The BQP Problem and Exhaustive Search Algorithms Based on Evolutionary Algorithms

ABSTRACT

We theoretically explore some of the properties of evolutionary algorithms (EAs). We discover that under certain conditions, it is more advantageous to utilize a restarting procedure for the evolutionary algorithm than to continue to allow the algorithm to run due to an exponentially increasing time required for transitions between optima. In the cases that restarting is indicated, we discover that an exhaustive algorithm based on a given evolutionary algorithm may be able to outperform the evolutionary algorithm on which it is based. We demonstrate the application of this algorithm to the BQP problem on test problems found in the literature, recovering the best performance reported in the literature.

track:evolutionary combinatorial optimization

1. INTRODUCTION

Evolutionary algorithms (EAs) are search methods that are inspired by natural selection[17, 18], typically operating on populations of individuals. EAs typically function in cycles known as generations during which individuals from one population are used to create a new population, using a variational process and a selection process which allow those elements of the population scoring well on a real-valued evaluation metric to contribute to the next population. Continual application of these cycles eventually produces populations filled with high scoring individuals.

Recently, researchers have been using memetic algorithms which combine exhaustive search algorithms with evolutionary algorithms[11]. The result has been algorithms that are quite good at discovering optima that neither algorithm can find individually in comparable computation times. This indicates that properties of evolutionary algorithms exist that can be useful to exhaustive algorithms, and vice-versa. We believe that the completeness of the search of an exhaustive algorithm is its strength while the adaptivity of the evolutionary algorithm is its strength. Combining these two properties has proven to be a useful technique in optimization.

While the marriage of these two methodologies seems to create an algorithm with capabilities that exceed those of either one, this by no means indicates that it is impossible to have a similar effect in approaching the same thing from another point of view. In this paper, we examine an algorithm that is more exhaustive than evolutionary, but contains the strong points from both parts of the standard memetic algorithm. We demonstrate that the algorithm has performance that is at worst comparable to an evolutionary algorithm and at best much superior to that of an evolutionary algorithm constructed with the same evolutionary operators.

We motivate our work in this paper by first theoretically examining evolutionary algorithms in general. We explore the performance of an evolutionary algorithm, characterizing it as a combination of two processes - traversing a pathway to an optimum and transitioning from one pathway to another. We demonstrate that under certain conditions, using long runs can be advantageous, while under other conditions, restarting the algorithm is more advantageous. The transition regions between local optima are defined and used to explore the conditions under which either strategy is preferred.

Two corollaries of the current theoretical formalism give rather suprising results. The first is that under certain conditions, it is a mathematical certainty that the genetic algorithm will *always find the optimum of the search space*. The conditions under which this is true are given. The second theoretical result is that a dynamic change of encoding, under certain conditions, is advantageous, and can be exponentially so. While these results are not further explored in this work, they do serve to clarify some of the empirical results that have appeared over the years.

In applying our algorithm in this paper, we focus on the *binary quadratic programming* (BQP) problem [11, 2, 3, 5] . This problem can be defined in the following way: Given a real symmetric square matrix M of rank n, find a rank n binary vector $\vec{v}$ such that $\vec{v^T} M \vec{v}$ is maximized. The problem finds uses in a wide variety of fields including financial analysis [9], traffic management [4], machine scheduling [1], molecular conformation [12], and many others. BQP has been studied in the context of both conventional optimization algorithms and evolutionary algorithms[14]. This paper develops a new optimization method known as *Directed Pseudoexhaustive Search* and applies it to the BQP problem, demonstrating the algorithm's ability to perform the search as accurately as both conventional and evolutionary algorithms[7, 8].

The remainder of this paper is organized as follows. Sections 2 and 3 examine theoretical issues related to evolutionary search. Section 4 explores the BQP problem and describes our search algorithm. Section 5 discusses the simulation results and their meaning in terms of the greater EA literature. Section 6 offers some concluding remarks.

2. THEORETICAL CONSIDERATIONS

Our examination begins with a description of the basic assumptions of the class of evolutionary algorithms examined here. Once the basic assumptions have been clarified, we describe an *extended diversification operate* which is a generalized mutation/crossover operator. This operator is tied in with the reproduction operator, and can be used to understand why the genetic algorithm is not generally limited to specific subspaces[6].

We continue with an examination of the meaning of different representations in optimization algorithms. This examination leads to the idea that optimization algorithms behaving like evolutionary algorithms tend to move between regions of the search space. The movement is essentially connected to the connectiveness of the space, which itself is a result of the representation. The representation, therefore, is important in the sense that different regions of the search space may be "further" or "closer", according to the number of steps required to reach them, as a result of which representation is used. This view of representation is used to determine the effect of the various random representations.

In what follows, we omit the proofs of the various propositions and corollaries as space is limited. The proofs will be included in an upcoming longer version of this paper.

2.1 Extended Diversifications

We assume in the following work that the systems to which we apply this formalism have the following properties: (1) The algorithm maintains a population, which is a set of vectors, that are stored in memory. The vectors are stored in memory and acted on by operators which define the evolutionary algorithm. The state of the evolutionary algorithm is defined by the population, which is a set of individual vectors stored in memory, and the current operator being employed. (2) The diversification operator(s) introduce new elements to the population using a combination of crossover and mutation events. Diversification operators do not include selection. (3) Selection culls the population by replacing some elements with others, preferentially replacing lower scoring individuals with higher scoring individuals. (4) We assume we are working with a finite search space Γ. The details of these behaviors are not important for the arguments made below, though the arguments made from here on out will be true as long as these assumptions are true.

There are, of course, two radically different types of populations. In the first type, the population is finite (which is required for most practical applications involving EAs), and in the second, the population is infinite. In this paper, we will explore the finite population case. In this case, the population has a specific number of elements which is maintained by the evolutionary algorithm. Additions of new elements to the population must be accompanied by removals of elements from the population.

The diversification operator generally operates in the following way: based on the population (or in some cases independently of the population), the diversification creates new vectors for consideration. These vectors either immediately become part of the population, replacing vectors in the population, or are subject some culling. The methods for including these are varied. They may include finding a vector whose score is lower than the new vector and replacing it, or randomly choosing a new vector to replace. When this step is combined with selection one typically finds a lower-scoring vector to replace with the new vector. If one cannot be found, the new vector is not added to the population. Generational selection typically consists of a removal of the lower scoring individuals from the population and replacement either with an individual produced by a diversification event or by an individual which is a copy of an existing individual in the population.

Diversification operators can be limited in the sense that a single application cannot transform any given vector into any other given vector in the search space. As an example diversification operators derived from single point mutation and any finite-point crossover operations cannot change all binary vectors into any other binary vector. In this case, it is the repeated action of the diversification operator which allows the vector to be changed completely from one vector to another vector. In population-based search algorithms, it is possible for diversification steps to act additively, extending the capability of the diversification operator. We call such a pseudo-operator an ***extended diversification operator*** and each sequence of connected diversifications an ***extended diversification event***.

Let us now take the original population of N elements and enumerate them. We represent these as $\{x_1, \ldots, x_N\}$. Then, each element discovered by the extended diversification operator can be added to this list of elements. Thus, if M extended diversification events have occurred, then the sequence of elements is given by $\{x_1, \ldots, x_{N+M}\}$. We may think of the current population as being the subset of elements from this sequence of elements that is being considered when the next element is developed. Thus, we may write the next element as a function of the current sequence, with an emphasis on the current population. That is, we may write a recursion relation as

$$x_{N+M+1} = f\left(P; x_1, \ldots, x_{N+M}\right) \tag{1}$$

where f is the diversification operator. Let us designate the entire set of numbers a sequence X.

What this means, then, is that the entire evolutionary algorithm can be likened to a method for generating an infinite sequence. The sequence of numbers is complex to analyze, but it is still a *deterministic* sequence.

The main problem is that the sequence is often times a *repeating* sequence of vectors containing multiple copies of many of the elements[6, 18]. Simply because an element has been removed from the population does not require this element to never again reappear in the population. Let us now consider the subsequence of elements of X which do not appear earlier in the sequence. Let us designate this sequence of elements as $Y = \{y_i\}_{i=1}^{Max}$ where Max represents the final unique discovery in the sequence X. For finite spaces, Max is finite; for infinite spaces, Max may be infinite.[1]

We can then define different types of algorithms. An optimization algorithm can be defined as *eventually stagnant* if $Max < |\Gamma|$ where Γ is the number of elements in the entire search space. An example of an algorithm that is eventually stagnant no matter the starting point is a hillclimbing algorithm, no matter the number of starting vectors or the position of the starting vectors.

One important question is whether or not evolutionary algorithms are eventually stagnant. This is dependant on the evolutionary algorithm's diversification operator. The following proposition addresses the future of optimization algorithms whose diversification operators or extended operators have no limitation in their reach. Let us define the probability of an extended diversification operator D changing vector $\overrightarrow{v_1}$ to vector $\overrightarrow{v_2}$ as $p_{D(\overrightarrow{v_1}, \overrightarrow{v_2})}$.

PROPOSITION 1 *Suppose that an evolutionary algorithm has an extended diversification operator D such that given any two vectors $\overrightarrow{v_1}$ and $\overrightarrow{v_2}$ in the search space Γ, $p_{D(\overrightarrow{v_1}, \overrightarrow{v_2})} > 0$. Then the evolutionary algorithm is not eventually stagnant.*

The importance of this proposition comes from its application to the optimization, and forms the motivation for the use of evolutionary algorithms. The following Corollary illustrates its use in optimization.

COROLLARY 1 *An evolutionary algorithm which has an extended diversification operator d such that given any two vectors $\overrightarrow{v_1}$ and $\overrightarrow{v_2}$ in the search space Γ, $p_{D(\overrightarrow{v_1}, \overrightarrow{v_2})} > 0$ will always find the space's optimum.*

Corollary 1 provides a clear description of the motivation for using evolutionary algorithms. It is clear that, using evolutionary algorithms, the optimum will eventually be visited by the algorithm. In order to do this, it is merely necessary to either construct a diversification operator that, in one step, has a p which is nonzero for all possible mutations, or an extended diversification operator which does the same.

2.2 Representational issues

In fact, most evolutionary algorithms have a limited diversification capability. Binary mutation operators and crossover operators can typically not change any single vector into any other in a single iteration. However, repeated uses of the operators have this effect, and so the extended diversification operator can be seen to also have positive p in this case.

Proposition 1 and Corollary 1 also indicate that the longer the optimization operator is run, the better the result, as even those better results that are unlikely to be visited can be found, given enough time. This result provides the theoretical motivation for the recent result of Cantu-Paz and Goldberg [6] in which the same result was theoretically and empirically found. [2]

Let us assume that we have a population P, which is a subset of the sequence of elements X. In general, this population is made up of copies of the maximal element in the population, and other members that do not score as well as the maximal element. We may view these as two types of elements. The first is the current best, and the others are intermediate states of a diversification event seeking a new best. In systems involving elitism, the current best is a protected element in that it will not be replaced by a less fit individual. In those not employing elitism, the best individual is not protected, and so it can be replaced by a less fit individual, depending on the selection operator employed. In what follows, we will assume that the methodology utilizes elitism.

We define a *basin of attraction of a local maximum* to be the the maximal set of vectors such that (1) the set is an open set in the set theoretic sense, (2) the local maximum is in the interior of the set, and (3) the fitness values of all the elements in the set are lower than that of the local maximum. We define the ***depth of a basin of attraction around a vector*** $\overrightarrow{v}$ to be the minimum

[1]Note that this sequence is identical to the one used in the arguments for the No Free Lunch theorems.

[2]Despite the positive impact of these considerations, the simple fact is that no indication is given as to how long one might expect to wait in order to obtain these improvements.

number of diversification steps required during a diversification event in order to reach another vector whose fitness value is greater than the local maximum.

Using these definitions, we can generate the following immediate consequences.

PROPOSITION 2 *Suppose that an evolutionary algorithm utilizes a diversification operator in which each separate step is uncorrelated, unbiased, and chooses between the same number of potential alternatives. Then the computation time required to find a vector outside of the basin of attraction increases at least exponentially with the depth of the basin.*

This result is important because it indicates that an evolutionary algorithm is likely to get trapped in a basin of attraction for an exponentially increasing amount of time, depending on the depth of the basin. Thus, even though the algorithm cannot be trapped, as Corollary 1 indicated, the algorithm's completion time tends to become continually larger.

Since basins of attraction have depths that are functions of the diversification operator, and the diversification operator is a function of the encoding of the search operators, the encoding of the space has a very large effect on the search algorithm. Some encodings will cause some basins of attraction to be very shallow. In this case, the search algorithm should progress very quickly. However, another encoding might make the basins quite deep. This would make the optimization happen very slowly, with a relative speed exponentially related to the basin depths. As a result, a re-encoding might be able to increase the speed of the algorithm. This is the topic of the next corollary.

Before we delve into the corollary, we pause to consider what a re-encoding of the diversification operator means. The diversification operator may be thought of as a recursion relation which takes the current population and generates a new vector. That is,

$$D(\mathbf{P}) = v \tag{2}$$

where v is some vector. Re-encoding this diversification operator causes a change in the mapping. Random re-encoding changes the mapping in a random way, generating a completely new mapping. The basins of attraction for specific maxima may be shredded by this process, and the new locations of optima may be completely unrelated to their previous positions. Under these circumstances, a re-encoding of the diversification operator will have an unpredictable effect on the overall shape of basins of attraction around the optima. Moreover, it the relative distance between any two optima will generally change, with the new positions unrelated to the old ones.

The act of re-encoding the diversification operator can have a profound effect on a continuing optimization. Not only will the population have completely new connectivity, and therefore be able to spread itself throughout the space very differently, but this new connectivity can change the distance (in number of diversifications) between any two vectors. This means that those reencodings which bring them closer together will tend to increase the speed of optimization, while those that make them further apart will tend to decrease the speed of optimization.

COROLLARY 2 *An evolutionary algorithm employing nonbiased mixing will converge to an optimum faster than an evolutionary algorithm which does not employ mixing with a probability that falls off as 2^{-n} with n mixing events.*

Note that this result does not mean that the improvements cannot themselves still have very long convergence times. Moreover, this does not discount the possibility that the improvements will form a convergent chain well below the global optimum. It simply means that mixing events will have an exponentially decreasing probability that different basins will be brought together[3].

3. EVOLUTIONARY PATHWAYS

Once a problem encoding and a fitness function have been chosen, the fitness of each of the points in the space is defined. Once the encoding of the diversification operator has been defined, the connectivity of the search space is defined. Each point has a well-defined number of positive and negative transitions. That is, a single diversification event applied to any single point will generate a new vector whose fitness value is likely to differ from the original one. A reduction in fitness value is viewed as a negative transition, while an increase in fitness is a positive transition. We define a point as a ***local maximum (minimum)*** if the point has no positive (negative) transitions. Note that these definitions mirror similar ones given earlier.

The set of all positive transitions defines a structure in the search space. This structure defines the way a hillclimbing algorithm might progress. Evolutionary algorithms tend to follow the same paths that hill- climbing algorithms follow if the improvements are relatively direct and quick. If, however, the improvements require more time, the evolutionary algorithm will tend to spread out in many directions, choosing another optimization directon from those built into this structure. We define a ***path*** to be a finite ordered set of points in which any given point could give rise to the next point using a single diversification yielding a positive transition. We define a ***path bundle between point a and b*** to be the set of all paths through state space which follow positive transitions only, begin at **a**, and end at

[3]In their 2005 paper, Rand and Riolo [13] demonstrate that a dynamic re-encoding of the search space has significant advantages over a static encoding of a search space. This result has no theoretical basis to date, but has been reported by other researchers as well. We turn now to a description of why this is so.

b. Let **b** be fixed. We define the set of all trajectories ending at **b** to be the ***heap of trajectories leading to b***. Let us represent this as $H(b)$. Finally, suppose that **b** is a local maximum (minimum). Then, we call the heap of trajectories leading to **b** the ***maximal (minimal) heap of trajectories leading to b***. We denote this as $H_m(b)$. Note that $\partial H_m(b)$, ***the boundary of the maximal heap of trajectories leading to b***, is a set such that at each point at least one transition is part of a path that leads to **b** and at least one is a transition that is not part of such a path.

Note that the boundary regions can be located within a basin of attraction. Since individual basins of attraction can contain smaller basins of attraction, they can also contain boundary regions. The regions, we will see, are locations in which "decisions" are made by the algorithm about which path bundle to climb.

PROPOSITION 3 *Suppose that* $H_m(b) \cap H_m(b') \neq \emptyset$. *Then at each point in the set* $H_m(b) \cap \partial H_m(b') \cup \partial H_m(b) \cap H_m(b')$ *a transition is possible which excludes (except with a backward transition) the local maximum* b *or* b'.

Because of this, we may designate the set $H_m(b) \cap \partial H_m(b') \cup \partial H_m(b) \cap H_m(b')$ ***the decision region for the sets*** $H_m(b)$ ***and*** $H_m(b')$. Note that if there are only two local maxima, this region will always choose between the two different maxima.

The decision region is an important part of the space, as it makes an ascending evolutionary algorithm choose between the quick transitions that lead it to a single maxima. As the population progresses through the search space, it passes through the transition regions, effectively "choosing" between future maxima. Once the population has made a decision, it becomes increasingly unlikely to return to the decision region and to make a new decision.

The search space can be characterized by the number of disparate starting points and decision regions a population will pass through as it moves toward a local optimum. As each decision region forces the population to choose between at least two local maxima, the number of potential end points must be at least $m2^k$ where m represents the number of disparate starting populations possible in the space [4] and k represents the number of transition regions that the population is likely to encounter during the optimization. This forms a lower limit on the number of times an unbiased evolutionary algorithm would have to be restarted in order to guarantee that the population visited the optimum. The value of m is very problem-dependent, and may be infinite. Moreover, k is highly dependent on the path bundle taken. These two factors make this type of analysis rather unreliable in determining a solid lower bound on the number of attempts one must make.

[4]Normally the initial population is distributed across the search space, making m 1.

4. MERGING APPROACHES

In the preceding analysis, we've examined properties of the system that lead to two competing approaches to the optimization. In the first approach, a single long run is attempted. Given enough time, we have seen that the extended diversification operator will produce vectors that will move the population from one basin of attraction to another, meaning that this approach will eventually succeed.

The second approach is generally to start and re-start the algorithm so as to traverse as many path bundles as possible. This approach will generally allow the algorithm to make it to basins of attraction that cannot be easily attained by simply using a single run. As we have seen, when the basins of attraction are far apart, then the single run has an extremely hard time finding its way between basins of attraction. When the number of path bundles is small compared to the number of evaluations required to transfer from one basin of attraction to the next, then it is expected to be quicker to start the algorithm over in order to improve the performance rather than to wait for the population to make the desired transition.

A third option that has rarely been utilized is to use a completely exhaustive algorithm. The reason this has been overlooked is that it is generally assumed that the search space is so vast that the use of an exhaustive algorithm would be prohibitive. However, as with an evolutionary algorithm, a well-made exhaustive algorithm might be exhaustive in design, but still exploit the pertinent parts of the search space in the same way the an evolutionary algorithm or other algorithm might.

What we have seen is that the search space is made up of path bundles leading to local maxima which are separated by differing step distances. The evolutionary algorithm will generally explore the search space along these path bundles, and traverse the intermediate locations with a much lower probability. Therefore, our algorithm must be designed to do the same thing. Our design must have the property that it can search exhaustively through the space, but do so in such a way that it can apply the same advantages one might find in an evolutionary algorithm.

4.1 Pseudoexhaustive search

Our algorithm is called *Directed Pseudoexhaustive Search (DPS)* [5] . This algorithm performs a *pseudoexhaustive search*, or a search which does not reach the entire space, but systematically exhaustively searches a subspace, which might include the entire space. This algorithm is meant to utilize the strengths of searches such as evolutionary algorithms, while avoiding the sometimes debilitating repetition of evolutionary algorithms.

DPS assumes that the space is discrete, as is the set of possible variations. Moreover, since the set of di-

[5]This has also been called *Directed Exhaustive Search (DES)* in earlier work on this subject [7, 8].

versifications is discrete and therefore enumerable, it is assumed that the diversification operator used can be invoked using a vector and a chosen diversification ranging from 1 to the maximum mutation. This makes it possible to keep track of which mutations have been attempted with a single vector and which have yet to be attempted.

Our algorithm utilizes two main data structures. The first data structure is a linked list. The linked list contains the vectors that are currently being examined. These vectors are ordered according to a decreasing fitness value. The linked list is augmented with a list pointer which determines which element in the list will be utilized at a specific moment. The list pointer's control strongly influences the search. We return to it presently.

Each node in the linked list is a record that contains three pieces of information. The first is the vector. The second is an integer known as a variation pointer indicating which variation was last attempted on this vector. The integer is incremented with every mutation operation on a given vector. The third is the fitness value for the vector.

The second data structure used is a lexicographically-ordered binary tree. The binary tree stores the vectors that have been visited by DPS. This is utilized so as to avoid revisiting the vectors. The binary tree is source of greatest memory consumption, and so care must be taken to avoid storing too many vectors. This is done by periodically purging the tree. We describe how this is done without losing the efficacy of the algorithm shortly.

DPS makes use of a reset threshold, which is a floating point number initially set to 1, and which controls the likelihood that the algorithm will "spontaneously" reset the linked list pointer to the head of the list.

DPS starts by creating a predetermined set of binary vectors. This initial population is then inserted into the linked list. Once the initial population is inserted in the linked list, the list pointer is reset to the top of the list and DPS enters continual cycles.

During each cycle, the vector to which the list pointer is pointing is varied, calling on the variation indicated by its variation pointer. The old vector's variation pointer is incremented once the variation occurs. If the old vector's variation pointer exceeds the maximum variation number, it is removed from the linked list. If the new vector does not exist in the binary tree, it is stored in the binary tree. If it also has a fitness that exceeds that of the old vector, it is stored in the linked list with a newly initialized variation pointer set to 1. The linked list pointer is reset to the top of the list and the next cycle begins. If the new vector's fitness is the highest thus far found, the reset threshold is reset to 1. Otherwise, it is replaced by the the product of itself and a number between zero and 1, typically 0.999.

If the new vector does exist in the binary tree or if its fitness is lower than the old vector's fitness, then the list pointer is incremented, and the new vector discarded. The reset threshold is multiplied by the same number as used in the previous paragraph in this case. Moreover, if a random number falls between 1 and the reset threshold, the list pointer is reset to the top of the list. This serves to bias the search so as to carry out deeper rather than wider searches if it goes too long without discovering a new optimum.

The binary tree will eventually expand to fill all the memory on a computer, even for moderately sized problems, without maintenance. As a result, we've developed a procedure for moderating its size. Each time an element is found in the binary tree (it has been produced by a mutation earlier), a counter in its node is incremented. Periodically, the binary tree is updated by creating a new tree from the old one, while deleting the old one. The new tree only contains those elements from the old one that have been visited recently. This not only removes many of the nodes, but also serves to include only those that need to be included. Many of the nodes can only be created by going through other nodes. If these other nodes are kept in the binary tree, the nodes created using them can be safely removed from the tree without fear that they will be added again to the linked list.

4.2 Application to the BQP

In our simulations, both the GA and BQP use a binary representation. The genetic algorithm utilized is a Goldberg style GA with proportional reproduction, and single or double point mutation, and elitism. As the mutation is assumed to be enumerable, mutations consists of passing of a vector and a random choice of mutation from between the minimum and maximum index of mutation. The GA does not use a crossover operator as the design of a suitable DPS algorithm utilizing crossover requires a tremendous variation operator. This is planned for a later study.

The DPS utilized is as descibed above. Both the DPS and the GA utilize the same fitness function, which returns the product of the vector, the matrix, and the vector's transpose.

We apply both DPS and the GA to 15 different instances of the BQP; these problems can be found in OR-LIB[2], a computer program library for basic models of operations. Among these instances are 10 small 50 x 50 instances and 5 medium instances of 100 x 100 problems. Each problem instance is tested 30 times where the best fitness for each problem is known.

We give the performance of the two algorithms in Table 1. This table gives the best found individual in ten runs of the each algorithm along with the best known solution. Interestingly, the new algorithm is capable of recovering the best known solution in all the examined cases. Moreover, the DPS algorithm outperforms the evolutionary algorithm from which it was cast in terms of optimization and reliability. This indicates a potential solution to the optimization for problems that EAs have trouble with, without changing the operators

Table 1: This table gives the average performance of the GA and the DPS on the various BQP problems. In all cases, the DPS is capable of finding the optimum.

Problem	DPS	GA	Best	%	%
	Best	Best	Known	DPS	GA
50.01	2098	2098	2098	100%	60%
50.02	3702	3702	3702	100%	100%
50.03	4626	4626	4626	100%	100%
50.04	3544	3544	3544	100%	76.7%
50.05	4012	4012	4012	100%	80%
50.06	3693	3693	3693	100%	100%
50.07	4520	4520	4520	100%	40%
50.08	4216	4216	4216	100%	96.7%
50.09	3780	3780	3780	100%	36.7%
50.1	3507	3507	3507	100%	43.3%
100.01	7970	7970	7970	60%	10%
100.02	11036	11036	11036	90%	30%
100.03	12723	12723	12723	100%	30%
100.04	10368	10368	10368	100%	20%
100.05	9083	9083	9083	100%	30%

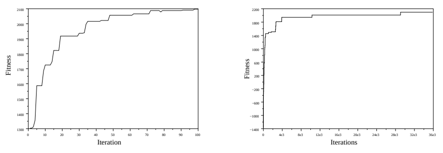

Figure 1: This figure illustrates the performance of the GA (left) and the DPS (right) on the BQP optimization problem. These graphs illustrate the behavior on the same 50x50 optimization problem.

utilized, and with a relative minimum of programming complexity.

We also present graphical data indicating the performance of the two algorithms during typical runs. The data, given in Figure 1, illustrates that though the two algorithms are very different in their details, the data processing of both is similar. In each case, the optimization may be characterized by periods of inactivity punctuated by sharp changes in the current best fitness.

5. DISCUSSION

Optimization algorithms such as evolutionary algorithms behave differently according to the part of the search space that they are currently exploring. However, the "magical" part of the evolutionary algorithm is often thought to be tied up in the randomness of the mutation and crossover operators. It is this randomness, it is often times argued, that makes evolutionary algorithms capable of doing things that are outside of more standard optimization algorithms, worthy of an entirely discrete research center.

Recent theoretical work on the No Free Lunch (NFL) Theorem and its consequences has begun to overturn this conceptual thinking [15, 16, 10]. Many researchers interpret the NFL as tending to assign equality among all problems to every optimization algorithm. However, this is neither implied nor explicitly stated by the NFL theorem as it does not include any information about the repetitive nature of the evolutionary algorithm. In fact, as all evolutionary algorithms exhibiting *eventual stagnation* are not addressed by the NFL Theorem, and many evolutionary algorithms are, in practice, eventually stagnant, this analysis does not apply to any of these.

What the NFL theorem does correctly do is to indicate that evolutionary algorithms capable of being implemented on a computational device and not eventually stagnant are, in fact, not random at all. Their apparent randomness is a result of a complex set of equations that exhibit many of the properties of randomness, but are not really random. Nonetheless, these algorithms have an impressive ability to yield the global optimum of a search space. As a result, it must be concluded that these algorithms are capable of yielding good results because of how they process the search space. Their randomness would seem to be irrelevent.

The search space processing capability of the algorithm can be clarified by examining the search space structure imposed by the diversification operators (mutation and crossover). This structure defines the distance in diversification steps between different vectors, the probability of each of these steps, and indirectly defines the expected amount of time that will pass between the various transitions. Therefore, an algorithm that processes the search space in the same way as evolutionary algorithms should be able to perform similarly, even if it is not evolutionary in nature.

In our simulations, we have taken care to generate algorithms that are fashioned from the same genetic operators used in our evolutionary algorithms. Because of the nature of crossover and the difficulty in implementing it in an exhaustive way, we have bypassed the use of crossover in our evolutionary algorithms. However, despite the limited capability of our evolutionary algorithms, both they and the versions of DPS formed from each EA instantiation performed quite well on the problems they were applied to.

The BQP problem represents a problem of moderate computational difficulty. While the problem is certainly NP complete, recent work using memetic algorithms can do quite well in completing the optimization. However, as we've seen in this work, some EAs have difficulty in finding the optimum, particularly in the larger dimension cases. This finding might be generalized to other problems that are tackled with more specialized functions. In much the same way that the performance of our very simple GA has been improved by the DPS formulation, it might be possible to improve other algorithms.

This particular problem is complex enough that it also illustrates an important property of search algorithms. As we've seen, the performance of the two related algorithms is similar, though there are significant advantages obtained when the DSP is used for larger

BQP problems. It is interesting that a non-stochastic algorithm can be generated that seems to have similar information processing capability and processes that evolutionary algorithms have, despite being exhaustive in nature. EAs must be understood in a way that does not depend on their stochastic nature, but instead focuses on their ability to extract information, react to the fitness landscape, and generate more desirable vectors. This might lead to localized economies on particular problems, and may make it possible to generate algorithms that self-adapt so as to take advantage of these economies as they carry out the optimization.

6. CONCLUDING REMARKS

In this paper, we've explored some of the theoretical issues concerning evolutionary algorithms, and applied a new type of search algorithm to the BQP problem. The theoretical issues are interesting in the sense that they validate some of the expected or oft observed but heretofore unexplained behaviors of EAs. The ability of the genetic algorithm, or for that matter any EA with a diversification operator that isn't eventually stagnant, to find the global optimum has never before (to our knowledge) been demonstrated theoretically. This serves to solidify our understanding of the scope and design of the EA.

Of prime importance in the design of any search algorithm is the representation of the various vectors and operators. Our theoretical results have demonstrated that random representations of these operators can be effective in generating improvements in performance, This is similar to the use of uniform crossover, which has various advantages over single point and dual point crossover.

Our examination of the search space and characterization of it as a set of paths has allowed us to understand how one might utilize an exhaustive or pseudoexhaustive algorithm which seems to behave similarly to the EA. Our analysis has indicated that while the EA is an interesting paradigm that seems to perform well with a modicum of machinery and using random operators, the part of the EA which makes it useful and important is its ability to process information gleaned during the search and use it to direct its later actions. When this information processing capability was utilized by our pseudoexhaustive algorithm, the algorithm was able to replicate the performance of the earlier optimized memetic algorithms, though it wasn't itself optimized.

7. REFERENCES

[1] B. Alidaee, G. Kochenberger, and A. Ahmadian, *0-1 Quadratic Programmin Approach for the Optimal Solution of Two Scheduling Problems*, **International Journal of Systems Science**, 25: 401-408, 1994.

[2] J. Beasley. *OR-library: Distrubing Test Problems by Electronic Mail.* http://people.brunel.ac.uk/ matjjb/jeb/orlib/files.

[3] J. Beasley. *Heuristic Algorithms for the Unconstrained Binary Quadratic Programming Problem*, Tech. Rep., Management School, Imperial College, London, UK, 1998

[4] G. Gallo, P. Hammer, and B. Simeone. *Quadratic Knapsack Problems*, **Mathematical Programming**, 12: 132-149, 1980.

[5] F. Glover, B. Alidaee, C. Rego, and G. Kochenberger. *One-Pass Heuristics for Large-Scale Unconstrained Binary Quadratic Problems*

[6] E. Cantu-Paz and D. Goldberg. *Are multiple runs of genetic algorithm better than one?* **Lecture Notes in Computer Science, 2723**, pp. 801-812, 2003.

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

[8] S. Kazadi, M. Lee, L. Lee. *A Case for Exhaustive Optimization.* **Proceedings of Gecco 2005 Conference, Late Breaking Papers**, Washington D.C., USA, June 2005.

[9] D. Laughunn. *Quadratic Binary Programming.* **Operations Research**, 14: 454-461, 1970.

[10] W. Macready and D. Wolpert. *What makes an optimization problem hard?*, **Complexity**, 5: 40-46, 1996.

[11] P. Merz and B. Freisleben. *Genetic Algorithms for Binary Quadratic Programming.* **Proceedings of GECCO 1999**, 417-424, 1999.

[12] A. Phillips, and J. Rosen. *A Quadratic Assignment Formulation of the Molecular Conformation Problem.* **Journal of Global Optimization**, 4: 301-328, 1994.

[13] Rand and Riolo. *The problem with a self-adaptive mutation rate in some environments. A case study using the shaky ladder hyperplane-defined functions.* **Proceedings of GECCO 2005**, Washington D. C., USA, pp. 1493-1500, 2005.

[14] C. Witsgall. *Mathematical Methods of Site Selection for Electronic System (EMS).* NBS Internal Report, 1975.

[15] D. Wolpert and W. Macready. *No free lunch theorems for optimization.* **IEEE Trans. Evol. Comput.**, 1(1): 67-83, 1997.

[16] D. Wolpert and W. Macready. *Coevolutionary free lunches.* **IEEE Trans. Evol. Comput.**, 9(6): 721-735, 2005.

[17] M. Mitchell. **An Introduction to Genetic Algorithms**. MIT Press: Boston, MA, 1996.

[18] L. Davis, M. Vose, and K. DeJong. **Evolutionary Algorithms. Volumes in Mathematics and its Applications Vol III**. Springer-Verlag: Berlin, 1996.