A Case for Exhaustive Optimization

S. Kazadi*
Jisan Research Institute
28 N. Oak Ave.
Pasadena, CA 91107
sanza@jisan.org

M. Lee
Jisan Research Institute
28 N. Oak Ave.
Pasadena, CA 91107
mlee@jisan.org

L. Lee
Jisan Research Institute
28 N. Oak Ave.
Pasadena, CA 91107
lauren@jisan.org

ABSTRACT

Evolutionary algorithms have enjoyed a great success in a variety of different fields ranging from numerical optimization to general creative design. However, to date, the question of why this success is possible has never been adequately determined. In this paper, we examine two algorithms, a genetic algorithm and a pseudo-exhaustive search algorithm dubbed Directed Exhaustive Search. We examine the GA's apparent ability to compound individual mutations, and its role in the GA's optimization. We then explore the use of the DES algorithm using a suitably altered mutation operator mimicking the GA's surreptitious compounding of the mutation operator. We find that the DES algorithm is capable of performing comparably to or outperforming the GA over all test problems, as predicted by theory.

1. INTRODUCTION

Evolutionary algorithms [6, 4] have been in existence for more than forty years. During that time, the uses that these algorithms have been applied to have been widely varied. Evolutionary algorithms are extremely useful on problems for which there is little known analytically, or for which very quick results are needed that might later be improved upon. Though a great deal of work has been done on the development of evolutionary algorithms, much basic work still needs to be carried out.

One basic question that seems to have been sidestepped is the question of how evolutionary algorithms compare to exhaustive algorithms. While this would seem to be a nonsensical question because exhaustive algorithms cannot realistically be expected to provide the optimum in a reasonable amount of time, one might ask the same of an evolutionary algorithm. Indeed, it is realistic to ask whether or not an exhaustive algorithm *that had access to approximately the same search space an evolutionary algorithm might have access to* would perform comparably to an evolutionary algorithm.

We view the search event similarly to the view expressed by Inman Harvey when discussing his generational genetic algorithms [3]. In this view, the path followed by the evolutionary algorithm is a bundle of overlapping paths that forms a tube through state space, eventually ending up at some local optimum. This paper therefore begins with an examination of the likelihood that either an exhaustive search or evolutionary search will make it to the best optimum in this case. Section 3 introduces an "exhaustive algorithm" that would seem to satisfy some of the design constraints of such an algorithm involved in optimization. Section 4 then explores some of the finer points of genetic algorithms which are beneficial in search. Section 5 reports the results of a comparative study of a GA and an adequately constructed DES. Finally, Section 6 provides some conclusions.

*To whom correspondences should be addressed.

2. THEORETICAL CONSIDERATIONS

We begin this section with a look at evolutionary pathways. We assume in these analyses that the evolutionary algorithm, much like an exhaustive algorithm, can only make progress if it is capable of finding vectors that outperform (in the fitness sense) other vectors using diversification operators (mutation and crossover being two favorites) perhaps repeatedly before selection culls these changes. These topics lead naturally to the question of how repeated improvement steps form paths, and how long it might take the average algorithm (or multiply restarted algorithm) to find its way to the best individual.

Search spaces contain a number of different points with respect to the search. Consider a point which can be reached from some initial point via a finite number of mutations. Let us assume that the new point's fitness is greater than the original point's fitness. Let us also assume that the number of mutations required is smaller than the limit imposed by selection. Then the original point is called an *interior point.* If no such new point exists, then the initial point is called a *local maximum.* Because we will consider paths through state space, we denote the local maximum as an *end point.* The optimization problem is a search through state space for the end point with the optimum fitness.

The interconnection of individuals through the diversification operator defines the structure of the state space. The optimization algorithm attempts to find a way through the state space from the initial point(s) to end points of hopefully near optimal score. We define a *path* to be a set of individuals which is ordered by fitness, connected by a finite number of diversification steps between adjacent vectors, and which contains an end point. The space is made up, generally, of interconnecting paths. Paths generally have

different lengths, though these lengths are typically much smaller than the number of individuals in the space. Let us define a *path bundle* as the set of all paths that start at the same initial point or group of points and end at the same point.

We can characterize a space as *simple* if all of the paths end at the same end point. We characterize the space as *complex* if the number of end points is a a large subset of the search space. As the term "large subset" is itself a characteristic tied to the magnitude of computational resources available to the researcher for the particular problem, the imprecision in this definition may be forgiven, leaving the term complex as a relative term.

Considering a single point in a search space, it is clear that the search space can generally be altered by the diversification operator into one of M different potential elements. We do not consider in this work the case that M is infinite. At each point only some of these M points will be improvements. We define $P_{\vec{v}}$ to be the number of points out of the total number of diversifications of the vector $\vec{v}$ which lead to an improvement in fitness. Typically, P is very large at earlier stages of the search while it becomes relatively small at later stages of the search. In an effort to simplify the analysis, we assume that P is static throughout the search.

Let us assume that we have two paths p and p' and that the two paths have differing end points. Then, at some point, the paths diverge. This means that the vectors making up the paths become so different that it is impossible for a vector on one of the paths to transition to a vector on the other path.

It is clear that there must be some points on each path which have the potentiality of being mutated into one or more of the points on the other path. It is at these points that optimization runs might "jump ship" from one path over to a the other path; beyond these points, the optimization is "locked in" to the path it is currently on. We call these points *transition points*. Contiguous sets of transition points are called *transition regions*, and we shall see that these have a great effect on the future performance of the algorithm.

In general, in transition regions, an active optimization will have the option of taking one or another direction leading to one of potentially many different end points. In what follows we use the simplifying assumption that each transition region forces the choice between k different path bundles. Moreover, we can assume that each bundle of paths is equally likely encounter m disparate transition regions before coming to an end point. Then the following theorem holds.

Directed Exhaustive Search Theorem: The expected number of evaluations required for an evolutionary optimization algorithm given the above assumptions is given by

$$\langle N_{\text{eval}} \rangle = mk^m \frac{M}{P} \tag{1}$$

whereas the greatest number of possible evaluations required to reach each end point in the search space is given by

$$N_{\max} = N_p m M \frac{k^m - 1}{m\,(k-1)}. \tag{2}$$

The proof of the DES Theorem is beyond the scope of this paper, but it will appear in an upcoming lengthened version of this paper.

The significance of this theorem is that if it is possible to explore each path bundle individually and without repetition, then it will be possible to duplicate the functionality of the evolutionary algorithm despite using an exhaustive search. Moreover, the number of evaluations one might expect to use in order to reach a specific optimum is considerably smaller using the exhaustive search than the evolutionary search if the number of paths explored per bundle is near one and the value of P is small.

Note that this result does not contradict the well known No Free Lunch Theorems [9, 10] as that result does not consider the number of evaluations required to reach the final result, but rather the number of *unique* evaluations. Since it is well known that EAs often times have redundant evaluations, the NFL Theorems cannot be used to directly evaluate the speed of convergence or the optimization speed.

3. AN EXHAUSTIVE SEARCH ALGORITHM - DES

In the previous section, we examined a model of the search space and the path through search space. In this model, we likened the search space to a set of interconnecting path bundles defined by the diversification operator whose function is tempered by selection. In this model, we identified local maxima which we called end points, interior points, path bundles, transition points, and transition regions. Using these we examined the expected number of evaluations required to explore all path bundles. The DES theorem describes the number of evaluations one might expect to use with an EA with potential restart after stagnation as well as the same for an exhaustive algorithm. Suprisingly, if the number of paths is restricted to one per bundle, the number of evaluations is comparable when considering the *worst exhaustive case* and the *average case for the EA*.

In this section, we first describe an exhaustive algorithm we call DES. This algorithm approximates the functionality of the algorithm described above. We then examine the aspects of DES more closely in order to better understand how this algorithm can be effectively deployed.

3.1 DES

DES is an exhaustive search algorithm in the sense that it will systematically explore all branches of the search space that may be reached using positive fitness transitions from the initial point(s) of the search space. DES *is not an exhaustive algorithm over the entire scope of the search space.*

DES maintains two data structures which help it to avoid duplicating evaluations of vectors and to keep track of where it is in the search. The first of these is an ordered linked list. This linked list contains all of the vectors currently

involved in the search, and it is from this list that the search progresses. The second data structure is a binary tree which maintains a record of all recently visited vectors. This keeps the search from revisiting individual vectors and entire branches once they have been visited. More on how this is achieved will appear later in this subsection.

DES uses a diversification operator which takes as an input a vector and returns a related vector which has one or more vector components altered. The diversification operator is assumed to be capable of a finite and enumerable set of potential changes to the vector. Each vector is bundled in a data structure which stores not only the vector, but also the current diversification that the vector is on. Each time a diversification occurs, this counter is increased, and the vector will be deleted from the linked list once the counter has reached the maximal count. Newly formed vectors will be added to the linked list if and only if their fitness exceeds that of the original vector.

The pseudocode is as given below.

```
initialize linked list with initial vector set.
store vectors in the binary tree.
set ptr to top of list.
set reset_random to 1
while (linked list is not empty or iteration number¡max iteration)
{
  increase iteration count
  if (diversification count ¡ maximal diversification count)
  {
    generate new vector and increase diversification count of current vector
    if (new vector isn't in binary tree)
    {       insert new vector in binary tree
      calculate new vector's fitness
      if (new vector's fitness ¿ old vector's fitness)
      {
        insert new vector in linked list
        reset pointer to top of the list
      }
    }
  }
  else
  {
    delete vector from linked list.
    advance pointer
  }
  every so often
    purge binary tree of unvisited vectors
  if (random number is greater than reset_random)
    reset pointer to top of linked list
  every so often
    recalculate reset_random based on length of linked list
}
```

This pseudocode implements a DES algorithm. The algorithm continues until the linked list is exhausted, which means that it has explored all potential paths from the initial positions. Three parameters affect the functionality of the algorithm, though they do not affect the completeness of the search. First, the frequency of binary tree purges affects the size of the binary tree. Having a binary tree with few elements makes the search recalculate many vectors, while a huge binary tree causes a concommittent reduction in efficiency. Empirically, we have found that a frequency and size that allows 10% of the vectors to remain in the binary tree per purge works well.

The second parameter is defined by the mechanism for assigning the reset_random parameter. If the reset_random number is very small, then the algorithm will tend to perform a depth-first search. If it is not small, then the algorithm will perform a parallel search (NOT a breadth-first search). Adaptivity is a useful characteristic which allows the same algorithm to perform well on a variety of different search spaces.

3.2 A closer inspection of DES' functionality

Perhaps the most important part of the algorithm is that it is capable of doing three things. The first thing is the generation of new vectors that are similar to those vectors that a genetic algorithm might be able to generate. The second is to limit the search of each of the bundles to a small number of paths in the bundle, preferably only one. Third, the algorithm must be able to recover when it reaches an end point, even one several optimization steps from another viable path to another end point.

All three objectives are accomplished using the DES algorithm. Firstly, as the diversification operator is borrowed from an EA, the diversification per step is identical to that from the same EA algorithm. What this does not take into account is the ability of the EA (as will be discussed below) to invoke multiple diversifications on a single vector. The remedy for this deficiency is to utilize diversification operators that invoke multiple steps within the DES. We shall return to this subject in the next two sections, as it can be demonstrated that this is a crucial property of evolutionary algorithms.

The second objective is accomplished using the binary tree. If multiple paths go through the same few vectors, then examining these vectors is sufficient to block subsequent exploration of these paths. That is, once these vectors have been added to the binary tree, they cannot be passed, and the path bundle is effectively blocked. Moreover, as the binary tree maintains vectors that are visited only, the vectors that can only be accessed beyond these few vectors will not be visited, and can be removed from the binary tree. This reduces the memory requirement and thereby improves the algorithm's speed. This capability requires that vectors are not removed from the binary tree "too quickly", as this will lead to their reinsertion in the binary tree and exploration of connected (already searched) regions.

The third objective follows from the parallel nature of the algorithm. Since the algorithm cannot retry already tried areas of the search space, but maintains vectors from potentially very different paths and path bundles, the end of a path leads to the associated vectors' deletion from the search list. However, this does not affect the other threads of search. Thus, the search continues unabated along multiple path bundles, and the algorithm avoids local maxima

traps.

As indicated above, one important property of the algorithm is that it does not take backward steps beyond the range of the diversification operator. This means that it is impossible for the algorithm to move backward in fitness and explore paths not connected to the initial points. However, at early stages of EA runs, the run tends to make very quick improvements which can put severe strain on backwards-reaching vectors. The inclusion of backwards-reaching vectors, or those that arise from them, which cannot be reached by an extended mutation operator is unlikely.

4. THE REACH OF THE GENETIC ALGORITHM

In the remainder of this study, we describe computational experiments done utilizing a genetic algorithm and DES on various computational problems and a set of neural network training problems. In this section, we examine the GA's performance on these problems when the range of mutations is limited and unlimited.

In these studies, we utilize a genetic algorithm which uses a discrete decimal encoding for the vectors. All vectors are represented by integer arrays whose elements range between 0 and 9. These vectors are broken into groups of three elements whose values represent the real values of problem parameters. For instance, the string '342028' might represent two parameters of value 3.42 and 0.28. Our genetic algorithm has a population size of 1000, proportional selection, and utilizes a mutation operator only as a diversification operator. Our mutation randomly changes a single element at a time, allowing multiple mutations to occur only if the mutation hits the same vector multiple times. The mutation probability is 0.05. We use a uniform selection in our genetic algorithm. Each DES and GA run continues for 10^6 evaluations[1].

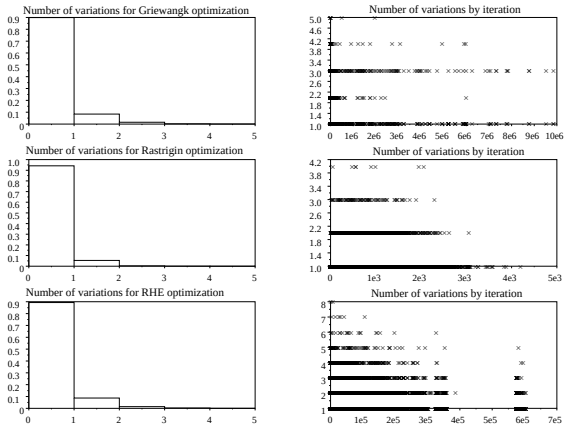

Figure 1: This figure illustrates the number of mutations that occur on a single vector between improvements. We represent it in a histogram (on the left) and as a time series (on the right). Note that all of the algorithms make use of multiple mutations during large portions of the search.

[1]Note that evaluations are not the same as iterations. With a mutation probability of 0.05 and a population size of 1000, this means an average of fifty evaluations per iteration

When utilizing a population-based EA which does not utilize replacement after each selection / diversification event, individual vectors can have multiple diversifications. This, in effect, extends the range of the algorithm. That is, the algorithm benefits from this because it can utilize diversifications that cause more significant changes to the vector than a single element diversification by combining the effects of each of the diversification events. This can have a significant effect on the final outcome of the search algorithm.

Consider, for instance, the data in Figure 1, which present data taken during the genetic algorithm optimization of three functions taken from the literature [7] (the Rastrigin, the RHE, and the Griewangk functions). These data illustrate the number of changes given in a specific vector between improvements. Note that many of the algorithms have large numbers of changes, which are not necessarily limited by the method of change. That is, our mutation operator changes only a single element each time. However, the selection operator is capable of removing only a few vectors per iteration, allowing mutations to accrue in different generations. The RHE function shows that a large number of mutation events can accrue. The others use many fewer mutations to get variations demonstrating that each function defines its own best mutation rate. Often times, when single mutations no longer seem to provide improvements, one or more multiple mutations will occur, restarting the single mutations.

5. NUMERICAL EXPERIMENTS

In the previous section, we examined evolutionary systems and found that evolutionary algorithms tend to execute multiple steps surreptitiously. Our goal in this section is to observe that an exhaustive algorithm that mimics this behavior can perform comparably well to an EA. Each run records the best vector during a 10^6 or 10^7 evaluation evoluationary simulation. We evaluate these vectors by reporting the mean, standard deviation, and minimum and maximum values out of the total number of runs. Because of computational limitations, we report fewer than 100 runs on some problems. More runs would have meant a more accurate mean and may have extended the maximum and minimum values, but we do not believe that they would have altered the general results significantly.

A practical optimization task for evolutionary algorithms is to train neural networks. We adopt this as a test problem. Our training data is made from networks with randomly generated weights. Networks that we train should appproximate the performance of the old networks. The networks we use are feed-foward, fully-connected, and 3-layered with arctan as the node function. For our preliminary tests, we use 2 networks: one that has 2 input nodes, 1 hidden node, and 1 output node, and another that has 3 input, 3 hidden, and 3 output nodes. The job of the search algorithms, given the random inputs used to train the networks, is to find the network weights. The score that is produced is the sum of the deviation between the output values and the training data output values. If the search algorithm can generate a

score of 0 (or very near 0 because of rounding done throughout the process), we say that the algorithm has generated an optimal network. Our weights are restricted to numbers between -5 and 5.

In our tests the two algorithms we use two neural networks. Because of the simplicity of the 9-dimensional neural network, both the DES and the GA reached the optimum, a score of 0.00808877. However, the more complex 54-dimensional network did not reach an optimal score in 10^6 evaluations. Because the DES and the GA were both able to find the optimum for the simple neural network problem, we can compare their performance using the number of evaluations it took for the algorithms to get to that specific score. The DES was able to reach that optimal score in 14029 evaluations (standard deviation of $\pm$12421.65) while the GA found that same score in 124875 evaluations (standard deviation of $\pm$194825).

We also test the DES using the problems mentioned in section 4. We allow it to mutate up to three components at once. For each run, the DES performs 10^7 iterations. In most cases, the DES outperforms the GA. In all other cases, the DES performs comparably well.

6. SUMMARY AND CONCLUDING REMARKS

This study was motivated by the discovery of the DES Theorem, which indicates that there are cases in which "exhaustive search algorithms" can perform nearly as well as, or potentially outperform, evolutionary algorithms. Suitably designed "exhaustive algorithms" might be designed based on the very operators that their evolutionary counterparts use, and would seem to have the potential to outperform the evolutionary algorithm counterparts in terms of their reliability and, in some cases, speed. This raises the question of whether or not it is possible to remove some of the inefficiencies in evolutionary algorithms by constructing their "exhaustive counterparts".

We have seen that much of the performance of evolutionary algorithms can be reproduced using our DES. As an example, we have noted that the performance of the DES is comparable to or superior to its evolutionary counterpart in the same number of function evaluations. We have also seen, anecdotally, that longer runs in the DES tend to produce better results, in agreement with a recent study from Cantú-Paz and Goldberg [1]. Future work can proceed in a number of ways. One might be to explore the class of functions over which the DES seems to outperform the EA (which might be those classes with small P values and narrow path bundles). Another might be to explore for which functions various methodologies found in the literature can be brought over to DES, and which cannot (one which cannot is Uniform Crossover [8]). Finally, one potential use of this algorithm might be to apply it to practical problems to which EAs have been applied and report on its comparative performance. We intend to pursue all of these fronts.

7. ACKNOWLEDGEMENTS

We would like to thank D. Zitter for his assistance in both this study and in preparing data for this paper. D. Zitter will be a contributer to the longer version of this paper.

8. REFERENCES

[1] E. Cantú-Paz and D. Goldberg. *Are multiple runs of genetic algorithms better than one?* **Proceedings, Genetic and Evolutionary Computation Conference**, p. 801-812, Chicago, Il., USA, July 2003.

[2] D. Fogel. **Evolutionary computation: Principles and Practices for Signal Processing**. Bellingham: Spie Press. 2000.

[3] I. Harvey. *Evolutionary robotics and SAGA: the case for hill crawling and tournament selection* C. Langton (ed.), **Artificial Life III, Santa Fe Institute Studies in the Sciences of Complexity**, Proc. Vol. XVI, New York: Addison Wesley, pp. 299-326., 1994.

[4] L. Kallel, B. Naudts, A. Rogers, (eds). **Theoretical Aspects of Evolutionary Computing**. New York: Springer-Verlag. 1998.

[5] S. Kazadi, D. Johnson, J. Melendez, B. Goo. *Exhaustive Directed Search.* **Proceedings of the Genetic and Evolutionary Computation Conference, 2004** , Seattle, WA, USA, 2004.

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

[7] S. Rana and L. Whitley. *Bit representations with a twist.* **International Conference on Genetic Algorithms**, Ann Arbor, Michigan, 1997.

[8] G. Syswerda. *Uniform crossover in genetic algorithms.***Proceedings of the 3rd International Conference on Genetic Algorithms**, p. 2-7, Fairfax, Virginia, 1989.

[9] D. Wolpert and W. Macready. *No free lunch theorem for search.* **Technical Report, SFI-TR-05-010**, available at www.santafe.edu, 1995.

[10] D. Wolpert, W. Macready. *No free lunch theorems for optimization.* **IEEE Trans. Evolutionary Computation** 1(1): 67-82 (1997)

[11] X. Yao (1993a), *A review of evolutionary artificial neural networks.* **International Journal of Intelligent Systems**, 8(4):539–567, 1993.

SIMULATION DATA				
Problem	GA/DES	dim	min/max	mean ±stdev
Griewangk	DES	5	0, 0.008025	0.004170 ±0.002011
Griewangk	GA	5	0, 0.02297	0.008589 ±0.004093
Griewangk	DES	10	0.001403, 0.01038	0.006851 ±0.004366
Griewangk	GA	10	0.001768, 0.07274	0.01939 ±0.01528
Rastrigin	DES	5	0, 0.0006241	0.0002496 ±0.0001861
Rastrigin	GA	5	0, 0.001041	0.0005084 ±0.0002394
Rastrigin	DES	10	0.0004160, 0.0008321	0.0006241 ±0.0002942
Rastrigin	GA	10	0.0002060, 0.001665	0.0009979 ±0.0003197
RHE	DES	5	0.0001760, 0.001056	0.0003813 ±0.0001999
RHE	GA	5	0.000172, 0.002576	0.0005667 ±0.0003780
RHE	DES	10	0.0007039, 0.007743	0.002816 ±0.0033295
RHE	GA	10	0.0005150, 0.004293	0.001238 ±0.0006487
Neural Network	DES	9	0.008089, 0.008089	0.008089±0
Neural Network	GA	9	0.008089, 0.008089	0.008089±0
Neural Network	DES	54	0.1290, 0.2099	0.1551 ±0.1315
Neural Network	GA	54	0.1287, 0.7034	0.1627 ±0.0556

Table 1: This Table reports the performance of the GA and DES on various problems for several different dimensions.