Continuous Optimisation Theory Made Easy? Finite-element Models of Evolutionary Strategies, Genetic Algorithms and Particle Swarm Optimizers

R. Poli[1], W. B. Langdon[2], M. Clerc[3], and C. R. Stephens[4]

[1] Department of Computer Science, University of Essex, UK
[2] Independent Consultant, Groisy, France
[3] Instituto de Ciencias Nucleares, UNAM, México

Abstract. We propose a method to build discrete Markov chain models of continuous stochastic optimisers that can approximate them on arbitrary continuous problems to any precision. We discretise the objective function using a finite element method grid which produces corresponding distinct states in the search algorithm. Iterating the transition matrix gives precise information about the behaviour of the optimiser at each generation, including the probability of it finding the global optima or being deceived. The approach is tested on a (1+1)-ES, a bare bones PSO and a real-valued GA. The predictions are remarkably accurate.

1 Introduction

Markov chains are important in the theoretical analysis of evolutionary algorithms operating on discrete search spaces. So far they have been of little use for EAs searching on continuous spaces. Naturally, Markov chains with continuous state spaces can be defined and powerful general results have been obtained using them [14]. However, the complexity of the calculations involved makes them less than ideal for the detailed theoretical analysis of continuous optimisers. Instead a variety of different tools have been used. Despite these, generally making theoretical progress in the continuous domain is extremely difficult. As a result, while there are a few continuous domain optimisers, such as evolutionary strategies [2], for which we have a reasonably clear mathematical understanding, in most other cases, the reasons why an algorithm works (or does not) are totally unclear. For example, how differential evolution [16] works is considered by most to be a mystery. In other cases, detailed models of only some components of an algorithm are available, as in the case of genetic algorithms applied to continuous functions [12].

Even where substantial theoretical progress has been made, this has virtually always required either working at a highly abstract level or considering in detail very special cases. For example, in evolutionary strategies, most theory has been restricted to the class of sphere functions. While in the case of particle swarm optimisers, the theory available (see for example [13, 5, 19, 18]) assumes: isolated single individuals, the search stagnates (i.e., no improved solutions are found) and, until very recently, even that there is no randomness. (In general none of these are true.) So, there is a large gap between theory and practice for continuous optimisers.

We suggest an idea which has the potential to radically improve the situation. It is general and can be applied to most continuous optimisers and arbitrary fitness functions. The inspiration has come from the Finite Element Method. FEM has been very successfully used to model continuous systems in a variety of disciplines [3]. It divides continuous systems into elements. These are sufficiently small that the behaviour of each can safely be modelled by a numerically simple function and so the whole system is accurately modelled by simply combining all its elements. Naturally, the accuracy of the results depends on the resolution of the mesh of elements used (which can be different for different parts of the system). When the mesh is fine enough, the analysis can be extremely accurate.

We discretise the system (in our case, the optimisation algorithm and the fitness function) and then study the dynamics of the discretised system (see Section 2). The new system is in one of a finite number of states. Crucially we will assume the optimisation algorithm's future behaviour can be captured by the current state. This is true of most evolutionary algorithms, which only depend on the current population and not on older populations. Hence we can model the new system as a Markov chain. By studying the chain we can then learn about the behaviour of the original (continuous) system. As we will see there is a notion of resolution. If the discretisation mesh is chosen appropriately, the accuracy with which the chain models the continuous system, over many generations, sometimes even for quite coarse grids can be remarkable.

This gives us an effective general technique to produce discrete Markov chain models of continuous stochastic optimisers that can approximate them on continuous problems to arbitrary precision and for arbitrary fitness functions. The model is complete and includes the ability to estimate arbitrary statistics, such as the evolution of average fitness, best fitness and population diversity. In particular, it is very easy to estimate the probability of an optimiser finding global optima or being deceived.

Our objective is to introduce the idea and to provide a proof of concept for it. So, we will apply the approach to a small, but diverse set of optimisers – (1+1) Evolutionary Strategies (Section 3), Particle Swarm Optimisers (Section 4) and real-valued Genetic Algorithms(Section 5) – and show how easily we can estimate important properties such as the probability of finding the global optimum and the expected runtime of each algorithm (Section 6). We will consider four different problems and will compare the behaviour of the resulting chains with actual runs (Section 7). We postpone the analysis and comparison of the resulting models. We draw some conclusions in Section 8.

2 Discretisation

We can obtain a discrete model of a continuous optimiser in two ways. Firstly, we can perform a formal FEM *approximation of the exact Markov chain representing the optimiser* over continuous search/state-spaces. We illustrate this approach in Section 2.1. Secondly, we can construct a discrete approximation of the optimiser and then obtain an *exact model for an approximation of the optimiser* as shown in Section 2.2. As we will see, when piecewise constant functions are used in FEM, the two approaches lead to the same type of model: a discrete Markov chain.

2.1 FEM approximation of exact Markov chain

Rudolph's EA model Let us recall the key elements of the generic model of EA presented by Rudolph in [14]. In this model the EA is seen as a homogeneous Markov chain $(X_t : t \geq 0)$ on a probability space $(\varPhi, \mathcal{F}, \mathsf{P})$ with image space $(E, \mathcal{A})$, where $\varPhi$ is the set of outcomes, $\mathcal{F}$ is the set of events (subsets of $\varPhi$) and P is a probability measure. Formally $\mathcal{F}$ must be a σ-algebra over $\varPhi$, i.e., it must be closed under complementation and countable unions of its members, and $\mathsf{P} : \mathcal{F} \to [0,1]$ must be a measure and $\mathsf{P}(\varPhi) = 1$. The set E is the state space for the system, while $\mathcal{A}$ is a σ-algebra over E. Since an EA consists of a population of N individuals represented by the N-tuple $(x_1, \cdots, x_N)$, where the x_i belong to some domain $\mathcal{M}$ (e.g., $\mathcal{M} = \mathbb{R}$) for $i = 1, \cdots, N$, typically the state space is $E = \mathcal{M}^N$, but there are more complex cases.

In Rudolph's EA model the probabilistic modifications on the population caused by the genetic operators are represented by a stochastic kernel $\mathsf{K}(.,.)$. The map $\mathsf{K} : E \times \mathcal{A} \to [0,1]$ is termed a Markovian kernel for the chain if $\mathsf{K}(., A)$ is measurable for any fixed set $A \in \mathcal{A}$ and $\mathsf{K}(x, .)$ is a probability measure on $(E, \mathcal{A})$ for any fixed state $x \in E$. In particular, $\mathsf{K}(x_t, A) = \mathsf{P}\{X_{t+1} \in A | X_t = x_t\}$.

The t-th iteration of the Markovian kernel given by

$$\mathsf{K}^{(t)}(x, A) = \begin{cases} \mathsf{K}(x, A) & \text{if } t = 1, \\ \int_E \mathsf{K}^{(t-1)}(y, A)\mathsf{K}(x, dy) & \text{if } t > 1, \end{cases} \tag{1}$$

describes the probability of the EA's state being in some set $A \subseteq E$ within t steps when starting from the state $x \in E$, i.e., $\mathsf{K}^{(t)}(x, A) = \mathsf{P}\{X_t \in A | X_0 = x\}$.

Let $\pi(.)$ denote the initial distribution over subsets A of $\mathcal{A}$, e.g., the probability distribution for the initial population at step $t = 0$. Then

$$\mathsf{P}\{X_t \in A\} = \begin{cases} \pi(A) & \text{if } t = 0, \\ \int_E \mathsf{K}^{(t)}(y, A)\pi(dy) & \text{if } t > 0. \end{cases} \tag{2}$$

FEM applied to Rudolph's model The starting point of FEM is the definition of a mesh and the assumption that the solution to the problem can be expressed as a piecewise linear, quadratic, or higher order function over the mesh.

There is no limitation as to the simplicity (or complexity) of the elements. They can even be constant. This is the type of elements we will use here, although one could extend the results to the case of more sophisticated elements.

In the case of an EA or other stochastic optimiser exploring a continuous search/state space, the function $\mathsf{P}\{X_t \in A\}$ in (2) provides a full probabilistic description of the system. This is, however, clearly a function of the kernel $\mathsf{K}^{(t)}(x, A)$ in (1). We will therefore take the latter as the solution of our problem that we will represent by finite elements.

Let us assume that E is divided up into n disjoint sets E_i such that $E = \bigcup_i E_i$. These represent our mesh. We assume that the family of functions $\mathsf{K}^{(t)}(x, A)$ is *piecewise constant* on each element, i.e., $\forall t, \forall i, \forall x', x'' \in E_i, \forall j$: $\mathsf{K}^{(t)}(x', E_j) = \mathsf{K}^{(t)}(x'', E_j)$.

Let us now focus on A's which are obtained as the union of *some* E_i, i.e., $A = \bigcup_{i \in I} E_i$, where $I \subseteq \{1, \cdots, n\}$. Then we can write $\mathsf{P}\{X_t \in A\} = \sum_{i \in I} \mathsf{P}\{X_t \in E_i\}$. We can, therefore, focus our attention on the quantities $\mathsf{P}\{X_t \in E_i\}$. For $t > 0$, from (2) we obtain

$$\begin{aligned}
\mathsf{P}\{X_t \in E_i\} &= \int_E \mathsf{K}^{(t)}(y, E_i)\pi(dy) \\
&= \sum_j \int_{E_j} \mathsf{K}^{(t)}(y, E_i)\pi(dy) \\
&= \sum_j \int_{E_j} \mathsf{K}^{(t)}(y_j, E_i)\pi(dy) \\
&= \sum_j \mathsf{K}^{(t)}(y_j, E_i)\pi(E_j)
\end{aligned}$$

where y_j is any representative element of the set E_j (e.g., its centroid, if the set is compact). From (1) we obtain

$$\begin{aligned}
\mathsf{K}^{(t)}(y_j, E_i) &= \int_E \mathsf{K}^{(t-1)}(y, E_i)\mathsf{K}(y_j, dy) \\
&= \sum_n \int_{E_n} \mathsf{K}^{(t-1)}(y, E_i)\mathsf{K}(y_j, dy) \\
&= \sum_n \int_{E_n} \mathsf{K}^{(t-1)}(y_n, E_i)\mathsf{K}(y_j, dy) \\
&= \sum_n \mathsf{K}^{(t-1)}(y_n, E_i) \int_{E_n} \mathsf{K}(y_j, dy) \\
&= \sum_n \mathsf{K}^{(t-1)}(y_n, E_i)\mathsf{K}(y_j, E_n)
\end{aligned}$$

If M is a matrix with elements $m_{ij} = \mathsf{K}(y_j, E_i)$ we have that $\mathsf{K}^{(t)}(y_j, E_i)$ is the (i, j)-th of M^t and $\mathsf{P}\{X_t \in E_i\}$ is the i-th element of $M^t p$, where p is a vector whose elements are $\pi(E_i)$. That is, *the FEM approximation with order-0 elements to Rudolph's exact EA model is an ordinary discrete Markov chain.*

2.2 Exact Markov model of approximate optimiser

An alternative to using FEM is to first obtain a discrete optimiser whose behaviour strongly resembles the behaviour of the original (continuous) optimiser, and then use standard-type Markov chain theory to model such an optimiser. This approach effectively stands to the previous as the finite difference method (FDM) stands to FEM. FDM is a method for integrating differential equations. The difference between FEM and FDM is that, while in FEM one approximates the solution to a problem, in FDM one discretises the equations of motion of the system. However, it is well known that in certain conditions, e.g., when using piecewise constant functions like we did in Section 2.1, the two methods coincide. Because of its simplicity in the remainder of the paper we will use the second approach.

Fitness Function f Discretisation We partition a continuous N-dimensional search space Ω into a finite number (n) of compact non-overlapping sub-domains Ω_i. We give each sub-domain Ω_i a fitness value f_i, which can be computed as the mean of f over Ω_i, i.e., $f_i = \int_{\Omega_i} f(x)dx / \int_{\Omega_i} dx$, or simply $f_i = f(x_{c_i})$ where x_{c_i} is the centroid of cell i, i.e., $x_{c_i} = \int_{\Omega_i} xdx / \int_{\Omega_i} dx$. We will call the pair $S_i = (\Omega_i, f_i)$ a *plateau*. So, effectively we turn our continuous fitness function into a piecewise-constant function, which looks like a multidimensional histogram.

If, for example, we consider the case where Ω is a N-dimensional cube which we partition using a regular grid of hypercubic cells, then we can represent each sub-domain as

$$\Omega_i = [x_{c_{i_1}} - r, x_{c_{i_1}} + r] \times [x_{c_{i_2}} - r, x_{c_{i_2}} + r] \times \cdots \times [x_{c_{i_N}} - r, x_{c_{i_N}} + r] \tag{3}$$

where r is the cell "radius" and $x_{c_{i_j}}$ is the j-th component of a lattice point x_{c_i} (the centroid of each sub-domain). So, when r is known and fixed, we can simply represent each plateau using its centroid and fitness value. That is $S_i = (x_{c_i}, f_i)$. Figure 1 shows two one-dimensional fitness functions and two corresponding piecewise-constant functions obtained by discretising them with $r = 0.25$.

Naturally, the choice of the mesh is crucial in determining the accuracy of the resulting model. Clearly, the finer the grid, the more accurate the results. However, also the method with which the fitness of plateaus is computed is important. When these are computed with $f_i = f(x_{c_i})$, as we will do in the rest of the paper, we run the risk of missing important landscape features of sub-element size. This is less likely when f_i is mean of f over Ω_i. The disadvantage of this method is that one needs to compute integrals of the fitness function.

Algorithm Discretisation Most optimisers store information about one or more points in Ω which are used to determine which areas of the search space to sample next. Let us assume that there are P such points, which we will call a population. The population is the state of the optimiser. To discretise the optimiser we need to discretise its population s so that instead of taking continuous values it can only take a finite number of states. We use the same discretisation mesh $\{\Omega_i\}$ as for the fitness function. So, in the discretised optimiser the population s is in one of the states $\{x_{c_i}\}^P$.

Some optimisers use additional variables and parameters to control the search, and these are often adapted dynamically. E.g., an ES may change the mutation strength, while the velocities of the particles change in a PSO. When these quantities adapt during the search they are part of the state of the algorithm. Hence they must also be discretised but the lattice used will depend upon the algorithm.

We should note that both discretisation methods discussed above assume that the search is contained within a finite domain Ω (typically a multidimensional box). This is what most problems require and what many optimisers do. However, some optimisers have unbounded state spaces. So, one cannot be sure whether a certain state variable will stay permanently in pre-defined bounds. There are many strategies to circumvent this problem. One could, for example, use boundary elements of infinite size (but with an artificial, finite, centroid), or use mapping/squashing functions to map an infinite space into a new, finite one. These and other strategies put forward in the FEM community (e.g., [1, 7, 6, 17]), however, are beyond the scope of this article.

3 Evolutionary Strategy Model

To start with, let us consider the simplest possible evolutionary strategy: a (1+1)-ES with Gaussian mutations but *without* adaptation of the mutation standard deviation σ.

Naturally, at any given time the only member of the population, x_p, will be located in some sub-domain Ω_i. After discretisation, x_p can only take one of a discrete set of values, namely $x_p = x_{c_k}$ for k in $\{1, \cdots, n\}$. So, our (1+1)-ES can only be in one of n states. We will indicate the state of the ES with an integer s.

Our objective is to model this simple ES as a Markov chain with states of this form. What we need to do is to compute the state transition matrix $M = (m_{ij})$, where m_{ij} is the probability of the ES moving from state i to state j at the next iteration. When M is available, we can compute the probability distribution π_t of the discretised ES being in any particular state at generation t, given its state probability distribution at the start, π_0, from $\pi_t = M^t \pi_0$.

Let $p(x|x_p)$ be the sampling probability density function when the parent is x_p. Normally in an (1+1)-ES random numbers are chosen independently for each dimension when computing mutants. In our discretised ES we do the same thing. So, we have separability of p. That is, p is given by a product of independent probability distributions for each separate dimension:

$$p(x|x_p) = \prod_{j=1}^{N} p(x_j|x_{p_j})$$

where x_j and x_{p_j} are the j-th components of the vectors x and x_p, respectively.[4]

The probability of sampling sub-domain Ω_i is given by

$$\Pr(\Omega_i|x_p) = \int_{\Omega_i} p(x|x_p)\; dx$$

So, if sub-domains Ω_i have the product structure shown in Equation 3, we have

$$\Pr(\Omega_i|x_p) = \prod_j \Pr([x_{c_{i_j}} - r, x_{c_{i_j}} + r]|x_{p_j}) \tag{4}$$

where

$$\Pr([x_{c_{i_j}} - r, x_{c_{i_j}} + r]|x_{p_j}) = \int_{x_{c_{i_j}}-r}^{x_{c_{i_j}}+r} p(x_j|x_{p_j})\; dx_j. \tag{5}$$

The standard sampling distribution used in the ES is a Gaussian distribution, i.e., $p(x_j|x_{p_j}) = G\left(x_{p_j}, \sigma\right)$ with $G(\mu,\sigma) = \frac{1}{\sqrt{2\pi}\,\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$. Let erf be the integral of the Gaussian distribution. Therefore,

$$\begin{aligned} &\Pr([x_{c_{i_j}} - r, x_{c_{i_j}} + r]|x_{p_j}) \\ &= \frac{1}{2}\left(\operatorname{erf}\left(\frac{x_{c_{i_j}} + r - x_{p_j}}{\sigma\sqrt{2}}\right) - \operatorname{erf}\left(\frac{x_{c_{i_j}} - r - x_{p_j}}{\sigma\sqrt{2}}\right)\right). \end{aligned} \tag{6}$$

Let us now put the sub-domains in order of their fitness so that $f_i \leq f_j$ for $i < j$. Since the population can only change if there is a fitness improvement, only certain state transitions can occur. That is, a transition from state s to state s' is possible only if $s \leq s'$.

Suppose the parent is in domain k, then the probability of it changing to domain l is given by:

$$\Pr(l|k) = \begin{cases} \Pr(\Omega_l|x_{c_k}) & \text{if } l \text{ and } k \text{ are such that } f_l > f_k \text{ (NB: } f_l > f_k \implies l > k \text{ but not } \textit{vice versa}\text{)}, \\ 0 & \text{if } k \neq l \text{ and } f_l \leq f_k, \\ 1 - \sum_{l: f_l > f_k} \Pr(l|k) & \text{if } l = k \text{, to guarantee the conservation of probability.} \end{cases}$$

This effectively means that the population remains in domain k if any of the following three conditions is met: (a) the new sample is in Ω_k, (b) the new sample is in an Ω_j (different from Ω_k) with $f_j \leq f_k$, or (c) the sample is outside Ω. So, we can then write the state transition probability for the ES as

$$m_{s,s'} = \Pr(s'|s) = \Pr(\Omega_{s'}|x_{c_s})\delta(f_{s'} > f_s) + (1 - \sum_{l: f_l > f_s} \Pr(\Omega_l|x_{c_s}))\delta(s' = s), \tag{7}$$

where $\Pr(\Omega_{s'}|x_{c_s})$ and $\Pr(\Omega_l|x_{c_s})$ can be computed using Equations 4 and 6. The function $\delta(z)$ returns 1 if z is true and 0 otherwise.

[4] Note that, while separability of the sampling distribution makes the model's calculations simpler, it is not a requirement.

As an example, consider a domain $\Omega = [-2,2] \times [-2,2] = [-2,2]^2$, and let us divide it into four squared sub-domains $\Omega_1 = [-2,0)^2$, $\Omega_2 = [-2,0) \times [0,2]$, $\Omega_3 = [0,2] \times [-2,0)$, and $\Omega_4 = [0,2]^2$ of radius $r = 1$. These have centroids $x_{c_1} = (-1,-1)$, $x_{c_2} = (-1,1)$, $x_{c_3} = (1,-1)$ and $x_{c_4} = (1,1)$. Let us further assume that the fitness function f takes the following values at the centroids: $f_1 = 1$, $f_2 = 2$, $f_3 = 3$, and $f_4 = 4$. Then by applying the equations above, for $\sigma = 1$, we obtain the transition matrix:

$$M = \begin{pmatrix} 0.9499 & 0.0000 & 0.0000 & 0.0000 \\ 0.0232 & 0.9731 & 0.0000 & 0.0000 \\ 0.0232 & 0.0037 & 0.9768 & 0.0000 \\ 0.0037 & 0.0232 & 0.0232 & 1.0000 \end{pmatrix}.$$

By iterating the corresponding chain one can compute the distribution of states of our discretised fixed-σ ES acting on fitness function f at any generation. However, given the very low resolution chosen, we would not expect the predictions of the chain to exactly reflect the real behaviour of the continuous optimiser. As we will see later, however, with higher resolutions, predictions can be very accurate.

Let us now generalise this model to include a more interesting version of (1+1)-ES: one where σ adapts during evolution. To keep our description simple, we will focus on an adaptive scheme which updates σ at each iteration [2, page 84]. If the offspring produced by mutation is better than its parent (and is in Ω) we increase σ according to the rule $\sigma' = \sigma c$ where c is a suitable constant > 1. If, the offspring is invalid or is not better than the parent, we reduce σ using the rule $\sigma' = \sigma / c$.

Naturally, for this new ES we can still discretise the parent individual using the regular mesh adopted for the fitness function, as we did for the fixed-σ case. However, we will use a non-uniform discretisation for σ. Indeed, it is apparent that σ can only take discrete values already, all σ's being of the form $\sigma = \sigma_0 \cdot c^i$ for some integer i, where σ_0 is the value of σ at generation 0. So, in any finite run of G generations, $\sigma \in \{\sigma_0 \cdot c^{-G}, \sigma_0 \cdot c^{-G+1}, \cdots, \sigma_0 \cdot c^{G}\}$, that is it can only take $2G+1$ different values. Following standard practice, in our ES we will limit σ so that it never becomes too little or too big. This effectively means that we can use a smaller range $\{\sigma_0 \cdot c^{-Z}, \sigma_0 \cdot c^{-Z+1}, \cdots, \sigma_0 \cdot c^{Z}\}$, with $Z < G$. So, we can represent the state of the ES with the tuple (s_1, s_2), where $s_1 \in \{1, \cdots, n\}$ represents the position of the parent and $s_2 \in \{-Z, \cdots, Z\}$ gives the mutation σ used to create its child. For the purpose of indexing the elements of the array M, we then convert tuples into natural numbers by using the odometer ordering, whereby $(1, -Z)$ maps to 1, $(1, -Z+1)$ maps to 2, etc. (i.e., $(s_1, s_2) \mapsto (2Z+1)s_1 + s_2 - Z$).

The calculations to compute M for the adaptive ES are based on the application of Equation 7, with minor changes. Firstly, when we compute the probability of a transition from state $s = (s_1, s_2)$ to state $s' = (s'_1, s'_2)$, we use the σ corresponding to s_2, i.e., $\sigma = \sigma_0 \cdot c^{s_2}$. Secondly, for all state pairs where $s_1 < s'_1$ (there was a fitness improvement) but where $s'_2 \neq s_2 + 1$ (σ was *not* increased according to our update rule), we know that $m_{s,s'} = 0$, so we don't apply Equation 7. Likewise, for all state pairs where $s_1 \geq s'_1$ and where $s'_2 \neq s_2 - 1$.

4 Particle Swarm Optimisation Model

4.1 Background

Particle Swarm optimisers (PSOs) [8, 10] have been with us a few years. However it is fair to say that most work on PSOs has been experimental confirmations of their effectiveness, extensions to new applications or new algorithms. With very few exceptions (e.g., see the dynamical system model in [5, 18] or the probabilistic stagnation analysis in[4]), analytical, theoretical and mathematical analysis of them is still relatively unexplored.

In a simple PSO, the swarm consists of a population of identical particles which move across a problem landscape looking for high-fitness regions. The particles have momentum and are accelerated by forces applied to them. The PSO's integration of Newton's laws of motion is discrete and the particles only sample the fitness landscape at discrete time steps. Thus the PSO particles draw samples from the search space only at some points in their trajectories. In the classic PSO, there are two attractive forces. The first pulls the particle towards the best point it personally has ever sampled, whilst the second pulls it towards the best point seen by any particle in its neighbourhood. The strengths of the various forces are randomly controlled. It is the stochastic nature of the PSO which allows it to effectively explore and ensures that the loci of the particles are not closed trajectories. Instead, the particles randomly sample the region nearby and between the particle's own best and the swarm best.

One of the recent advances has been Jim Kennedy's "Bare Bones" PSO (BB-PSO) [9]. This optimiser is inspired by the observation that, at least until a better location in the search space is sampled, the pseudo chaotic particle orbits can be approximated by a fixed probability distribution centred on the point lying halfway between the particle best and the swarm best. Its width is modulated by the distance between them. The exact nature of the distribution is not clear: it is bell shaped like a Gaussian distribution [11] but the tails appear to be heavier, like a Cauchy distribution. The essential "bare bones" PSO, cuts out the integration needed to find each particle's position, and instead draws it from a random distribution. This means we no longer need to track exactly each particle's position and velocity. As with other swarm intelligence techniques, there has been little theoretical work on this essential PSO. The model we are about to present addresses this.

4.2 Model of "bare bones" PSO

Let us consider a fully-connected bare bones PSO to start with. In this PSO the particles have no dynamics, but simply sample the neighbourhood of their personal best and swarm best using a fixed probability density function. This continues until either their personal best or the swarm best is improved. When this happens, the parameters of the sampling distribution are recomputed and the process is restarted.

In the unlikely event that more than one particle's personal-best fitness is the same as the best fitness seen so far by the whole swarm, we assume that swarm leadership is shared. That is, each particle chooses as its swarm best a random individual out of the set of swarm bests.

Naturally, at any given time the personal best for each particle and the swarm best will be located in some sub-domain Ω_i. In a discretised BB-PSO both the particle best x_p and swarm best x_s can only take one of a discrete set of values, namely $x_p = x_{c_k}$ and $x_s = x_{c_j}$ for some j and k in $\{1, \cdots, n\}$. So, the discretised algorithm can only be in a finite set of states. However, we don't need to represent explicitly the swarm best, since the information is implicit in the fitness values f_i associated to each centroid. So, if P is the population size, there are n^P such states – one for each particle's personal best – and we can represent states as P dimensional vectors with integer elements, e.g.

$$s = (s_1, \cdots, s_P).$$

Let us now focus on computing state transition probabilities.

Let $p(x|x_s, x_p)$ be the sampling probability density function when swarm best is x_s and particle best is x_p. The standard sampling distribution used in the BB-PSO is a Gaussian distribution. (Our approach could also be applied to Cauchy or other distributions.) So, we have

$$p(x_j|x_{s_j}, x_{p_j}) = G\left(\frac{x_{s_j} + x_{p_j}}{2}, \left|x_{s_j} - x_{p_j}\right|\right).$$

Note that this distribution becomes a Dirac delta function when $x_{s_j} = x_{p_j}$. Normally in PSOs random numbers are chosen independently for each dimension when computing force vectors. In a bare bones PSO we do the same thing. So, again we have separability of p and we can write

$$p(x|x_s, x_p) = \prod_{j=1}^{N} p(x_j|x_{s_j}, x_{p_j})$$

where x_j, x_{s_j}, and x_{p_j} are the j-th components of the vectors x, x_s and x_p, respectively.

Similarly to the ES case, the probability of sampling domain Ω_i is given by the integral of p across Ω_i, and, if sub-domains Ω_i have the product structure shown in Equation 3, we have

$$\Pr(\Omega_i|x_s, x_p) = \prod_j \Pr([x_{c_{i_j}} - r, x_{c_{i_j}} + r]|x_{s_j}, x_{p_j}) = \prod_j \int_{x_{c_{i_j}}-r}^{x_{c_{i_j}}+r} p(x_j|x_{s_j}, x_{p_j})\, dx_j. \tag{8}$$

For a Gaussian sampling distribution we have

$$\begin{aligned}&\Pr([x_{c_{i_j}} - r, x_{c_{i_j}} + r]|x_{s_j}, x_{p_j})\\ &= \begin{cases} \frac{1}{2}\left(\mathrm{erf}\left(\frac{x_{c_{i_j}}+r-\frac{x_{s_j}+x_{p_j}}{2}}{\left|x_{s_j}-x_{p_j}\right|\sqrt{2}}\right) - \mathrm{erf}\left(\frac{x_{c_{i_j}}-r-\frac{x_{s_j}+x_{p_j}}{2}}{\left|x_{s_j}-x_{p_j}\right|\sqrt{2}}\right)\right) & \text{if } x_{s_j} \neq x_{p_j}, \\ \delta(x_{p_j} \in [x_{c_{i_j}} - r, x_{c_{i_j}} + r]) & \text{otherwise.} \end{cases}\end{aligned}$$

Again, let us order sub-domains so that $f_i \leq f_j$ for $i < j$. Since particle personal bests can only change if there is a fitness improvement, only certain state transitions can occur. That is, a transition from state $s = (s_1, \cdots, s_P)$ to state $s' = (s'_1, \cdots, s'_P)$ is possible only if $s_1 \leq s'_1$, $s_2 \leq s'_2$, etc. We will denote this by $s \leq s'$.

Let us identify the location of the swarm best for a PSO in a state s. Typically in a fully-connected PSO there is only one particle with the best fitness value, but, within a discretised PSO, it is not uncommon to have more than one. So, in general, we have a set of swarm bests:

$$\mathcal{B}(s) = \bigcup_{i:f(s_i)=f_m(s)} \{s_i\}$$

where $f_m(s) = \max_j f_{s_j}$ and $|\mathcal{B}(s)| \geq 1$. More generally, to allow other communication topologies, we need to talk about sets of neighbourhood bests – one set for each particle. We will denote these sets as $\mathcal{B}(s,i)$, for $i = 1, \cdots, P$.

Let us consider a PSO in state s. In a BB-PSO, at each iteration, the particles sample the search space independently. So, if the i-th particle's best is in plateau k (that is, $s_i = k$), then the probability of it changing to plateau l is given by:

$$\Pr(l|\mathcal{B}(s,i),k) = \begin{cases} \frac{1}{|\mathcal{B}(s,i)|}\sum_{b\in\mathcal{B}(s,i)} \Pr(\Omega_l|x_{c_b}, x_{c_k}) & \text{if } l \text{ and } k \text{ are such that } f_l > f_k, \\ 0 & \text{if } k \neq l \text{ and } f_l \leq f_k, \\ 1 - \sum_{l:f_l>f_k} \Pr(l|\mathcal{B}(s,i),k) & \text{if } l = k \text{ (to guarantee the conservation of probability).} \end{cases}$$

Like for the ES case, this effectively means that the particle remains in plateau k if any of the following three conditions is met: (a) the new sample is in Ω_k, (b) the new sample is in an Ω_j (different from Ω_k) with $f_j \leq f_k$, or (c) the sample is outside Ω.

Because of the independence of the particles (over one time step), we can then write the state transition probability for the whole PSO as

$$\begin{aligned} m_{s,s'} &= \prod_i \Pr(s'_i|\mathcal{B}(s,i), s_i) \\ &= \prod_{i:f_{s'_i}>f_{s_i}} \frac{1}{|\mathcal{B}(s,i)|} \sum_{b\in\mathcal{B}(s,i)} \Pr(\Omega_{s'_i}|x_{c_b}, x_{c_{s_i}}) \\ &\times \prod_{i:f_{s'_i}\leq f_{s_i}} \left[\left(1 - \sum_{l:f_l>f_{s_i}} \frac{1}{|\mathcal{B}(s,i)|} \sum_{b\in\mathcal{B}(s,i)} \Pr(\Omega_l|x_{c_b}, x_{c_{s_i}})\right) \delta(s'_i = s_i)\right]. \end{aligned}$$

Naturally, further decompositions can be obtained using Equation 8.

As an example, let us consider again the domain $\Omega = [-2,2]^2$, which we divide into four sub-domains Ω_i of radius $r = 1$, with centroids $x_{c_1} = (-1,-1)$,

$x_{c_2} = (-1, 1)$, $x_{c_3} = (1, -1)$, $x_{c_4} = (1, 1)$ and associated fitness $f_1 = 1$, $f_2 = 2$, $f_3 = 3$, and $f_4 = 4$, respectively.

If the population includes only two particles ($P = 2$), then we have only 16 different states for the PSO: (1,1), (1,2), ..., (4,4). For example, the state $(1, 1)$ represents the situation where both particles are in the lowest plateau (so, both are swarm bests); in state $(1, 2)$ one particle is in the lowest plateau, while the other (the swarm best) is in the second lowest plateau; etc.

By applying the previous equations we can then obtain the following transition matrix:

$$\left(\begin{array}{cccccccccccccccc|c}
(1,1) & (1,2) & (1,3) & (1,4) & (2,1) & (2,2) & (2,3) & (2,4) & (3,1) & (3,2) & (3,3) & (3,4) & (4,1) & (4,2) & (4,3) & (4,4) & \\
\hline
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (1,1) \\
0 & 0.659 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (1,2) \\
0 & 0 & 0.659 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (1,3) \\
0 & 0 & 0 & 0.651 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (1,4) \\
0 & 0 & 0 & 0 & 0.659 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (2,1) \\
0 & 0.341 & 0 & 0 & 0.341 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (2,2) \\
0 & 0 & 0 & 0 & 0 & 0 & 0.766 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (2,3) \\
0 & 0 & 0 & 0.117 & 0 & 0 & 0 & 0.659 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (2,4) \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.659 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (3,1) \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.766 & 0 & 0 & 0 & 0 & 0 & 0 & (3,2) \\
0 & 0 & 0.341 & 0 & 0 & 0 & 0.117 & 0 & 0.341 & 0.117 & 1 & 0 & 0 & 0 & 0 & 0 & (3,3) \\
0 & 0 & 0 & 0.117 & 0 & 0 & 0 & 0 & 0 & 0.117 & 0 & 0.659 & 0 & 0 & 0 & 0 & (3,4) \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.651 & 0 & 0 & 0 & (4,1) \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.117 & 0.659 & 0 & 0 & (4,2) \\
0 & 0 & 0 & 0 & 0 & 0 & 0.117 & 0 & 0 & 0 & 0 & 0 & 0.117 & 0 & 0.659 & 0 & (4,3) \\
0 & 0 & 0 & 0.117 & 0 & 0 & 0 & 0.341 & 0 & 0 & 0 & 0.341 & 0.117 & 0.341 & 0.341 & 1 & (4,4)
\end{array}\right)$$

where we have added one extra row and column to more clearly identify states.

5 Real-valued Genetic Algorithm Model

We consider a simple real-valued GA with finite population, fitness proportionate selection, no mutation, and 100% recombination. Recombination produces the offspring, $o = (o_1, \cdots, o_n)$, by sampling uniformly at random within the hyper-parallelepiped defined by the parents, $p' = (p'_1, \cdots, p'_N)$ and $p'' = (p''_1, \cdots, p''_N)$. That is, $o_i = \rho_i(p''_i - p'_i) + p'_i$, where ρ_i is a uniform random number in $[0, 1]$ for $i = 1, \cdots, N$. We will refer to this type of recombination as blend crossover.

We use the same state representation as for BB-PSO, $s = (s_1, \cdots, s_P)$, except that we interpret each s_i as the position of an individual in the search space, rather than a particle's best.

The (offspring) sampling distribution for parents p' and p'' under blend recombination is

$$p(o|p', p'') = \prod_i p(o_i|p'_i, p''_i)$$

where

$$p(o_i|p'_i, p''_i) = \begin{cases} 1/|p'_i - p''_i| & \text{if } o_i \in [\min(p'_i, p''_i), \max(p'_i, p'_i)], \\ 0 & \text{otherwise.} \end{cases}$$

Note that the sampling distribution becomes a Dirac delta function when $p' = p''$.

As before, the probability of sampling domain Ω_i is given by the integral of p across Ω_i. So, for sub-domains Ω_i as in Equation 3, we have

$$\Pr(\Omega_i|p',p'') = \prod_j \int_{x_{c_{i_j}}-r}^{x_{c_{i_j}}+r} p(o_j|p'_j,p''_j)\; do_j \tag{9}$$

where

$$\int_{x_{c_{i_j}}-r}^{x_{c_{i_j}}+r} p(o_j|p'_j,p''_j)\; do_j$$
$$= \begin{cases} \max\left(0, \frac{\min(x_{c_{i_j}}+r,\max(p'_j,p''_j))-\max(x_{c_{i_j}}-r,\min(p'_j,p''_j))}{|p'_j-p''_j|}\right) & \text{if } p'_j \neq p''_j, \\ \delta(p''_j \in [x_{c_{i_j}}-r, x_{c_{i_j}}+r]) & \text{otherwise.} \end{cases}$$

By adding the contributions from all possible pairs of parents (with their selection probabilities) we can now compute the total probability that the offspring will sample domain Ω_i in a particular population $\mathcal{P} = (p_1,\cdots,p_P)$:

$$\Pr(\Omega_i|\mathcal{P}) = \sum_{p'\in\mathcal{P}} \sum_{p''\in\mathcal{P}} \Pr(\Omega_i|p',p'')\phi(p')\phi(p'') \tag{10}$$

where $\phi(x)$ is the selection probability of parent x in population $\mathcal{P}$. For fitness proportionate selection $\phi(x) = f(x)/\sum_{y\in\mathcal{P}} f(y)$.

Naturally, when $\mathcal{P}$ is the population associated to state s, Equation (10) gives us the probability, $\Pr(\Omega_i|s)$, of generating an individual in domain i for a population in state s. Because each individual in a population is generated by an independent Bernoulli trial, we can then trivially compute the Markov chain transition probability from any state s to any state s' as

$$m_{s,s'} = \prod_i \Pr(\Omega_{s'_i}|s).$$

6 Success probability and expected run time of continuous optimisers

As mentioned above, when M is available, we can compute the probability distribution π_t of a discretised continuous optimiser being in any particular state at generation t, given its state probability distribution at the start, π_0, from $\pi_t = M^t\pi_0$. Since for each optimiser we know what π_0 is, to compute the probability with which the element containing the global optimum is visited at a particular generation t, one only needs to add up the appropriate components of the π_t vector. We will informally call this quantity the *success probability*. For example, in an ES with fixed σ we have that the components of π_0 are all

$1/n$ and the success probability is simply given by the last component of π_t (assuming domains are ordered by fitness).

We can also estimate the *expected run time* of continuous optimisers by computing the expected waiting time of the corresponding discrete Markov chain to visit a particular target state or set of states J. Following [15, pages 168–170] we have that the *mean passage time* for going from state i to the set of states J, given that it is currently outside the set is given by:

$$\eta_{i,J} = \sum_{j \in J} m_{i,j} + \sum_{k \notin J} m_{i,k}(1 + \eta_{k,J}) \tag{11}$$

where $m_{i,j}$ are the elements of Markov matrix for the system. Simple algebraic manipulations of (11) lead to the following system of simultaneous equations:

$$\eta_{i,J} - \sum_{k \notin J} m_{i,k}\eta_{k,J} = 1 \tag{12}$$

Once solved, we can then compute the *expected waiting time* to reach state J, given a random initial state (described by the distribution $p(.)$) as

$$EWT_J = \sum_{i \notin J} p(i)\eta_{i,J}. \tag{13}$$

If the calculation is applied to an initial distribution where all states are equally likely (the standard initialisation strategy in EAs) and with J being the element containing the global optimum, we have

$$E[\text{runtime}] = \frac{\sum_{i \notin J} \eta_{i,J}}{\text{number of elements} - 1}. \tag{14}$$

Note that this calculation assumes that the algorithm has a way of identifying when the element containing the global optimum is sampled and stops when this happens. Often this is not the case. In this case, (14) should be interpreted as the average first hitting time.

7 Experimental Results

We first apply the Markov chain models described in the previous sections to the two one-dimensional ($N = 1$) fitness functions in Figure 1. These are effectively continuous versions of the onemax problem (Ramp) and the deceptive trap function. The results of these tests are reported in Sections 7.1– 7.3. Then, in Section 7.4, we study two-dimensional problems.

7.1 Evolutionary Strategies

In a series of experiments we applied the model of the (1+1)-ES with fixed-σ and compared its behaviour with the behaviour of the real algorithm acting

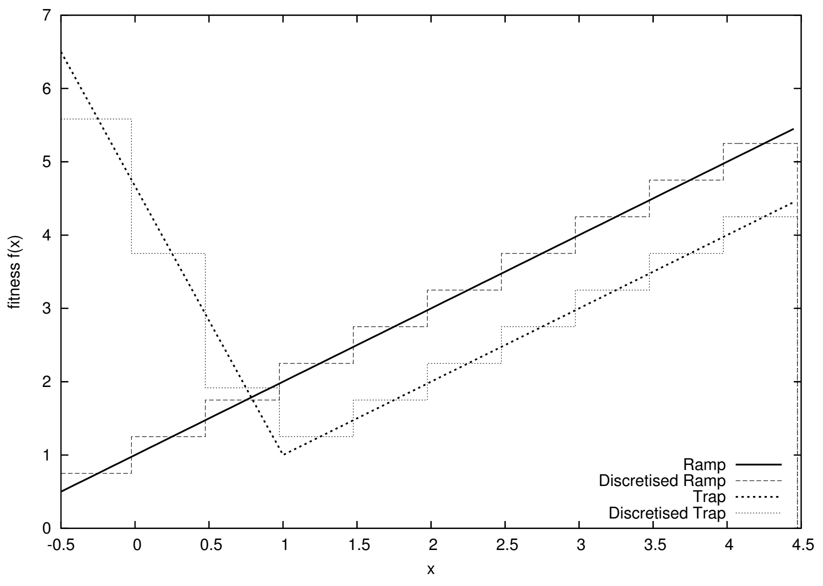

Fig. 1. Ramp and Deceptive test functions and two corresponding piece-wise constant discretisations.

on the Ramp and Deceptive continuous fitness functions. We chose the domain $\Omega = [-0.5, 4.5)$. This was divided into n sub-domains $\Omega_i = [x_{c_i} - r, x_{c_i} + r)$ with equally spaced centroids x_{c_i}. To assess the behaviour of the algorithm in real runs we performed 1,000 independent runs, where, in each run and each generation, we recorded the sub-domain Ω_i occupied by the parent. In particular we were interested in comparing the proportion of runs in which the individual was in the domain containing the global optimum Ω_n vs. the frequency predicted by the Markov chain over a number of generations.

Figure 2 shows the results of the comparison for different values of σ and for the case of $n = 10$ ($r = 0.25$), i.e., where we discretise the evolutionary strategy using only 10 states. As the figure indicates, as long as σ is bigger than the cell width, $2r$, the model predicts the success rate with considerable accuracy throughout our runs (50 generations). When σ is comparable to r, there are errors of up to around 10% in the prediction. Similar accuracies were obtained for the deceptive fitness function (see Figure 3). In all cases, despite its tiny size, the chain was able to predict that Deceptive is harder than Ramp.

To illustrate how one can use our Markov model to study how the computational complexity of an algorithm varies as the parameter σ varies and as a function of the fitness function we also computed (as described in Section 6) the expected first hitting time for the global optimum for the Ramp and Deceptive functions. In this case we used a model with $n = 40$ elements to ensure good accuracy also at small values of σ. Figure 4 shows the results for Ramp. As one can see too small values of σ slow down the march towards the optimum, while

Table 1. Comparison between the probability of sampling the optimum domain predicted by the Markov chain for a variable-σ ES and empirical data (averages over 1,000 independent runs) after 50 generations for different discretisation resolutions, n. (NB the size of the optimum domain reduces as the resolution increases.)

	Ramp		Deceptive	
Resolution (n)	Model	Runs	Model	Runs
5	0.973	1.000	0.328	0.378
10	0.998	1.000	0.376	0.388
20	0.998	1.000	0.374	0.381
40	0.988	0.985	0.344	0.349

too big values make the search excessively random (note, resampling and the rejection of samples outside Ω make the search even slower than pure enumeration which on average would require 20 trials to find the optimum). So, the optimum σ for this function appears to be between 0.5 and 1, as also suggested by the success rates reported in Figure 2. As shown in Figure 5, the results for for Deceptive are radically different. Firstly, for very low values of σ the problem of finding the expected hitting time becomes unstable and so we cannot compute reliable values. It is clear, however, that the search for the global optimum becomes easier as σ grows. This is to be expected. Most evolutionary algorithms do worse than random search on deceptive problems. So, by increasing the search variance, we turn our ES more and more into a random searcher, thereby improving performance (although resampling and the rejection of samples outside Ω prevent performance to ever reaching the pure enumeration limit of 20).

We then considered the (1+1)-ES with variable σ and studied the proportion of runs in which the individual occupied the domain containing the global optimum Ω_n. In our tests we allowed both the discretise algorithm and real one to use a range of 21 different σ's in the range $\{\sigma_0 \cdot c^{-Z}, \sigma_0 \cdot c^{-Z+1}, \cdots, \sigma_0 \cdot c^{Z}\}$ with $\sigma_0 = 1$, $Z = 10$ and $c = 1.1$. Table 1 shows how the accuracy of the model varies as a function of the number of domains (n). Only the smallest chain, where $n = 5$ and σ can take 21 values (i.e., the Markov chain has 105 states), deviates significantly from the success probability[5] observed in real runs. Figure 6 shows how accurate the predictions of the model can be throughout a run. Note that as one increases the grid resolution, the size of the element containing the optimum, $|\Omega_n|$, decreases. So, it becomes harder and harder to hit such a region (for both the model and the algorithm). This is the reason why in Table 1 the figures for $n = 20$ are bigger than for $n = 40$.

7.2 Bare-bones PSO

In the experiments with the bare bones PSO we performed 5,000 runs for each setting. Runs lasted 100 generations. In this case we wanted to compare not just the success probability, but the whole state distribution at the end of the runs.

[5] More precisely, the probability of sampling Ω_n.

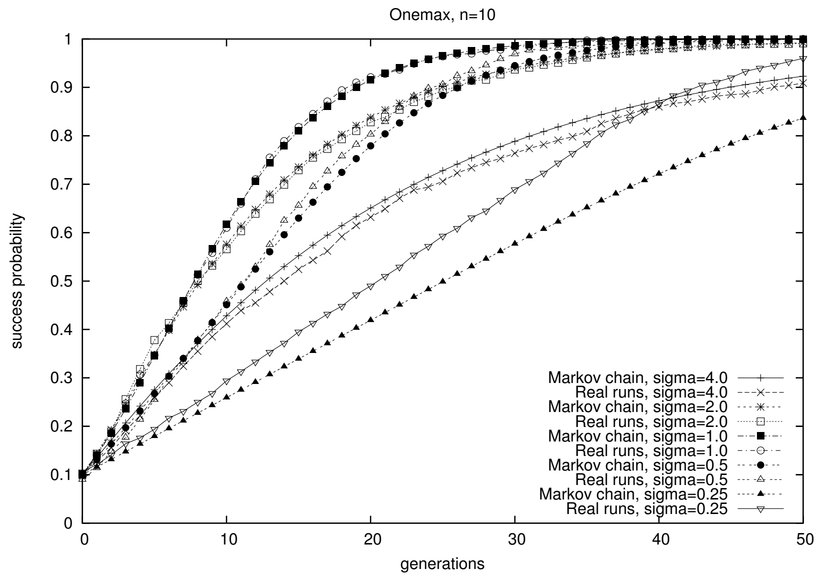

Fig. 2. $(1 + 1)$-ES with fixed σ: comparison between the success probability predicted by the chain and that recorded in real runs for the Ramp function discretised with $n = 10$ plateaus and for different values of σ.

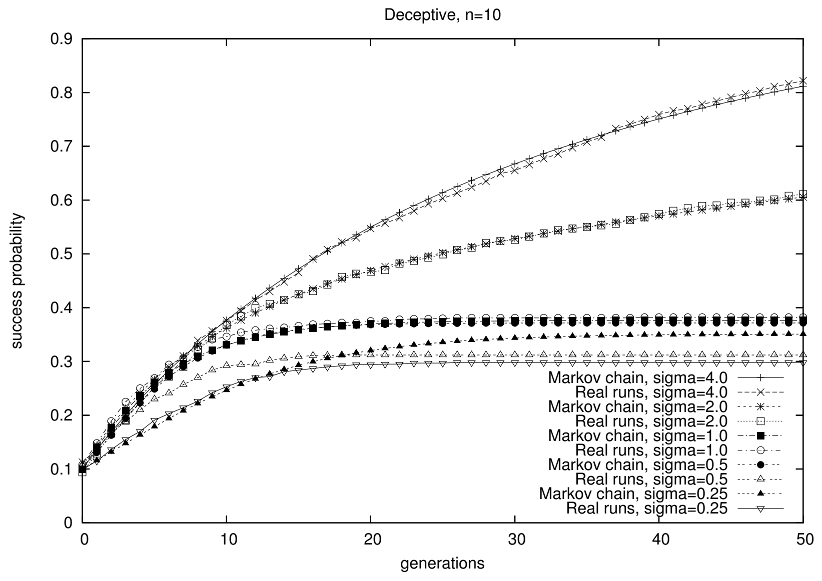

Fig. 3. $(1 + 1)$-ES with fixed σ: comparison between the success probability predicted by the chain and that recorded in real runs for the Deceptive function discretised with $n = 10$ plateaus and for different values of σ.

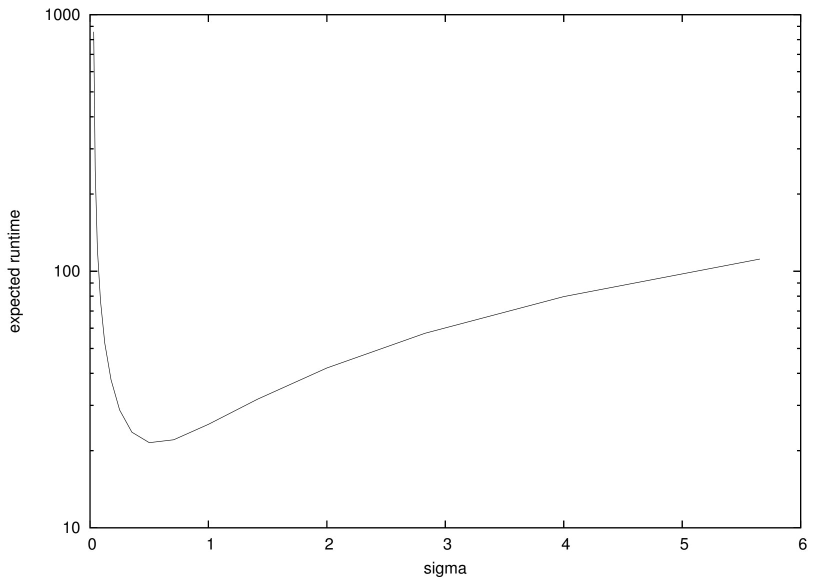

Fig. 4. $(1+1)$-ES with fixed σ: expected first hitting time for the domain containing the global optimum for the Ramp function discretised with $n = 40$ plateaus.

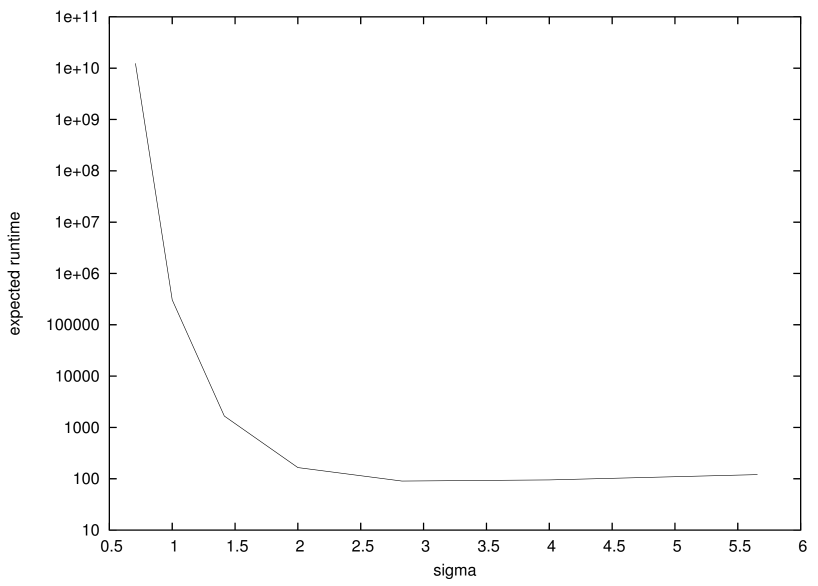

Fig. 5. $(1+1)$-ES with fixed σ: expected first hitting time for the domain containing the global optimum for the Deceptive function discretised with $n = 40$ plateaus.

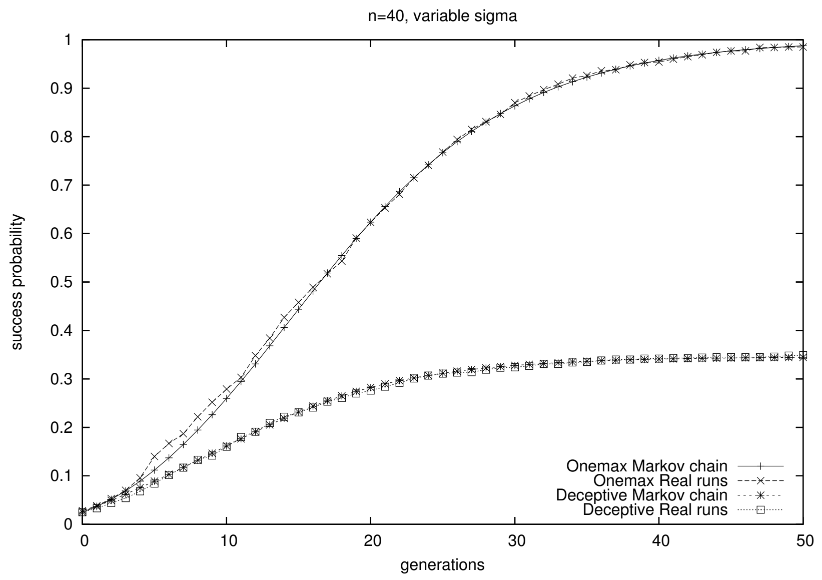

Fig. 6. Comparison between the probability of sampling the optimum domain predicted by the Markov chain for a variable-σ (1+1)-ES and empirical data (averages over 1,000 independent runs) on Ramp and Deceptive. Grid with $n = 40$ elements.

Figures 7–12 compare the distributions obtained in real runs with those predicted by the chain for Ramp and Deceptive and for population sizes $P = 2$, $P = 3$ and $P = 4$ in the case where the domain is divided into just $n = 5$ subdomains. Because the number of states grows very quickly with the resolution, n, and the population size, P, and only very few states have non-zero probabilities, it is very hard to obtain a meaningful plot of the full state distribution. So we plot only the 20 states with the largest probabilities. Despite the low resolution used, we obtain an extremely good match between the distributions for all population sizes tested. Also, the success probabilities (the rightmost point in the plots for Ramp, and the one just before the last for Deceptive) match very closely.

Naturally, as with the ES models, increasing the resolution n (see Figure 7) improves fidelity and provides more accurate information on the distribution of the population.

7.3 Real-valued GA

We performed 5,000 real-valued GA runs for each parameter setting. Runs lasted 100 generations. As with the PSO, we focused on the state distribution at the end of the runs.

Figures 13–18 compare the distributions obtained in real runs with those predicted by the chain for the case where the domain is divided into just $n = 5$

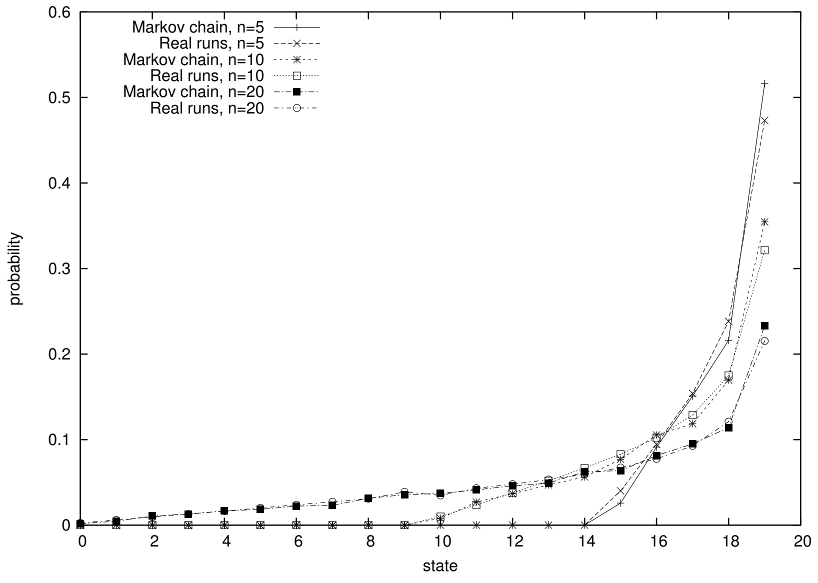

Fig. 7. Comparison between predicted and observed state distributions at generation 100 for a BB-PSO (population size $P = 2$) applied to the Ramp function and for grid resolutions $n = 5$, $n = 10$ and $n = 20$.

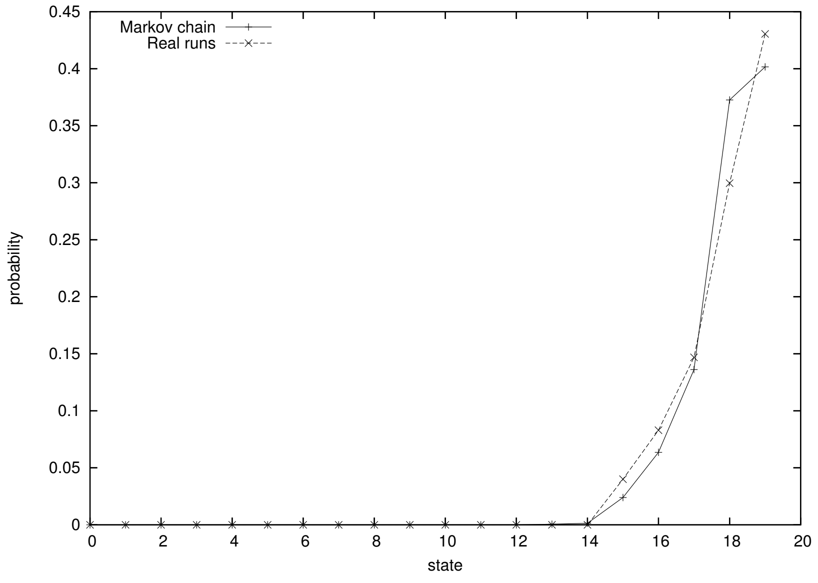

Fig. 8. Comparison between predicted and observed state distributions at generation 100 for a BB-PSO (population size $P = 2$) applied to the Deceptive function and for a grid resolutions $n = 5$.

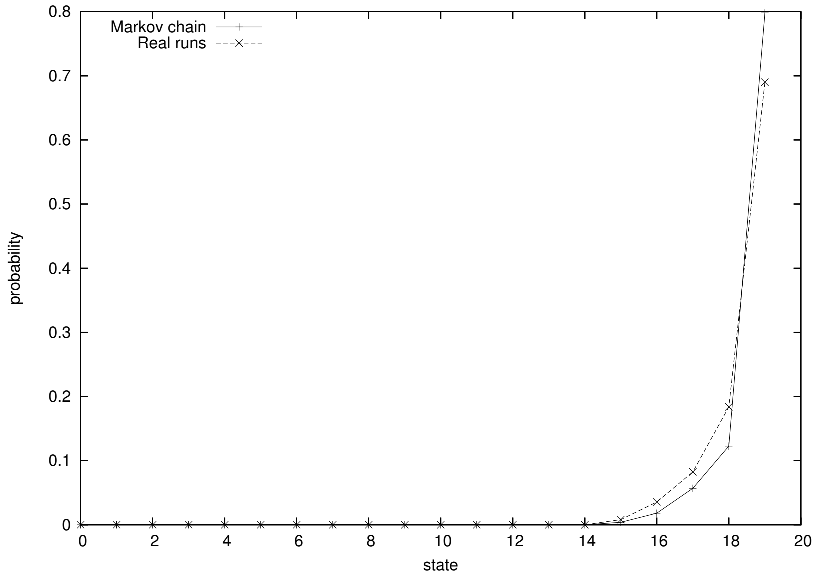

Fig. 9. As in Figure 7 but for $P = 3$ and $n = 5$.

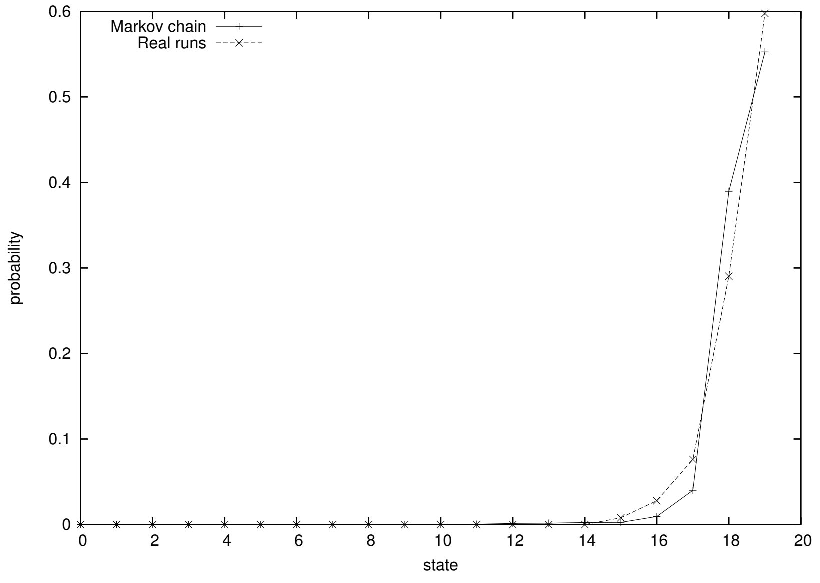

Fig. 10. As in Figure 8 but for $P = 3$.

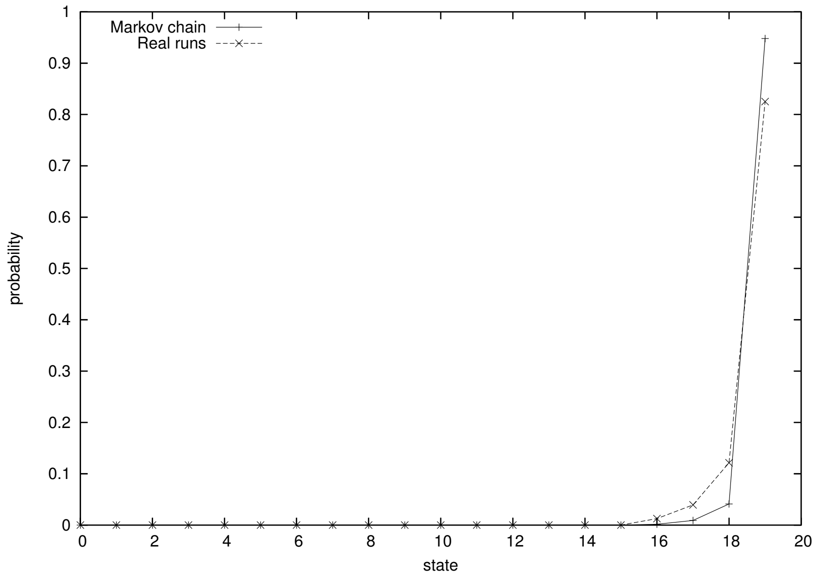

Fig. 11. As in Figure 7 but for $P = 4$ and $n = 5$.

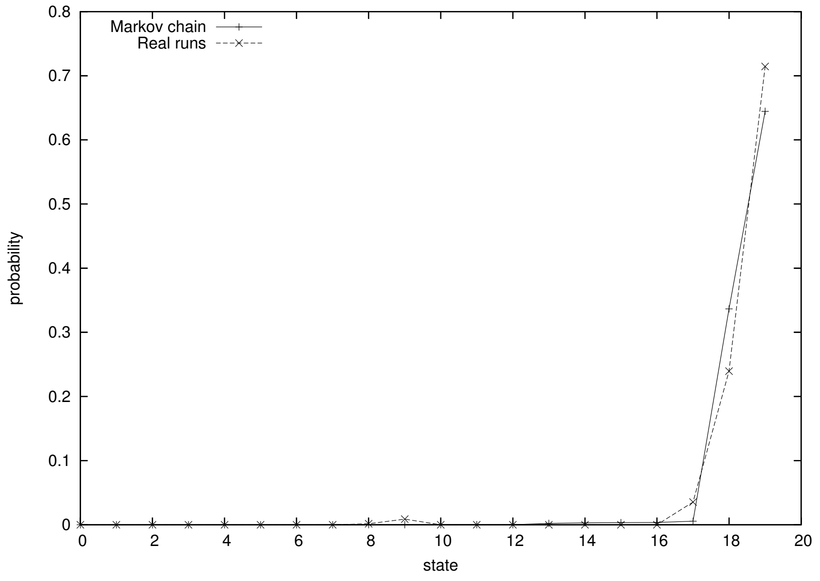

Fig. 12. As in Figure 8 but for $P = 4$.

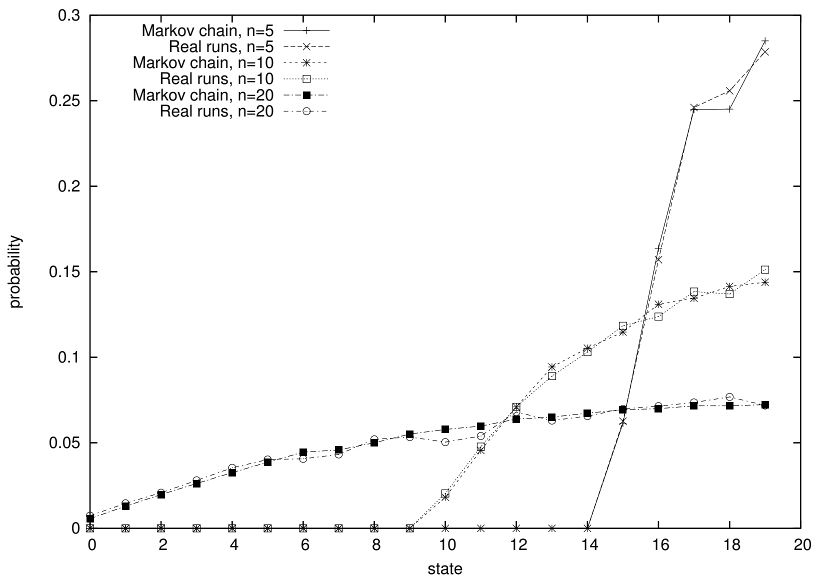

Fig. 13. Comparison between predicted and observed state distributions at generation 100 for a GA (population size $P = 2$) applied to the Ramp function and for grid resolutions $n = 5$, $n = 10$ and $n = 20$.

sub-domains and for population sizes $P = 2$, $P = 3$ and $P = 4$. Again, the figures plot the 20 states with the largest probabilities. Despite the low resolution used, we obtain an extremely good match between the distributions for all population sizes tested and the success probabilities match very closely.

Again, increasing the resolution n (see Figure 13) provides more accurate information on the distribution of the population.

7.4 Two-dimensional Problems: Sphere and Rastrigin

Very accurate results can also be obtained for higher dimensional and realistic test functions. Figure 19, for example, compares the success probability estimated by the chain and the actual success rate in 100,000 independent runs for the variable-σ $(1 + 1)$-ES used in Section 7.1 on a 2–D sphere function over the interval $[-5, 5)^2$ discretise with a $21 \times 21 = 441$ element grid. Since we allowed 21 different σ's, the total number of states in the Markov chain was 9261. This might appear large, however the transition matrix is very sparse and the chain can be computed and iterated in minutes on an ordinary personal computer.

Results of a similar quality were obtained when we applied the approach to a 2–D Rastrigin function over the interval $[-5, 5)^2$. Because of the complexity of this function (it presents 100 optima in the interval chosen), we used a more sophisticated variable meshing technique. The discretisation proceeded at the 21×21 resolution until an element with high fitness was found. When this

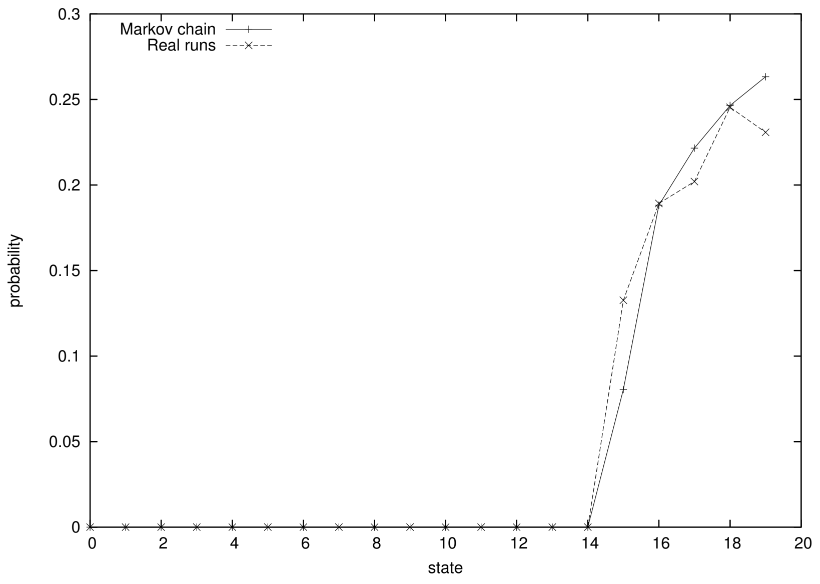

Fig. 14. Comparison between predicted and observed state distributions at generation 100 for a GA (population size $P = 2$) applied to the Deceptive function and for a grid resolutions $n = 5$.

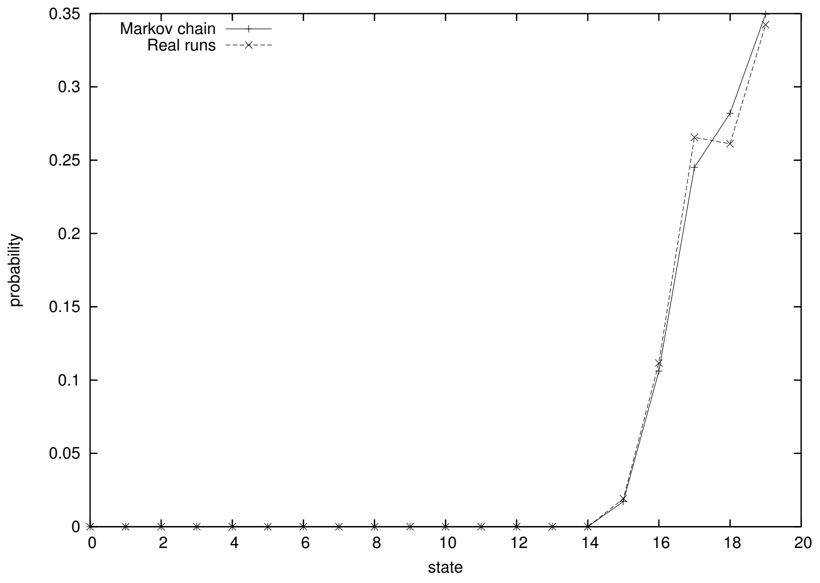

Fig. 15. As in Figure 13 but for $P = 3$ and $n = 5$.

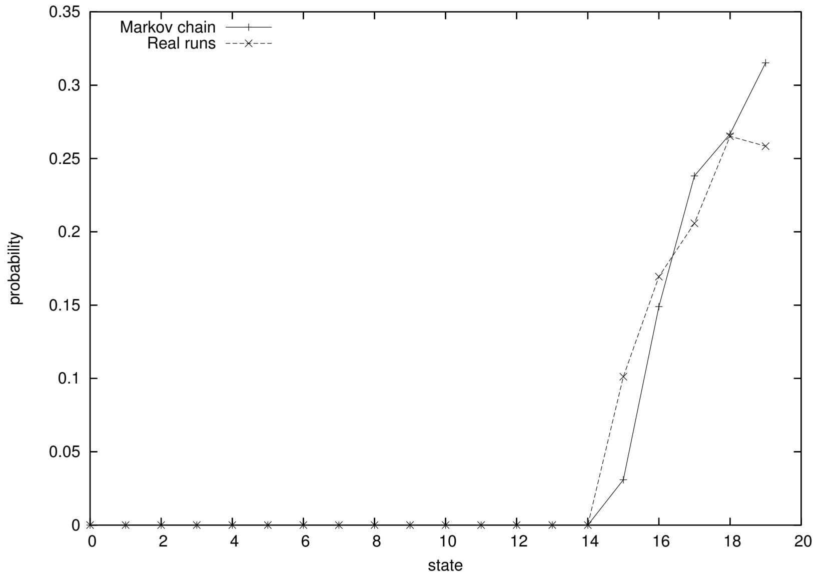

Fig. 16. As in Figure 14 but for $P = 3$.

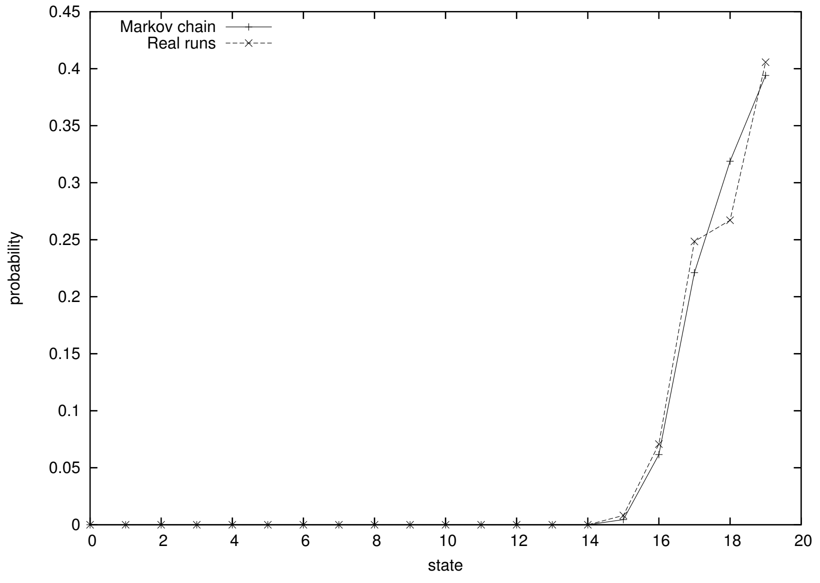

Fig. 17. As in Figure 13 but for $P = 4$ and $n = 5$.

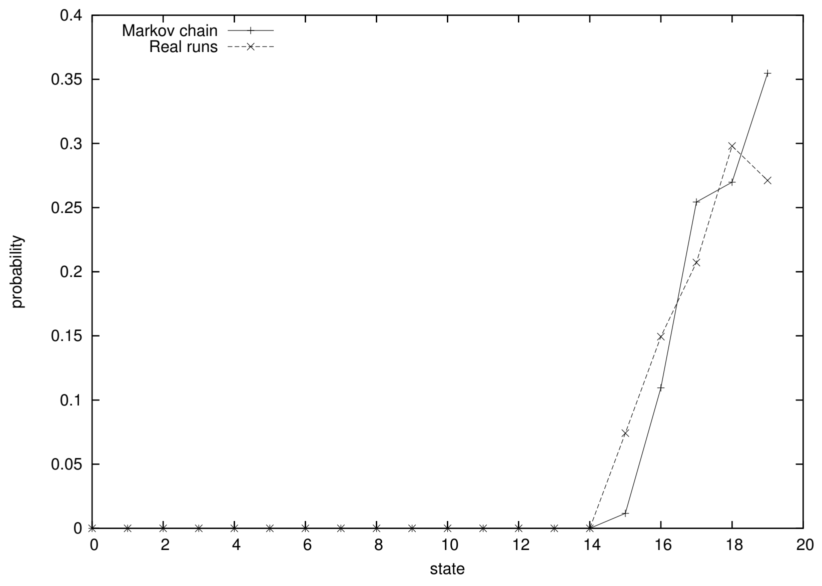

Fig. 18. As in Figure 14 but for $P = 4$.

happened the element was replaced by a set of smaller ones, effectively locally increasing the resolution to that of a 61×61 grid. This gave a finite element grid of 1329 elements instead of the 441 used for the sphere function. Consequently the total number of states was about three times higher, namely 27909. As shown in Figure 20, chain and experiments are in excellent agreement. Note that the element containing the optimum is 9 times smaller that for the sphere function. So, to obtain reliable statistics we performed 1,000,000 runs.

8 Conclusions

We have introduced a finite element method to construct discrete Markov chain models for continuous optimisers and we have tested it on two types of evolutionary strategies, on the "bare bones" particle swarm optimiser and on a genetic algorithm with continuous gene values. Whilst the models are approximate, they can be made as accurate as desired by reducing the size of the sub-domains used to quantise the system.

Being Markov chains, the models allow one to compute everything that one needs to estimate about the distribution of states of a search algorithm over any number of generations and for any fitness function. This is a complete characterisation of the behaviour of the search algorithm. For example, in this single framework, in addition to the success probability and the expected runtime, one could calculate the evolution of mean fitness, the population diversity and the size of basins of attraction. We can also compare the behaviour of algorithms by

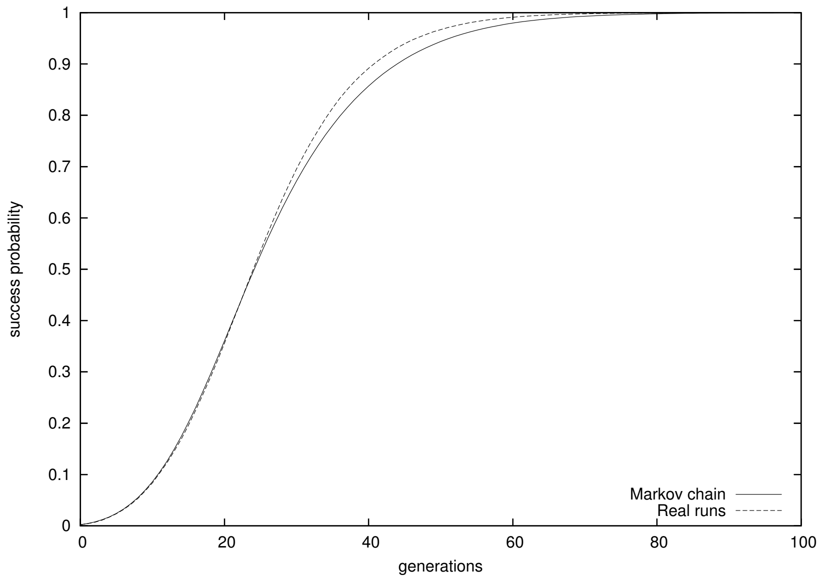

Fig. 19. ES with variable σ: comparison between predicted and observed success probabilities for a 2–D sphere function. Observations are means of 100,000 runs.

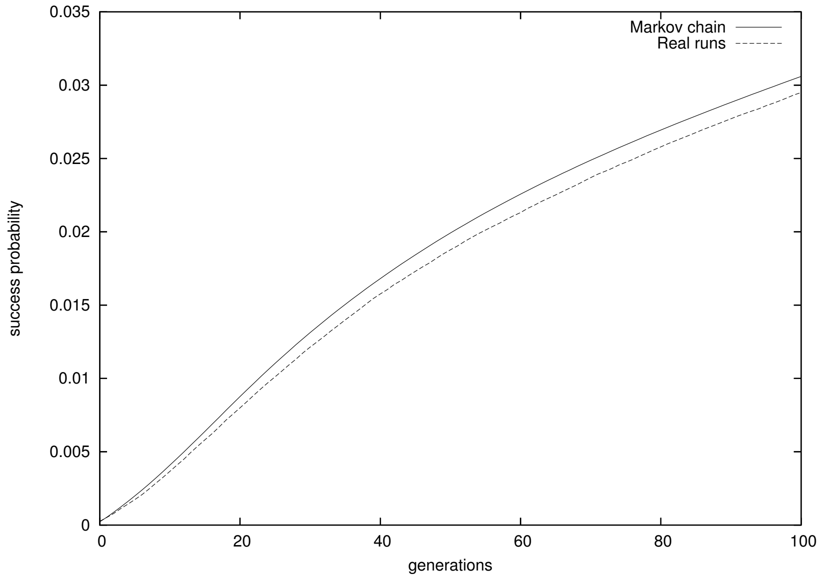

Fig. 20. ES with variable σ: comparison between predicted and observed success probabilities for a 2–D Rastrigin function. Observations are means of 1,000,000 runs.

comparing their Markov chains for different problems and compare how different fitness functions influence the behaviour of an algorithm by comparing the corresponding chains.

This is remarkable, but there is of course a price to pay. The price is that, unsurprisingly, like most other models of evolutionary algorithms, the model scales exponentially with the population size, or more generally the size of the memory used by a search algorithm.

In future research we intend to present a deeper analysis and comparison of the Markov chains obtained for different algorithms. We will look at the limiting behaviour of system in the infinite time limit by applying traditional Markov chain analysis techniques. We will also study our models mathematically in the limit of the discretisation resolution going to zero. Finally we want to apply the method to a broader variety of search algorithms, including simulated annealing, traditional particle swarms with velocities, $(\mu+\lambda)$-ESs, and differential evolution.

The method also opens the way to using the mathematical power of Markov chains, specifically existing results on their limiting distribution and rates of convergence, for a far wider range of practical evolutionary algorithms and realistic fitness functions, than has previously been the case.

Acknowledgements

The authors would like to acknowledge support by EPSRC XPS project (GR/T11234/01) and Leverhulme Trust (F/00213/J).

References

1. I. Babuska. The Finite Element Method for Infinite Domains. I. *Mathematics of Computation*, 26(117):1–11, 1972.
2. T. Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms.* Oxford University Press, New York, 1996.
3. K. J. Bathe. *Finite Element Procedures in Engineering Analysis.* Prentice-Hall, Englewood Cliffs, New Jersey, 1982.
4. M. Clerc. Stagnation analysis in particle swarm optimisation or what happens when nothing happens. Technical Report Technical Report CSM-460, Department of Computer Science, University of Essex, August 2006. Edited by Riccardo Poli.
5. M. Clerc and J. Kennedy. The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *IEEE Transaction on Evolutionary Computation*, 6(1):58–73, February 2002.
6. D. Givoli. *Numerical methods for problems in infinite domains.* Elsevier New York, 1992.
7. C. Goldstein. The Finite Element Method with Nonuniform Mesh Sizes for Unbounded Domains. *Mathematics of Computation*, 36(154):387–404, 1981.
8. J. Kennedy. The behavior of particles. In *Evolutionary Programming VII: Proceedings of the Seventh Annual Conference on evolutionary programming*, pages 581–589, San Diego, CA, 1998.

9. J. Kennedy. Bare bones particle swarms. In *Proceedings of the IEEE Swarm Intelligence Symposium 2003 (SIS 2003)*, pages 80–87, Indianapolis, Indiana, 2003.
10. J. Kennedy, R. C. Eberhart, and Y. Shi. *Swarm Intelligence*. Morgan Kaufmann, San Francisco, 2001.
11. R. A. Krohling. Gaussian particle swarm with jumps. In D. Corne, Z. Michalewicz, M. Dorigo, G. Eiben, D. Fogel, C. Fonseca, G. Greenwood, T. K. Chen, G. Raidl, A. Zalzala, S. Lucas, B. Paechter, J. Willies, J. J. M. Guervos, E. Eberbach, B. McKay, A. Channon, A. Tiwari, L. G. Volkert, D. Ashlock, and M. Schoenauer, editors, *Proceedings of the 2005 IEEE Congress on Evolutionary Computation*, volume 2, pages 1226–1231, Edinburgh, UK, 2-5 September 2005. IEEE Press.
12. T. Nomura and K. Shimohara. An analysis of two-parent recombinations for real-valued chromosomes in an infinite population. *Evolutionary Computation*, 9(3):283–308, 2001.
13. E. Ozcan and C. K. Mohan. Particle swarm optimization: surfing the waves. In *Proceedings of the IEEE Congress on evolutionary computation (CEC 1999)*, Washington DC, 1999.
14. G. Rudolph. Convergence of evolutionary algorithms in general search spaces. In *International Conference on Evolutionary Computation*, pages 50–54, 1996.
15. W. M. Spears. *The Role of Mutation and Recombination in Evolutionary Algorithms*. PhD thesis, George Mason University, Fairfax, Virginia, USA, 1998.
16. R. Storn. Designing digital filters with differential evolution. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, Advanced Topics in Computer Science, chapter 7, pages 109–125. McGraw-Hill, Maidenhead, Berkshire, England, 1999.
17. L. Thompson and P. Pinsky. Space-time finite element method for structural acoustics in infinite domains. Part 1: formulation, stability and convergence. *Computer Methods in Applied Mechanics and Engineering*, 132(3):195–227, 1996.
18. I. C. Trelea. The particle swarm optimization algorithm: convergence analysis and parameter selection. *Information Processing Letters*, 85(6):317–325, 2003.
19. F. van den Bergh. *An Analysis of Particle Swarm Optimizers*. PhD thesis, Department of Computer Science, University of Pretoria, Pretoria, South Africa, November 2001.