Exhaustive Directed Search

Sanza Kazadi, Daniel Johnson, Jhanisus Melendez, Brian Goo

Jisan Research Institute
28 N Oak Ave
Pasadena, CA 91107

Abstract. We explore the development of an exhaustive directed search of state space based on concepts from evolutionary computation. A brief investigation of the evolvability of an evolutionary algorithm illustrates that evolutionary algorithms are capable of reaching optimal solutions when the diversification operator (which may be a *pseudo-operator* which acts over many different diversification steps) is capable of reaching, at every improvement point, another, more improved population element. Moreover, we demonstrate that the upper limit on the time to the optimal point is identical to that of an *exhaustive directed search.* This search is exhaustive, but borrows the diversification operator from the evolutionary algorithm and proceeds in such a way that, if left alone, it would exhaustively search the space. However, we demonstrate that this type of search can perform comparably with the evolutionary algorithm, avoiding deceptive search tracks that might trap an evolutionary algorithm.

1 Introduction

For nearly thirty years evolutionary systems have been under increasing investigation and scrutiny. A great deal of progress has occurred in the intervening years, with a great many different algorithms and methods being proposed and investigated [3,4,5,6]. Almost in parallel, and quite separately, optimization methods have been under extreme study, again resulting in the production of a number of promising methods [7]. Both groups of research papers have similar goals - general methods of performing the function optimizations, where the function may or may not correspond to a physical system, device, or performance. The main difference between the two approaches is the dependence on strict provable methodologies. Those functions for which provable optimization methods are available are limited, and some that provably work often times fail to reach global optima in acceptable computation times. Evolutionary systems, which may reach optima quickly, suffer from the lack of "provable" results in the sense that a global optima has been identified and is irrefutable without external knowledge.

In evolutionary systems, it is often assumed that provable optimization techniques cannot be applied, due to the highly complex nature of the fitness functions in question. As a result, a number of different methods have been used for optimization, though a careful anaysis of the number of computations expected is not carried out very often. Rather, an analysis of the amount of computation actually used is performed in some studies while other studies contain analyses of the quality of the optimization performed. However, the authors of this study are not aware of an analysis of the absolute required computation one might expect for a given evolutionary optimization method. Such an analysis would be very useful in generating upper bounds on the computation time of the evolutionary approach.

A central issue in optimization is the path through state space. Branch and bound methods use the exclusion of large subspaces during a walk through state space to direct the path through state space. Evolutionary methods are directed as a result of the absolute and relative qualities of the individual states. However, it is not currently known how one might determine whether or not a particular set of individuals is capable of being connected, in a reasonable computation time, to another individual with a higher state. Thus, it is not known how to determine when to terminate the current simulation outside of using a heuristic. Moreover, if a suitable "next step" is to be found, it is not clear that the time required wouldn't be comparable to that found using an exhaustive search of all potential solutions that might come from the use of a particular diversification [1] operator.

In this paper, we examine the paths through state space taken by a simple genetic algorithm and an exhaustive method called *directed exhaustive search.* We compare the two methods in terms of their expected or maximal completion times. We demonstrate that the performance of the genetic algorithm is comparable to that of the directed exhaustive search, though the GA's progress can be rerouted by deceptive evalutation functions, taking it away from possible improvement. Section 2 will explore the expected time fo the computation. Section 3 will examine the application of applicable algorithms to various functions containing single maxima with deceptive and non-deceptive fitness functions. Finally, Section 4 will offer some concluding remarks.

2 Searching Through State Space

2.1 The Evolutionary Algorithm

In general, the state of any evolutionary algorithm (EA) may be represented by a *single* vector $\vec{v}$. This vector may be separated into various sections, each of which carries with it a score commensurate with using that section as input to a fitness function (for instance if the vector represents multiple elements of a population). Under this type of formalism, the evolutionary algorithm can be characterized by two operators: diversification and reproduction. The diversification operator is responsible for changing the vector in a way that may include copies, swaps, and variations. We are not immediately concerned here with the

[1] Here diversification is used to refer to any operation which alters an individual, which may include crossover.

details of how one might produce any specific diversification, so we ignore this for the remainder of the study. The second operator specifically carries out copies of large parts of the vector commensurate with the relative scores of each of these parts of the vector.

Let us suppose, for concreteness, that the vector $\overrightarrow{v}$ is the open product of N other vectors $\overrightarrow{x_i}$. I.e.

$$\overrightarrow{v} = \overrightarrow{x_1} \otimes \overrightarrow{x_2} \otimes \cdots \otimes \overrightarrow{x_N}. \tag{1}$$

We may assume that each vector $\overrightarrow{x_i}$ is a complete system vector, representing a potential solution to the problem at hand. In a uniform population, all elements $\overrightarrow{x_i}$ will be identical. Now, let us suppose that all elements are part of a degenerate class C_i of identical or nearly identical fitness. The immediate question is the number of iterations required to move to another higher, potentially degenerate class of elements. For this to occur, one of the vector regions must transition to the next class. Let us assume that the probability of transferring to the next class C_{i+1} is $T_{i,i+1}$. Then, the expected number of changes to an individual sub-vector $\overrightarrow{x_i}$ required for a change to the next class is $\langle T \rangle = \frac{1}{T_{i,i+1}}$. The number of iterations required is thereby $\langle N_{it} \rangle = \frac{1}{NT_{i,i+1}}$ while the number of evaluations is still $\langle N_e \rangle = \frac{1}{T_{i,i+1}}$.

Now, suppose that we have a set of M different classes, each of which has a probability of transitioning to the next class given by $T_{i,i+1}$. Then the total expected number of evaluations required to complete the entire set of transitions is

$$\langle T_{total} \rangle = \sum_{i=1}^{M-1} \frac{1}{T_{i,i+1}} \tag{2}$$

This gives the total expected time that the evolutionary algorithm should take assuming that each step is capable of being reached with a single diversification. This, of course, assumes that all states are ordered, or that each state leads exactly to the next. Moreover, this assumes that no other competing states are present.

Let us assume that the diversification operator is discrete, and so the set of vectors capable of being reached by the diversification operator can be enumerated. Let us define the *diversification set of a class* C_i as the set of all vectors such that may be reached by an element of C_i by a single diversification. Let us denote this as $S(D, C_i)$ where D represents the diversification operator. Then, it is trivially the case that

$$C_i \subseteq S(D, C_i). \tag{3}$$

Let $B_{S(D,C_i),C_i}$ represent the set of all elements in $S(D, C_i)$ whose fitness exceeds that of C_i. Now, let us assume that the diversification operator is unweighted. Then, the probability of a diversification leading to a new vector whose fitness exceeds that of the original vector is

$$T_{i,i+1} = \frac{\left|B_{S(D,C_i),C_i}\right|}{|S(D, C_i)|} \tag{4}$$

where $|\bullet|$ represents the cardinality of the set. Thus, in particular, if each set C_i overlaps only C_{i-1} and C_{i+1} then

$$\langle T_{total} \rangle = \sum_{i=1}^{M-1} \frac{|S(D, C_i)|}{\left|B_{S(D,C_i),C_i}\right|} \tag{5}$$

As an example, in the case that we have binary vectors of length 50 elements, single point mutation, no crossover, and fitness values equal to the size of a block of ones starting from the most significant bit, this would reduce to

$$\langle T_{total} \rangle = \sum_{i=1}^{M-1} \frac{50}{1} = 49 * 50 = 2450 \tag{6}$$

assuming that the population started uniformly from a binary set containing only zeros.

2.2 Directed Exhaustive Search

In actuality, the power of the evolutionary algorithm comes from two different areas. First, the evolutionary algorithm is capable of exploring a number of different areas which may be remote from the current areas using diversity operators that are not necessarily bounded by locality. Second, the evolutionary algorithm is capable of focusing exploration on areas of relatively high fitness, or basing exploration on areas of relatively high fitness. Together, this can allow the algorithm to find optima that are remote from the given starting point, and to relatively quickly find those optima whose location in state space is close in mutation space to the current set of vectors.

Thus, in order to develop an exhaustive search algorithm, it is important to try to create an algorithm that is similar to the evolutionary algorithm in the sense that it can explore remote areas of the state space and that it can concentrate its efforts on areas that seem to be promising. It is also advantageous to correct the two main flaws with evolutionary algorithms: it cannot be proven that there are no other states that will improve the state of the algorithm, and so the algorithm will continue running until a "convergence condition" is reached; the evolutionary algorithm has no memory, and so when led down deceptive paths, has no way to recover[2].

[2] In order to have an idea of how damaging this last point might be, we can consider the completion of a complex evolution from a single starting point. If there are N branching points along the "correct" evolutionary path leading either down the path or down a dead-ending path, then the probability of choosing the correct path is p^N, assuming that each decision has a probability of p of going in the right direction. This means that the likelihood of making it to the optimal design falls off exponentially with increasing complexity, and any algorithm incapable of retracing its steps once the end of an incorrect path has been reached is increasingly unlikely to make it to the end. If there are only two choices per branching point, ten branching points

We define a *directed exhaustive search (DES)* in the following way. As with the EA, we assume the existence of a diversification operator D. Such an operator, we assume, has a countable number of potential states to which a vector may be mapped. As a result, all the states may be enumerated, and the operator's effects cycled through. We represent the set of all vectors which may be reached by D in a single mutation by $S(D, \vec{v})$. Now, as before, we suppose the existence of a set of vectors whose fitness exceeds that of the vector $\vec{v}$, and denote this by $B_{S(D,\vec{v}),\vec{v}}$.

The exhaustive search proceeds as follows:

1. The first vector is placed on a stack. The mutation number of the vector is initialized to zero.
2. If the mutation number is at the maximal mutation, the vector is popped of the stack and discarded. Go to step 3 otherwise go to Step 6.
3. If the stack is empty, stop. The maximum cannot be reached.
4. A vector is produced from the top vector on the stack using the vector's mutation number. If this has the maximum value, stop. The maximum has been reached.
5. If the new vector's fitness exceeds the top vector, it is added to the stack in order of its fitness. The new vector's mutation number is set to zero, and the stack pointer is reset to the top. If the maximal vector has been reached, stop.
6. The stack pointer is moved down.
7. Go to step 2.

Now, if a second vector is to be found, the number of evaluations cannot exceed $|S(D, \vec{v})| - \left|B_{S(D,\vec{v}),\vec{v}}\right|$. Thus, if a path containing an optimium is found, the number of computations required to reach the end of the path is bounded by

$$\begin{aligned} T_{total} &= \sum_{i=2}^{M} \left(|S(D, \vec{v}_i)| - \left|B_{S(D,\vec{v_i}),\vec{v_i}}\right| \right) \\ &\leq \sum_{i=2}^{M} |S(D, \vec{v}_i)| \,. \end{aligned} \tag{7}$$

where N is the number of states on the path. The two will be comparable depending on the relative size of $\left|B_{S(D,\vec{v_i}),\vec{v_i}}\right|$. If the size of this set is relatively large, then the evolutionary algorithm can be expected to converge significantly faster than the exhaustive search algorithm; that is, if the dispersion of $B_{S(D,\vec{v_i}),\vec{v_i}}$ in $S(D, \vec{v_i})$ is concentrated in a region that is unlikely to be found, the evolutionary algorithm may converge more quickly. Moreover, if the sizes of the sets

mean that the probability of making it to the end is less than 10^{-3}. This means that 1000 complete runs need to be implemented in order to be able to expect a single run to succeed.

$S(D, \overrightarrow{v_i})$ are identical (which depends on the nature of the diversity operator), then equation (7) reduces to

$$T_{total} \leq (M-1)\left|S(D, \overrightarrow{v}_i)\right|. \tag{8}$$

This is a hard upper limit on computation. As indicated above, the performance as compared to the expected performance of the evolutionary algorithm depends on the size of $B_{S(D,\overrightarrow{v_i}),\overrightarrow{v_i}}$. In fact, the two are related by

$$\frac{\langle T_{total}\rangle}{T_{total}} = \frac{|S(D, C_i)|}{\left|B_{S(D,C_i),C_i}\right| |S(D, v_i)|}. \tag{9}$$

This gives the expected relative performance of the two algorithms. As the size of $B_{S(D,C_i),C_i}$ is defined by the fitness function, and those of $S(D, v_i)$ and $S(D, C_i)$are defined by the diversity operator D, the relative performance is strongly affected by the diversification operator. Put another way, as the sizes of $S(D, v_i)$ and $S(D, C_i)$ increase, the relative advantage, if any, of the evolutionary algorithm decreases significantly, particularly if the growth of $S(D, C_i)$ outstrips that of $B_{S(D,C_i),C_i}$, which is likely to be the case.

3 Comparing Performance

We examine the performance of the two methodologies using a very simplistic GA and the DES algorithm. We choose as our sample space a binary search space of vectors of length fifty (50), one hundred (100), and one hundred fifty (150) digits. We choose a particularly simple diversification operator so that enumeration of the mutations is straightforward. In our case, this operator is a single point mutation operator. The use of a genetic algorithm which does not include crossover allows us to explore the effects in directly comparable ways. Our genetic algorithm utilizes a mutation probability of 10%, fifty individuals, and proportional selection.

We choose as our fitness functions two minimally deceptive functions and one very deceptive function. The functions are described as follows:

1. This function returns values equal to the size of the block of ones beginning at the most significant bit.
2. This function returns the value of the largest block of ones in the vector.
3. This function returns the size of the block of ones beginning at the most significant bit, up to half the size of the string. However, if the most significant bit-half of the number is populated by zeros, it returns a number equal to twice the number of one's beginning at the least significant bit. We refer to this function as the "Narrow Peak" function.

The first two functions are remarkably simple, though the first function has fewer paths to the maximum than does the second. The third is particularly deceptive, with the basin of attraction shrinking exponentially as a function of the overall search space size. We compare the overall performance when the algorithms are initialized with random vectors and when they are initialized with all-zero vectors. Because of their simple form, we can analyze the performance in terms of the convergence time and the percentage optimized, and compare these values to predicted expected completion times.

As indicated earlier, we can make rather specific predictions as to the number of expected evaluations needed for optimality for rather simple functions. In the case of function 1, we estimate forty-nine steps between the zero vector and the maximum vector. According to the analysis in Section 2.1, this should yield an average of 2450 evaluations for the maximal state to be reached with a stochastic algorithm. Moreover, predicted values for 100 and 150 elements are 9900 and 22350, respectively. Table 1 gives the actual evaluation times, and Fig.1 gives histograms of completion times for both the exhaustive and GA paradigms with random and non-random initial points. Note that Fig.1 excludes the uniformly distributed initial state for DES. This is because only one vector can be expected to be created using this algorithm, and so no interesting detail can be found about this performance from the graph.

Table 1. Performance of the DES and the GA on the first function for different vector sizes, with comparisons to predicted values for EA performance. The GA performance is very close to the predicted values (within one standard deviation). In this case, the DES does very well, finding the final values much faster than the GA. We initialize DES uniformly with a zero vector.

Bits	Pred. Eval.	DES	GA
50	2450	1275	2639 ± 332
100	9900	5050	9840 ± 1012
150	22350	11325	21570 ± 1864

This agreement between the GA completion times and the EA expectation times is striking. In each case, the average value is less than one standard deviation away from the predicted value. The DES algorithm performs very well on function 1, producing completion times (from an initial vector containing all zeros) well under both the predicted EA and actual EA performance.

Figure 2 illustrates similar data for function 2.

The Narrow Peak Function has a very different behavior than the other functions. This is because for a random sampling of vectors, it is very unlikely that the basin of attraction for the highest peak will be found. In fact, if the function has N components, then, $N2^{\frac{N}{2}}$ elements in a space of $2^N$are in the basin of attraction. That means that the relative size is $N2^{-\frac{N}{2}}$, which falls off as N increases. It is the case that for moderate values of N, it is very unlikely that this basin will be reached. Moreover, the probability of a single element from a GA reaching the basin of attraction is N^{-p} if the distance from the basin is p.

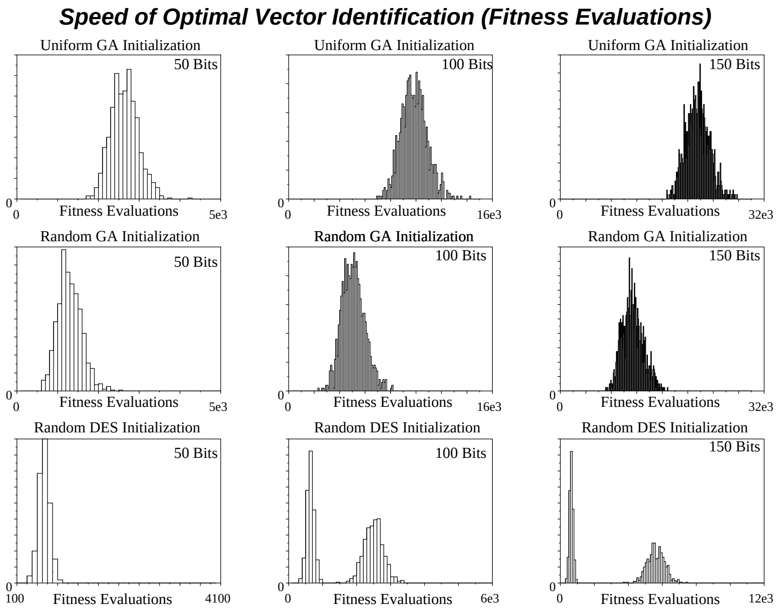

Fig. 1. This Figure illustrates the completion time of the algorithms on differently sized instances of Function1

As we've seen, this translates to an expected time to reach the basin (from static vectors) to N^p, which, for moderate values of N and p can be quite large.

As a result, we examine the performance of two different regimes on this function. The first is the random initial vector regime, and the second is the uniform zero vector regime. We report in Table 2 the performance of both algorithms on these regimes.

Table 2. performance of the DES and GA on the uniform initial state regime. While the GA runs faster than the DES when it finishes, owing to its not wasting time exploring the dead end, it moves toward the higher peak fewer times

Bits	DES It.	DES $Comp$	GA $Comp$	GA It.
50	2200	100%	67.8%	1439 ± 252
100	8775	100%	38.5%	5403 ± 723
150	19725	100%	33.4%	11707 ± 1261

As we can see in the table, the GA performance is faster than that of the DES, but less reliable. Moreover, the increase in speed seems to be less than a factor of

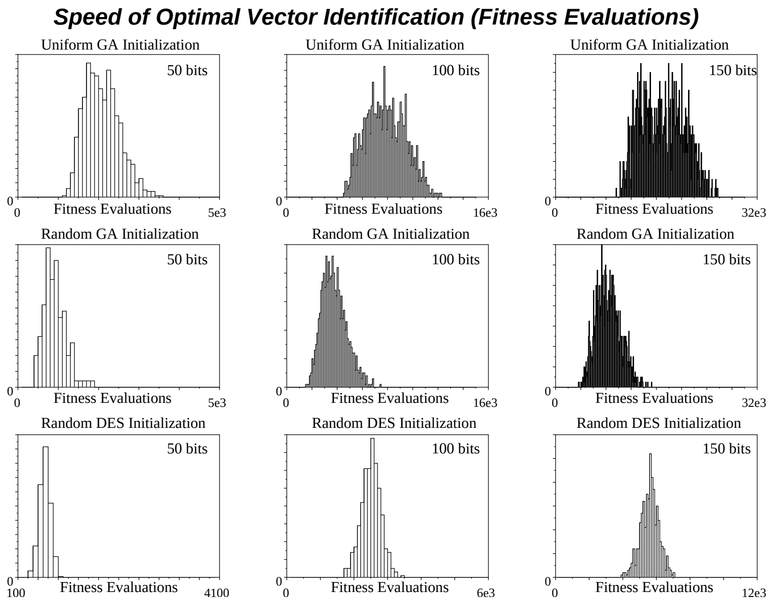

Fig. 2. This Figure illustrates the completion time of the algorithms on differently sized instances of Function 2

two in this case, which is an indication of the number of tracks traversed by the DES. On the other hand, the overall computation of the DES seems to decrease with increasing dimension (which is an indicator of problem complexity). Thus, in order to reach the final correct maximum, it would seem that the DES has a competitive advantage.

One interesting fact is that, as expected, neither the GA nor DES were able to find the optimum at all for $N > 10$. This is a reflection of the small basis of attraction.

4 Discussion and Concluding Remarks

This paper has primarily explored a single small detail in evolutionary systems. That is, that evolutionary systems have a specific amount of computation required to reach a global maximum, and that this computation time is predictable in the general sense. This is a very useful detail because it helps us understand how long one might expect an evolutionary algorithm to take before reaching a global maximum. Our suprising result is that it is possible to take an exhaustive approach using the genetic operators from evolutionary systems to explore the *entire space that the evolutionary system is likely to reach.* This exploration seems to require an amount of computation that is comparable (the same order of magnitude) to that required by an evolutionary algorithm to find the maximum, but does not seem to suffer from premature convergence, or trapping from deceptiveness. Moreover, while the method explores a limited search space, which may be slightly more limited than that explored carefully by the EA, the method is provable in the sense that one can be certain that at the completion of the search, the vector found is the optimum capable of being found by the diversification operator in question using the initial population of the EA.

The method can be compared to other exhaustive or directed methods, such as tabu search [2] or branch-and-bound [1] methods. In the first, certain steps in state space become taboo for a period of time based on their effectiveness in producing better vectors. This is meant to guarantee that the search algorithm moves beyond the local global maxima, and explores other maxima. The main weakness with this method is that once the taboo is removed, the vectors can come back to the same area, and there is no guarantee that the entire search space will be explored. Branch and bound methods require the development of a bounding condition which, at best, works for a small subset of potential problems. In our case, there is no limitation on which problems this may be applied to, and the method is not redundant in the sense that it creates loops which the system may go through as temporary restrictions are lifted.

Another criticism which may be levied against this method is its inability to handle neutral mutations, which have been seen to be very useful and important in evolutionary systems. However, it is indeed the case that the dependance on neutral mutations is, in effect, a dependence on an extended diversification operator which can take many more steps than the specific diversification operator in use. The remedy for handling such mutations is to create a more robust mutation operator capable of the desired number of mutations. As we have seen, the expected delivery time is not significantly lengthened by using the exhaustive method over strict evolutionary methods, so the potential increase in computation would seem to be unimportant.

We have also seen that the analysis presented here is capable of making rather accurate predictions of the behavior of the system under the action of a GA. However, this predictive power comes primarily from a knowledge of the fitness function, which gives us the number of vectors yielding improvements. Without this knowledge, the prediction of the behavior of the system's convergence time could not be done. Thus, it is an important consideration when designing evolutionary systems; limiting the number of plateaus would seem to be a requirement for quick convergence.

5 Acknowledgements

J. Melendez is supported by a grant from the Center for Neuromorphic Systems Engineering at the California Institute of Technology under NSF grant number EEC-0313658.

References

[1] Desmet, J., et. al.: The dead end elimination theorem and its use in protein side chain positioning. Letters to Nature. **356** (1992) 538–542

[2] Glover, F.: Tabu Search - Part I. ORSA Journal on Computing. **1 (3)** (1989) 190–206

[3] Kallel, L., Naudts, B., Rogers, A.: Theoretical Aspects of Evolutionary Computing. New York, New York: Springer-Verlag, 1998.

[4] Kazadi, S.: Conjugate Schema in Genetic Search. Proceedings of the Seventh International Conference on Genetic Algorithms. San Mateo, Ca: Morgan Kaufmann Publishers, pp. 10–17, 1997.

[5] Kazadi, S., Lee, D., Modi, R., Sy, J., Lue, W.: Levels of Compartamentalization in Artificial Evolution. Proceedings of GECCO 2000. pp. 841–849, 2000.

[6] Michalwicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs. New York, New York: Springer-Verlag, 1999.

[7] Rhee, S., Chung, J., Kazadi, S., Lin, H.: Conjugate Schema-Based Search. JRI Technical Report 1. 1999.