Tournament Selection, Iterated Coupon-Collection Problem, and Backward-Chaining Evolutionary Algorithms

Riccardo Poli

Department of Computer Science, University of Essex, UK
rpoli@essex.ac.uk

Abstract. Tournament selection performs tournaments by first sampling individuals uniformly at random from the population and then selecting the best of the sample for some genetic operation. This sampling process needs to be repeated many times when creating a new generation. However, even upon iteration, it may happen not to sample some of the individuals in the population. These individuals can therefore play no role in future generations. Under conditions of low selection pressure, the fraction of individuals not involved in any way in the selection process may be substantial. In this paper we investigate how we can model this process and we explore the possibility, methods and consequences of not generating and evaluating those individuals with the aim of increasing the efficiency of evolutionary algorithms based on tournament selection. In some conditions, considerable savings in terms of fitness evaluations are easily achievable, without altering in any way the expected behaviour of such algorithms.

1 Introduction

Tournament selection is one of the most popular forms of selection in evolutionary algorithms (EAs). In its simplest form, a group of n individuals is chosen randomly uniformly from the current population, and the one with the best fitness is selected (e.g., see (Bäck et al., 2000)). The parameter n is called the *tournament size* and can be used to vary the selection pressure exerted by this method (the higher n the higher the pressure to select above average quality individuals).

Different selection methods, including tournament selection, have been analysed mathematically in depth in (Blickle and Thiele, 1995, 1997; Motoki, 2002). The main emphasis of previous research has been the evaluation of the changes produced by selection on the fitness distribution of the population. The proportion of individuals of a population that is not selected during the selection phase is one of the quantities that have been used to characterise the behaviour of selection algorithms. This quantity is called the *loss of (fitness) diversity*. Under the implicit assumption that the population is wholly diverse (each individual has a unique rank), the loss of diversity p_d for tournament selection was estimated in (Blickle and Thiele, 1995, 1997) as

$$p_d = n^{-\frac{1}{n-1}} - n^{-\frac{n}{n-1}},$$

and later calculated exactly in (Motoki, 2002) as

$$p_d = \frac{1}{M}\sum_{k=1}^{M}\left(1 - \frac{k^n - (k-1)^n}{M^n}\right)^M,$$

where M is the population size.

If one assumed that selection only is used or that we use selection to form a mating pool[1], the creation of a new generation would require exactly M selection steps. These are exactly the conditions assumed in the work mentioned above. However, in this paper we do not make this assumption. Instead, we consider the case where each genetic operator directly invokes the selection procedure to provide a sufficient number of parents for its application (e.g., twice in case of crossover). So, there are situations where more than M selection steps are required to form a new generation. More precisely, in a generational selecto-recombinative algorithm, where crossover is performed with probability p_c and reproduction is performed with probability $1-p_c$, the number of selection steps required to form a new generation is a stochastic variable with mean

$$M(1-p_c) + \rho M p_c = M[1 + (\rho - 1)p_c],$$

where $\rho = 1$ for a crossover operator which returns two offspring after each application, and $\rho = 2$ if only one offspring is returned[2]. The two-offspring version of crossover is more efficient in terms of tournaments required, and also, since $\rho = 1$, the number of selection steps required to form a new generation is not stochastic and is simply M. For brevity in the following we will use the definition $\alpha = [1 + (\rho - 1)p_c]$.

So, because in each tournament we need n individuals, tournament selection requires drawing $n\alpha M$ individuals uniformly at random (with resampling) from the current population. An interesting side effect of this process is that not all individuals in a particular generation are necessarily sampled within the $n\alpha M$ draws, and this is particularly true for small values of the tournament size n.

[1] The mating pool is an intermediate population which gets created by using selection only and from which other operations, such as reproduction and crossing over, draw individuals uniformly at random

[2] In the following we will ignore the (potential) stochasticity of the number of selection steps required to create a new generation. This is justifiable for various reasons: a) it simplifies the analysis (but without significant loss in terms of accuracy of the results obtained, as empirically verified), b) when $\rho = 1$ (two-offspring crossover or mutation only algorithm) there is no stochasticity (and so the analysis is exact), c) even with $\rho = 2$ (one-offspring crossover) it is possible to slightly modify the evolutionary algorithm in such a way that there is no stochasticity

For example, let us imagine to run an evolutionary algorithm starting from a random population containing 4 individuals, which we will denote as 1, 2, 3 and 4. Let us assume that we are creating the next generation using tournament selection with tournament size 2 and mutation only. Then, the creation of the first individual will require randomly picking two individuals from the current population (say individuals 1 and 4) and selecting the best for mutation. We repeat the processes to create the second, third and fourth new individuals. It is not unconceivable that in so doing maybe individual 3 was never involved in any tournament.

It is absolutely crucial, at this stage, to stress the difference between *not sampling* and *not selecting* an individual in a particular generation. The latter refers to an individual which was involved in one or more tournaments, but did not win any, and this is exactly what previous work on tournament selection has concentrated on. The former, instead, refers to an individual which did not participate in any tournament at all, simply because it was not sampled during the creation of the required αM tournament sets. It is individuals such as this that are the focus of this paper. Therefore, the results in this paper are orthogonal to those appeared in the previous work mentioned above and are not limited by uniqueness assumptions.

Continuing with our argument, in general, how many individuals should we expect not to take part in any of αM tournaments? As will be shown in the next section, an answer comes straight from the literature on the coupon collector problem. However, before we explain the connection in more detail, we may want to reflect briefly on why this effect is important.

In general those individuals that do not get sampled by the selection process have no influence whatsoever on future generations. However, these individuals use up resources, e.g., memory, but also, and more importantly, CPU time for their creation and evaluation. For instance, individual 3 in the previous example was randomly generated and had its fitness evaluated in preparation for selection, but neither its fitness nor its genetic make up could have any influence on future generations. So, one might ask, why did we generate such an individual in the first place? And what about generations following the first two? It is entirely possible that an individual in generation two got created and evaluated, but was then neglected by tournament selection, so it had no effect whatsoever on generations 3, 4, etc. Did we really need to generate and evaluate such an individual? If not, what about the parents of such an individual: did we need them? What sort of saving could we obtain by not creating unnecessary individuals in a run?

In this paper we intend to analyse the relationship between tournament selection and the coupon collector's problem, and attempt to answer all of the questions above and more. In particular, we want to rethink the way evolutionary algorithms are run for the purpose of making best use of the available resources *without altering in any way the expected behaviour of such algorithms.* As will become clear in the next sections, in some conditions, saving of 20% fitness evaluations or, in fact, even more are easily achievable.

2 Coupon Collection and Tournament Selection

In the *coupon collector problem*, every time a collector buys a certain product, a coupon is given to him. The coupon is equally likely to be any one of N types. In order to win a prize, the collector must have at least one coupon of each type. The question is: how many products will the collector have to buy on average before he can expect to have a full set of coupons? The answer (Feller, 1971) is obtained by considering that the probability of obtaining a first coupon in one trial is 1 (so the expected waiting time is just 1 trial), the probability of obtaining a second coupon (distinct from the first one) is $\frac{N-1}{N}$ (so the expected waiting time is $\frac{N}{N-1}$), the probability of obtaining a third coupon (distinct from the first two) is $\frac{N-2}{N}$ (so the expected waiting time is $\frac{N}{N-2}$), and so on. So, the expected number of trials to obtain a full set of coupons is

$$E_N = 1+\frac{N}{N-1}+\frac{N}{N-2}+\cdots N = N\times\left(\frac{1}{N}+\frac{1}{N-1}+\cdots 1\right) = N\log N+O(N).$$

It is well known that the $N\log N$ limit is sharp. For example, if X is a random variable representing the number of coupons collected, for any constant c

$$\lim_{N\to\infty} \Pr\{X > N\log N + cN\} = 1 - e^{-e^{-c}}.$$

So, for $c \approx 3$ this probability is less than 5%.

How is the process of tournament selection related to the coupon collection problem? We can imagine that the M individuals in the current population are $N = M$ distinct coupons and that tournament selection will draw (with replacement) $n\alpha M$ times from this pool of coupons. Because of the sharpness of the coupon-collector limit mentioned above, if $M\log M + cM < n\alpha M$, i.e., if $n\alpha > \log M + c$ for some suitable positive constant c, then we should expect tournament selection to sample all individuals in the population most of the time. However, for sufficiently small tournament sizes or for sufficiently large populations the probability that there will be individuals not sampled by selection becomes significant.

So, how many different coupons (individuals) should we expect to have sampled at the end of the $n\alpha M$ trials? In the coupon collection problem, the expected number of trials necessary to obtain a set of x distinct coupons is

$$E_x = 1 + \frac{N}{N-1} + \frac{N}{N-2} + \cdots \frac{N}{N-x+1} = N\log\frac{N}{N-x} + O(N).$$

By setting $E_x = n\alpha M$, $N = M$ and ignoring terms of order $O(N)$, from this we obtain an estimate for the number of distinct individuals sampled by selection

$$x \approx M\left(1 - e^{-n\alpha}\right). \tag{1}$$

This indicates that the expected proportion of individuals *not* sampled in the current population varies approximately like a negative exponential of the tournament size.

This approximation is quite accurate. However, we can calculate the expected number of individuals neglected after performing $n\alpha M$ trials directly. We first calculate the probability that one individual is not involved in one trial as $1 - 1/M$. Then the expected number of individuals not involved in any tournaments is simply

$$M(1 - 1/M)^{n\alpha M} = M\left(\frac{M}{M-1}\right)^{-n\alpha M},$$

which also varies like a negative exponential of the tournament size.

As shown in Figure 1 for $\alpha = 1$ (two-offspring crossover or no crossover), typically for $n = 2$ over 13% of the population is neglected, for $n = 3$ this drops to 5%, for $n = 4$ this is 2%, and becomes negligible for bigger values of n.

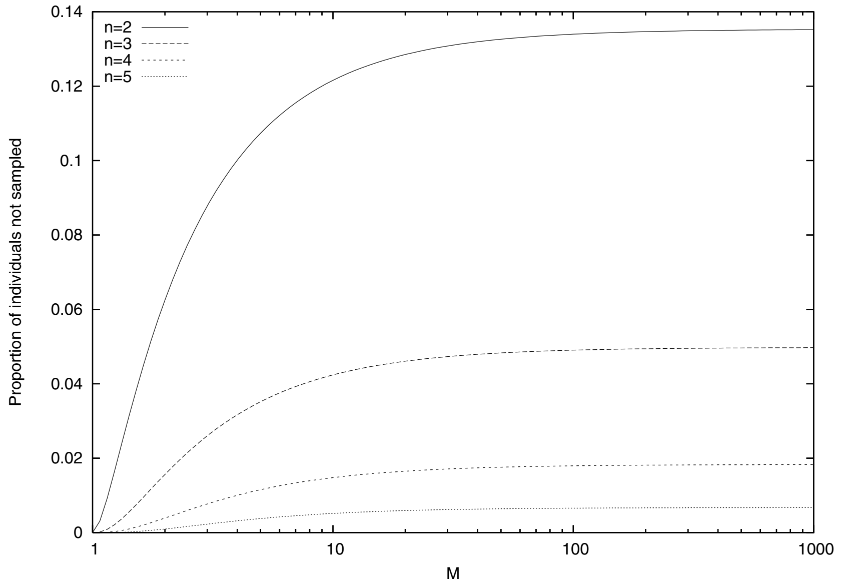

Fig. 1. Proportion of individuals not sampled in one generation by tournament selection for different tournament and population sizes assuming two-offspring crossover or mutation only are used

This simple analysis suggests that saving computational resources by avoiding the creation and evaluation of individuals which will not be sampled by the tournament selection process appears to be possible only for relatively low selection pressures. However, tournament sizes in the range 2–5 are quite common in practice, particularly when attacking hard, multi-modal problems which require extensive exploration of the search space before zooming the search onto any particular region. By rethinking how we perform selection, in this paper we will show how we can achieve substantial computational savings in *any evolutionary algorithm* based on tournament selection on *any problem* where low selection pressure is appropriate *without changing* in any way the course of a run! We will start exploring how we can achieve all this in the next section.

3 Iterated Coupon Collector Problem

Now, let us consider a new game, that we will call the *iterated coupon collection problem*, where the coupon set changes at regular intervals, but the number of coupons available, N, remains constant. Initially the collector is given a (possibly incomplete) set of $m(0)$ old coupons. Each old coupon allows the collector to draw n new coupons. So, he can perform a total of $nm(0)$ trials, which will produce a set of $m(1)$ distinct coupons from the new set. The coupon set now changes, and the player performs $nm(1)$ trials to gather as many as possible new distinct coupons. And so on. Interesting questions here are: what happens to $m(t)$ as t grows? Will it reach a limit? Will it oscillate? In which way will the values of n, $m(0)$ and N influence its behaviour?

Before we answer these questions let us motivate our analysis a bit. How is this new problem related to evolutionary algorithms and tournament selection? The connection is simple (we will assume $\alpha = 1$ for the sake of clarity). Suppose an oracle told us which individuals in a particular generation, G, are not involved in any way in future generations, because selection will not sample them. Then we could concentrate on the other individuals in the population, creating and evaluating only them. Let $m(0)$ be the number of such individuals (these are like the initial set of old coupons given to the player). Clearly, in order to create such individuals, we will need to know who their parent(s) were. This will require running $m(0)$ tournaments to select such parents. In each tournament we randomly pick n individuals from generation $G - 1$ (each distinct individual in that generation is equivalent to a coupon in the new coupon set). After, $nm(0)$ such trials we will be in a position to determine (without the need for an oracle) which individuals in generation $G - 1$ will contribute to future generations, we can count them and denote this number with $m(1)$ [3]. So, again, we can concentrate on these individuals only. They are the equivalent of the new set of coupons the collector has gathered. We can now perform $nm(1)$ trials to determine (again without the need for an oracle) which individuals in generation $G - 2$ (the new coupon set) will contribute to future generations, we can count them and denote this number with $m(2)$ and so on until we reach the initial random generation. There the game stops. So, effectively, *the iterated coupon collector problem is a model for the sampling behaviour of tournament selection over multiple generations in a generational evolutionary algorithm.*

Knowing the sequence $m(t)$ for a particular evolutionary algorithm would tell us how much we could save by not creating and evaluating individuals which will not be sampled by selection. Naturally, we will not have an oracle to help us choose G and to give us $m(0)$. For now, while we concentrate on understanding more about the iterated coupon collector problem, we could think of G as the

[3] Because at this stage we are only interested in knowing the number of individuals playing an active role in generation $G-1$, there is no need to determine the winners of the tournaments: we just need to know who was involved in which tournament. So, we do not even need to evaluate fitness, and, therefore, we do not need to know the genetic makeup of any individual

number of generations we are prepared to run our evolutionary algorithm for, and we might imagine that $m(0) = M$ (the whole population).

In the classical coupon collection problem, the shopper will typically perform as many trials as necessary to gather a full collection of coupons. As we have seen before, however, it is quite easy to estimate how many distinct coupons one should expect at the end of any given *fixed* number of trials. Because the iterated coupon collection game starts with a known number of trials, we can calculate the *expected value* of $m(1)$. However, we cannot directly apply the theory in the previous section to gather information about $m(2)$. This is because $m(1)$ is a stochastic variable, so in order to estimate $m(2)$ we would need to know the probability distribution of $m(1)$ not just its expected value.

Exact probabilistic modelling can be obtained by considering the coupon collection game as a Markov chain (Feller, 1971), where the state of the chain is the number of distinct coupons collected. The transition matrix for the chain can easily be constructed by noticing that the chain can be in state k (i.e., the collector has k distinct coupons) at the next time step only if either it was already in state k and a new coupon has been acquired that is a duplicate (which happens with probability $\frac{k-1}{N}$) or it was in state $k-1$ and the coupon just acquired is a new one (which, of course, happens with probability $\frac{N-k+1}{N}$). So, the number of distinct individuals in the previous generation sampled when randomly picking individuals for tournament selection can be described by the following Markov transition matrix:

$$A = \frac{1}{M}\begin{pmatrix} 0 & 0 & 0 & 0 \cdots & 0 \\ M & 1 & 0 & 0 \cdots & 0 \\ 0 & M-1 & 2 & 0 \cdots & 0 \\ 0 & 0 & M-2 & 3 \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \ddots & \vdots \\ 0 & 0 & 0 & 0 \cdots & M \end{pmatrix}.$$

The process always starts from state 0, which can be represented using the state probability vector $e_0 = (1\,0\,0\,0 \cdots 0)^T$. So, the probability distribution over the states after x time steps (i.e., coupon draws or random samples from the population) is given by $A^x e_0$, which is simply the first column of the matrix A^x.

So, if we are interested only in $m(0)$ individuals in generation G, the probability distribution of $m(1)$ (the number of distinct individuals we need to know from generation $G-1$) is given by $A^{n\alpha m(0)} e_0$ [4]. For example, if the population size is $M = 3$, the tournament size is $n = 2$, we use a two-offspring version of crossover ($\alpha = 1$) and we are interested in $m(0) = 1$ individuals, then the probability distribution of $m(1)$ is represented by the following probability vector

[4] This, of course, gives us also the probability distribution over the number of draws, $n\alpha m(1)$, we will need to make from generation $G-2$ in order to fully determine the $m(1)$ individuals we want to know at generation $G-1$

$$\left(\frac{1}{3}\right)^2 \begin{pmatrix} 0\,0\,0\,0 \\ 3\,1\,0\,0 \\ 0\,2\,2\,0 \\ 0\,0\,1\,3 \end{pmatrix}^2 \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \frac{1}{3} \begin{pmatrix} 0 \\ 1 \\ 2 \\ 0 \end{pmatrix}.$$

If we were interested in $m(0) = 2$ individuals at generation G, the probability distribution over the number $m(1)$ of unique individuals sampled would be $A^4 e_0 = (\,0.0\;0.0370\;0.5185\;0.4444\,)^T$. Finally, if we were interested in the whole population ($m(1) = M = 3$), the distribution would be $A^6 e_0 = (0\;0.0041\;0.2551\;0.7407)^T$, which reveals that, in these conditions, even when building a whole generation there are still more than 1 in 4 chances of not sampling the whole population at the previous generation. Of course, if $m(0) = 0$, the probability vector for $m(1)$ is e_0, i.e., $m(1) = 0$.

Although this example is trivial, it reveals that for any given $m(0)$ we can compute a distribution over $m(1)$. That is, we can define a new *Markov chain to model the iterated coupon collector problem.* In this chain a state is exactly the same as in the coupon-collector chain (i.e., the number of distinct coupons sampled), *except that now a time step corresponds to a complete set of draws from the new coupon set rather than just the draw of one coupon.* The transition matrix B for this new chain can be obtained very simply: column i of B is $A^{\alpha i} e_0$. That is

$$B = \left(e_0 \middle| A^{\alpha} e_0 \middle| A^{2\alpha} e_0 \middle| \cdots \middle| A^{M\alpha} e_0 \right).$$

For instance, for the case $M = 3$, $n = 2$ and $\alpha = 1$ considered above

$$B = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0.3333 & 0.0370 & 0.0041 \\ 0 & 0.6667 & 0.5185 & 0.2551 \\ 0 & 0 & 0.4444 & 0.7407 \end{pmatrix}.$$

Of course, once the transition matrix is defined, the chain can be iterated to compute the probability distributions of $m(2)$, $m(3)$ and so on, as back as necessary to reach generation 0.

In general B is block diagonal of the form

$$B = \left(\begin{array}{c|c} 1 & \mathbf{0}^T \\ \hline \mathbf{0} & C \end{array} \right),$$

where $\mathbf{0}$ is a column vector containing M zeros and C is a $M \times M$ stochastic matrix. Clearly B is not ergodic (from state 0 we cannot reach any state other than 0 itself), so we cannot expect a unique limit distribution for $m(t)$. However, because B is block diagonal, we have

$$B^x = \left(\begin{array}{c|c} 1 & \mathbf{0}^T \\ \hline \mathbf{0} & C^x \end{array} \right).$$

So, if we ensured that the probability of the chain initially being in state 0 is 0 (that is $\Pr\{m(0) = 0\} = 0$), the chain could never visit such a state at any future time. Because of this property, and because, objectively, state 0 is totally uninteresting (of course we already know that if we are interested in no individual at generation G, we do not need to know any individual at previous generations!) we can declare such a state of the iterated coupon-collection chain as invalid, and reduce the state set to $\{1, 2, \ldots, M\}$. In this situation C is the state transition matrix for the chain, and to model the sampling behaviour of tournament selection over multiple generations we just need to concentrate on the properties of C.

The transition matrix C is ergodic if $n\alpha > 1$ as can be easily seen by the following argument. If $n\alpha > 1$ then each old coupon gives us the right to draw more than one new coupon in the iterated coupon-collection problem. So, if the state of the chain is k ($k < M$), it is always possible to reach state $k+1$ in one stage of the game with non-zero probability. From there it is then, of course, possible to reach state $k+2$ and so on up to M. So, from any lower state it is always possible to reach any higher state in repeated iterations of the game. But, of course, the converse is always true: irrespective of the value of $n\alpha$ there is always a chance of getting fewer coupons than we had before in an iteration of the game, due to resampling. So, from any higher state we can also reach any lower state (in fact, unlike the reverse, we can achieve this in just one iteration of the game).

Since, $\alpha \geq 1$ and $n \geq 2$ for any practical applications, the condition $n\alpha > 1$ is virtually always satisfied and C is ergodic. So, the Perron-Frobenius theorem guarantees that the probability over the states of the chain converges towards a limit distribution which is independent from the initial conditions (see (Nix and Vose, 1992; Davis and Principe, 1993; De Jong et al., 1995; Rudolph, 1994; Poli et al., 2001) for other applications of this result to genetic algorithms and genetic programming). This distribution is given by the (normalised) eigenvector corresponding to the largest eigenvalue of C ($\lambda_1 = 1$), while the speed at which the chain converges towards such a distribution is determined by the magnitude of the second largest eigenvalue λ_2 (the relaxation time of an ergodic Markov chain is $1/(1 - |\lambda_2|)$). Naturally, this infinite-time limit behaviour of the chain is particularly important if G is sufficiently big that $m(t)$ settles into the limit distribution well before we iterate back to generation 0. Otherwise the transient behaviour is what one needs to focus on. Both are provided by the theory.

Because the transition matrices we are talking about are relatively small ($M \times M$), they are amenable to numerical manipulation. We can, for example, find the eigenvalues and eigenvectors of C for quite respectable population sizes, certainly well in the range of those used in most applications of evolutionary algorithms, thereby determining the limit distribution and the speed at which this is approached.

If $x(t)$ is a probability vector representing the probability distribution over $m(t)$, then the expected value of $m(t)$ is

$$E[m(t)] = \begin{pmatrix} 1 \; 2 \cdots M \end{pmatrix} \cdot x(t) = \begin{pmatrix} 1 \; 2 \cdots M \end{pmatrix} \cdot C^t x(0). \tag{2}$$

Typically $m(0)$ will be deterministic and so $x(0) = e_{m(0)}$ (where e_l is a base vector containing all zeros except for element l which is 1).

If x^* denotes the limit distribution for $x(t)$, then for large enough G, the average number γ of individuals (in generations 0 through to $G-1$) that have no effect whatsoever on a designated set of $m(0)$ individuals of interest at generation G is approximately $\gamma = M - \begin{pmatrix} 1 & 2 & \cdots & M \end{pmatrix} \cdot x^*$. So, a question that naturally springs to mind is whether we could run an evolutionary algorithm without creating and evaluating these unnecessary individuals.

If we could do this, because of the ergodicity of the selection process, for large enough G, it is almost irrelevant whether $m(0)$ is as small as 1 or as large as M, and so we might want to know the entire makeup of generation G and still have a saving of approximately $G \times \gamma$ individual creations and evaluations.

In the next two sections we will consider two different ways in which we could modify our evolutionary algorithm to achieve this kind of saving.

4 Running Evolutionary Algorithms Efficiently

Normally, in each generation of an evolutionary algorithm we iterate the following phases:

a) the choice of genetic operator to use to create a new individual,
b) the creation of a random pool of individuals for the application of tournament selection,
c) the identification of the winner of the tournament (parent) based on fitness,
d) the execution of the chosen genetic operator,
e) the evaluation of the fitness of the resulting offspring.

Naturally, phases (b) and (c) are iterated as many times as the arity of the genetic operator chosen in phase (a), and the whole process needs to be repeated as many times as there are individuals in the new population.

Interestingly, the genetic makeup of the individuals involved in these operations is of interest only in phase (d) (we need to know the parents in order to produce offspring) and phases (c) and (e) (we must know the genetic makeup of individuals in order to evaluate their fitness). However, phases (a) and (b) do not require any knowledge about the actual individuals involved in the creation of a new individual. In most implementations these phases are just performed by properly manipulating numbers drawn from a pseudo-random number generator.

So, there is really no reason why we could not first iterate phases (a) and (b) as many times as needed to create a full new generation (of course, memorising all the decisions taken), and then iterate phases (c)–(e). This idea was first used in (Teller and Andre, 1997) for the purposed on speeding up genetic programming fitness evaluation[5].

[5] The main idea in (Teller and Andre, 1997) was to estimate the fitness of the individuals involved in the tournaments by evaluating them on a subset of the fitness cases available. On the basis of this estimate, for most tournaments it was often possible to determine with a small error probability which individual would win. These tournaments could therefore be decided quickly, while only in a subset of tournaments individuals ended up being evaluated using all fitness cases. This is what produced the speed up

In fact, we could go even further. In many practical applications of evolutionary algorithms, people fix a maximum number of generations they are prepared to run their algorithm for[6]. Let this number be G. So, at the cost of some memory space, we could iterate phases (a) and (b) not just for one generation but for a whole run from the first generation to generation G and then iterate phases (c)–(e) as required (that is, either until generation G or until any other stopping criterion is satisfied).

Because the decisions as to which operator to adopt to create a new individual and which elements of the population to use for a tournament are random, statistically speaking this version of the algorithm is exactly the same as the original. In fact, if the same seed is used for the random number generator in both algorithms, *they are indistinguishable*! However, the version of the algorithm we propose (let us call it an *EA with macro-selection*) makes it possible to avoid wasting the computation involved in generating and evaluating the individuals "neglected" by tournament selection: after macro-selection (the iteration of phases (a) and (b) up until generation G) is completed we analyse the information stored during that phase and identify which population members were not involved in any tournament in each generation, we mark them, and we avoid calculating and evaluating them when iterating phases (c)–(e). We will call this algorithm the *EA with efficient macro-selection (EA-EMS).*

Irrespective of the problem being solved and the parameter settings used, the behaviours of the standard algorithm and the efficient version proposed above will have to be on average identical. So, what are the differences between the two evolutionary algorithms?

Obviously, the standard algorithm requires more fitness evaluations and creations of individuals while the one proposed above requires more bookkeeping and use of memory. Also, clearly, in any particular run, the plots of average fitness and max fitness in each generation may differ (since in EA-EMS not all individuals are considered in calculating these statistics). However, when averaged over multiple runs the average fitness plots would have to coincide.

A more important difference derives from the fact that most practitioners keep track of the best individual seen so far in a run of an EA and designate that as the result of the run. In EA-EMS we can either return the best individual in generation G or the best individual seen in a run *out of those that have been sampled by tournament selection.* Because the fast algorithm does not create and evaluate individuals that did not get sampled, the end-of-run results may differ in the two algorithms. Of course, quite often the best individual seen in a run is actually a member of the population at the last generation. So, if one creates and evaluates all individuals in generation G (which leads to only a minor inefficiency in the EA with efficient macro-selection), most of the time the two algorithms will behave identically from this point of view too.

One remaining inefficiency in EA-EMS derives from the fact that there may be individuals which were sampled by selection when creating a particular indi-

[6] This is a limit that is virtually always present, even if another stopping criterion, e.g., based on fitness, is present

vidual in a new generation which in turn, however, was not sampled by selection. If the first individual was involved only in the creation of the second, further computational savings could be achieved by not creating and evaluating the first individual at all. This is just an example of a more general issue: it is possible that some of the "ancestors" of the individuals neglected by selection were unnecessarily created and evaluated. How could we improve our algorithm to get rid of these individuals too?

Clearly the iteration of phases (a) and (b) over multiple generations induces a graph structure containing $(G + 1)M$ nodes representing all the individuals evolved during a run and where edges connect each individual to the individuals which were involved in the tournaments necessary to select the parents of such an individual. If we are interested in calculating and evaluating all the individuals in the population at generation G, maximum efficiency would be achieved by considering (marking for evaluation) only the individuals which are directly or indirectly connected with the M individuals in generation G. So, the problem can be solved with a trivial connected-component algorithm. Let us call the resulting algorithm an *EA with efficient macro-selection and connected-component detection (EA-EMS-CCD).*

This would appear to be the best we can get from our EA: it saves on fitness evaluations and it is really as close as we can get to the original in terms of behaviour. However, the recursive nature of connected-component detection and the similarity between the mechanics of EAs and that of rule-based systems give us suggestions on how to make further substantial improvements, as will be discussed in the next section.

5 Backward-Chaining Evolutionary Algorithms

Running evolutionary algorithms from generation 0, to generation 1, to generation 2, and so on is the norm: this is what happens in nature, and this is certainly what has been done for decades in the field of evolutionary computation. This is similar to running a rule-based system in forward-chaining mode (Russell and Norvig, 2003). In these systems, we start with a working memory containing some premises, we apply a set of IF-THEN inference rules which modify the working memory by adding or removing facts, and we iterate this process until a certain condition is satisfied (e.g., a fact which we consider to be a conclusion is asserted). The rules in the knowledge base are a bit like the genetic operators in an evolutionary algorithm, the working memory is a bit like a population and the facts in it are a bit like individuals in an evolutionary algorithm.

This loose analogy between rule-based systems and evolutionary algorithms is not, in itself, terribly useful, except for one thing: it suggests the possibility of running an evolutionary algorithm in backward chaining mode, like one can do with a rule-based system. Broadly speaking, when a rule-based system is run in backward chaining, the system focuses on one particular conclusion that it wants to prove and operates as follows: a) it looks for all the rules which have such a conclusion as a consequent (i.e. a term following the "THEN" part of a rule), b) it analyses the antecedent (the "IF" part) of each such rule, c) if the antecedent is a fact (in other words, it is already in the working memory) then the original conclusion is proven and can be placed in the working memory, otherwise the system saves the state of the inference and recursively restarts the process with the antecedent as a new conclusion to prove (if there is no rule which has this term as a consequent, the recursion is stopped and another way of proving the original conclusion is attempted). If a rule has more than one condition (which is quite common), the system attempts to prove the truth of all the conditions, one at a time, and will assert the conclusion of the rule only if all conditions are satisfied. In this modality only the rules that can contribute to determining the truth or falsity of the target conclusion are ever considered, which of course can lead to major efficiency gains.

So, how would we run an evolutionary algorithm in backward-chaining mode? Let us suppose we are interested in knowing the makeup of the population at generation G and let us start by focusing on the first individual in the population. Let r be such an individual. Effectively r plays the role of a conclusion we want to prove. In order to generate r we only need to know what operator to apply to produce it and what parents to use. In turn, in order to know which parents to use, we need to perform tournaments to select them[7]. In each such tournaments we will need to know the makeup of n (the tournament size) individuals from the previous generation (which of course, at this stage we may still not know). Let us call $S = \{s_1, s_2, \ldots\}$ the set of the individuals that we need to know in generation $G-1$ in order to determine r. Clearly, $s_1, s_2, \ldots$ are like the premises in a rule which, if applied, would allowed us to work out r (this would require evaluating the fitness of each element of S, deciding the winners of the tournament(s) and applying the chosen genetic operator to generate r). Normally we will not know the makeup of these individuals. However, we can recursively consider each of them as a subgoal. So, we determine which operator should be used to compute s_1, we determine which set of individuals at generation $G-2$ is needed to do so, and we continue with the recursion. When we emerge from it, we repeat the process for s_2, etc. The recursion can terminate in one of two ways: a) we reach generation 0, in which case we can directly instantiate the individual in question by invoking the initialisation procedure for the particular EA we are considering, or b) the individual for which we need to know the genetic makeup has already been constructed and evaluated. Clearly the individuals in generation 0 have a role similar to that of the initial contents of the working memory in a rule-based system. Once we have finished with r we repeat the process with all the other individuals in the population at generation G, one by one.

Clearly, at its top-level, the algorithm just described is a recursive depth-first traversal of the graph mentioned at the end of the previous section. While we traverse the graph (more precisely, when we re-emerge from the recursion), we are in a position to know the genetic makeup of all the nodes encountered and so we can invoke the fitness evaluation procedure for each of them. Thus, we

[7] Decisions regarding operator choice and tournaments are trivial and can be made on the spot by drawing random numbers or can be all made in advance as in the EA with macro-selection

can label each node with the genetic makeup and fitness of the individual represented by such a node. Recursion stops when we reach a node without incoming links (a generation-0 individual, which gets immediately labelled randomly and evaluated) or when we reach a node that has been previously labelled. We will call an EA running in this mode a *Backward-Chaining EA (BC-EA).*

Statistically a BC-EA is fully equivalent to the EA-EMS-CCD, and so it presents the same level of equivalence to an ordinary EA. In particular, if the same seed is used for the random number generators and all decisions regarding operators and tournaments are performed in a batch before the graph traversal, generations G of a BC-EA and an EA are indistinguishable.

So, if there are no differences why bother with a BC-EA instead of using a simpler "forward-chaining" version of the algorithm? One important difference between the two modes of operation is the order in which individuals in the population are evaluated. Let us consider an example where we have a population of $M = 5$ individuals which we run for $G = 3$ generations using tournament selection with $n = 2$ and we use crossover with 50% probability. Let us further suppose that, in the first instance, we are interested in knowing the first individual in the last generation. The ancestors of this individual might form a graph like the one in Figure 2.

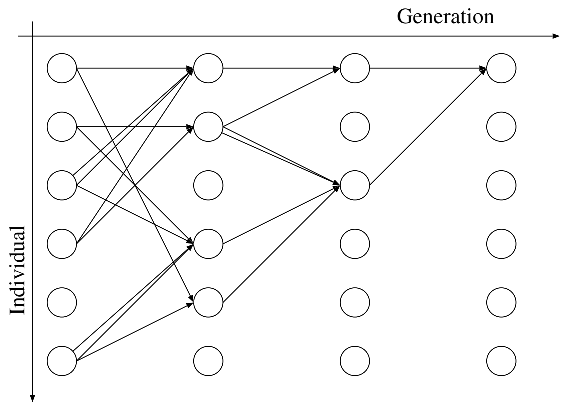

Fig. 2. Ancestors of the first individual in generation 3

Let us denote the nodes in row i (for individual) and column g (for generation) in the graph with the notation r_{ig}. In a forward chaining EA, even if we knew which individuals are unnecessary to define our target individual r_{13} (e.g., individuals r_{50}, r_{31}, r_{61}, and so on), we would evaluate individuals column by column from the left to the right in the following sequence: r_{10}, r_{20}, r_{30}, r_{40}, r_{60}, r_{11}, r_{21}, r_{41}, r_{51}, r_{12}, r_{32}, and finally r_{13}. That is, generation 0 individuals are computed before generation 1 individuals, which in turn are computed before generation 2 individuals, and so on. A backward chaining EA would instead evaluate nodes in a different order. For example, it might do it according to the sequence: r_{10}, r_{30}, r_{40}, r_{11}, r_{20}, r_{21}, r_{12}, r_{60}, r_{41},r_{51}, r_{32}, and finally r_{13}. So, the algorithm would move back and forth evaluating nodes at different generations.

Why is this important? Typically, in an EA the average fitness of the population and the maximum fitness in each generation grow as the generation number grows. In our forward chaining EA the first 3 individuals evaluated have an expected average fitness equal to the average fitness of the individuals at generation 0, and the same is true for the BC-EA. However, unlike for the forward-chaining EA, the fourth individual created and evaluated by BC-EA belongs to generation 1, so its fitness is expected to be higher than that of the previous individuals. Individuals 5 and 6 have same expected fitness in the two algorithms. However, the seventh individual drawn by BC-EA is a generation 2 individual, while the forward EA draws a generation 1 individual. So, again the BC-EA is expected to produce a higher fitness sample than the other EA. Of course, this process is not going to continue indefinitely, and at some point the individuals evaluated by BC-EA start being on average inferior. This is unavoidable since the sets of individuals sampled by the two algorithms are identical.

This behaviour is typical: for problems where fitness tends to increase generation after generation a BC-EA will converge faster than an ordinary EA in the first part of a run and slower in the second part. So, if one restricts oneself to that first phase, the BC-EA is not just faster than an ordinary EA because it avoids evaluating individuals neglected by tournament selection, *BC-EA is also a faster converging algorithm.* How can we make sure we work in the region where the BC-EA is superior to the corresponding forward EA? Simple: like in any ordinary EA, in a BC-EA one does not need to continue evolution until all the individuals in generation G are known and evaluated, e.g., we can stop the algorithm whenever the best fitness seen so far reaches a suitably high value. In this way we can avoid at least a part of the phase where BC-EA is slower converging than the forward EA.

It is worth noting that this faster convergence behaviour is present in a BC-EA irrespective of the value of the tournament size, although, of course, the differences in behaviour between the two algorithms depend on it.

6 Experimental Results

We have implemented a backward chaining version of genetic algorithm (BC-GA) and run a variety of experiments on the counting ones problem. The choice of algorithm and problem was simply dictated by simplicity, since the notion of BC-EA is completely general.

Let us start by corroborating experimentally the expected faster convergence behaviour of BC-EA. To assess this we performed 100 independent runs of both a backward and a forward chaining version of the algorithm applied to a 100-bit problem. In these runs the maximum number of generations G was set to 99 (i.e, we did 100 generations). The population size M was 100. Only tournament selection and mutation (mutation rate $p_m = 0.01$) were used. To make a comparison between the algorithms possible, in our BC-GA we computed maximum fitness and average fitness every 100 fitness evaluations, and we treated this interval as a generation. In the BC-GA we computed *all* the individuals at generation G.

Figure 3 shows the fitness vs. generation plots for the two algorithms when the tournament size n is 2. It is clear from the figure that BC-GA performs about 20% fewer fitness evaluations than the standard EA, reaching, however, the same average and maximum fitness values. So, as predicted in the previous sections this significant computational saving comes without altering in any substantial way the behaviour of the EA.

Figure 4 shows the fitness vs. generation plots for the two algorithms when the tournament size n is 3. With this tournament size, there is still a saving of about 6% which is definitely worth having, but clearly for higher selection pressures the disadvantages of using a BC-GA in terms of memory use and bookkeeping become quickly preponderant.

Similar results should be expected when using a two-offspring version of crossover, although, of course, higher fitness values would be observed.

These experiments illustrate how a BC-EA typically converges faster than the traditional version of the algorithm. However, the most important question about BC-EA is how the expected number of fitness evaluations changes as a function of M, n, and G. We investigate this in the remainder of this section.

In order to assess the impact of both the transient and the limit-distribution behaviour of BC-EA, we performed a series of experiments where the task was to evaluate just one individual at generation G (that is $m(0) = 1$). In these experiments, we set G to be big enough so that all transients had finished well before generation 0, thereby revealing also the limit-distribution sampling behaviour. In the experiments we used populations of size $M = 10$, $M = 1000$ and $M = 100\,000$. We set $G = 49$ (i.e., we performed exactly 50 generations, 0 through to 49), so forward runs required exactly 500, 50 000 and 5 000 000 fitness evaluations to complete, respectively. For each setting we did 100 independent runs.

Figure 5 shows the average proportion of individuals evaluated by BC-EA when *mutation only* is used ($\alpha = 1$) for tournaments sizes $n = 2$ and $n = 3$ as a function of the population size M, while Figure 6 shows the average proportion of individuals evaluated by BC-EA when *one-offspring crossover* with $p_c = 0.5$ is used ($\alpha = 1.5$) for the same tournaments sizes[8].

From these figures we can see that, as expected, the limit-distribution saving is largely independent from the size of the population. E.g., for $n = 2$, after the transient about 80% of the population is an "ancestor" of the individual of interest in generation G if mutation only is used, while this goes up to 94% when $\alpha = 1.5$. For EAs where long runs are used, these percentages provide an approximate estimation of the total proportion of fitness evaluations required by a backward chaining version of the algorithm w.r.t. the standard algorithm.

The figures also show that during most of the transient the number of individuals sampled by tournament selection grows very quickly (backward from generation 49). As clearly shown in Figure 7, the growth is exponential. The reasons for this are quite simple: when only a few samples are drawn from a

[8] Plots for crossovers producing two offspring in each application would show the same behaviour as the mutation only case

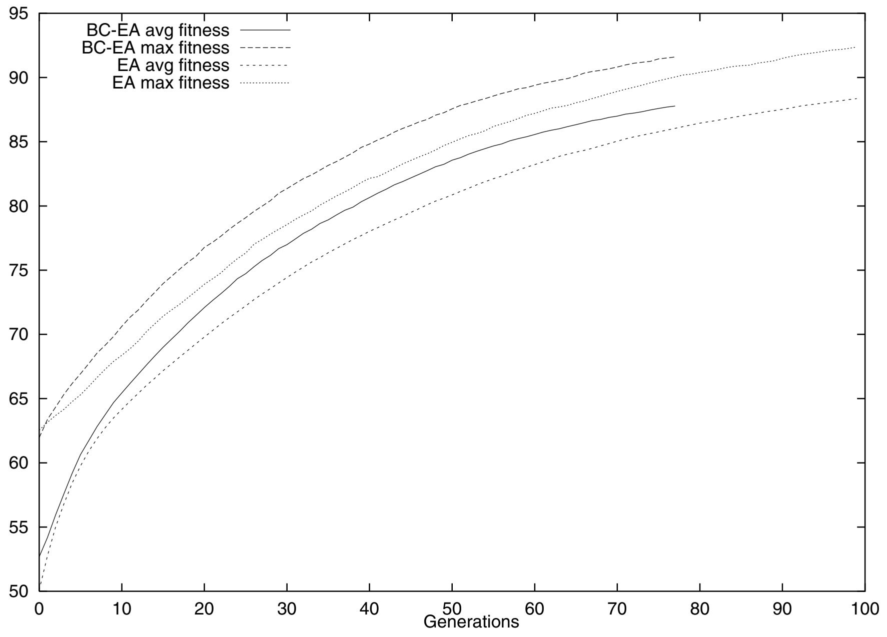

Fig. 3. Comparison between BC-GA and standard GA when a tournament size of 2 is used. Means over 100 independent runs

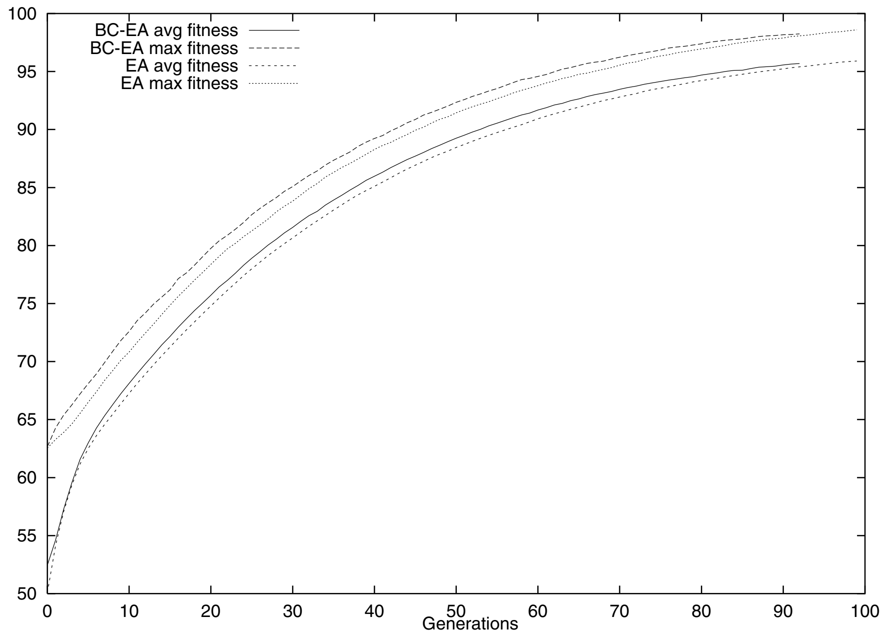

Fig. 4. Comparison between BC-GA and standard GA when a tournament size of 3 is used. Means over 100 independent runs

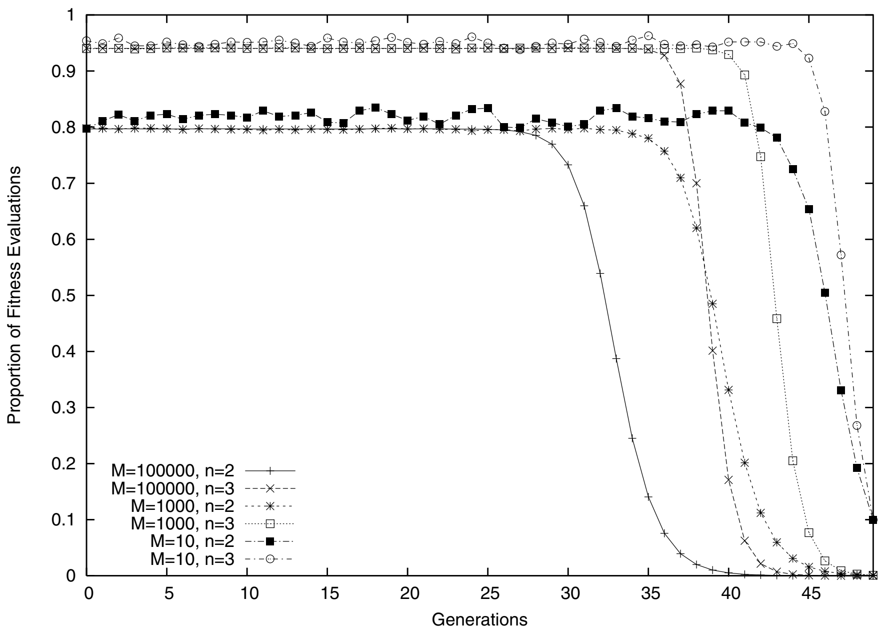

Fig. 5. Average proportion of individuals evaluated by BC-EA when mutation only is used ($\alpha = 1$) for tournaments sizes $n = 2$ and $n = 3$. Means over 100 independent runs

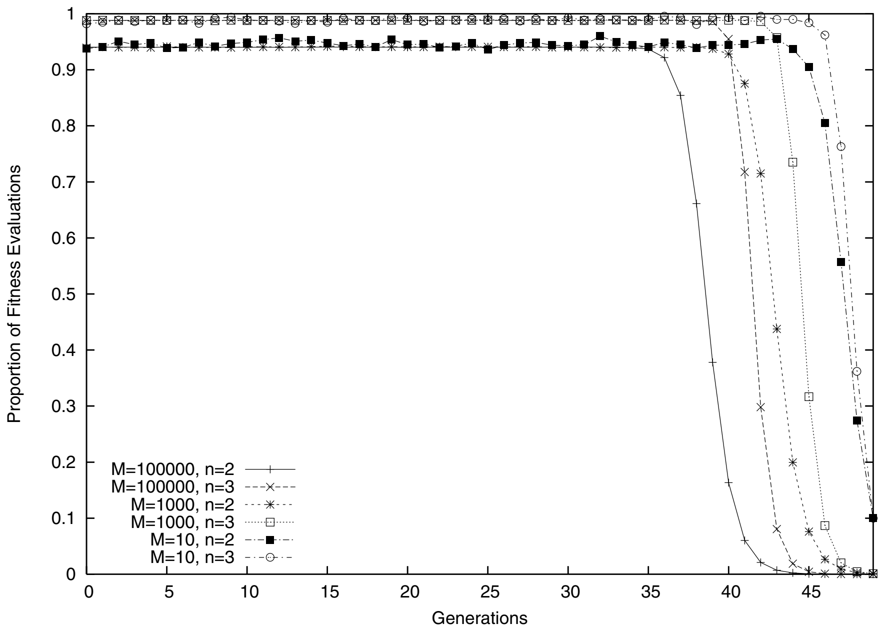

Fig. 6. Average proportion of individuals evaluated by BC-EA when one-offspring crossover with $p_c = 0.5$ is used ($\alpha = 1.5$) for tournaments sizes $n = 2$ and $n = 3$. Means over 100 independent runs

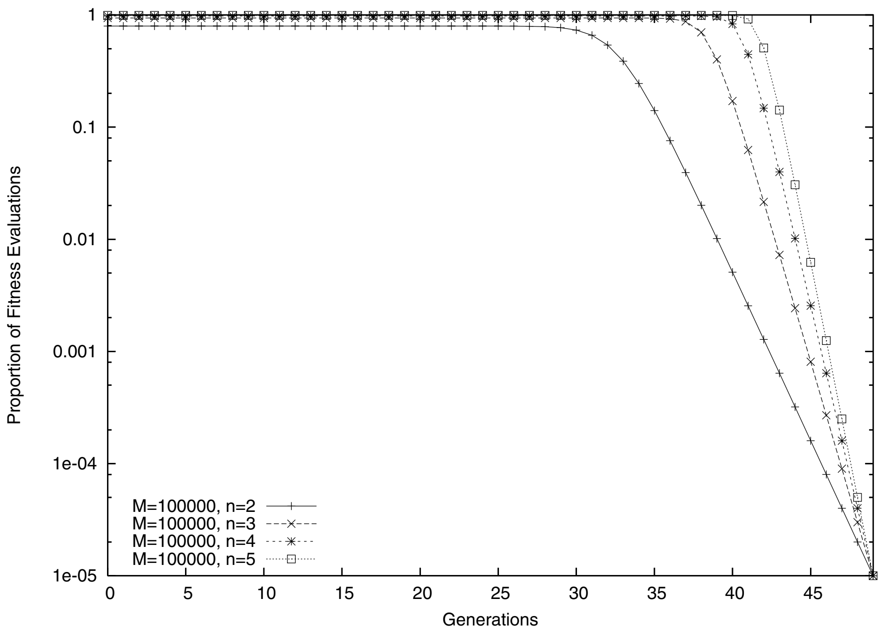

Fig. 7. Logarithmic plot of the average proportion of individuals evaluated by BC-EA when mutation only is used ($\alpha = 1$) for tournaments sizes $n = 2$–5 and a population size $M = 100\,000$

population, resampling is very unlikely, and, so, the ancestors of the individual of interest will tend to form a tree rather than a graph for at least some generations before the last. The branching factor of the tree is $n\alpha$. So, for small enough g, generation $G - g$ will include $(n\alpha)^g$ ancestors of the individual of interest in generation G. Naturally, this exponential growth continues only until $(n\alpha)^g$ becomes comparable with the expected number of individuals processed in the limit distribution case.

For small populations or high selective pressures the transient is short. However, there are cases where the transient lasts for many generations. For example, in a population of $M = 100\,000$ individuals and $\alpha = 1$ (i.e., in the case of mutation only or two-offspring crossover), the transient lasts for almost 20 generations (although it is exponential only for around 16 or 17). This population size may appear very big and these generation numbers may appear quite small. However, big populations and short runs are actually typical of at least one important class of EAs: genetic programming. So, it is worth evaluating the impact of the transient on the total number of fitness evaluations.

Let us assume $G - g_e$ is the last generation in which the transient is effectively exponential. (E.g., for $n = 2$, $\alpha = 1$, a population of size M and, of course, $m(0) = 1$, we have an exponential transient of $g_e \approx \log_2 M$ generations.) Then the number of individuals evaluated during the last g_e generations of a run is

$$F = \frac{(\alpha n)^{g_e+1} - 1}{\alpha n - 1}.$$

Because by definition $(\alpha n)^{g_e} \leq M$, we have the following simple upper bound for the number of individuals required in the exponential transient:

$$F < \left(\frac{\alpha n}{\alpha n - 1}\right) M$$

Then, based on the previous equation, if we ran our BC-EA for $G = g_e$ generations, whatever α and n the total number of fitness evaluations F is upper bounded by $2M$ – i.e., F is less than the effort required to initialise a population and create one generation in the standard version of the EA!

Even when runs last for more than g_e generations, the effects of the exponential transient are marked. To illustrate this, Table 1 reports the mean *total* number of fitness evaluations recorded in the experiments shown in Figures 5 and 6. Taking, for example, the case of $n = 2$ and no mutation, where the limit distribution effort would be around 80%, we can see that efforts of as low as 53.4% of those required by a forward EA are achieved.

Table 1. Mean number of fitness evaluations recorded during 50 generations in the experiments shown in Figures 5 and 6. Forward EA fitness evaluations are also reported for reference

M	BC-EA with mutation		BC-EA with two-offspring crossover		Forward EA
	$n = 2$	$n = 3$	$n = 2$	$n = 3$	
10	384	454	452	476	500
1 000	31 980	40 967	40 895	44 615	50 000
100 000	2 671 667	3 702 667	3 692 295	4 160 496	5 000 000

7 Discussion

In the previous sections we have developed a deeper understanding of the sampling behaviour of tournament selection, in particular focusing on a source of inefficiency: the creation and evaluation of individuals that cannot influence future generations. We have then proposed general methods to remove this source of inefficiency and speed up evolutionary algorithms based on tournament selection. One of these methods, the backward chaining evolutionary algorithm, provides the additional benefit of converging faster that a standard (forward) algorithm due to its constructing and evaluating individuals belonging to later generations sooner.

The implementation of a backward chaining evolutionary algorithm is not very complex and the added book keeping required is quite limited. However, there is no doubt that BC-EAs require more memory than their forward counterparts. So, if one did not want to leave the well known and safe terrain of forward chaining evolutionary algorithms, what could one do to mitigate the effects of the potential sampling inefficiency of tournament selection? One possibility is to modify the manner in which the "random samples" are taken for the tournaments: instead of using uniform random samples, which can sometimes fail to select a subset of the total sample domain, we could use "super-uniform" random samples, which mimic the expected behaviour of the uniform random sample.

In the context of tournament selection, and for the case of $\alpha = 1$ (e.g., in a mutation-only algorithm) the following method would capture the spirit of tournament selection while more uniformly distributing a subset of the samples used in its calculation. For the i-th tournament of size n, choose the first sample as the i-th member of the previous population, and choose the remaining $n - 1$ samples uniformly at random. Effectively, this amounts to choosing the first samples with a super-uniform distribution, since exactly M tournaments are required when $\alpha = 1$ [9]. Clearly it would be easy to adapt this scheme for the case $\alpha > 1$. For example, one could use the scheme just described for M of the required tournaments and then use standard tournament selection for the remaining $M(\alpha-1)$. With this revised version of tournament selection all samples would be used in the calculation of the next generation. So, this method is elitist, in that it is guaranteed to keep the best individual. As a result, the method may also present stronger selection pressure than the original. Future research will be needed to fully understand the properties of the proposed method.

If one, however, is prepared to adopt the ideas behind BC-EA, the computational savings can be very big. These are achievable not only when we exploit the transient behaviour of the algorithm (as we illustrated in the previous section), but also in the limit-distribution behaviour, as will be illustrated below.

Maximum savings are achieved when αn is minimum. The smallest value α can take is 1 and with standard tournament selection the minimum for n is 2. So, we already know that the best we can do is saving around 20% fitness evaluations. However, a form of tournament selection exists (e.g., see (Mitchell, 1996)) that we can modify to obtain even more spectacular savings.

In this form of tournament selection, one picks up two individuals at random and then chooses the one with the higher fitness with probability p, the other with probability $1 - p$. For $p = 1$ this form of selection is equivalent to standard tournament selection with $n = 2$, while it is a form of random selection for $p = 0.5$. By acting on p it is possible to vary the selection pressure of the method continuously between these two extremes. An alternative description of the method is that we choose the higher fitness individual with probability q and randomly between the two with probability $1 - q$ (naturally $p = q + (1 - q)/2 = (1 + q)/2$). In this case q can be varied in the interval $[0, 1]$.

This second version of the algorithm can be modified for our purposes. Instead of first choosing a pair of individuals and then deciding whether we select the best or we pick one at random, we first decide which selection strategy we are going to use, and then, based on this, we randomly draw individuals from the population. If we decide to go for the best in the tournament, then we must draw two individuals from the population. However, if we decide to choose randomly between the two members of the tournament, then we can just draw one

[9] Effectively, this method stands to standard tournament selection as stochastic universal selection stands to roulette wheel selection

random individual from the population (instead of drawing two individuals and then randomly discarding one).

With this method, the expected number of individuals drawn in each tournament is $n = 2 \times q + 1 \times (1 - q) = q + 1 \leq 2$. So, clearly the smaller q the bigger the saving we should expect in a BC-EA. Just to get a feel for the order of magnitude of these savings, let us assume $\alpha = 1$ and let us use Equation 1 to estimate the expected proportion of individuals not sampled. This is approximately $e^{-(1+q)}$. So, for very low selection pressures saving of over 35% fitness evaluations are possible.

Naturally, much more substantial savings can be obtained when exploiting the transient behaviour of BC-EAs. In the previous section we showed that when running a BC-EA with $m(0) = 1$, the effort is of the order of the population size. However, the reader will probably wonder about the usefulness of evaluating just one individual in the last generation. Normally we would want to have the whole generation. However, we need to remember that the individual provided by a BC-EA (with $m(0) = 1$) at generation G is effectively a random sample drawn from the population at that generation. Although we expect one individual to be insufficient, one important question is whether we really need to have the whole of generation G in order to solve a problem. Elsewhere (Poli and Langdon, 2005) we have experimented with a backward-chaining genetic programming implementation showing that even when run with $m(0) = 1$ BC-GP can solve problems well. So, the answer to the above mentioned question appears to be that, at least is some cases, we do not need the whole population. To get a more complete and satisfactory answer, future work on BC-EAs will need to include a thorough investigation of the best way to choose $m(0)$ and G.

As we already noted, BC-EAs are based on changing the order of various operations on an EA, which requires memorising choices and individuals over multiple generations. Let us evaluate the space complexity of BC-EA and compare it to the space complexity of a standard EA. For simplicity, we consider EAs where the representation of each individual requires a fixed amount of memory: b bytes. The space complexity of a forward generational EA is

$$C_F = 2 \times (b + 4) \times M$$

where we assumed that we store both the current and the new generation and that fitness values are stored in a vector of floats (4 byte each). So, for $b \gg 1$, $C_F \approx 2bM$. In BC-EA, instead, the space complexity is

$$C_B = G \times M \times (b + 4 + \frac{1}{8})$$

since we need to store one array of individuals, one of floats, and one bit array, all of size $G \times M$ [10]. So, for $b \gg 1$, $C_B \approx GbM$. So, the difference in space

[10] This calculation is based on an implementation where the graph structure induced by tournament selection on the population is not explicitly stored. Instead, it is created dynamically and recursively. The bit array is used to flag the individuals that have been constructed and evaluated in a previous recursions. The calculation ignores the small amount of memory required in the stack during recursion

complexity between the two algorithms is

$$\Delta C = C_B - C_F = (G-2) \times M \times (b+4) + \frac{GM}{8},$$

which indicates that in most conditions the use of BC-EA carries a significant memory overhead. However, this does not prevent the use of BC-EAs. For example, if the representation of an individual requires $b = 100$ bytes, and we run a population of $M = 1000$ individuals for $G = 100$ generations, BC-EA requires only around 9.6MB of memory to run.

8 Conclusions

In this paper we have analysed the sampling behaviour of tournament selection over multiple generations and exploited this analysis to come up with more efficient implementations of evolutionary algorithms based on this selection method. In particular we have proposed a new way of running evolutionary algorithms, the BC-EA, which offers a combination of fast convergence, increased efficiency in terms of fitness evaluations, complete statistical equivalence to a standard EA and broad applicability. Because of these interesting properties we think the class of BC-EAs is an area worthy of further investigation.

Acknowledgements

I would like to thank Bill Langdon, Chris Stephens and the referees for their extremely useful comments, criticisms and suggestions for improvements. Also, I would like to thank Bill for agreeing to give my talk at the workshop when I had to suddenly leave for family reasons.

References

T. Bäck, D. B. Fogel, and T. Michalewicz, editors. *Evolutionary Computation 1: Basic Algorithms and Operators.* Institute of Physics Publishing, 2000.

T. Blickle and L. Thiele. A mathematical analysis of tournament selection. In L. J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms (ICGA'95)*, pages 9–16, San Francisco, California, 1995. Morgan Kaufmann Publishers.

Tobias Blickle and Lothar Thiele. A comparison of selection schemes used in evolutionary algorithms. *Evolutionary Computation*, 4(4):361–394, 1997.

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

Kenneth A. De Jong, William M. Spears, and Diana F. Gordon. Using Markov chains to analyze GAFOs. In L. Darrell Whitley and Michael D. Vose, editors, *Proceedings of the Third Workshop on Foundations of Genetic Algorithms*, pages 115–138, San Francisco, July 31–August 2 1995. Morgan Kaufmann. ISBN 1-55860-356-5.

W. Feller. *An Introduction to Probability Theory and Its Applications*, volume 2. John Wiley, 1971.

Melanie Mitchell. *An introduction to genetic algorithms.* Cambridge MA: MIT Press, 1996.

Tatsuya Motoki. Calculating the expected loss of diversity of selection schemes. *Evolutionary Computation*, 10(4):397–422, 2002.

Allen E. Nix and Michael D. Vose. Modeling genetic algorithms with Markov chains. *Annals of Mathematics and Artificial Intelligence*, 5:79–88, 1992.

Riccardo Poli and William B. Langdon. Backward-chaining genetic programming. 2005.

Riccardo Poli, Jon E. Rowe, and Nicholas F. McPhee. Markov chain models for GP and variable-length GAs with homologous crossover. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, San Francisco, California, USA, 7-11 July 2001. Morgan Kaufmann.

Günter Rudolph. Convergence analysis of canonical genetic algorithm. *IEEE Transactions on Neural Networks*, 5(1):96–101, 1994.

S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prendice Hall, Englewood Cliffs, New Jersey, second edition, 2003.

Astro Teller and David Andre. Automatically choosing the number of fitness cases: The rational allocation of trials. In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 321–328, Stanford University, CA, USA, 13-16 July 1997. Morgan Kaufmann. URL http://www.cs.cmu.edu/afs/cs/usr/astro/public/papers/GR.ps.