phenotypic value* (fitness) of individuals. Possibly because of this and of the lack of a more precise definition, often people simply attribute the exploration features of an evolutionary algorithm entirely to the properties of the genetic operators used by the algorithm (e.g., crossover or mutation), while it is assumed that exploitation is achieved by selection.

This is an unsatisfactory interpretation as the reality appears to be much more complex: each search operator, including selection, has a certain bias in guiding the search, and the aggregation of these biases results in an emergent collective behaviour that exhibits a certain degree of exploration/exploitation at different times of the search.

1.3 Contributions of this Paper

In this paper, we develop an empirical coarse-grained model that captures and tracks a population's dynamics represented as topological distribution of various features describing populations from both a genotypic and a phenotypic perspective.[1] The model uses a *variable representation* that represents the population's dynamics at a relatively high level of abstraction which makes it easier to talk about collective behaviours of populations.

Our model represents the population using a *grid of nodes*, where each node points to a region of population activity and records information about genotypic density, fitness values, and distances for the individuals in such a region. During a run of an algorithm, as the regions of population activity change over time, the nodes in the model adjust to track these regions and continue gathering information about them. Thus, over a run, the model produces *a sequence of grids that capture the population dynamics*

[1]An initial formulation and some results of our model were presented in (Turkey and Poli, 2012).

represented as a topological distribution of genotypic density, fitness values and distances. This model share some features with previous theoretical models of EAs: it is similar to a schema-based model in that it uses sets of genotypes as elements of the representation, it is similar to Markov chain models in that it tracks the changes of the population, and it is similar to statistical mechanics models as it keeps track of the fitness distribution (but it does so at a finer level of granularity, i.e., at the level of activity nodes).

Higher levels of abstraction in the description of the population's collective dynamic behaviour can be obtained by grouping the nodes in our model into activity regions and by categorising these regions according to the type of activity carried out within them. As we will see later, this allows us to define a variety of measures that can be used to characterise different aspects of the collective behaviour of the population and to analyse its coarse-grained dynamics.

In addition, by exploiting our coarse-grained representation of population dynamics, we contribute to clarifying the notions of exploration and exploitation. In particular, we do so by quantifying by how much the fitness distribution guides the search activities of an algorithm. Although other proposed measures can be qualitatively used to characterise the exploitation/exploration behaviour, we suggest a specific new *entropy-based exploration indicator* of the dynamic behaviour of population. This assesses the fitness-dependency of the topological distributions of different features of the collective dynamic behaviour of an algorithm. This is particularly easy to do with our model, as it is designed to track the topological distribution of fitness values and other genotypic features of populations.

Many implementations of the proposed coarse-grained representation of populations are possible. In this article, a modified version of self-organising maps (SOMs) (Kohonen, 2001) are used to track population movement in the search space and to mine information about the emergent collective behaviour of an algorithm as it operates on solving a problem.[2] The model has been applied on different EAs that use different operators and techniques. The reported results show that the model was able to characterise their collective behaviours and reveal the effect of different biases the algorithms used.

The rest of this paper is organised as follows. Section 2 will present the proposed model in detail and will develop measures of the collective dynamic behaviour of populations. It will also aim at assessing the fitness-dependency of different aspects of population activities. In Section 3, we present the details of our implementation of the proposed model using SOMs. Section 4 reports and discusses the results of our experiments. Finally, we end the paper with some conclusions in Section 5.

2 Proposed Model of Collective Dynamic Behaviour

In this section, the formulation of the proposed model is presented. Then, we define features of the collective behaviour of an algorithm based on the model. Finally, the proposed exploitation indicator is defined.

2.1 Principles of the Proposed Model

The proposed model uses a grid of nodes to capture the topological distribution of individuals in a population-based algorithm at certain points in time.[3] Each node is a

[2]SOMs have previously been used to improve the performance of evolutionary algorithms by enhancing the search strategy and avoiding genetic drift (Amor and Rettinger, 2005).

[3]In this work, we assume that the algorithm is operating on real-coded problems. However, a grid of nodes for different types of representations can be defined.

vector representing the centre of a group of individuals in a localised area in the search space. In addition, each node also stores information describing the activities of the group in that area. To track the dynamic behaviour of a population over a run, the grid is updated to represent motion of the population and record information about its activities as the algorithm works on solving a problem. Emergent features that characterise collective dynamic behaviour are extracted by analysing the changes in node positions and other information stored in the nodes.

Formally, the grid, C^t, represents a snapshot of the population distribution at time t. Let C^0 be a grid of nodes that represent the distribution of the initial 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 the search space dimension, and $y_i \in \mathbb{R}$ is the associated fitness value. Furthermore, let the time-ordered set $P = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ represents a sequence of all the individuals created by an algorithm over a period of a run from the initial population, where N is the total number of calls to the fitness function. The grid is updated every τ individuals produced by the algorithm (i.e., τ represents the sampling period). For example, C^1 is computed starting from C^0 and using the sequence of individuals $P_1 = \{(x_1, y_1), \ldots, (x_\tau, y_\tau)\}$. That is, the grid's node vectors are adjusted to track individuals of the sequence P_1 and record information about them. Generally speaking, the grid C^t is created by tracking and recording information about the set of individuals $P_t = \{(x_{k+1}, y_{k+1}), \ldots, (x_{k+\tau}, y_{k+\tau})\}$, where $k = \tau \times (t-1)$. More specifically, if $P^{C^t_r} \subseteq P_t$ is the subset of individuals associated with a node, C^t_r, based on their distance from the node vector (as explained in Section 3), we have that the sets $x^{C^t_r}$ and $y^{C^t_r}$ representing the positions and fitness values of the individuals associated with C^t_r provide information about the population activity happening in the region represented by that node.

More formally, the grid C^t consists of a matrix of $n \times n$ nodes, where each node, C^t_r, is represented by the following tuple

$$C^t_r = \langle m^{C^t_r}, d^{C^t_r}, f^{C^t_r}, f^{C^t_r}_{\textit{best-so-far}}, h^{C^t_r} \rangle \tag{1}$$

where t is time, $r \in \{1 \ldots n\} \times \{1 \ldots n\}$ is the position in the grid and the elements of the tuple are as follows:

$m^{C^t_r} \in \mathbb{R}^D$ is a vector representing the centre of a region of activity of the population. As the algorithm works on redistributing the population around activity regions, the node vector tracks the newly created individuals in its vicinity (the mechanism of updating vector positions is explained in Section 3);

$d^{C^t_r} = \sum_{a \in x^{C^t_r}} |a - m^{C^t_r}|$ (*hit distance*) is the sum of the distances between $m^{C^t_r}$ and the individuals, $x^{C^t_r}$, associated with the node. This can be used as an indicator of how far away from the centre of activity an algorithm searches for new solutions;

$f^{C^t_r} = \frac{1}{|y^{C^t_r}|} \sum_{a \in y^{C^t_r}} a$ is the mean fitness of the individuals associated with C^t_r. This element gives an insight about the quality of the region of the search space represented by the node;

$f^{C^t_r}_{\textit{best-so-far}}$ is the best fitness value of an individual associated with C^t_r since time $t = 0$. Formally, if $f^{C^0_r}_{\textit{best-so-far}} = \max y^{C^0_r}$, then the best fitness for time $t > 0$ is defined as

$f_{\textit{best-so-far}}^{C_r^t} = \max\{f_{\textit{best-so-far}}^{C_r^{t-1}}, \max y^{C_r^t}\}$. This quantity tells us how well the population is searching this area of the search space, and whether this activity has led to achieve any growth in fitness;

$h^{C_r^t} = |P^{C_r^t}|$ (*hits counter*) is the number of individuals associated with the node. This number represents the volume of activity an algorithm allocates for a particular area of the search space.

As the algorithm starts exploring the fitness landscape and redistributing the population, some nodes (that represent identified areas of activity at a certain point in time) may be abandoned as the population moves towards different areas. At any point in time, we are only interested in the nodes that have observed some population activity. These nodes are called *active nodes* and they are defined as $AN^t = \{C_r^t | h^{C_r^t} > 0\}$. Active nodes in this model represent spots in the search space where population activities have taken place. Nodes are, in a way, similar to schemata, in that both represent subsets of the search space. However, differently from schemata, our nodes have the ability to track the population dynamics over time, and they store information about these dynamic activities. This feature enables us to capture the distributions of activity density, observed fitness values, and other features of the topological distribution of a population.

A higher level of abstraction in describing population activity is obtained by aggregating active nodes into *activity regions*. An activity region is a set of adjacent active nodes the vectors of which point towards similar directions. Formalising this definition requires introducing some further notions.

Let us define a function, $Nr(C_r^t)$, that returns a set of all immediate neighbours of a node that is a member of the set of active nodes AN^t:

$$Nr(C_r^t) = \left\{ C_{r'}^t \in AN^t \mid 0 < \|r - r'\| < \sqrt{2} \text{ and } \|m^{C_r^t} - m^{C_{r'}^t}\| < \omega \right\}$$

where $\omega = 0.05 \times \textit{Dia}$ is a distance threshold, *Dia* being the largest distance between any two points in the search space. The function above returns the empty set $\emptyset$ in case C_r^t has no neighbours. With this in hand, we can define a function $Nc(C_r^t, C_{r'}^t)$, that returns a set of consecutive neighbours between C_r^t and $C_{r'}^t$:

$$\begin{aligned} Nc(C_r^t, C_{r'}^t) = \big\{ X \in AN^t \,| X \in Nr(C_r^t) \text{ and } X \in Nr(C_{r'}^t), \text{ or} \\ Nc(C_r^t, X) \neq \emptyset \text{ and } Nc(X, C_{r'}^t) \neq \emptyset \big\} \end{aligned}$$

An activity region, $\textit{Region}(C_r^t)$ for $C_r^t \in AN^t$, is the set of all the nodes that are either direct or indirect neighbours of node C_r^t, that is:

$$\textit{Region}(C_r^t) = \big\{ X \mid Nc(C_r^t, X) \neq \emptyset \big\}$$

Figure 1 illustrates the proposed model and how it relates to the individuals in the population. The figure also shows how regions of active exploration in the search space are represented by activity regions within the model.

An even higher level of abstraction can be obtained considering *sets* of activity regions. For example, the set of all activity regions (set of sets) is defined as follows:

$$A^t = \bigcup_{X \in AN^t} \{\textit{Region}(X)\}$$

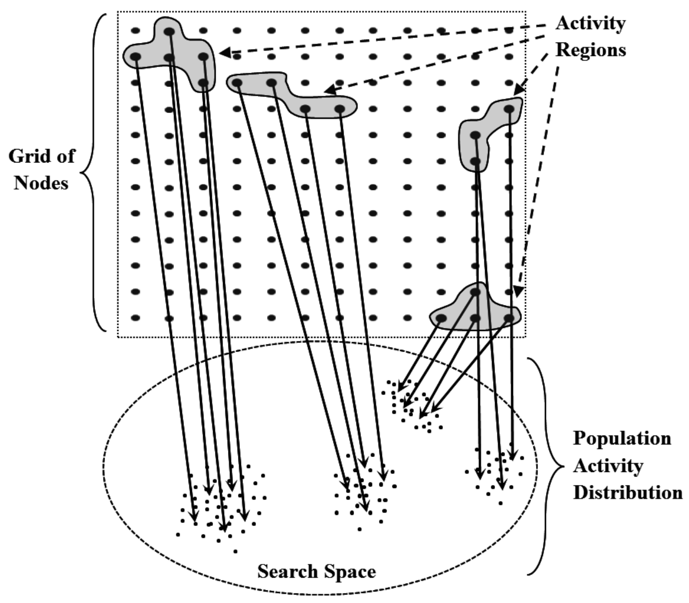

Figure 1: The proposed model and its relationship with the population. The model (at the top of the figure) is made up of nodes organised in a logical grid. Each node represents one or a small group of individuals out of the individuals generated by a population-based algorithm within a certain time period (typically they are the current population, but they can also represent the most recently produced individuals within a population). Nodes store multiple pieces of information about the individuals they represent, including a vector (the downward pointing arrows in the figure) that is the centroid of the individuals represented by a node. Not every node has an arrow, because nodes may have no individual associated with them. Node that do are said to be active nodes. Because self-organising maps are used to instantiate the model, neighbouring active nodes tend to represent neighbouring areas of the search space. Therefore, if we group active nodes into activity regions (the shaded regions in the model), collectively we find that each region represents a corresponding region of active search within the search space (represented by the clusters of individuals at the bottom of the figures).

This can be divided into the set of *growth regions* g^t and the set of *non-growth regions* ng^t. The set of growth regions contains all the regions where the population has managed to discover a better fitness value in the last τ fitness evaluations, while the non-growth region set contains the regions of the population where no fitness improvement has been achieved in the same period. Formally, these are defined as

$$g^t = \left\{ R^t \in A^t \mid \max_{X_1 \in R^t} f^{X_1}_{\textit{best-so-far}} > \max_{X_2 \in R^{t-1}} f^{X_2}_{\textit{best-so-far}} \right\} \qquad \text{and} \qquad ng^t = A^t \setminus g^t \, .$$

The sets of all the nodes belonging to growth regions and non-growth regions are defined as $G^t = \{X | X \in R \text{ and } R \in g^t\}$ and $NG^t = \{X | X \in R \text{ and } R \in ng^t\}$, respectively. Later we will also use the notion of best region, $\textit{Best}^t = \arg\max_{x \in A^t} f^x_{\textit{best-so-far}}$.

Note that the different levels of coarse-grained representation proposed above preserve the topological distribution of activity density, fitness values, and other features. In the next two subsections, we will define a set of features to represent the dynamic collective and emergent behaviour of a population based on these notions.

2.2 Extracting Features of Collective Dynamic Behaviour

Many features that characterise aspects of the collective dynamic behaviour of an algorithm can be extracted by monitoring changes in the activity regions of the model over time and analysing the corresponding recorded information. These features can be classified into three types.

Activity-related features describe the way an algorithm allocates its activities to the regions of population. The volume of activity can simply be measured using the number of individuals (hits counter $h^{C_r^t}$) that have been associated with nodes of a certain region. Region size (number of nodes in a region) is also used to describe aspects of population activity. These features characterise the population collective behaviour from a higher level (phenotype level). In evolutionary algorithms, selection methods and population handling techniques (e.g., niching techniques) have the significant influence on these emergent features and also affect other search operators.

Distance-related features describe two aspects of the population collective behaviour from the genotype point of view: how widely the population distributes its activities away from the centre (node vector), and the bias in population movement. Two distance measures can be calculated: the sum of the distances between the node's vector and input individuals ($d^{C_r^t}$) and the displacement which signifies the amount of shift in population centres of activity and it is calculated as the change of vector positions (the distance between $m^{C_r^t}$ and $m^{C_r^{t-1}}$).

Correlated features analyse the association between two different emergent features. Revealing and quantifying dependencies between emergent features could help characterising what directs the search and to what extent. In this work, we analyse the dependency of the two types of features mentioned above on fitness values. The resultant measures are used as indicators to the exploitation behaviour of the algorithm.

The first two types of features are introduced in the next subsection, while the third type is presented in Section 2.3.

2.2.1 Activity-related and Distance-related Features

Below we will define four features that characterise the dynamic collective behaviour of the population over time as an algorithm operates on the problem. For each feature we also define a related coarse-grained quantity, the average of that feature over the course of a run, to represent the overall behaviour of an algorithm across time.

The *activity rate* in a set of nodes S^t is the number of hits these nodes receive collectively at time t. It measures the intensity with which an algorithm explores these nodes and the regions they represent. It is defined as:

$$\mathit{ActivityRate}(S^t) = \frac{1}{\tau}\Big(\sum_{X \in S^t} h^X\Big) \tag{2}$$

where S^t is the set of all nodes in a set of regions and τ is the sampling period. This function can be applied to any set of nodes, but it makes most sense when applied to sets of nodes representing activity regions, particularly G^t, NG^t and $Best^t$. In these cases this measure tells us the way an algorithm tends to allocate its activities. For example, allocating more activities to growth regions implies that the algorithm exhibits exploitation behaviour as it tends to invest more individuals in regions where good fitness values have been discovered. The average of activity rate over a run of an algorithm is defined as $\mathit{ActivityRate}_{avg}(S) = \frac{1}{M}\sum_{t=1}^{M} \mathit{ActivityRate}(S^t)$, where $S = \{S^1 \dots S^M\}$ is a time-ordered sequence of sets of nodes representing a certain type of regions over the period of a run, $M = N/\tau$ represents the total number of sampling points and N is the total number of fitness function evaluations in a run.

Another useful features is the *size ratio of a region*, i.e., the number of active nodes of a certain set of regions relative to the total number of active nodes:

$$\mathit{SizeRatio}(s^t) = \Big(\sum_{R^t \in s^t} |R^t|\Big)/|AN^t| \tag{3}$$

where s^t is a set of regions (i.e., g^t, ng^t or $\{Best^t\}$).[4]

While $\mathit{ActivityRate}(S^t)$ measures the portion of search activity in a certain type of regions, $\mathit{SizeRatio}(s^t)$ tells us how concentrated these activities were. When $\mathit{ActivityRate}(S^t) = \mathit{SizeRatio}(s^t)$, that implies that the activities were evenly distributed among the nodes of s^t and each node has been hit by only one individual. On the other hand, if $\mathit{ActivityRate}(S^t) > \mathit{SizeRatio}(s^t)$, the activities were more concentrated and at least one node in S^t has been hit by more than one individual. The smaller $\mathit{SizeRatio}(s^t)$, the more concentrated the activities. The average size ratio over a run is defined as $\mathit{SizeRatio}_{avg}(s) = \frac{1}{M}\sum_{t=1}^{M} \mathit{SizeRatio}(s^t)$, where $s = \{s^1 \dots s^M\}$ is a time-ordered sequence of sets of regions of a certain type.

The proposed model updates its node vectors to track the population as it moves around the search space by monitoring every new individual created by the algorithm. These changes in vector positions reveal the speed of movement and the algorithm's bias. The average change in node vector positions (*displacement*) is measured as follows:

$$\mathit{Displacement}(S^t) = \frac{1}{|S^t|}\Big(\sum_{X^t \in S^t} \|m^{X^t} - m^{X^{t-1}}\|\Big) \tag{4}$$

This feature represents the extent to which the population distribution shifted, after the algorithm has created τ individuals. The total displacement achieved by an algorithm over the period of a run is measured as $\mathit{Disp}_{tot} = \sum_{t=1}^{M} \mathit{Displacement}(AN^t)$, where M is defined above and AN^t is the set of all active nodes at time t. High displacement values imply that the algorithm tends to move its areas of activity around the search space, while smaller values imply that the algorithm is searching in the same areas.

Another useful features is the *hit distance* — the distance between a newly created individual and the nearest node vector:

$$\mathit{HitDistance}(S^t) = \Big(\sum_{X \in S^t} d^X\Big)/\Big(\sum_{X \in S^t} h^X\Big) \tag{5}$$

The *hit distance* reveals the collective behaviour of the population in localised areas of the search space and characterises the effect of genetic operators. The average hit distance over a run is defined as $\mathit{HitDis}_{avg} = \frac{1}{M}\sum_{t=1}^{M} \mathit{HitDistance}(AN^t)$.

2.3 Using Fitness-Dependency as Indicator to an Exploitation Behaviour

Population activities can be guided (or biased) by various factors that an algorithm uses to make decisions. In addition to the observed fitness values, for example, distance or

[4]Please note the difference between s^t, which is a set of regions (set of sets), and S^t, which is the set of all nodes in these regions. Formally, $S^t = \{X | X \in R \text{ and } R \in s^t\}$.

similarity between individuals can be used (as in fitness-sharing techniques). In this section, we develop a measure to quantify the level of fitness-dependency in directing the search, and use it as an indicator to population exploitation behaviour.

In order to characterise fitness-dependency of population behaviour, we have to *measure the amount of certainty the presence of information (fitness values) brings to the activities of a population*. That is to say, we want to assess how much the features of the collective behaviour of populations — such as the activity rate, the displacement distance, or the hit distance — are dependent on the fitness distribution. To quantify this association, we used *an entropy-based measure*, which is a normalised variant of mutual information called the *uncertainty coefficient* (Press et al., 2007). This is defined as follows:

$$U(Y|X) = \frac{I(X,Y)}{H(Y)} \tag{6}$$

where

$$I(X,Y) = \sum_{x\in X}\sum_{y\in Y} p(x,y)\log\Big(\frac{p(x,y)}{p(x)p(y)}\Big) \tag{7}$$

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

$$H(X) = -\sum_{x\in X} p(x)\log(x) \tag{8}$$

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 (Shannon, 1948), while mutual information is a quantity that measures the mutual dependence of the two random variables. In Equation 6, the 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 the uncertainty about Y left after knowing X. So, in effect *the dependency between X and Y represents 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 knowing X can totally predict Y.

To use Equation (6) to assess the fitness-dependency of the behaviour of an algorithm, first, we need to compute the probability distribution of each feature of the collective behaviour over all activity regions. The probability distribution of a feature is estimated by binning active nodes according to the values of that feature and then computing a probability histogram. More specifically, we find the maximum and minimum values of each feature over all active nodes, we evenly partition the interval between them into m sub-intervals ($m = 10$ in this work), and we then divide the active nodes into classes according to of their feature values. If $L_i^{\phi_1}$ for $i = 1 \ldots m$, are the classes of the active nodes obtained by binning them based on feature ϕ_1, then the probability of each class is computed as

$$p(L_i^{\phi_1}) = \big(\sum_{X\in L_i^{\phi_1}} h^X\big)/\tau, \quad \text{for } \ i = 1\ldots m\,. \tag{9}$$

We also need to compute the joint probability of two classes of features. Given that the proposed model provides us with topological distributions of different features of population dynamics, finding the association and computing the joint probability for two features is easy. If $L_i^{\phi_2}$ for $i = 1 \dots m$, is another sequence of classes obtained by binning nodes using feature ϕ_2, then the joint probability of $L_i^{\phi_1}$ and $L_j^{\phi_2}$ is

$$p(L_i^{\phi_1}, L_j^{\phi_2}) = \Big(\sum_{X \in L_i^{\phi_1} \cap L_j^{\phi_2}} h^X \Big) / \tau, \quad \text{for } i, j = 1 \dots m \tag{10}$$

With these definitions of in hand, we can use the uncertainty coefficient (Equation 6) to analyse the dependency of hit distance, displacement, and activity (hit count) on fitness. For simplicity, we will denote the corresponding values of the uncertainty coefficient as $U_t(hd|fit), U_t(dis|fit)$ and $U_t(act|fit)$, respectively. These three measures, $U_t(act|fit)$ in particular, *are used as an exploitation behaviour indicator* at a certain point in time. The bigger the value they have the more dependent on fitness the algorithm behaviour is at that time.

The average of fitness-dependency of activity over one algorithm run is defined as $U_{avg}(act|fit) = \frac{1}{M} \sum_{t=1}^{M} U_t(act|fit)$. Similarly, we define $U_{avg}(hd|fit)$ and $U_{avg}(dis|fit)$. *The average of fitness-dependency rate is the quantity we will use as an exploitation indicator characterising the collective behaviour of an algorithm over a run.*

So far, in this section, we have presented a way to assess the exploitation characteristics of an algorithm in terms of the extent to which the algorithm exploits the sampled fitness values to direct the search. However, developing an accurate measure that quantifies the exploitation behaviour is complicated. This is because algorithms may make use of information other than fitness, to redistribute their activities; also, a precise definition of exploitation/exploration is lacking. The same can be said about assessing exploration behaviour. However, if we assume that any action made by an algorithm can be viewed as either exploitation or exploration depending on whether or not information influenced the decision leading to that action, then the proposed fitness-dependency measure quantifies a significant part of population exploitation behaviour.

3 Implementing the Proposed Model using Self-Organising Map

A Self-Organising Map (Kohonen, 2001) is an artificial neural network that can be trained using unsupervised learning. After training, SOMs can be used for mapping (classifying) or visualising high dimensional data. SOMs consist of a set of nodes (or neurons) arranged, usually, in a two-dimensional grid. A node, i, has an associated vector, m_i, of the same dimension of the input space and is connected to its neighbours according to a neighbourhood radius, as explained later in this section. The training is done by feeding a SOM with a large number of training samples drawn from the data space. Each time a sample, x, is fed into a SOM, the best matching node (BMN) is identified as the node whose vector has the smallest distance from the input sample, i.e., $BMN = \arg\min_i \|x - m_i\|$. Then m_{BMN} is updated by moving it slightly in the direction of x. The change to the BMN's vector results in changing the vectors of its neighbours as well (more on this below).

In this work, the dynamics of the grid nodes C_r^t of the proposed model is controlled by a SOM learning algorithm. In other words, a SOM is responsible for adjusting the position of each node vector, $m^{C_r^t}$.

We use SOMs in two stages. In the first stage the model node vectors are set using a SOM training algorithm fed with the individuals in the initial population. More specifically, after randomly generating initial vectors, in each training iteration, the SOM is fed with η (the size of population) individuals randomly selected from the initial population P_{init}. Node vectors, m^{C_r}, are then updated as follows:

$$m^{C_r} = m^{C_r} + \alpha e^{\left(-\frac{\|r - r_{BMN}\|^2}{2\sigma^2}\right)} \left(x - m^{C_r}\right) \tag{11}$$

where x is the input individual vector, α is the learning rate, σ is the neighbourhood radius and r_{BMN} is the index of the BMN. During this stage, the initial learning rate is set to $\alpha = 0.07$ and the neighbourhood radius is set to $\sigma = n/2$.[5] In this phase, these parameters are decreased over time as standard in SOMs. For training iteration c, the learning rate is computed as $\alpha_c = \alpha \exp\left(-c/I\right)$ and the neighbourhood radius is computed as $\sigma_c = \sigma \exp\left(-\frac{c}{I/log(\sigma)}\right)$, where I is the total number of training iterations. Note that, although an individual from the initial population may be used several times in SOM training, its information (as described by the proposed model, Equation 1) is recorded by the grid only once. The resulting grid, C^0, is topologically consistent and provides a representation of the initial population distribution in the search space.

In the second stage, the initial model is progressively modified (again using a SOM training algorithm) so as to track the collective dynamic behaviour of a population. In this stage, newly created individuals are fed into the SOM to adjust the node vectors. Here, we use a fixed non-zero learning rate and neighbourhood radius so that the nodes can track the population. In addition to the change that new individuals bring to the node vectors of the SOM, information of those individuals (as required in the model) is also recorded in the nodes. This process produces a sequence of grids C^t, for $t = 1, \ldots, N/\tau$, that represent population distributions and activities over time. Collectively they capture the collective dynamic behaviour of the population over the period of a run.

4 Experimental Results

In this section, we first describe a set of algorithms used in the experimentation as well as the operators and techniques used within them and then we experimentally characterise their collective dynamic behaviour using the model and features proposed above.

4.1 Algorithms and Experiments Description

To test the proposed model and features, a variety of operators, selection pressure levels and population handling techniques have been utilised which produce a drastically different collective behaviours. More specifically, we used both standard evolutionary algorithms [6] as well as EAs that utilise *fitness sharing*, *deterministic crowding* and *random immigrant* (details are in Appendix A). With each algorithm, *arithmetic crossover*, *blending crossover* (BLX-α), and *heuristic crossover* were used (their details and settings are explained in Appendix B). All EAs used tournament selection.

We chose arithmetic crossover, BLX-α, and heuristic crossover as they are representative of operators that exchange genetic material in a *stochastic*, *deterministic*, and *fitness-biased manner*, respectively. These crossover methods show different behaviours (in particular different levels of exploration and exploitation) because they handle the diversity of the population and they make use of the information available to them in

[5]We conducted our experiments on a grid of size 25×25

[6]By standard evolutionary algorithm we mean a generational evolutionary algorithm that uses no extra techniques. Full parameters settings are given in Appendix A

different ways (Herrera et al., 2003). However, their behaviour is also a function of their parameter settings.

Uniform arithmetic crossover (Michalewicz, 1996) handles the genetic material in a deterministic fashion. However, for $\lambda > 1$ or $\lambda < 0$, this operator exhibits an explorative behaviour and increases the population diversity (Herrera et al., 2003). BLX-α (Eshelman and Schaffer, 1992) creates new individuals by generating random values for their genes within ranges calculated from the parents gene values and α. BLX-α has a more stochastic behaviour and its exploration/exploitation behaviour can be tuned by adjusting α (Nomura and Shimohara, 2001). Heuristic crossover (Herrera and Lozano, 1996; Wright, 1990), on the other hand, creates offspring close to the best parent, which makes this operator more exploitative because it does not only make use of the provided genetic material, but also uses the fitness values to bias the search.

Unfortunately, we cannot assess the effect of recombination operators on the collective dynamic behaviour based only on the way they combine/alter genetic materials to produce new individuals. That is because the quality of the operators' output depends on the quality of the genetic material which is chosen by the selection mechanism. For this reason, we used tournament selection, where the selection pressure can easily be controlled by increasing/decreasing the tournament size. Larger tournaments induce a high selection pressure and lead to a high exploitation behaviour whereas small tournaments lead to low pressure and high exploration.

Our experiments focus on characterising the collective dynamic behaviour of EAs that use different parameter settings for their crossover methods over different levels of selection pressure. The interaction between the biases of different operator creates different collective dynamic behaviours. While tournament selection uses fitness values to bias the search, other population handling techniques involve other elements. Fitness sharing and crowding use distances in handling the population, while random immigrants tries to counterbalance the bias of selection by adding more stochastic behaviour (randomness) into the search.

We adopted a naming system to refer to the algorithms and their settings when presenting results. An algorithm name consists of four parts, representing, respectively, the population handling technique, the crossover method, the crossover control parameter, and the tournament size. For example, StBLX-0.5-3 is a standard EA using BLX crossover with control parameter value 0.5 and tournament size 3. Full details of the algorithm naming system along with parameter settings, algorithm techniques and benchmark problems are provided in Appendix A.

In showing the results of the experiments, we used two types of figures. The first describes the change of a certain feature over time (fitness function evaluations) (e.g., Figure 3), while the second characterises the effect of using different selection pressures (different tournament sizes) on a certain feature of the collective behaviour (e.g., Figure 5). In most of the figures of the second type, we compared the effect of selection pressure on different settings of each crossover (the first row of plots), and we compared the crossover methods with each other using different settings (the lower row of plots). We called these parameter settings *balanced*, *exploitative* and *explorative* depending on the qualitative behaviour they induced. We picked the most explorative settings (or the least exploitative in case of Heuristic crossover) from the range of used settings for the explorative parameter settings (BLX-0.75, Heur-0.4, and Arth-1.5). In a similar way, we chose the exploitative parameter settings (BLX-0.0, Heur-0.1, and Arth-0.2). The set of balanced parameter settings represents more moderate and common settings for each of the crossover methods (BLX-0.5, Heur-0.2, and Arth-0.4).

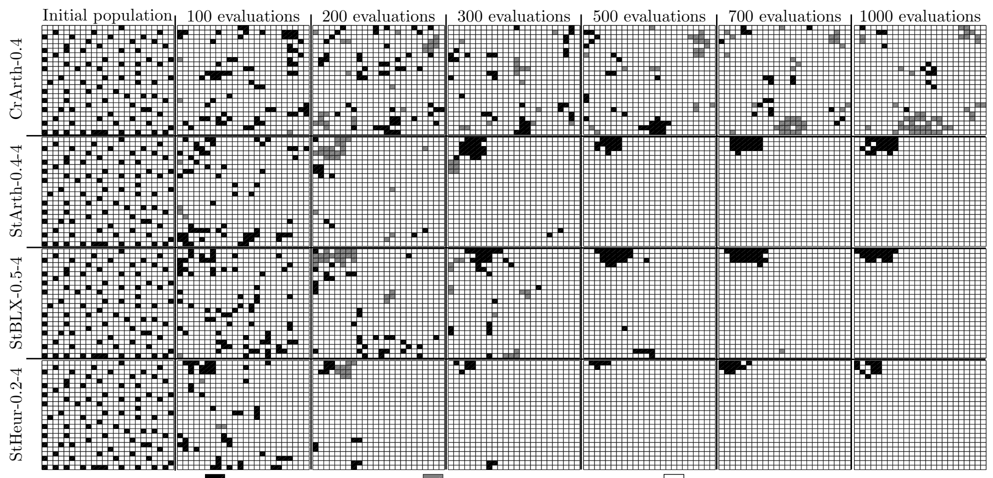

Figure 2: Growth and non-growth regions formation on the SOM grid of nodes.

4.2 Characterising Emergent Features of Collective Dynamic Behaviour

4.2.1 Illustration of the Operations of the Model

Figure 2 depicts the regions formation in the model for four of the algorithms we will characterise later in this section: three standard EAs with different types of crossover and one EA with crowding and arithmetic crossover. The figure shows 7 snapshots of the grid of nodes in the model, during the first 1000 fitness evaluations of a run of each of the four algorithms. All runs had the same initial random seed (and, hence, the same initial population and grid of nodes).

As one can see, only a small subset of the model nodes are used at any given time. Note how the model of the crowding EA holds many more non-growth nodes than the standard EAs and how these are distributed more evenly within the grid, indicating that correspondingly different regions of the search space are concurrently explored. Note also how the most exploitative of the four algorithms (StHeur-0.2-4) rapidly focuses its search on fewer and smaller (and, hence, more focused) growth regions.

4.2.2 Activity rate and size ratio

To characterise the effect of using different techniques on how the population distributes its focus among different regions of activity, we measured the activity rate (Equation 2) in growth, non-growth, and best regions, in a standard EA (StArth-0.4-4) and in EAs with fitness sharing (FSArth-0.4-4), deterministic crowding (CrArth-0.4) and random immigrants (RIArth-0.4-4). All the algorithms use tournament of size 4 (except for deterministic crowding which uses its own selection mechanism) and arithmetic crossover (Arth-0.4). Figure 3 shows the results.

By comparing Figures 3(A) and (B) one immediately sees that algorithms using niching techniques allocate little activity to the growth regions, while most of the population focuses on searching non-growth regions. This is a sign of the explorative nature of this type of algorithms. We also see that both the standard evolutionary algorithm and the random immigrants version have a different collective behaviour, as they initially focus on the growth regions more than on the non-growth ones. Allocating more

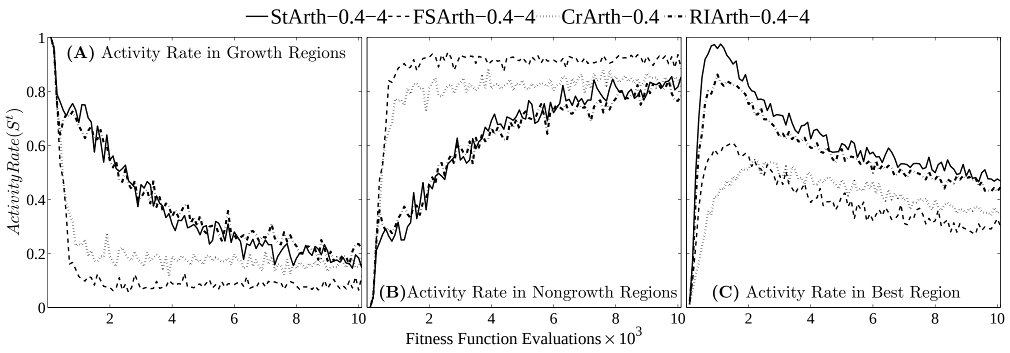

Figure 3: Activity rates of growth, non-growth, and best regions measured over the period of a run in standard, fitness-sharing, deterministic-crowding and random-immigrant EAs.

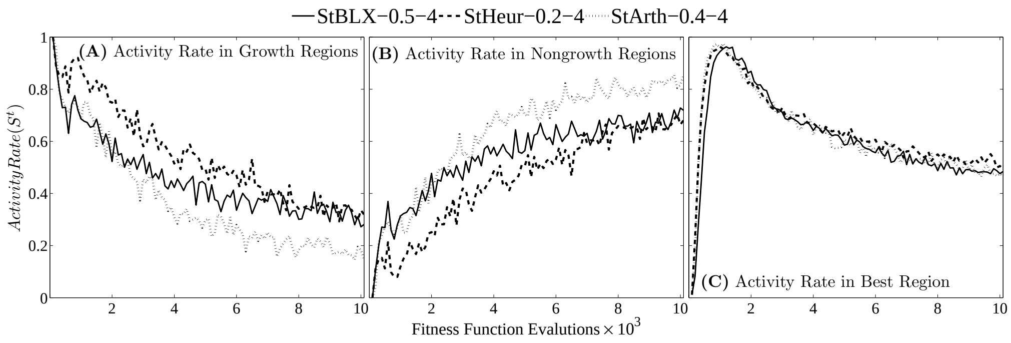

Figure 4: Activity rates of growth, non-growth, and best regions measured over the period of a run in three standard EAs utilising BLX-0.5, Arth-0.4 and Heur-0.2 crossover.

activities to pursue fitness growth is an indicator of exploitation behaviour. Naturally, the activity rate of these EAs in growth regions declines as the run proceeds as a result of the population's convergence and its losing the ability to achieve fitness growth.

Figure 3(C) depicts activity rate in the best region found by each algorithm. As we can see from the figure, after an initial increase, the activity rate in the best region declines over the course of a run. This phenomenon may contradict the standard interpretation of convergence, which is normally viewed as the population gathering in one region around the best-found individual (especially in standard EAs). At the beginning of a run, a standard EA form a large activity region around the best-discovered area. As the run goes on, this region shrinks. At some point, the bias of the genetic operators (e.g., BLX-0.75 or Arth-1.5) starts balancing the focusing bias of selection. Then the operators scatter individuals away from the best region, thereby creating little temporary regions in its vicinity. More results and discussion on this issue will be provided later in this section.

In Figure 4, we show the effect of using different crossover methods on the activity rates in different regions of the population in a standard EA. A comparison of Figures 4(A) and (B) shows that a standard EA with Heur-0.2 crossover allocates more

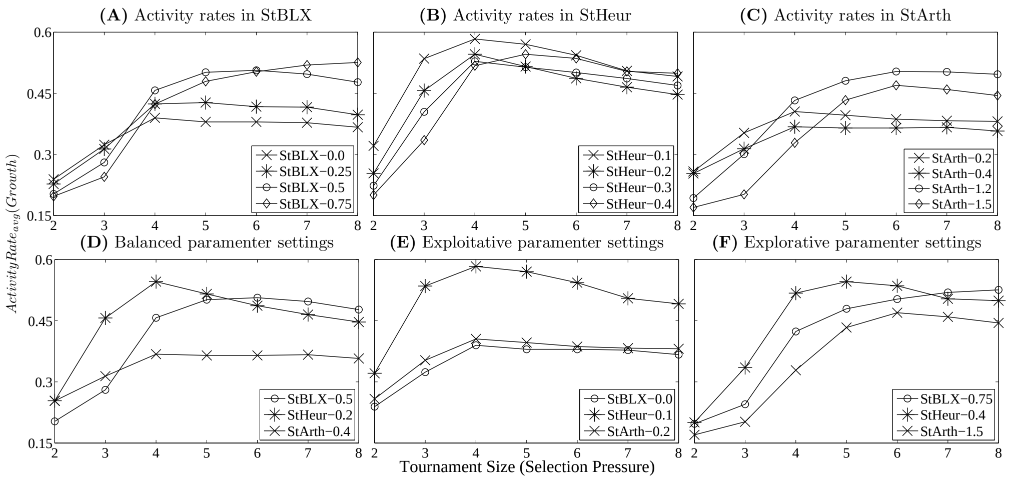

Figure 5: The effect of using different parameter settings for three crossover methods over different levels of selection pressure on the average of activity rate in the growth regions.

activities to the growth regions, which is, of course, due to the exploitative nature of this operator. Standard EAs with Arth-0.4 have the lowest interest in growth regions, while an EA with BLX-0.5 changes its character as the population loses diversity and converges. Interestingly, as shown in Figure 4(C), the three algorithms approximately have the same activity rate in the best region.

To analyse the combined effect of using different crossover methods and different selection pressures (tournament sizes) on a population's allocation of activities to regions, we used standard EAs utilising three crossover methods, with four parameter settings for each one, over different selection pressure levels. The parameters were chosen so that a range of exploitation and exploration (or less exploitation, in the case of Heur-λ) behaviours would be produced. Figure 5 depict the results.

Let us first focus on Figures 5 (A)–(C), which report the average activity rate in growth regions using BLX-α, Heur-λ and Arth-λ crossovers. When using *exploitative* parameter settings (BLX-0.0, BLX-0.25, Arth-0.2, Arth-0.4, Heur-0.1, . . . , Heur-0.4), the average activity rates in growth regions increases with the selection pressure, until a certain point beyond which increasing the selection pressure further has little effect of the activity rate. This happens because highly exploitative operators force the population to converge too quickly onto one region in the search space, thereby losing the ability to discover new good regions and produce further fitness growth. Thus, *stepping up selection pressure accelerates losing the ability of producing growth in exploitative crossover methods*. The faster the population converges, the fewer growth regions can be found and, thus, the less activity in these regions. Because of this, with exploitative crossovers and high selection pressure the population can only rely on mutation to bring about some diversity that may help discover new regions and move the search away from local optima.

Conversely, as shown in Figures 5 (A)–(C), increasing the selection pressure in EAs that use crossovers with *explorative* parameter settings (BLX-0.5, BLX-0.75, Arth-1.2 and

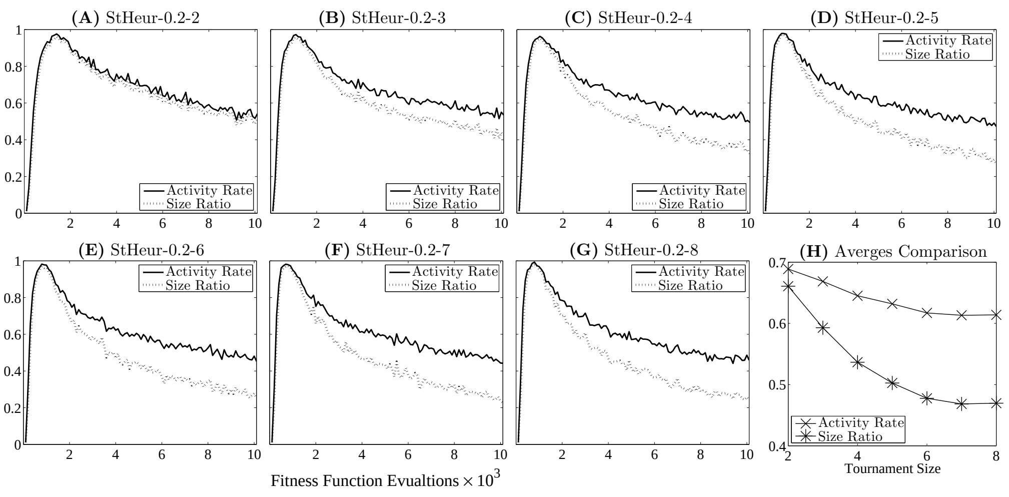

Figure 6: The effect of increasing the selection pressure on the size ratio of the best region for EAs using Heur-0.2 crossover.

Arth-1.5) drives the population to discover more growth regions.[7] With low selection pressure, explorative crossovers have a bias that prevents the population from converging and concentrating its activities only on the good regions that has so far discovered. Instead, crossover will "push" individuals away from these regions, thereby reducing the chance of finding better solutions in these regions and, thus, producing growth. However, also in these crossovers increasing the selection pressure makes the population converge, although less rapidly, and counterbalances their diversification effects. In this case, the population converges to a small region around the best solutions and the explorative crossover creates some small growth regions by scattering individuals around the best region.

To better illustrate these observations, in Figures 5(D)–(E) we have reproduced nine configurations from the previous plots but this time we have grouped them on the basis of how exploitative/explorative the crossover's parameters settings are. More specifically, Figure 5(D) shows the activity-rate averages for crossovers using balanced parameter settings, while Figures 5(E) and (F) report those for exploitative and explorative settings, respectively.

Search operators and selection methods have a major influence on the size of activity regions explored by an algorithm. For example, using a high selection pressure will shrink the population activity regions more than using a low selection pressure. Figures 6(A)–(G) illustrate this using the size ratio and the activity rate of the best region in an EA using Heur-0.2 crossover. The figures show that as the selection pressure increases, the activity rate becomes more concentrated. However, the effect of the selection pressure on the size ratio becomes weaker as the selection pressure increases. This is shown also in Figure 6(H) that plots size ratio and activity rate averages as a function of the tournament size.

Figures 7(A)–(C) show the effects of the selection pressure on the activity concentration in the best region in standard EAs with different types of crossover. Figure 7(D)

[7]Of course, growth in those regions does not mean that an algorithm also improves the best-so-far fitness of the search.

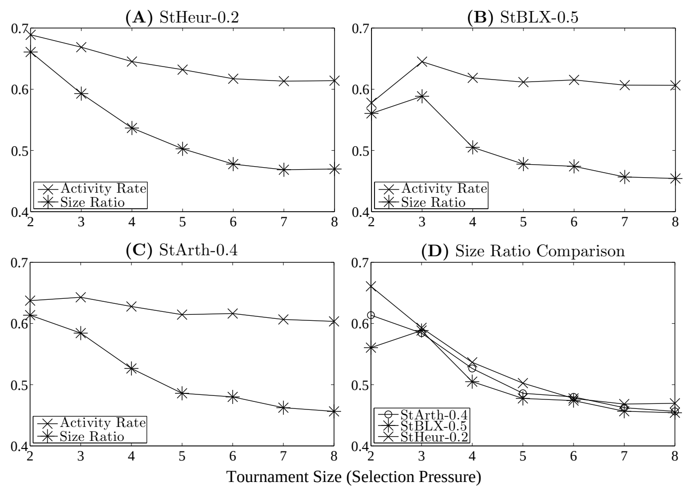

Figure 7: The averages of size ratio and activity rate of the best region for StHeur-0.2, StBLX-0.5, and StArth-0.4 EAs over different levels of selection pressure.

compares the size ratios of these algorithms and shows that they are more and less the same, which implies that selection, more than crossover, has a significant effect on changing the size ratio of the best regions of population.

4.2.3 Hit Distance and Displacement

Hit distance and displacement provide information about the population dynamics from a geometric perspective.

Hit distance measures how far a population searches away from previously sampled regions in the fitness landscape. For example, a large hit distance implies that

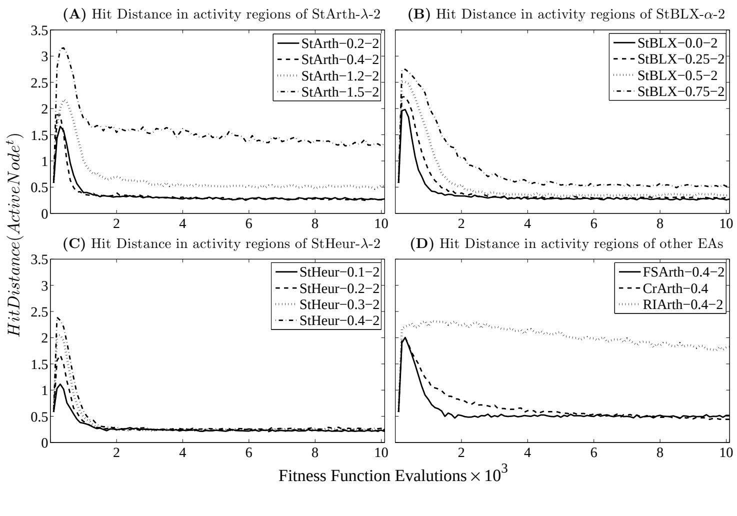

Figure 8: Hit distance of all activity regions for EAs with Arth-λ, BLX-α and Heur-λ crossover methods, and with different population handling techniques.

search operators produce/maintain genetic diversity. The hit distance varies as the run progresses and also depends on the diversity of population and the nature of the operators used. Figure 8 shows how using different crossover methods or population handling techniques affects the hit distance.

In particular, Figures 8(A)–(C) illustrate the effect of using different crossover methods. (We have used the same selection pressure level for all results, except for CrArth-0.4 that uses its own selection mechanism). Note that the ones with an explorative nature tend to have a large hit distance, while the hit distance in the more exploitative ones is smaller and declines rapidly to a low level as the run proceeds. Despite the fact that selection pressure has an impact on population diversity, explorative crossovers manage to maintain a higher level of hit distance than the exploitative ones. StArth-1.2-2, StArth-1.5-2, StBLX-0.5-2 and StBLX-0.75-2 (see Figures 8(A) and (B), respectively) start the run with a high hit distance and as the selection pressure forces the population to converge, the hit distance decreases, but maintain a higher level than other exploitative crossovers. On the other hand, exploitative crossovers start the run with different hit distances (according to how exploitative they are based on their parameter settings), but as the run goes on, the hit distances drop to the same level regardless the parameter settings. This is particularly noticeable with StHeur-λ-2 (Figure 8(C)) but the same can be said about both other algorithms (Figures 8(A) and (B)). In these algorithms, all the operators, apart from mutation, have an exploitative nature and there is no operator that can maintain the diversity of the population. Moreover, as the diversity goes to the lowest level, the population activity becomes limited to a certain region where the new individuals (hits) are created not far away from their parents. Figure 8(D) shows the hit distance of three different algorithms that use the same crossover (Arth-0.4) but with different population-handling techniques. The algorithms act on the diversity of population in different ways, which lead the hit distance to be different even if they use exactly the same crossover method.

To explore in greater depth the effects of selection pressure on the hit distance

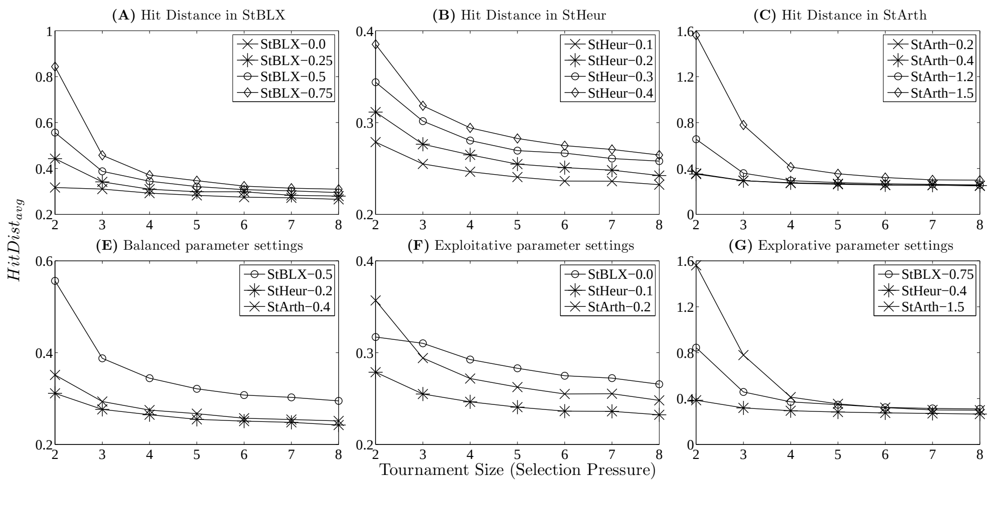

Figure 9: The effect of using different parameter settings for three crossover methods and different levels of selection pressure on the average hit distance of all activity regions.

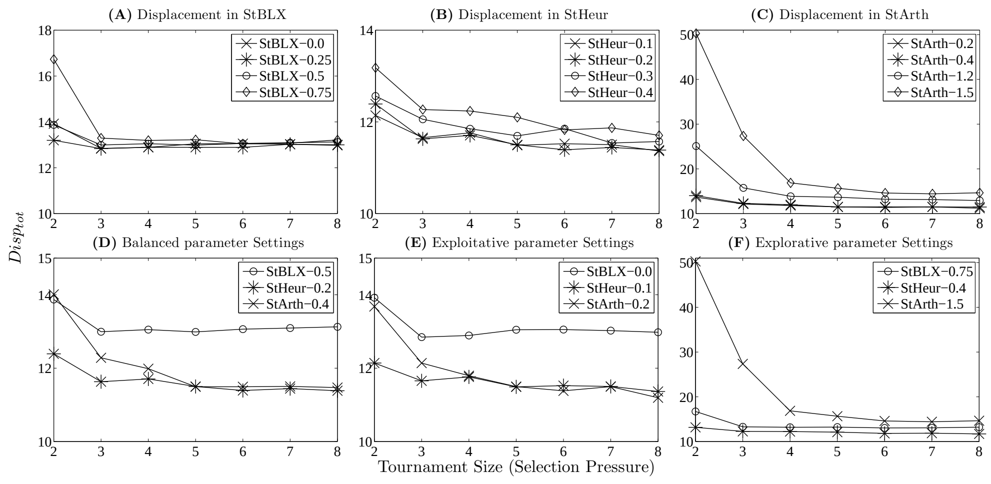

Figure 10: The effect of using different parameter settings for three crossover methods over levels of selection pressure on the total displacement of activity regions.

of different crossover methods, we measured the average of hit distance over different selection pressure levels, for the three crossover methods studied in this paper. Figures 9(A)–(C) show that the selection pressure has a large effect on explorative crossover methods, while the exploitative crossover methods undergo a lesser decline. The significant change in hit distance observed in explorative crossover methods is due to the conflicting biases of these crossovers and selection, where increasing the selection pressure overwhelms the bias of crossovers. Figures 9(D)–(F) compare the hit distance for crossovers with balanced, exploitative and explorative parameter settings, respectively.

As indicated in Section 2.2, the total displacement characterises a population's dynamics by measuring the amount of change population activities cause to the positions

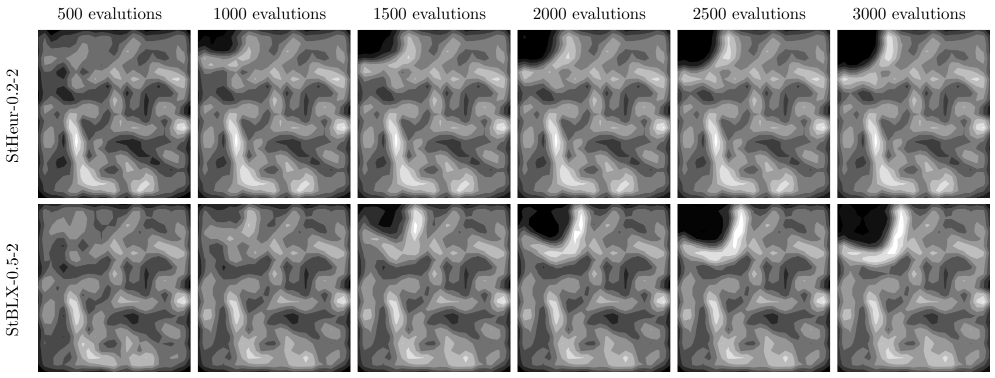

Figure 11: Model grid visualised using U-Matrix method to show the change in node vectors movements as they track population activity. Darker areas represent node vectors that are close to each other and *vice versa*.

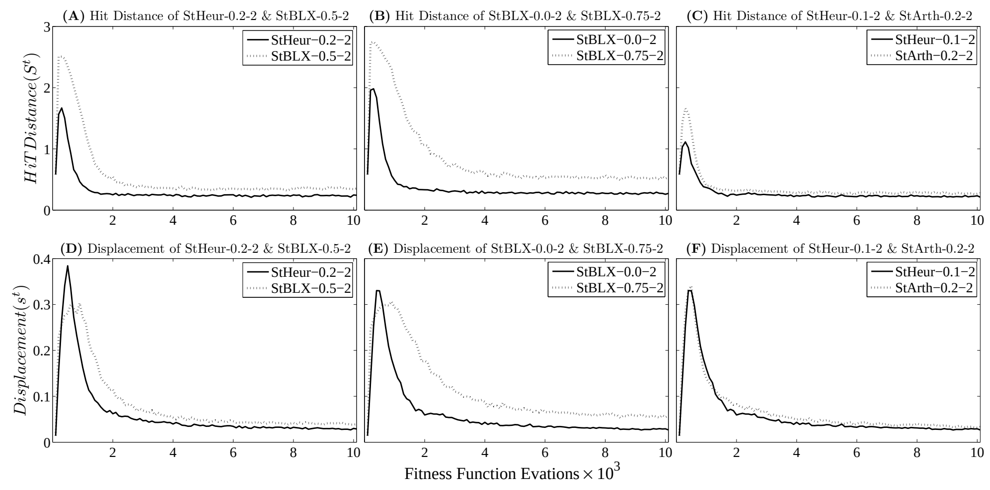

Figure 12: Hit distance and displacement comparison over the period of a run for EAs using different crossovers.

of node vectors. Populations that have a high total displacement exhibit a highly explorative nature, as they tend to transfer their activity regions around the search space and not to concentrate on certain regions. They usually have the ability to escape local optima, but they may risk abandoning the global optimum due to their tendency to widen and diversify the search. Figure 10 shows the effect of using different crossover methods and different level of selection pressure on the population's total displacement. Algorithms with explorative crossovers have a greater total displacement. Also, increasing the selection pressure affects them more than exploitative crossovers.

Figure 11 uses the U-Matrix method (Ultsch and Siemon, 1990) (which colours node vectors based on the average distance of their neighbours) to illustrate the movements of node vectors as they track population activity. The figure shows samples of two runs, StHeur-0.2-2 and StBLX-0.5-2, where the darker areas represent node vectors that are close to each other, while the light areas represent node vectors that point to different areas in the search space. We can see that StHeur-0.2-2 has a higher displacement, as the node vectors move faster towards the best-found region. More explanation on this particular example will be provided bellow.

Comparing population displacement and hit distance reveals information about different operator biases and the way they explore the search space. Figure 12 shows examples of how hit distance and displacement of different EA change over a run. In Figure 12(A) we notice that StHeur-0.2-2 has a low hit distance compared to StBLX-0.5-2. However, StHeur-0.2-2 displacement (Figures 12 (D)) is greater than that of StBLX-0.5-2 and this changes in a fashion that implies a rapid move to a region in the search space that is then followed by a sudden slowing down of its dynamics. This suggests that Heur-0.2 crossover *has a stronger bias* and tend to move the population in the direction of good observed fitness values, while BLX-0.5 *tends to explore around activity regions longer* and does not guide the search in such a directed way. The same explanation applies to StBLX-0.0-2 and StBLX-0.75-2 in Figures 12(B) and (E), while Figures 12(C) and (F) show that StHeur-0.1-2 and StArth-0.2-2 have approximately the same displacement behaviour despite the fact that they have a different hit distance.

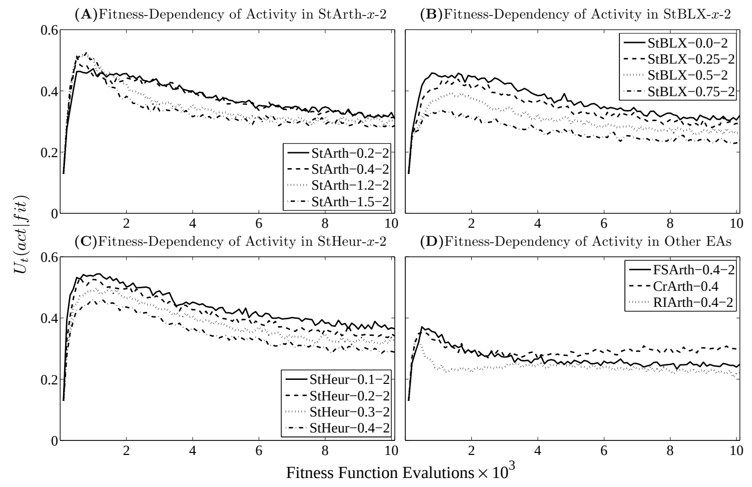

Figure 13: Fitness-dependency of activity, $U_t(act|fit)$, for EAs with Arth-λ, BLX-α and Heur-λ crossovers and different population handling techniques.

4.3 Characterising Exploitation Behaviour

Experiments were carried out to characterise exploitation behaviour of the same set of algorithms by assessing the fitness-dependency of our collective-behaviour features. Figure 13 shows the fitness-dependency of the activity of different algorithms over time. When a population is more diverse and has a wide range of fitness values, which is the case of the initial population, algorithms show a higher level of exploitation as they tend to be more selective and focus on regions of good fitness. As the run progresses and the activity regions become smaller and so does the range of

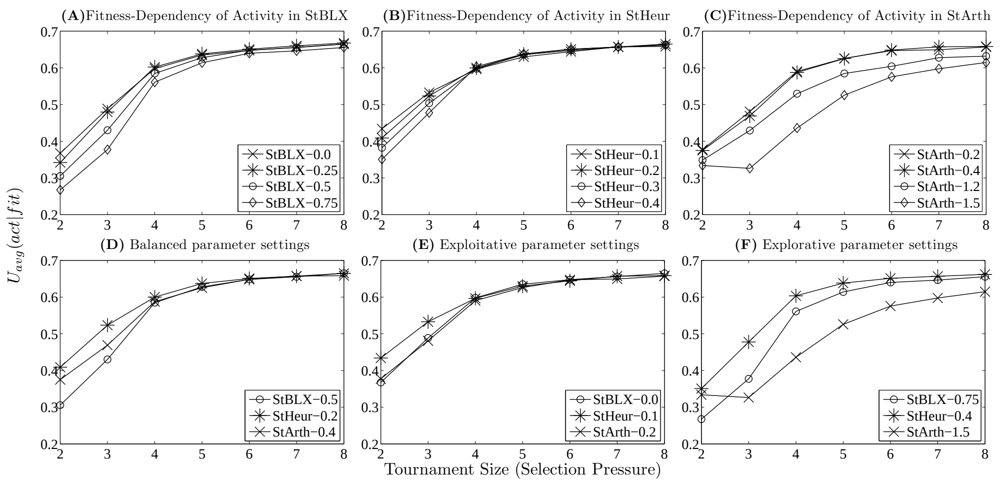

Figure 14: Average fitness-dependency of activity, $U_{avg}(act|fit)$, for different parameter settings for three crossover methods over different levels of selection pressure.

fitness values, our exploitation indicator declines towards a lower value which depends on the operators used and their parameter settings. Algorithms using heuristic crossover exhibit higher exploitation behaviour, while algorithms with diversity maintaining/generating techniques (fitness sharing, deterministic crowding and random immigrants) show the lowest level of exploitation.

Figure 14 shows the effect of changing the selection pressure on the average of fitness-dependency of activity $U_{avg}(act|fit)$. In particular, Figures 14(A)–(C) show $U_{avg}(act|fit)$ for the three crossover methods with four different settings for each, while Figures 14(D)–(F) compare algorithms with three different parameter settings. Notice that for most of these algorithms as we increase the selection pressure, the bias of selection tends to overcome the bias of crossover and the collective behaviour tends to become more exploitative. It is clear from the figures that when crossover has an exploitative bias (Heur-0.1, . . . , Heur-0.4, Arth-0.2, Arth-0.4, BLX-0.0 and BLX-0.25), increasing selection pressure forces the algorithms to have similar exploitation behaviours regardless of the parameter settings of crossover. That is because these operators have a bias consistent with selection. On the contrary, crossover methods with explorative parameter settings tend to be more resistant to the bias of selection as shown in Figure 14(G).

Figures 15 and 16 depict the effects of the selection pressure on fitness-dependency of hit distance ($U_{avg}(hd|fit)$) and displacement ($U_{avg}(dis|fit)$), respectively. These effects are more marked for fitness-dependency of activity than for the other features (more details are provided in Table 1), although our exploitation behaviour indicators are still increasing functions of the selection pressure. All crossover methods, approximately, have their bias overcome by the selection bias as the selection pressure increases However, algorithms using crossovers with explorative parameter settings are more resilient to the selection bias, as shown in Figures 15(G) and 16(G). Note also that in Figures 15(F) and 16(F), fitness-dependency of hit distance and displacement of algorithms with exploitative parameter settings vary to a lesser extent.

Table 1 summarises our exploitation indicators for Standard EAs. The table shows the minimum, maximum, and range of each of our three fitness-dependency averages

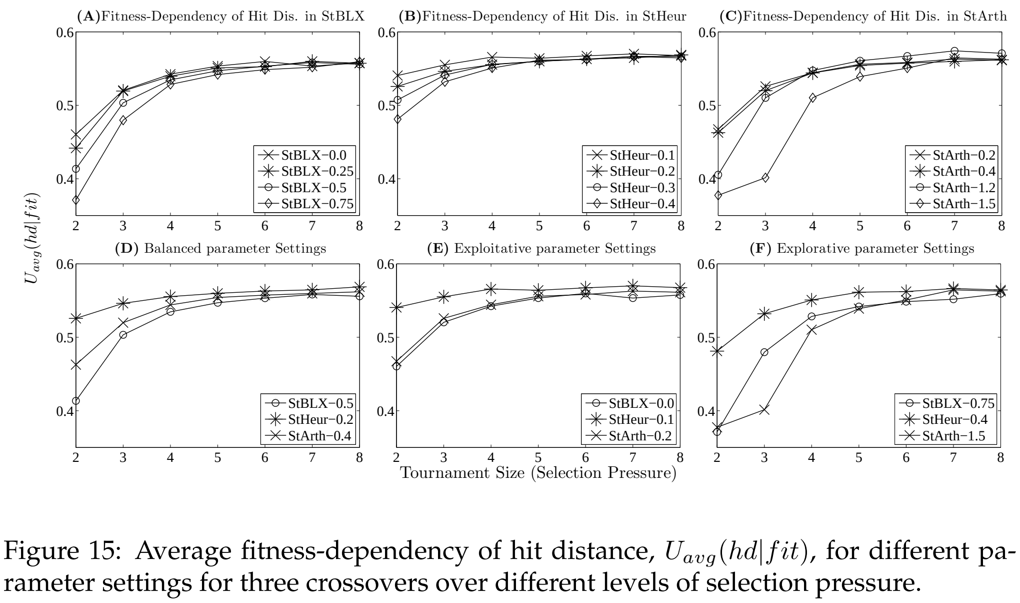

Figure 15: Average fitness-dependency of hit distance, $U_{avg}(hd|fit)$, for different parameter settings for three crossovers over different levels of selection pressure.

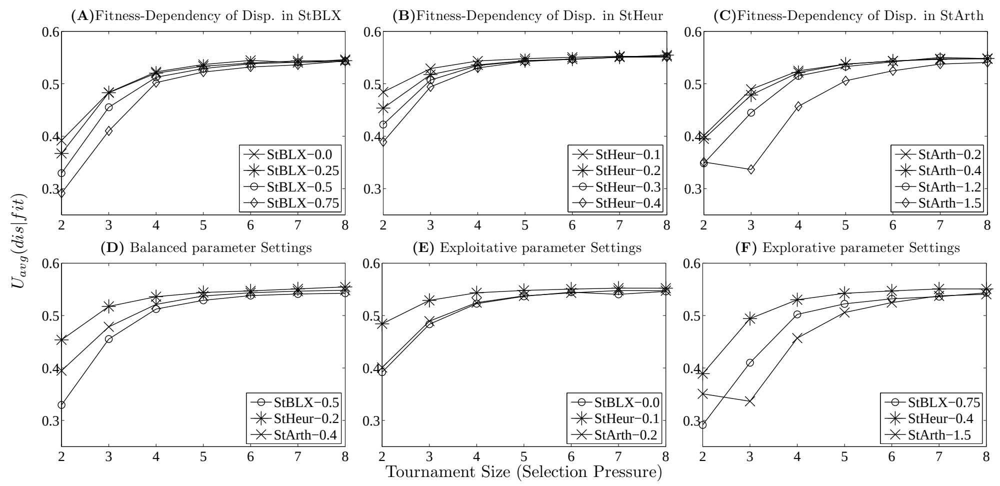

Figure 16: Average fitness-dependency of displacement, $U_{avg}(dis|fit)$, for different parameter settings for three crossover methods over different levels of selection pressure.

for the three crossover methods over all the possible settings and all selection pressure levels. From the table we see that $U_{avg}(act|fit)$ has the lowest minimum value and the biggest maximum value, thus, the largest range, among all dependency measures. On the other hand, we see that $U_{avg}(hd|fit)$ has the highest minimum value and the smallest range among all dependency measures. These two observations tell us that the activity is more driven by selection because increasing the selection pressure increases the fitness-dependency, while the hit distance is mainly dependent on crossover, as changing the selection pressure does not have a significant effect on fitness dependency. Displacement fitness-dependency ranges are in between these extremes, which suggests that both selection and crossover have an effect on its fitness dependency. By looking at each of the table parts separately, we notice that heuristic crossover (Heur-λ) has the highest fitness dependency, which is consistent with its fitness-exploiting na-

Table 1: Minimum, maximum and range of fitness-dependency averages of the three crossover methods across all parameter settings and selection pressures used in our experiments. <u>Underlined</u> and **Bold** values represent the smallest and biggest value for each column within each part of the table.

Dependency Measure	Algorithm	Minimum	Maximum	Range
$U_{avg}(act\|fit)$	StArth	0.3262	<u>0.6583</u>	0.3321
	StBLX	<u>0.2674</u>	0.6597	**0.3923**
	StHeur	**0.3505**	**0.6648**	<u>0.3142</u>
$U_{avg}(hd\|fit)$	StArth	0.3775	**0.5742**	**0.1967**
	StBLX	<u>0.3711</u>	<u>0.5599</u>	0.1888
	StHeur	**0.4812**	0.5701	<u>0.0889</u>
$U_{avg}(dis\|fit)$	StArth	0.3365	0.5502	0.2137
	StBLX	<u>0.2914</u>	<u>0.5461</u>	**0.2547**
	StHeur	**0.3892**	**0.5547**	<u>0.1655</u>

ture in handling the genetic material. We also see that hit-distance fitness-dependency of the same crossover is the least affected by selection pressure. On the other hand, blending crossover (BLX-α) has the lowest minimum fitness-dependencies for all the three exploitation indicators, which can be attributed to its stochastic nature (although, by looking at the last column of the table, we see that selection pressure increases the fitness-dependency of BLX-α).

5 Conclusions

In this paper, we have presented a model to represent the collective dynamic behaviour of population-based algorithms. As we saw in Section 1, previous models used either the genotypic or the phenotypic representation to capture such behaviour. Many such models are tailored to evolutionary algorithms (usually a simple version of them) and have a limited practical use. Here we have proposed an algorithm-independent empirical approach that uses a sequence of topological genotypic and phenotypic representations (at the same time) to capture the population collective behaviour. Our models allow one to represent the dynamics at different levels of abstraction, via the aggregation of the most elementary components of the model into more coarse-grained entities.

In addition, a number of measures have been proposed to represent different features of the collective dynamic behaviour of a population. These fall within two categories: activity-related features and distance-related features. Activity-related measures describe the way an algorithm distributes its activities around regions in the search space while distance-related measures describe the population's dynamics from a geometric viewpoint. These measures can be used to qualitatively characterise the exploration/exploitation behaviour of an algorithm. Also, a set of fitness-dependency measures are introduced that can be used as indicators of an algorithm's exploitation behaviour. These measures assess the dependency of different population activity on fitness values, in order to characterise the exploitation tendency of an algorithm. We used an entropy-based measure, namely the uncertainty coefficient, to quantify the dependency of different emergent features of the collective behaviour on the fitness.

The proposed model has been implemented using a modified version of a self-organising map. SOMs are used to capture the distribution of the initial population, and track its moves as the algorithm operates on solving the problem and creates new individuals in different areas of the search space. There is nothing special about this choice. Other topology preserving algorithms might have been used. Also, for a number of features, one does not need a topology and so a vector-quantisation algorithm would suffice. In the future, we will investigate the relative benefits and drawbacks of alternative techniques.

In this article, we have studied the collective dynamic behaviour of EAs focusing on the effect of using different crossover methods, various parameter settings, and different population handling techniques. The impact of combining consistent or conflictive biases on the emergent features of the collective behaviour has been studied. However, we believe our methods would also work well with other population-based algorithms. In the future, we will explore also this avenue of research.

Results of experimenting with different EAs have shown that the proposed measures are able to capture new aspects of the population dynamics and reveal important features of the collective behaviour. Experiments have also shown that our proposed measures of exploitation are useful in quantifying the exploitation behaviour of populations. The proposed measures enable a new level of algorithm comparison and provide tools to investigate the effect of combining different biases on the resulting collective behaviour of a population.

Our model is an effort to contribute to our understanding of the collective dynamic behaviour of population and the exploration/exploitation phenomena. It provides an empirical analysis tool that allows us to represent the dynamic behaviour at different levels of abstraction and to identify and measure different emergent features of the dynamic behaviour, thus, enabling us to better describe and, potentially, tune that behaviour. By studying the dependency between the emergent features, we tried to define what constitutes exploitation (or exploration) and to understand the effect of operators with different biases on the collective dynamic behaviour of populations.

This model can also be of use by EA practitioners for analysing the effects of operators and for tuning their control parameters. It can help characterise the way contradictory/consistent biases combine together to produce the collective behaviour, or how populations behave differently according to the nature of the fitness landscape they are operating on. This can help identify what make certain algorithms fail/succeed in searching certain fitness landscapes. All these issues will be investigated in future work.

Acknowledgement

This work was supported by EPSRC Doctoral Training Account (DTA) research studentship.

References

Amor, H. and Rettinger, A. (2005). Intelligent exploration for genetic algorithms: using self-organizing maps in evolutionary computation. In *Proceedings of the 2005 Genetic and Evolutionary Computation Conference (GECCO 2005)*, pages 1531–1538. ACM.

Back, T., Fogel, D., and Michalewicz, Z. (1997). *Handbook of evolutionary computation*. IOP Publishing Ltd.

Bethke, A. (1981). *Genetic algorithms as function optimizers*. PhD thesis, Unversity of Michigan.

Cobb, H. G. and Grefenstette, J. J. (1993). Genetic algorithms for tracking changing environments. In Forrest, S., editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 529–530, San Mateo, CA. Morgan Kaufmann.

Davis, T. E. and Principe, J. C. (1993). A Markov chain framework for the simple genetic algorithm. *Evolutionary Computation*, 1(3):269–288.

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

Droste, S., Jansen, T., and Wegener, I. (2002). On the analysis of the (1+1) evolutionary algorithm. *Theoretical Computer Science*, 276(1-2):51–81.

Eiben, A. and Schippers, C. (1998). On evolutionary exploration and exploitation. *Fundamenta Informaticae*, 35(1):35–50.

Eshelman, L. J. and Schaffer, J. D. (1992). Real-coded genetic algorithms and interval-schemata. In Whitley, L. D., editor, *FOGA*, pages 187–202. Morgan Kaufmann.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley.

Goldberg, D. E. and Richardson, J. (1987). genetic algorithms with sharing for multimodal function optimization. In *Genetic Algorithms and their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, pages 41–49. Lawrence Erlbaum.

Graff, M. and Poli, R. (2008). Practical model of genetic programming's performance on rational symbolic regression problems. In O'Neill, M., Vanneschi, L., Gustafson, S., Esparcia Alcazar, A. I., De Falco, I., Della Cioppa, A., and Tarantino, E., editors, *Proceedings of the 11th European Conference on Genetic Programming, EuroGP 2008*, volume 4971 of *Lecture Notes in Computer Science*, pages 122–133, Naples. Springer.

Herrera, F. and Lozano, M. (1996). Heuristic crossovers for real-coded genetic algorithms based on fuzzy connectives. In Voigt, H.-M., Ebeling, W., Rechenberger, I., and Schwefel, H.-P., editors, *PPSN*, volume 1141 of *Lecture Notes in Computer Science*, pages 336–345. Springer.

Herrera, F., Lozano, M., and Sánchez, A. (2003). A taxonomy for the crossover operator for real-coded genetic algorithms: An experimental study. *International Journal of Intelligent Systems*, 18(3):309–338.

Holland, J. H. (1992). *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. The MIT press.

Igel, C. and Toussaint, M. (2003). On classes of functions for which no free lunch results hold. *Information Processing Letters*, 86(6):317–321.

Jansen, T., Jong, K. A. D., and Wegener, I. (2005). On the choice of the offspring population size in evolutionary algorithms. *Evolutionary Computation*, 13(4):413–440.

Jansen, T. and Wegener, I. (2002). The analysis of evolutionary algorithms - a proof that crossover really can help. *Algorithmica*, 34(1):47–66.

Jones, T. (1995). *Evolutionary Algorithms, Fitness Landscapes and Search*. PhD thesis, University of New Mexico, Albuquerque.

Kohonen, T. (2001). *Self-organizing maps*. Springer series in information sciences. Springer.

Liang, J., Suganthan, P., and Deb, K. (2005). Novel composition test functions for numerical global optimization. In *Swarm Intelligence Symposium, 2005. SIS 2005. Proceedings 2005 IEEE*, pages 68–75. IEEE.

Macready, W. G. and Wolpert, D. (1998). Bandit problems and the exploration/exploitation tradeoff. *IEEE Trans. Evolutionary Computation*, 2(1):2–22.

Mahfoud, S. W. (1992). Crowding and preselection revisited. In Männer, R. and Manderick, B., editors, *PPSN*, pages 27–36. Elsevier.

Mahfoud, S. W. (1995). *Niching methods for genetic algorithms*. PhD thesis, University of Illinois at Urbana-Champaign, Urbana, IL, USA. IlliGAL Report 95001.

Michalewicz, Z. (1996). *Genetic algorithms and data structures - evolution programs (3. ed.)*. Springer.

Mitavskiy, B. and Rowe, J. (2006). Some results about the Markov chains associated to GPs and to general EAs. *Theoretical Computer Science*, 361(1):72–110.

Mitchell, M., Forrest, S., and Holland, J. (1992). The royal road for genetic algorithms: fitness landscapes and ga performance. In Varela, F. J. and Bourgine, P., editors, *Toward a Practice of Autonomous Systems, Proceedings of the First European Conference on Artificial Life*, pages 245–254. The MIT Press.

Neumann, F. and Wegener, I. (2007). Randomized local search, evolutionary algorithms, and the minimum spanning tree problem. *Theoretical Computer Science*, 378(1):32–40.

Nomura, T. and Shimohara, K. (2001). An analysis of two-parent recombinations for real-valued chromosomes in an infinite population. *Evolutionary Computation*, 9(3):283–308.

Poli, R., Graff, M., and McPhee, N. F. (2009). Free lunches for function and program induction. In Garibay, I. I., Jansen, T., Wiegand, R. P., and Wu, A. S., editors, *FOGA*, pages 183–194. ACM.

Poli, R. and McPhee, N. F. (2003a). General schema theory for genetic programming with subtree-swapping crossover: Part I. *Evolutionary Computation*, 11(1):53–66.

Poli, R. and McPhee, N. F. (2003b). General schema theory for genetic programming with subtree-swapping crossover: Part II. *Evolutionary Computation*, 11(2):169–206.

Poli, R., McPhee, N. F., and Rowe, J. E. (2004). Exact schema theory and Markov chain models for genetic programming and variable-length genetic algorithms with homologous crossover. *Genetic Programming and Evolvable Machines*, 5(1):31–70.

Poli, R. and Vanneschi, L. (2007). 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*, pages 1335–1342. ACM.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., Flannery, B. P., and Flannery, B. P. (2007). *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press.

Prügel-Bennett, A. and Shapiro, J. (1994). Analysis of genetic algorithms using statistical mechanics. *Physical Review Letters*, 72(9):1305–1309.

Schumacher, C., Vose, M. D., and Whitley, L. D. (2001). The no free lunch and problem description length. In Spector, L., Goodman, E. D., Wu, A., Langdon, W. B., Voigt, H.-M., Gen, M., Sen, S., Dorigo, M., Pezeshk, S., Garzon, M. H., and Burke, E., editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 565–570, San Francisco, California, USA. Morgan Kaufmann.

Shannon, C. E. (1948). A mathematical theory of communications. *The Bell Systems Technical Journal*, 27:379–423, 623–656.

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

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

Stephens, C. R. and Waelbroeck, H. (1999). Schemata evolution and building blocks. *Evolutionary Computation*, 7(2):109–124.

Storch, T. and Wegener, I. (2004). Real royal road functions for constant population size. *Theoretical Computer Science*, 320(1):123–134.

Tomassini, M., Vanneschi, L., Collard, P., and Clergue, M. (2005). A study of fitness distance correlation as a difficulty measure in genetic programming. *Evolutionary Computation*, 13(2):213–239.

Turkey, M. and Poli, R. (2012). An empirical tool for analysing the collective behaviour of population-based algorithms. In Chio, C. D., Agapitos, A., Cagnoni, S., Cotta, C., de Vega, F. F., Caro, G. A. D., Drechsler, R., Ekárt, A., Esparcia-Alcázar, A. I., Farooq, M., Langdon, W. B., Guervós, J. J. M., Preuss, M., Richter, H., Silva, S., Simões, A., Squillero, G., Tarantino, E., Tettamanzi, A., Togelius, J., Urquhart, N., Uyar, S., and Yannakakis, G. N., editors, *EvoApplications*, volume 7248 of *Lecture Notes in Computer Science*, pages 103–113. Springer.

Ultsch, A. and Siemon, H. P. (1990). Kohonen's Self Organizing Feature Maps for Exploratory Data Analysis. In *Proceedings of International Neural Networks Conference (INNC)*, pages 305–308, Paris. Kluwer Academic Press.

Vanneschi, L. (2007). Investigating problem hardness of real life applications. In Riolo, R. L., Soule, T., and Worzel, B., editors, *Genetic Programming Theory and Practice V*, Genetic and Evolutionary Computation, chapter 7, pages 107–125. Springer, Ann Arbor.

Vanneschi, L., Clergue, M., Collard, P., Tomassini, M., and Vérel, S. (2004). Fitness clouds and problem hardness in genetic programming. In Deb, K., Poli, R., Banzhaf, W., Beyer, H.-G., Burke, E., Darwen, P., Dasgupta, D., Floreano, D., Foster, J., Harman, M., Holland, O., Lanzi, P. L., Spector, L., Tettamanzi, A., Thierens, D., and Tyrrell, A., editors, *Genetic and Evolutionary Computation – GECCO-2004, Part II*, volume 3103 of *Lecture Notes in Computer Science*, pages 690–701, Seattle, WA, USA. Springer-Verlag.

Vose, M. and Liepins, G. (1991). Punctuated equilibria in genetic search. *Complex Systems*, 5:31–44.

Whitley, D. and Watson, J. P. (2005). Complexity theory and the no free lunch theorem. In Burke, E. K. and Kendall, G., editors, *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, chapter 11, pages 317–339. Springer US.

Witt, C. (2006). Runtime analysis of the (μ + 1) ea on simple pseudo-boolean functions. *Evolutionary Computation*, 14(1):65–86.

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

Wright, A. H. (1990). Genetic algorithms for real parameter optimization. In Rawlins, G. J. E., editor, *FOGA*, pages 205–218. Morgan Kaufmann.

Appendix

A EAs Parameter Settings, Naming Convention, Techniques and Benchmark Problems

In this work, we used generational evolutionary algorithms with four population handling techniques: standard, fitness sharing, deterministic crowding and random immigrants. Except deterministic crowding that uses its own selection mechanism, we used tournament selection with different tournament sizes to provide different levels of selection pressure. Table 2 summarises the common parameter settings among all the EAs that have been used in our experiments.

We used a naming standard for EAs to capture four facts about an algorithm: population handling technique, crossover method, crossover control parameter and tournament size (selection pressure). The algorithm name has the format **PPCCCC-r-t** where **PP** denotes population handling technique as follows: **St**: Standard EA, **FS**: Fitness Sharing EA, **Cr**: Deterministic Crowding EA and **RI**: Random Immigrant EA. **CCCC** refers to the crossover method, **Arth**: Arithmetic Crossover, **BLX**: Blending Crossover and **Heur**: Heuristic Crossover. **r** is a real number representing the control parameter of the crossover method used and **t** is an integer representing the tournament size. For example, FSArth-0.4-4 is an EA using fitness sharing and the arithmetic crossover with control parameter 0.4 and tournament size 4.

Table 2: Parameter settings of the evolutionary algorithms used in our experiments

Parameter	Value
Population size	100
Crossover rate	0.75
Mutation rate	0.2
Mutation step	[0, 1.0]
Individual length	5
Total number of evaluation per run	10,000

In addition to standard generational EA, we used three population handling techniques, they are: **i) Fitness sharing** (Goldberg and Richardson, 1987) is a fitness scaling mechanism that alters only the fitness evaluation stage of an EA and can be used with any other techniques. In this method, similar individuals (located in the vicinity of each other on the fitness landscape) have to share the fitness among them. Thus, the number of individuals that can reside in an area is limited by the fitness of that area. A full description can be found in (Back et al., 1997). **ii) Crowding techniques** (DeJong, 1975) insert new individuals into the population by replacing similar individuals. However, replacement errors may prevent crowding technique from maintaining individuals on different but nearby peaks. Deterministic crowding (Mahfoud, 1992, 1995) is designed to minimise the replacement error. A full description and of the algorithm and a pseudocode can be found in (Back et al., 1997). **iii) Random immigrants mechanism** (Cobb and Grefenstette, 1993) replaces part of the population with randomly created individuals. In this work, we replace 10 per cent of the worst individuals in population by random ones. We perform this replacement at the end of every EA round.

We tested the algorithms on 5-dimensional real-valued optimisation problems obtained by composing a random combination of 10 basic functions taken from the following family: *Sphere, Ranstrigin and Griewank*. These functions were randomly generated, shifted, rotated and combined. In our experiments, we conducted 100 runs for each algorithm settings (e.g., crossover method and its control parameter, tournament size ...etc.). Each run operated on a random fitness landscape. Full details on the benchmark functions and how to compose them can be found in (Liang et al., 2005).

B Crossover Operators for Real-coded Evolutionary Algorithms

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) (Michalewicz, 1996): 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) (Eshelman and Schaffer, 1992): Two offspring individuals are produced by randomly (uniformly) generating values for their 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)(Herrera and Lozano, 1996; Wright, 1990): This method that 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 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 $C_2 = P_1 + \lambda \times (P_2 - P_1)$.