A Diversity Maintaining Population-Based Incremental Learning Algorithm

Mario Ventresca *

Systems Design Engineering Department, University of Waterloo, Waterloo, ONT, N2L 3G1, CANADA

Hamid R. Tizhoosh

Systems Design Engineering Department, University of Waterloo, Waterloo, ONT, N2L 3G1, CANADA

Abstract

In this paper we propose a new probability update rule and sampling procedure for population-based incremental learning. These proposed methods are based on the concept of opposition as a means for controlling the amount of diversity within a given sample population. We prove that under this scheme we are able to asymptotically guarantee a higher diversity, which allows for a greater exploration of the search space. The presented probabilistic algorithm is specifically for applications in the binary domain. The benchmark data used for the experiments are commonly used deceptive and attractor basin functions as well as 10 common Travelling Salesman problem instances. Our experimental results focus on the effect of parameters and problem size on the accuracy of the algorithm as well as on a comparison to traditional population-based incremental learning. We show that the new algorithm is able to effectively utilize the increased diversity of opposition which leads to significantly improved results over traditional population-based incremental learning.

Preprint submitted to Elsevier

Key words: Population-based incremental learning, opposition-based computing, diversity maintenance, diversity control.

1 Introduction

Diversity is a measure of a sample or population, independent of the evaluation function [33]. This measure typically represents the relative distance between samples with respect to their solution representation (vis-à-vis the problem), where distance is determined by a user-defined function. A lack of diversity in a population corresponds to sample solutions being very similar with respect to the distance metric. Conversely, when samples are not very similar then the degree of diversity is high.

Population-based algorithms such as incremental learning or evolutionary algorithms have a tendency to converge to local optima. This premature convergence is due to various algorithmic properties, but a major reason is due to a lack of diversity in the population. Typically, diversity is highest at the onset of the algorithm when all solutions have been randomly generated. As the algorithm progresses diversity normally decreases due to selection pressure or sample generating bias and therefore increasing the difficulty of escaping a local optima.

Evolutionary algorithms maintain a set of solutions at each generation, and perform crossover and mutation operations on this set to generate different solutions and use a selection mechanism to guide the search process. Contrast-

* Corresponding author.
Email addresses: mventres@pami.uwaterloo.ca (Mario Ventresca), tizoosh@uwaterloo.ca (Hamid R. Tizhoosh).

ingly, population-based incremental learning (PBIL) stores a single prototype vector instead of a larger set of solutions. The vector represents probability distributions over the allowable values at each locus of the solution representation. Do to this property, PBIL belongs to the family known as Estimation of Distribution Algorithms (EDAs), which use this vector to generate sample solutions. Furthermore, in lieu of a selection mechanism the probability model represented by the prototype vector is updated. It has been experimentally shown that population based incremental learning has the capacity to outperform evolutionary algorithms [6].

The main benefits of PBIL over traditional evolutionary algorithm approaches are (1) lowered memory requirements (since the population does not needed to be stored); (2) increasing the sample size has no effect on memory (versus increasing population size); and (3) typically a lowered computational cost (mainly because of not employing genetic operators) [22]. Nevertheless, PBIL (and generally EDAs) still suffer from issues of diversity loss, as described in [2, 20, 36], and so methods which improve diversity should be of importance.

PBIL's implementation ease and robustness have allowed it to be employed for solving a variety of real-world problems. In [35] multiple probability vectors and an adaptive updating strategy are proposed and the resulting algorithm is tested on the geometrical design of the end region of power transformers. It was shown to outperform other common heuristic methods.

Power system controller design under various operating conditions was examined in [4]. It was found that PBIL designed controllers are able to adaquately stabalize the system over the varying conditions. Additionally, for large disturbances in conditions PBIL is shown to outperform a genetic algorithm ap-

proach. An initial investigation into the use of PBIL for evolutionary robotics was given in [22]. The proposed Floating Point PBIL is able to handle floating point results and is shown to outperform a traditional genetic algorithm.

The telecommunications problem of radio network design (placement of antennas) was investigated in [28]. This was a novel application area for PBIL and it yielded very good results. Another interesting problem was that of university curriculum scheduling, where PBIL is shown to also provide usable course schedules [9]. And, in [13] PBIL was employed to provide a measure of the uncertainty in parameters for reservoir models which quanitify the amount of risk associated with alternative production scenarios.

In this paper we propose a new probability update rule and sample generation procedure which are rooted in concepts of opposition-based computing. Given some sample, we generate its corresponding opposite by taking the complement of the initial binary solution. The complement is dynamic in that we do not always consider the entire solution to complement, rather a subset of decreasing size based on the number of iterations. Using opposite samples allows for diversity maintenance and aids in slowing premature convergence and exploring the search space. As the algorithm progresses the degree of opposition between opposite samples is decreased in order to allow the algorithm to actually converge towards good areas of the search space. Without employing this concept the algorithm could not converge very easily as truly opposite samples do not lie close together in the search space.

The alternate probability update rule allows for the effective use of the opposite sampling procedure by not only tending toward quality areas of the search space, but also forces diversity in the samples. Depending on the quality of

results, the update strategy will use the global or population best solution to update the probability vector. The global best solution has an increased probability proportional to the number of iterations since a new best was discovered, preventing the algorithm from forgetting probabilities. The local best solution is used to explore the current probability matrix and exploit diversity.

The remainder of this paper is organized as follows: In Section 2 we introduce the idea behind opposition-based computing and population-based incremental learning, which form the basis for our approach. We then describe how opposition can be employed to improve diversity and provide mathematical proofs to this effect as well as outline our proposed algorithm in Section 3. The benchmark problems and experimental results are described in Section 4 followed by concluding remarks in Section 5.

2 Background

This section will describe the motivation behind opposition-based computing (OBC). Also, a brief description of the population-based incremental learning algorithm will be provided.

2.1 Opposition-Based Computing

Opposition-based computing is a recently introduced computational intelligence framework [23]. The underlying notion is based on observations in nature and society whereby the current state of an object or entity has some notion of an opposite state or entitys. Some common examples of opposition include;

- *Logic:* The relation between two propositions in virtue of which the truth or falsity of one of them determines the truth or falsity of the other [27];
- *Linguistics:* Contrast in a language between two phonemes or other linguistically important elements [27].
- *Electricity:* The condition that exists when two waves of the same frequency are out of phase by one-half period [27].
- *Philosophy:* Chinese philosophy's complementary natures of the principles of the Yin and Yang [10] as well as the Pythagorean Table of Opposites [7].

In societal situations the idea of opposition manifests itself, for instance, via political revolutions which typically occur rather rapidly towards a more desirable form of government (at least in perception). In any event, there exists some form of balance or completeness to a situation where opposites exist. OBC aims at exploiting the relationship between opposites in order to improve computational intelligence algorithms with regards to their ability to search, generalize, yield accurate results, convergence faster, etc.

The fundamental intuition behind the application of OBC is to simultaneously consider the current state/solution to the problem, as well as it's opposite(s). Then, depending on the evaluation criteria, the most desirable of the two solutions is selected, or some form of combination between the two solutions is used to generate a "common ground solution". For example, searching for the minimum of an unknown function that is bounded to the interval $[0, 1]$. Here, a guess x could simply be a random value in the interval, and we can define the opposite of x to be $\breve{x} = 1 - x$ (this relationship between x and $\breve{x}$ is defined by an opposition map [31]). Assuming the algorithm makes guesses in pairs, then after each pair of guesses the algorithm will return $\min(x, 1 - x)$ as opposed to $\min(x, y)$ for some random $y \in [0, 1]$ where x, y have been

independently sampled. Due to the properties of the mapping $x \mapsto \breve{x}$ and the min(...) function the expected value of the original evaluation function is lowered, and therefore it is reasonable to expect a more desirable outcome using OBC [31]. Furthermore, the opposition map may be defined such that it also considers properties such as symmetry or other a-priori knowledge of the evaluation function.

To date, explicit opposition-based computing has led to an improvement in accuracy, convergence rate and/or generalization ability in differential evolution [14, 15], reinforcement learning [21, 24, 25], simulated annealing [31] and backpropagation learning [29, 30]. We believe that it will also be shown to yield improvements in other areas of computational intelligence, although it should be noted that many paradigms and approaches currently exist which fall under the umbrella of opposition-based computing. However, due to the recent proposal of the OBC framework most of these existing methods were not classified as oppositional at the time of their inception.

2.2 Population-Based Incremental Learning

Population-based Incremental Learning (PBIL) combines elements from evolutionary computation (EC) and reinforcement learning (RL) [1]. PBIL is a population-based stochastic search where the population is essentially a random sample based on an estimated probability distribution for each variable. So, in reality the population does not exist as it does in traditional EC. After a sample is generated, the best is retained and the probability model for each variable is updated to reflect the belief regarding the structure of the best solution. This is accomplished according to a similar update rule as used in

RL. The result is a statistical approach to evolutionary computation.

An evolutionary algorithm's population can be thought of as representing an estimated probability distribution over the possible values for each gene. In PBIL the population is replaced by a $d \times c$ dimensional probability matrix $\mathbf{M} := (m_{i,j})_{d \times c}$ which corresponds to a probability distribution over possible values for each element (d is the problem dimensionality each having c variables). For example, if a binary problem is under consideration then a solution $\mathbf{B} := (b_{i,j})_{d \times c}$ where $b_{i,j} \in \{0, 1\}$ and so each $m_{i,j} \in [0, 1]$ corresponds to the probability of $b_{i,j} = 1$.

Learning in PBIL consists of utilizing the current $\mathbf{M}$ to generate a set G_1 of k samples. These samples are evaluated according to the objective function for the given problem and the "best" sample $\mathbf{B}^* := (b_{i,j})_{d \times c} \in \{0, 1\}$ is maintained. Then, the probability distributions represented in $\mathbf{M}$ are updated by increasing the probability of generating solutions similar to $\mathbf{B}^*$. The update rule to accomplish this is similar to that found in learning vector quantization [1]:

$$\mathbf{M}_t = (1 - \alpha)\mathbf{M}_{t-1} + \alpha\mathbf{B}^*, \tag{1}$$

where $0 < \alpha < 1$ represents a user-defined learning rate and the subscript $t \geq 1$ corresponds to the current iteration of PBIL. Without prior information $(m_{i,j}) = 0.5$.

Another contrast to evolutionary computation is the lack of a crossover operator or selection mechanism, instead the values in $\mathbf{M}$ are "mutated" once per iteration. During this phase a small random value is added or subtracted from

a random subset of the values in $\mathbf{M}$. Furthermore, since at each iteration a new subset of samples is generated and only the best is maintained then no selection mechanism is required.

The pseudocode for PBIL is presented in Algorithm 1. It assumes constants to control the maximum number of iterations and samples, ω and k, respectively. Additionally, in line 13 a user-defined parameter $0 < \gamma < 1$ controls the amount that any $m_{i,j}$ can be perturbed. This change is applied according to the user provided constant $0 < \beta < 1$ in line 12.

Algorithm 1 Population-Based Incremental Learning [1]

```
1: {Initialize probabilities}
2: M_0 := (m_{i,j}) = 0.5
3: for t = 1 to ω do
4:    {Generate samples}
5:    G_1 = generate_samples(k,M_{t-1})
6:    {Find best sample}
7:    B* = select_best({B*} ∪ G_1)
8:    {Update M}
9:    M_t = (1 − α)M_{t-1} + αB*
10:   {Mutate probability vector}
11:   for i = 0...d and j = 0...c do
12:      if random(0,1) < β then
13:         m_{i,j} = (1 − γ)m_{i,j} + γ · random(0 or 1)
14:      end if
15:   end for
16: end for
```

Initially, in line 2 we let $\mathbf{M} := (m_{i,j}) = 0.5$ to reflect the lack of a priori information regarding the probability distribution of each variable. In line 5 we generate the k samples using the current probability matrix and select the best sample (w.r.t. some user-defined criteria) in line 7. Matrix $\mathbf{M}$ is updated in line 9 using the best sample to guide the direction of probability update. Finally, lines 11-15 probabilistically perform the mutation rule.

It has been shown that for a given discrete search space PBIL will converge to a local optima [8, 16, 17]. PBIL algorithms for continuous spaces have also been explored (for examples see [18, 19]), although only the discrete binary case is considered in this paper.

The Double Learning PBIL algorithm, based on an elitist strategy using both current best and global best solutions for updating is studied in [37], where improvements in convergence speed and accuracy are observed. Additionally, the usefulness of PBIL for dynamic problems has also been investigated [34].

Expanding PBIL to multi-objective problems has been examined in [3]. Using a method based on non-dominance, the new approach was compared to a multi-objective genetic algorithm on the problem of designing pie networks in terms of cost, loss and reliability. It was found that the multi-objective PBIL algorithm was able to estimate and explore the true Pareto front better than the genetic algorithm.

3 Proposed Approach

This section provides the theoretical motivation from a diversity perspective for utilizing opposition-based computing to improve PBIL. Firstly, we prove that we can yield a greater diversity during the sampling stages of PBIL using opposition. Then, we propose a new update rule made possible because of the properties of opposition. This new rule requires a new procedure to mutate the probability matrix $\mathbf{M}$.

3.1 Improved Diversity

As mentioned above, diversity is a measure of a sample or population independent of the evaluation function [33]. A common approach is to measure the distance between all possible pairs of samples in the population. Under this approach samples which differ in only a few variable values will contribute less to the overall diversity than samples which differ in many values. In the case of binary problems, the Hamming distance is commonly utilized to measure the difference between pairs of samples [11, 33].

Since the Hamming distance (d_{HAM}) is symmetric we need only to compare pairs of elements once (i.e. $d_{HAM}(p_1, p_2) = d_{HAM}(p_2, p_1)$ for samples p_1 and p_2). Then, the all-possible-pairs diversity of a population $\mathcal{P}$ of n binary samples is formulated as [33]

$$V(\mathcal{P}) = \sum_{i=1}^{n} \sum_{j=1}^{i-1} d_{HAM}(p_i, p_j), \tag{2}$$

where $p_i, p_j \in \mathcal{P}$. In total this formulation will result in $\frac{n(n+1)}{2}$ Hamming distance calculations.

We now provide some definitions and notations required for the subsequent proofs. The definitions and proofs are presented without loss of generality assuming d-dimensional problems where each dimension is composed of $c = 1$ variables.

Definition (Opposite Binary Sample) Given some binary sample g of length l where $g_i \in \{0, 1\}$ and $i = 0...l$, the *opposite binary sample* $\breve{g}$ is defined to be the binary negation of g:

$$\breve{g}_i = \begin{cases} 1, \text{if } g_i = 0, \\ 0, \text{otherwise} \end{cases} \quad \text{for } i = 0...l \tag{3}$$

Definition (Comparison Set) Given some universal set $\mathcal{U}$ and subsets $A \subset \mathcal{U}$ and $B \subset \mathcal{U}$, we define a *comparison set* S as the Cartesian product $A \times B = \{< a, b > \mid a \in A \text{ and } b \in B\}$. We will denote this set as $S = A \rightarrow B$, inferring set A is being compared to set B.

Definition (Opposite Set) Given some universal set $\mathcal{U}$ and subset $A \subset \mathcal{U}$ we define the *opposite set* as $A^\circ = \{< a_i, \breve{a}_i > \mid a_i \in A \text{ and } \breve{a}_i \in \mathcal{U}\}$, where $A^\circ \subset \mathcal{U}$ and the relationship between a given guess a and its opposite $\breve{a}$ is according to a user-defined function ξ. Note $A^\circ \neq A \times \mathcal{U}$ and $|A^\circ| \geq 1$ since at least 1 guess pair must be made, where $|\cdot|$ represents the cardinality of the set.

Definition (Diversity of Binary-valued Opposite set) Given some universal set $\mathcal{U}$ and opposite set $A^\circ \subset \mathcal{U}$ of size m, we calculate the all-pairs-diversity of this opposite set as

$$V(A^\circ) = \sum_{i=1}^{m} d_{HAM}(p_i, \breve{p}_i) \tag{4}$$

where $p_i, \breve{p}_i \in A^\circ$ represent the i^{th} guess and opposite guess, respectively.

Using these criteria we can now prove that the diversity in a set of k samples, where $k/2$ are randomly chosen and $k/2$ are opposites, is greater than the case where the set was composed of k random samples. The strategy used to prove this is based on a decomposition of the calculations of the all-pairs-diversity into only the random guesses, only the opposite guesses and between

the random and opposite guesses (non inclusive).

We also assume the samples are generated according to a probability matrix **M** as is the case with PBIL. We also make the assumption that only binary problems are being considered.

Lemma 1 (Diversity of Binary-valued Opposite set) *Given a d-dimensional binary space $\mathcal{B}^d$ and sets $R^\circ, R_1 \subset \mathcal{B}^d$ where $|R^\circ| = |R_1|$, then $V(R^\circ) \geq V(R_1)$ and furthermore $V(R^\circ) = constant$.*

Proof By definition of diversity on an opposite set,

$$V(R^\circ) = \sum_{i=1}^{m} d_{HAM}(p_i, \breve{p}_i)$$

where $p_i, \breve{p}_i \in R^\circ$. Since $d_{HAM}(p_i, \breve{p}_i) = l \ \forall \ i$ then $V(R^\circ) = constant$ and by definition of an opposite guess $\max(d_{HAM}(p_i, \breve{p}_i)) = l$.

Since R_1 is composed entirely of i.i.d. samples the only instance where $d_{HAM}(r_i, r_j) = l$ is if $r_j = \breve{r}_i$, where $r_i, r_j \in R_1$. It follows that $V(R^\circ) \geq V(R_1)$. Furthermore, by definition of an opposite guess, this value must be constant such that $V(A^\circ) = m \cdot l$.

■

From Lemma 1 we can now prove that the diversity of opposite guesses is greater than that of only random guesses if for k samples, $k/2$ are shared by each guessing strategy.

Theorem 3.1 (Diversity of Opposite Guesses) *Given a d-dimensional binary space $\mathcal{B}^d$, sets $R_1, R_2, \breve{R}_1 \subset \mathcal{B}^d$ each of size $k/2 \geq 1$ the all-pairs-diversity*

$V(R_1 \to \breve{R}_1) \geq V(R_1 \to R_2)$.

Proof We begin by extracting the opposite set S° from $R_1 \to \breve{R}_1$, resulting in

$$S^\circ = (R_1 \to \breve{R}_1) \setminus S_2$$

where S_2 is the set of non-opposites such that $S^\circ \cap S_2 = \varnothing$.

We can also write $R_1 \to R_2$ as the composition of two subsets,

$$R_1 \to R_2 = R_a \cup R_b$$

where R_a and R_b are composed of randomly selected elements (without replacement) from $R_1 \to R_2$ such that $|R_a| = |S^\circ|$ and $|R_b| = |S_2|$.

By Lemma 1 $V(S^\circ)$ is maximal and constant and if we assume that in general $V(S_2) = V(R_b)$ since S_2 and R_b are essentially composed of random samples, then,

$$\begin{aligned} & V(R_1 \to \breve{R}_1) - V(R_1 \to R_2) \\ &= V(S^\circ) + V(S_2) - (V(R_a) + V(R_b)) \\ &= V(S^\circ) - V(R_a) \\ &\geq 0 \end{aligned}$$

■

Now we can extend Theorem 3.1 to the complete situation where we consider

the all-pairs-diversity of an entire set of guesses. We will prove that using opposite guesses yields a higher diversity than traditional independent random sampling.

Theorem 3.2 (Diversity of the Opposite Guessing Strategy) *Given a probability matrix* $\mathbf{M} := (m_i)_n$*, d-dimensional binary space* $\mathcal{B}^d$ *and sets* $R_1, R_2, \breve{R}_1 \subset \mathcal{B}^d$ *with* $|R_1| = |R_2| = |\breve{R}_1| = k$*, if* $G_1 = R_1 \cup R_2$ *and* $G_2 = R_1 \cup \breve{R}_1$*, then* $V(G_2) \geq V(G_1)$*.*

Proof We begin by decomposing $V(G_1)$ and $V(G_2)$ into

$$V(G_1) = V(R_1) + V(R_2) + V(R_1 \rightarrow R_2),$$
$$V(G_2) = V(R_1) + V(\breve{R}_1) + V(R_1 \rightarrow \breve{R}_1).$$

These decompositions include all distance computations required for determining the all-pairs-diversity for G_1 and G_2, respectively.

Since R_1 and R_2 are generated using $\mathbf{M}$, it can be assumed that since the relationship between a guess and its opposite guess is symmetric, then $1 - \mathbf{M}$ can be seen as generating $\breve{R}_1$. Therefore, we make the assumption that in general $V(\breve{R}_1) = V(R_2)$. So, by Theorem 3.1,

$$V(G_2) - V(G_1) = V(R_1 \rightarrow \breve{R}_1) - V(R_1 \rightarrow R_2) \geq 0$$

■

We can also prove the following property concerning the convergence of di-

versity between generating samples by opposition versus solely independent random guessing.

Corollary 1 (Diversity Convergence) *Given a probability matrix* $\mathbf{M} := (m_i)_n$*, a d-dimensional binary space* $\mathcal{B}^d$ *and sets* $R_1, R_2, \breve{R}_1 \subset \mathcal{B}^d$ *(where* R_1 *and* R_2 *are i.i.d. samples based on* $\mathbf{M}$*), if the values of* $\mathbf{M}$ *converge (i.e.* $m_i = 0$ *or* 1*), then for* $k = |R_1| = |R_2|$ *samples,*

$$\lim_{V(R_1 \to R_2) \to 0} V(R_1 \to \breve{R}_1) - V(R_1 \to R_2) = l \cdot \frac{k(k+1)}{2},$$

where the notation $V(R_1 \to R_2) \to 0$ *represents the convergence of diversity of guesses based on* $\mathbf{M^t}$ *as* $t = 0, ..., \infty$ *(i.e. as the values of* $\mathbf{M}$ *converge).*

Proof Since we assumed the values of $\mathbf{M}$ converge, and since R_1 and R_2 are i.i.d. based on $\mathbf{M}_t$ it must be that if $t = 0..\infty$ is the current iteration of the algorithm, then

$$\lim_{t \to \infty} V(R_1^t \to R_2^t) \to 0$$

where R_1^t and R_2^t represent random samples based on $\mathbf{M}_t$. This is intuitive since in the limit both R_1^t and R_2^t are composed of the same identical element therefore $d_{HAM}(a, b) = 0 \ \forall a \in R_1 \text{and } b \in R_2$.

By a similar argument and by definition of opposite guesses the sets R_1 and $\breve{R}_1$ approach opposite solutions. Since the calculation $V(R_1 \to \breve{R}_1)$ will require $\frac{k(k+1)}{2}$ Hamming distance computations and since the length of a solution is l it must follow that

$$\begin{aligned}
&\lim_{V(R_1 \to R_2) \to 0} V(R_1 \to \breve{R}_1) - V(R_1 \to R_2) \\
&= \lim_{V(R_1 \to R_2) \to 0} V(R_1 \to \breve{R}_1) \\
&= l \cdot \frac{k(k+1)}{2}.
\end{aligned}$$

■

From this corollary we see that diversity is infused into a system utilizing opposite guessing strategy, which can lead to improved results. However, this can also be a detriment to a search as it may cause confusion during the searching process and lead away from quality solutions [38]. Therefore, it is imperative to determine a correct strategy for managing this diversity such that it decreases with an increase in the number of iterations of the algorithm.

3.2 Proposed Use for Opposite Guessing Strategy

From the previous subsection we have seen that opposition can maintain, and indeed increase diversity over traditional independent sampling. The problem now is that as a search algorithm converges, the need for such a high diversity tends to decrease [38]. Thus, in order for opposition to be useful the definition of opposite guesses must adapt to this scenario. Furthermore, without this adaptation to convergence the usefulness of the diversity resulting from opposite guesses tends to decrease as the algorithm converges [31]. This is due to the property of a search algorithm to discover high quality areas in the search space and therefore decreasing the probability that an opposite guess (with respect to the entire search space) will be useful.

In general, the notion of opposition has an implied utilization of distance. For example, given some search space $\mathcal{U}$ and reference point u^* (usually in the center of $\mathcal{U}$) and some initial random position $p \in \mathcal{U}$, its opposite position $\breve{p}$ is typically that which is located an equal distance from u^*. Moreover, if the distance between p and u^* is d, then the distance between p and $\breve{p}$ is $2d$. However, by changing the position of u^* we can relax the magnitude of the distance between two points. Moving u^* to some location other than the center of $\mathcal{U}$ yields the following consequences (which are also shown in Figure 1):

- the new location of u^* allows for two physically close points to be considered as opposites and essentially redefines the search space bounds.
- this effectively decreases the search space size since some points no longer have an opposite (according to the situation described above). In reference to a search algorithm, these points are too far from the current focus of the search to yield a desirable evaluation and hence are ignored.

[Fig. 1 about here.]

The translation of u^* need not be explicit. In the case of a binary domain (as is the case in this paper) we can instead alter the allowable distance between two points to be considered opposite. By shrinking the distance between opposite points, they become increasingly similar and u^* is implicitly placed between the two points. Similarly, increasing the distance between opposite points decreases their similarity. Therefore, if we have a mechanism to control the distance between opposite points we can control the amount of diversity infused into the search algorithm by opposite guesses.

We can model the decrease in distance between opposite points with a decision function $\xi(t)$ which returns the distance between opposite elements and

decreases as the number of iterations t increases. For binary problems we use this to reflect the hamming distance. While a variety of decision functions are possible we utilize a exponentially decaying function in the flavor of

$$\xi(t) = le^{(ct)}, \tag{5}$$

where l represents the maximum number of bits in a guess and $c < 0$ is a user defined constant. We then take the complement of $\xi(t)$ randomly selected bits from the current solution to yield the opposite guess.

The shape of $\xi(t)$ directly effects the convergence of diversity of the population. This function will also impact on the convergence of the output of the algorithm, although to a lesser extent than on diversity.

3.3 Alternate Probability Update Rule

The role of PBIL's probability mutation rule is to allow for further exploration of the search space by randomly forcing a change in one or more values of the probability matrix $\mathbf{M}$. For binary problems, this is accomplished by adding or subtracting a small random value between 0 and 1 to arbitrary elements of $\mathbf{M}$ (i.e. mutate the probability). Of course there is no guarantee that a mutation will improve the likelihood of generating a high quality sample from $\mathbf{M}$.

The proposed approach does not employ a mutation operation as PBIL does. Instead, we opt for a more advanced mutation rule which adjusts probabilities of $\mathbf{M}$ through decay or amplification fuctions, with respect to either the current sample or global best found solution. This contrasts with PBIL, which employs

a purely random mutation strategy. However, if the rate of decay is too high, it will result in a lack of exploitation (moving quickly away from current best solution) and could possibly result in a lack of convergence (decay and amplification neutralize each other) or premature convergence (amplification too high). To stabalize this situation we employ the following logic:

- Whenever a new optima $\mathbf{B}^*$ is found, values of $\mathbf{M}$ are always amplified towards it to guide the search towards that region of the search space.
- As the number of iterations increases from the previous discovery of a new $\mathbf{B}^*$, the sample best is used to update (according to some probability) as if it was a new global optima. This allows the search to exploit the current best solution, but also slowly tends away if that region of the search space does not produce any more optima.
- As the number of iterations increases, the probability of decay decreases. Too many decays will cause divergence. This, along with the previous idea allow the search to self-control exploration and exploitation.

In order to control the decay and amplification we introduce two parameters $0 < \tau < 1$ and $0 < \rho < 1$, respectively. These three logics are implemented as described in the following.

When a new optima $\mathbf{B}^*$ is discovered we utilize the same update as PBIL, replacing the learning rate with ρ,

$$m_{i,j} = m_{i,j} \cdot (1 - \rho) + \mathbf{B}^*_{i,j} \cdot \rho. \tag{6}$$

We also apply this rule to expoit the current sample-best solution, which is increasingly important as the difference Δ in iterations between discovering

$\mathbf{B}_i^*$ and $\mathbf{B}^{*-1}$, respectively, increases. The probability function we use in this paper to represent this situation is

$$p_{amp}(\Delta) = 1 - e^{-b\Delta} \tag{7}$$

where $b > 0$ is a user defined constant. We use the sample best solution η instead of the global best $\mathbf{B}^*$ (we found experimentally that b=0.01 yields the better average results on the problems tested). Of course, other forms of this probability function are possible, our experimentation has led to the selection of this one, but we make no claim that it is the optimal choice.

If we do not amplify the probabilities we probabilitistically decay them to allow for exploration. This probability is controlled by (as with Equation (7) we make no claim to its optimality)

$$p_{decay}(\Delta; f(\mathbf{B}^*), f(\eta)) = \frac{1 - \frac{f(\mathbf{B}^*) - f(\eta)}{f(\mathbf{B}^*)}}{\sqrt{\Delta + 1}} \tag{8}$$

where $f(\cdot)$ is the evaluation function. Since $\sqrt{\Delta + 1} \to \infty$ as no new global best solution is found, then accordingly $p(\Delta; f(\mathbf{B}^*), f(\eta)) \to 0$. Therefore, as Δ increases the decay portion of the algorithm becomes less likely to occur which serves the purpose of avoiding divergence. Then, the update rule is used with probability given by (8) for the case where $\eta_{i,j} = \mathbf{B}_{i,j}^*$ is:

$$m_{i,j}^{t+1} = m_{i,j}^t \cdot \begin{cases} 1 - \tau \cdot random(0,1), & \text{if } \eta_{i,j} = 1, \\ 1 + \tau \cdot random(0,1), & \text{otherwise} \end{cases} \tag{9}$$

and when $\eta_{i,j} \neq \mathbf{B}^*_{i,j}$ we use

$$m^{t+1}_{i,j} = m^t_{i,j} \cdot \begin{cases} 1 + \tau \cdot random(0,1), \text{ if } \eta_{i,j} = 1, \\ 1 - \tau \cdot random(0,1), \text{ otherwise} \end{cases} \tag{10}$$

where the *random(0,1)* function returns a uniformly distributed random number between 0 and 1 and the superscript t represents the iteration of OPBIL. To further aid in exploitation we can employ the global best solution in lieu of η in Equation (9). This can also be done in a probabilistic or heuristic manner. For simplicity we choose the probability of Equation (8).

3.4 Summary of the Proposed Algorithm

In this subsection we outline the OPBIL algorithm which is based on the aforementioned adjustments to the PBIL algorithm. Specifically, we have altered the probability update rule and sample generation procedure while removing the mutation rule. The pseudocode for OPBIL is presented in Algorithm 2.

In line 2 we initialize the probability matrix to reflect the amount of prior knowledge we have about the solution, $m_i = 0.5$ reflects a total lack of prior information. The main algorithm is then listed in lines 3 - 30 and will terminate after the predefined number of iterations, ω, has been reached.

The $k/2$ random samples are generated in line 5 based on the probabilities represented in $\mathbf{M}$, where k is the total number of samples desired. From this set R_1 we then create an opposite for each of the $k/2$ elements (in line 6) represented by the set $\breve{R}_1$ in accordance with Section 3.2. In lines 8 and 9

the sample best η and global best $\mathbf{B}^*$ solution are updated from the newly generated samples and the previous global best solution, respectively. The function *select_best*$(\cdot)$ will select the "best" solution from its arguments with respect to the evaluation function f. Typically, this will be the min or max function. Then, the probabilities required for update of $\mathbf{M}$ are computed in lines 11 and 12, respectively.

Probability updating is performed in lines 14 to 28. First we determine whether a new global best solution was found ($\eta = \mathbf{B}^*$) or if the sample best solution should be used to move the focus of the search ($random(0, 1) < p_{amp}$, where $b = 0.01$). So, this probability update serves both to exploit the global best solution, and to quickly tend away from it when no new optima are found in its vacinity. The update itself is peformed in line 16.

If the first test fails, we may then perform a decay of the values of $\mathbf{M}$, with probability p_{decay}, which decreases as Δ increases (line 17). This test is intended to be successful in the iterations directly following the update in line 16. The decay in lines 21 to 27 serves to slowly tend away from the global best solution. The update in line 23 pushes the values of $\mathbf{M}$ away from the local or global best (determined in line 18) and the update in line 25 pushes those values which differ between the η and $\mathbf{B}^*$. So, this part of the algorithm is intended to prevent convergence and aide in exploration by very small updates, resulting in smooth transitions.

Algorithm 2 Pseudocode for the OPBIL algorithm

Require: Maximum iterations, ω
Require: Number of samples per iteration, k

{Initialize probabilities}
$\mathbf{M}_0 = m_{i..l} = 0.5$

for $t = 1$ to ω **do**
  {Generate samples}
  $R_1 =$ generate_samples($k/2$,$\mathbf{M}$)
  $\breve{R}_1 =$ generate_opposites(R_1)

  {Find best sample}
  $\eta =$ select_best($\{R_1 \cup \breve{R}_1\}$)
  $\mathbf{B}^* =$ select_best($\mathbf{B}^*, \eta$)

  {Compute probabilities}
  $p_{amp}(\Delta) = 1 - e^{-b\Delta}$
  $p_{decay}(\Delta; f(\mathbf{B}^*), f(\eta)) = \dfrac{1 - \frac{f(\mathbf{B}^*) - f(\eta)}{f(\mathbf{B}^*)}}{\sqrt{\Delta + 1}}$

  {Update $\mathbf{M}$}
  if $\eta = \mathbf{B}^*$ OR $random(0,1) < p_{amp}$ **then**
    $\Delta = 0$
    $\mathbf{M}_t = (1 - \rho)\mathbf{M}_{t-1} + \rho\eta$
  else if $random(0,1) < p_{decay}$ **then**
    if $random(0,1) < p_{decay}$ **then**
      use $\mathbf{B}^*$ in line 23 instead of η
    end if
    for all i, j each with probability $< p_{decay}$ **do**
      if $\eta_{i,j} = \mathbf{B}^*_{i,j}$ **then**
        $m_{i,j} = m_{i,j} \cdot \begin{cases} 1 - \tau \cdot random(0,1), & \text{if } \eta_{i,j} = 1, \\ 1 + \tau \cdot random(0,1), & \text{otherwise} \end{cases}$
      else
        $m_{i,j} = m_{i,j} \cdot \begin{cases} 1 + \tau \cdot random(0,1), & \text{if } \eta_{i,j} = 1, \\ 1 - \tau \cdot random(0,1), & \text{otherwise} \end{cases}$
      end if
    end for
  end if
  $\Delta = \Delta + 1$
end for

4 Experimental Results

In this section we first provide a description of the benchmark functions used for our analysis. Then we provide a comparison between the accuracy achieved by PBIL and OPBIL algorithms on these functions. A comparison between PBIL and OPBIL on 10 Traveling Salesman Problems (TSP) is also provided. It must be noted that the exact same number of samples was used to compare all algorithms (i.e. no overhead for using opposition) and the probabilities are bounded such that $m_{i,j} \in [1/(c \times d), 1 - 1/(c \times d)]$ for both PBIL and OPBIL where c, d are the number of dimensions and bits per dimension, respectively.

The results presented below for the benchmark functions were obtained by fixing PBIL's learning rate $\alpha = 0.25$, mutation shift $\gamma = 0.1$ and mutation probability $\beta = 0.1$. These values were empirically decided from the set $\{0.05, 0.10, 0.15, 0.25, 0.5\}$. Parameters for OPBIL were determined in a similar manner, we set the amplification $\rho = 0.05$, decay $\tau = 0.0005$. The function we use to determine the distance between opposite guesses is

$$\xi(t) = \max(1, e^{-0.01t}) \tag{11}$$

where t is the current iteration of the algorithm. We did not fully explore the range of possible functions for $\xi(t)$, but of those attempted Equation (11) performed best. Values of the decay constant from the set $\{0.005, 0.01, 0.05, 0.10, 0.25\}$ were examined, resulting in the choice 0.05, which tended to yield the most desirable results.

We also examined two slightly different versions of OPBIL. The first uses $\xi(t)$

as a hard limit (i.e. opposites differ by exactly $\xi(t)$ bits) which will be referred to as hard-OPBIL. The second version, referred to as soft-OPBIL, uses $\xi(t)$ as a soft limit whereby opposite points will differ by a maximum value of $\xi(t) > 0$. The value is actually randomly selected according to a uniform distribution over $[1, \xi(t)]$. While the behavior and results are very similar for both approaches, both are provided as their results do differ in some situations.

4.1 Benchmark Functions

In order to evaluate the efficacy of the OPBIL algorithms we have utilized four common optimization problems. Each of these functions have been widely used in the field of evolutionary computation as benchmark tests. They were designed to examine deceptivity in searches, meaning the propensity of a search to be led away from the globally optimal value by seemingly more desirable local optimums [32]. Also, two of the Whitley functions (described below) were designed to also consider attractors. That is, areas of local optimality, as opposed to a single locally optimal value. These functions are important because many difficult real world problems exhibit these characteristics. Each of these functions take the form $f\colon (\{0,1\}^m)^n \rightarrow \mathbb{Z}^*$, where n is the number of dimensions of size m in a solution representation.

4.1.1 Goldberg's 3-bit Deceptive Function

Goldberg's multi-modal 3-bit deceptive function [5] utilizes 3 bits per each of the n dimensions and is evaluated according to Equation (12). This is a multi-modal function, meaning there exist more than one local optima, but only a single global optima.

$$f(x) = \sum_{i=1}^{n} h(x_i) \tag{12}$$

Each 3-bit pattern is evaluated by the $h(\cdot)$ function which is summarized in Table 1. The values represent the corresponding evaluation for a given three bit pattern with the goal of minimizing $f(x)$. The global optimum for this minimization problem occurs when each dimension has a bit pattern (111), which has an evaluation of 0.

[Table 1 about here.]

4.1.2 Whitley's Functions

Whitley's 3 and 4-bit deceptive functions are similar to Goldberg's, but are considered more difficult to solve [32]. They are also multi-modal, and were designed to examine deceptivity and attractor basins in searching. The corresponding evaluation for dimensions composed of 3 or 4 bits are given in Tables 2, 3 and 4. The former two functions contain attractive basins. As with Goldberg's function, the optimal values occur when the evaluation of the solution is equal to zero.

[Table 2 about here.]

[Table 3 about here.]

[Table 4 about here.]

4.2 Experiments on Diversity

In this section we will briefly compare the amount of diversity for PBIL and the OPBIL algorithms. We will only show a single result because the diversity of OPBIL is controlled through equation (11) and therefore will always appear similar. In order to compare the expected behavior of diversity we will arbitrarily use the results from Goldberg's 3-bit deceptive function with 100 dimensions and a sample size of 4. The results are the result of an average of the best solution found over 30 trials of the algorithms.

Figure 2 shows the amount of all-pairs-diversity as calculated by equation (2). As expected, the hard-OPBIL diversity is greatest but essentially the same curve is also observed for the soft-OPBIL algorithm since they are both controlled by the same mechanism. The diversity for PBIL converges to a steady state at an extremely early stage in the learning process and remains at that level. Both OPBIL algorithms exhibit a much higher diversity for approximately 75% of the 2500 iterations. This allows for a greater exploration of the search space during the first half of learning and permits the algorithms to fine-tune their results during the latter stages.

[Fig. 2 about here.]

In order to provide insight into the relationship between the diversity and the actual results yielded by each algorithm we now examine Figure 3. It is apparent that the increased diversity results in a slower convergence rate for the OPBIL algorithm as it explores more of the search space and in fact has not yet converged at iteration 2500. Nevertheless, the relationship between diversity, convergence speed and accuracy is highlighted. It should also be

pointed out that in general too small a sample size may not make effective use of the increased diversity, although not observed in these experiments.

[Fig. 3 about here.]

Figure 4 shows the results of increasing the number of samples from 4 to 20. Comparing with the results in figure 3, the rate of convergence is increased and a more desirable final result is obtained by all the algorithms. As described above, this is a consequence of more information being provided to the algorithm, implying the increased diversity was more useful than previous. This behavior is also observed in PBIL, but since the diversity is relatively much lower than for the OPBIL algorithms the impact of these added samples on convergence rate is much lower.

[Fig. 4 about here.]

4.3 Experiments on Parameter Control

In this section we will examine the effect of the amplification (ρ) and decay (τ) parameters on the outcome of OPBIL. To do this we will fix the dimensionality of the problem to 100 and select Goldberg's 3-bit deceptive problem as a case study. We will also restrict ourselves to the soft-OPBIL algorithm (both hard- and soft-OPBIL behave similarily) using a total of 10 samples at each iteration. We first fix $\tau = 0.0005$ and vary the value for $\rho = \{0.01, 0.05, 0.15, 0.25, 0.50\}$. This experiment is aimed at examining the influence of the reinforcement signal towards new best solutions $\mathbf{B}^*$. The presented results have been averaged over 30 trials.

A plot of the behavior of the OPBIL algorithm for these experiments is presented in figure 5 where the impact of ρ is observed in terms of convergence rate and accuracy. In general, the convergence rate increases as ρ approaches 1. For $\rho = 0.01$ the convergence is extremely low which results in the algorithm not finding a very good solution by termination at iteration 2500. The other settings all achieve a similar final result, although values of $\rho = \{0.15, 0.25, 0.50\}$ converge at the higher rate. By experimentation we found that a setting of $\rho = 0.05$ has the tendancy to yield the best final result (at least for the benchmark functions and TSP data used in this paper). Nevertheless, these experiments clearly show the effect of ρ on OPBIL.

[Fig. 5 about here.]

Figure 6 presents the consequence of varying $\tau = \{0.0005, 0.001, 0.01, 0.05, 0.1\}$ while fixing $\rho = 0.05$ with respect to the convergence and accuracy of OPBIL. From this case study, it can be seen that larger values of τ imply a lower degree of exploration and results in a rapid convergence to a poor local optima. This behavior is a result of moving away from the current best solution at a high rate without fully exploring the nearby area and thus it is very hard to discover a quality solution. As τ is lowered the rate of convergence is slowed and the algorithm can explore more of the search space at a more reasonable rate. Essentially, the difference between relatively high and low values of τ is the rate of diffusion through the search space where a larger value effectively jumps from solution to solution and lower values have a smoother transition. Through empirical tests we have found that $\tau = 0.0005$ usually yields the most desirable final result.

[Fig. 6 about here.]

These results have provided insight into the control of exploration/expoitation ability of the soft-OPBIL algorithm (the hard-OPBIL behaves similarily). The tradeoff between exploration and exploitation can easily be controlled through the τ and ρ parameters, and it is important to realize that the specific values of τ and ρ for a given problem could differ greatly. Additionally, the shape of $\xi(t)$ which controls the distance between a guess and its opposite will also impact the quality of the search. As with any parameterized algorithm, the better result will occur when the parameters complement each other to yield the desired response.

4.4 Experiments on Accuracy

This section will provide a comparison between PBIL, soft-OPBIL and hard-OPBIL with regards to the final solution quality. To do this we utilize all four benchmark problems described in the previous subsection and vary the dimensionality from $D = 50, 100, 200$. For the Goldberg and Whitley 3-bit functions we run the algorithms for 2000, 2500 and 3000 iterations, corresponding to each dimensionality, and 2500, 3000, 3500 iterations for the 4-bit Whitley functions, respectively. Additionally, we examine four different sample sizes, $S = 4, 10, 20, 30$, during each iteration of the algorithm. All results are averaged over 30 runs where we report the mean μ and standard deviation σ in each table.

To examine whether the results are statistically significant we employ a two sample Kolmogorov-Smirnov test at a 0.95 confidence level. In the following tables, bolded values represent statistically significant results. If PBIL is bolded then its value is significantly superior to both soft-OPBIL and

hard-OPBIL; if the value is italicized then it is significant with respect to max(soft-OPBIL,hard-OPBIL). When a value for soft or hard-OPBIL is bolded then it is statistically significant when compared to the result found by PBIL.

The results for the 3-bit Whitley attractor function are provided in Table 5. For $D = 50$ all except the 4 sample experiment was found to be statistically equivalent, PBIL was found more desirable only when compared to hard-OPBIL. When the dimensionality is increased to 100 we find that for the smaller sample size both OPBIL algorithm results are significantly lower than PBIL for the small sample size of 4. For sample sizes of 10 and 20, the results favor PBIL, and the three algorithms are statistically the same for $S = 30$. Further increasing the dimensionality to $D = 200$ increases the problem difficulty significantly and PBIL is statistically outperformed in all experiments by very wide margins.

[Table 5 about here.]

Table 6 presents the results for Goldberg's 3-bit deceptive problem. When the dimension is 50, PBIL is found to have significant results for sample sizes of 10, 20 and 30, respectively. However, when $D = 100$ this situation is reversed and the OPBIL algorithms result in more desirable outcomes for sample sizes of 4,10 and 20. The sample size of $S = 30$ is statistically equivalent. For the 200 dimensionality instances OPBIL finds significantly better results for all sample sizes. Furthermore, for all dimensionalities the results for OPBIL are essentially the same, within each dimension size, when $S \geq 10$. To a lesser extent this seems true for PBIL, but for dimension sizes of 100 and 200, it seems as though the results are approaching those found by OPBIL. Also for these instances OPBIL algorithms require only 10 samples to achieve better

results than those found by PBIL at 30 samples.

[Table 6 about here.]

Table 7 exhibits the results from experiments on Whitley's 4-bit attractor problem. In all experiments both OPBIL algorithms achieve results which are statistically favorable compared to PBIL. Similar to the previous results for Goldberg's function OPBIL is able to find results with only 10 samples that are statistically significant at the 0.95 confidence level when compared to larger sample sizes for PBIL. In most cases, especially those where $D = 200$ the margin between the PBIL and OPBIL results is relatively large. We also find that in most of the results the standard deviation of OPBIL results is relatively low compared to PBIL.

[Table 7 about here.]

Figure 7 shows an example of the convergence of the three compared algorithms for Whitley's 4-bit attractor problem for $D = 200$ and $S = 20$. The PBIL algorithm rapidly converges within the first 500 iterations. On the other hand both OPBIL algorithms converge at a comparatively slower rate. At the 1500^{th} iteration the OPBIL results begin to improve over the already converged PBIL algorithm, and eventually yield a much better outcome. This rapid PBIL convergence was characteristic of its behavior for all of the experiments and the much slower convergence rate of OPBIL was also common throughout. The two curves for the OPBIL algorithms are also very similar, which tended to be found for all experiments.

[Fig. 7 about here.]

The results from the 4-bit deceptive Whitley function are given in Table 8.

Except for the instance where $D = 50$ and $S = 30$ all results are statistically in favor of the OPBIL algorithms. The actual difference in results obtained by each OPBIL algorithm is relatively large compared to PBIL, as the problem size increases this difference is many factors large, for example for $D = 100$, $S = 30$ the final result of OPBIL is about 1% that of PBIL. As with the previous two experiments, OPBIL requires much less samples to achieve much better results than those found with PBIL.

[Table 8 about here.]

4.5 Summary Results for Benchmark Funtions

We first provided a representative example of the increased diversity induced by opposition into the OPBIL algorithms by comparing the all-pairs-diversity. We showed that OPBIL does in fact maintain a higher degree of diversity than traditional PBIL, and that the diversity is controlled according to the $\xi(t)$ function. After examining the relationship between diversity and the final result obtained at each iteration we observed that as the sample size increased, OPBIL was typically able to make a more effective use of the diversity.

We have also compared the results obtained with the hard and soft versions of the OPBIL algorithm to those discovered by PBIL. To summarize the above results:

- 35/48 (hard and soft) OPBIL results were statistically significant.
- 6/48 PBIL results were significant.
- 7/48 values were statistically indistinguishable.

Additionally, for the Whitley 4-bit problems OPBIL algorithms were found to yield statistically significant results in 23 or the 24 instances. This highlights the ability of PBIL to successfully utilize the diversity for quickly exploring the search space. To provide a general idea as to the improvement of OPBIL over PBIL for all instances we calculate the average percentage improvement as

$$P_{i,j} = \frac{PBIL_{i,j} - OPBIL_{i,j}}{\max(PBIL_{i,j}, OPBIL_{i,j})} \tag{13}$$

where we substitute the results for the soft and hard OPBIL versions into $OPBIL_i$, $i = 1..3$ is the current dimension size ($D = 50, 100, 200$) and $j = 1..4$ is the current sample size ($S = 4, 10, 20, 30$). We calculate this value for each of the four problems. A value of $-\infty$ corresponds to instances where PBIL found the optimal solution and OPBIL did not. In contrast we use ∞ to represent the opposite situation where OPBIL discovers an optimal and PBIL does not. If $P_{i,j} = 0$ then both algorithms found the optimal.

Table 9 shows the average improvement of results using soft-OPBIL versus PBIL over all four test problems (we arbitrarily decided on soft-OPBIL since the results between soft- and hard-OPBIL were statistically insignificant). In total 9/48 of the instances have $P_{i,j} < 0$, corresponding to PBIL finding better results (not all significant). In three cases each of PBIL and OPBIL found optimal values (two instances both found the same optimal), and the remaining 35/48 values OPBIL improved over PBIL. For values where $P_{i,j} > 0$ the closer to 1, the larger the improvement. A value > 0.99 means that OPBIL's final result was only about 1% that of PBIL. Correspondingly, a value of $P_{i,j} < 0$ is shows the percentage improvement PBIL exhibited over

OPBIL. Although not all values are significant this table provides a general idea as to the improvement OPBIL has.

[Table 9 about here.]

We also discovered that as D increases, the benefit of using opposition typically also increases with respect to PBIL. That is, the accuracy of results found with OPBIL increases as the problem becomes more difficult. Also, the rate of convergence of OPBIL as implemented here is slower than PBIL but can be made more rapid by a different choice of $\xi(t)$ and learning parameters ρ and τ. In many of the instances results obtained by OPBIL required less samples to achieve (or better) the results found using more samples and PBIL.

Additionally, the two possible versions of the OPBIL algorithm were found to yield very similar results in all the experiments. This behavior leads to the conclusion that there is no significant difference in the results of soft-OPBIL and hard-OPBIL. Therefore, either version of the algorithm can be arbitrarily selected for these problems.

4.6 Experiments on the Traveling Salesman Problem

Benchmark functions typically do not capture the complexities of real-world situations. Ten Traveling Salesman Problem (TSP) instances were selected from the TSPLIB [26] to further examine the ability of OPBIL. We test its performance against results obtained by PBIL as well as a Genetic Algorithm (GA).

Let a graph $G = (V, E)$ of vertices $i, j \in V$ and undirected, weighted edges

$e_{i,j} \in E$ which connect vertices i and j represent cities and travel routes between cities, respectively. Further, let each $e_{i,j}$ have an associated weight $w_{i,j} \geq 0$ which represents the travel cost between cities, where if $w_{i,j} = 0$, then the edge is not travelable. Then, the goal is to determine which of the $\frac{(|V|-1)!}{2}$ Hamiltonian cycles has minimum total cost (sum of weights along chosen edges), corresponding to the shortest path from a starting city which visits all other cities and returns back. Typically, only an approximate solution is possible due to the large search space size for large $|V|$ (assuming a large number of edges as well).

To encode the TSP we use the binary representation adopted in [1] whereby each solution (and also probability matrix $\mathbf{M}$) is of dimensions $|V| \times \lceil \log_2 |V| \rceil$, where $|\cdot|$ represents the cardinality of the set (number of cities). Each index of this representation corresponds to a city, and the binary value at each index is converted to a decimal value when constructing a tour/path. The tour begins with the city with the lowest associated integer value and continues in this manner until all cities have been visited. In the event that two or more cities have the same integer value, they are visited in the order of increasing index value. As pointed out in [1] this representation is not ideal, and many other are in existence which have led to better results. However, since the goal here is to simply compare the results to each other, the final result with respect to the optimal or best known solution is not as important as the relative difference between these algorithms.

To widen this comparison we also include results obtained by a standard genetic algorithm (GA). During the reproduction phase we use an n-point crossover [12] with probability 0.75. Alternatively, with probability 0.25 we mutate the representation by flipping $|V|/10$ bits at random. The selection

mechanism used was a 2-way tournament method with a selection pressure of 0.8 [12]. The parameter settings for OPBIL are the same as above ($\tau = 0.0005$, $\rho = 0.05$), $\xi(t)$ is as shown in Equation (11), and the remainder of the OPBIL algorithm is the same as listed in Algorithm 2. For PBIL we use a learning rate $\alpha = 0.15$, mutation probability $\beta = 0.01$ and mutation shift $\gamma = 0.20$. For all three algorithms the population/sample size at each generation/iteration was equal to the number of cities ($|V|$).

Table 10 shows the results for the 10 TSP problem instances averaged over 30 runs. Instances eil51 and berlin52 were run for 2000 iterations, eil76 was run for 2500 iterations and the remaining problems were all run for 3500 iterations. According to a Kolmogorov-Smirnov test at a 0.05 significance level, we find that the average of all final results (μ) are statistically in favor of OPBIL. Also, for all experiments the standard deviation σ was lower for OPBIL than for PBIL or the GA, implying more reliable results.

[Table 10 about here.]

In Figure 8 a plot of the averaged results for the kroA100 problem is presented. As with previous experiments PBIL converges relatively quickly compared to the slower rate of OPBIL. This same behavior is seen with the GA. However, at approximately iteration 2000 OPBIL overtakes PBIL and continues to improve until termination (it crosses the GA at about iteration 1500). This curve is characteristic of the setting of OPBIL parameters used in this paper, and allows for improved exploration during early stages of the algorithm.

[Fig. 8 about here.]

5 Conclusions and Future Work

In this paper we have proposed the use of opposition-based computing concepts to improve population based incremental learning. Our proposed method is based on an increased diversity in the generated samples. We were also able to provide theoretical evidence to support the claim of increased diversity when considering opposition as opposed to independent random sampling. This increase in diversity and altering of the sampling strategy led to a modification of the probability update rule of PBIL, and further led to the creation of the soft and hard versions of the OPBIL algorithm. Although, we find that there was generally no statistical difference between there results.

Our experiments confirm that using opposition according to the OPBIL framework can lead to an improvement in accuracy (at the expense of slower convergence) over traditional PBIL. Indeed, we observed that $35/48 = 72.9\%$ of the results from OPBIL were statistically significant and only $6/48 = 12.5\%$ results from PBIL favored. Moreover, results obtained using OPBIL for problems of high dimensionality were found to be far superior to PBIL. We showed experimentally that as the sample size increased it was usually the situation that both the OPBIL algorithms made increasingly efficient use of the diversity induced by opposition. Additionally, when considering 4-bit benchmark functions OPBIL yielded $23/24 = 95.5\%$ statistically significant results. Many real-world problems are of very high dimensionality and therefore it seems as though OPBIL would be a relatively significant improvement over PBIL.

We also compared the two approaches on 10 TSP instances composed of betweeen 50 and 130 cities. In all situations OPBIL was able to achieve a statis-

tically significant result over PBIL and a GA. We used a simple and not very robust, yet common, problem representation and no heuristic information. Incorporating more advanced options in these two areas can greatly improve the results. However, for the purpose of this paper the comparison between OPBIL, a GA and PBIL does not require such additions.

Possible directions for future research involve both theoretical and practical aspects. Firstly, research into a better understanding of opposition-based computing as a whole is required. With reference to the OPBIL algorithms, deriving convergence proofs, as well as examining the rate of error decrease with respect to the number of samples per iteration are important questions, as is the amount of information gain per sample. The latter question can lead to a better understanding of selecting an appropriate sample size. Further experiments on larger real-world problems and comparisons to other search algorithms such as simulated annealing or evolutionary approaches are also possible directions to follow.

Acknowledgements

We would like to thank the anonymous reviewers for their valuable comments. This work was funded by the Natural Sciences and Engineering Research Council of Canada.

References

[1] S. Baluja. Population Based Incremental Learning - A Method for Integrating Genetic Search Based Function Optimization and Competitive

Learning. Technical report, Carnegie Mellon University, 1994. CMU-CS-94-163.

[2] J. Branke, C. Lode, and J. L. Shapiro. Addressing sampling errors and diversity loss in UMDA. In *Proceedings of the 9th annual Genetic and Evolutionary Computation Conference*, pages 508–515, 2007.

[3] S. Bureeret and K. Sriworamas. *Soft Computing in Industrial Applications*, volume 39/2007 of *Advances in Soft Computing*, chapter Population-Based Incremental Learning for Multiobjective Optimisation, pages 223–232. Springer, 2007.

[4] K. A. Folly. Robust controller design based on a combination of genetic algorithms and competitive learning. In *International Joint Conference on Neural Networks*, pages 3045–3050, 2007.

[5] D. Goldberg, B. Korb, and K. Deb. Messy Genetic Algorithms: Motivation, Analysis, and First Results. *Complex Systems*, 3:493–530, 1989.

[6] T. Gosling, J. Nanlin, and E. Tsang. Population Based Incremental Learning Versus Genetic Algorithms: Iterated Prisoners Dilemma. Technical report, University of Essex, 2004.

[7] W. K. C. Guthrie. *The Earlier Presocratics and the Pythagoreans (A History of Greek Philosophy*, volume 1. Cambridge University Press, 1979.

[8] M. Hohfeld and G. Rudolph. Towards a Theory of Population-Based Incremental Learning. In *Proceedings of the 4th IEEE Conference on Evolutionary Computation*, pages 1–5, 1997.

[9] Q. Jiang, Y. Ou, and D. Shi-Du. Optimizing curriculum scheduling problem using population based incremental learning algorithm. In *Second Workshop on Digital Media and its Application in Museum and Heritages*, pages 448–453, 2007.

[10] S. Little. *Taoism and the Arts of China*. University of California Press,

2000.

[11] S. J. Louis and G. J. E. Rawlins. Syntactic Analysis of Convergence in Genetic Algorithms. In L. D. Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 141–151. Morgan Kaufmann, 1993.

[12] M. Mitchell. *An Introduction to Genetic Algorithms*. The MIT Press, 1998.

[13] I. Petrovska and J. N. Carter. Using population-based incremental learning algorithm to quantify the uncertainty in model parameters. In *69th EAGE Conference and Exhibition incorporating SPE EUROPEC 2007*, 2007.

[14] S. Rahnamayn, H. R. Tizhoosh, and S. Salama. Opposition-based Differential Evolution Algorithms. In *IEEE Congress on Evolutionary Computation*, pages 7363–7370, 2006.

[15] S. Rahnamayn, H. R. Tizhoosh, and S. Salama. Opposition-based Differential Evolution Algorithms for Optimization of Noisy Problems. In *IEEE Congress on Evolutionary Computation*, pages 6756–6763, 2006.

[16] R. Rastegar and A. Hariri. The population-based incremental learning algorithm converges to local optima. *Neurocomputing*, 69(13–15):1772–1775, 2006.

[17] R. Rastegar, A. Hariri, and M. Mazoochi. A Convergence Proof for the Population Based Incremental Learning Algorithm. In *Seventeenth IEEE International Conference on Tools with Artificial Intelligence*, pages 387–391, 2005.

[18] S. Rudlof and M. Koppen. Stochastic hill climbing with learning by vectors of normal distributions, 1996.

[19] M. Sebag and A. Ducoulombier. Extending Population-Based Incremental Learning to Continuous Search Spaces. *Lecture Notes in Computer*

Science, 1498:418–427, 1998.

[20] J. L. Shapiro. Diversity loss in general estimation of distribution algorithms. In *Parallel Problem Solving from Nature IX*, pages 92–101, 2006.

[21] M. Shokri, H. R. Tizhoosh, and M. Kamel. Opposition-based Q(lambda) Algorithm. In *IEEE International Joint Conference on Neural Networks*, pages 646–653, 2006.

[22] F. Southey and F. Karray. Approaching Evolutionary Robotics Through Population-Based Incremental Learning. In *IEEE International Conference on Systems, Man, and Cybernetics*, volume 2, pages 710–715, 1999.

[23] H. Tizhoosh and M. Ventresca, editors. *Oppositional Concepts in Computational Intelligence*. Studies in Computational Intelligence. Springer-Verlag, 2008.

[24] H. R. Tizhoosh. Reinforcement Learning Based on Actions and Opposite Actions. In *International Conference on Artificial Intelligence and Machine Learning*, 2005.

[25] H. R. Tizhoosh. Opposition-based Reinforcement Learning. *Journal of Advanced Computational Intelligence and Intelligent Informatics*, 10(4):578–585, 2006.

[26] TSPLIB. http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsplib.html.

[27] Dictionary.com Unabridged (v 1.1). opposition. (n.d.). Retrieved February 19, 2007, from Dictionary.com.

[28] M. Vega-Rodriguez, D. Vega-Perez, J. Gomez-Pulido, and J. Sanchez-Perez. Radio network design using population-based incremental learning and grid computing with boinc. In *Applications of Evolutinary Computing*, volume 4448/2007 of *Lecture Notes in Computer Science*, pages 91–100. 2007.

[29] M. Ventresca and H. R. Tizhoosh. Improving the Convergence of Back-

propagation by Opposite Transfer Functions. In *IEEE International Joint Conference on Neural Networks*, pages 9527–9534, 2006.

[30] M. Ventresca and H. R. Tizhoosh. Opposite Transfer Functions and Backpropagation Through Time. In *IEEE Symposium on Foundations of Computational Intelligence*, pages 570–577, 2007.

[31] M. Ventresca and H. R. Tizhoosh. Simulated Annealing with Opposite Neighbors. In *IEEE Symposium on Foundations of Computational Intelligence*, pages 186–192, 2007.

[32] D. Whitley. Fundamental Principles of Deception in Genetic Search. In G. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 221–241. Morgan Kaufmann, 1991.

[33] M. Wineberg and F. Oppacher. Metrics for Population Comparisons in Evolutionary Computation Systems. In *Intelligent Systems and Control*, 2003.

[34] S. Yang and X. Yao. Experimental Study on Population-Based Incremental Learning Algorithms for Dynamic Optimization Problems. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, 9(11):815–834, 2005.

[35] S. Y. Yang, S. L. Ho, G. Z. Ni, J. M. Machado, and K. F. Wong. A new implementation of population based incremental learning method for optimizations in electromagnetics. *IEEE Transactions on Magnetics*, 43(4):1601–1604, 2007.

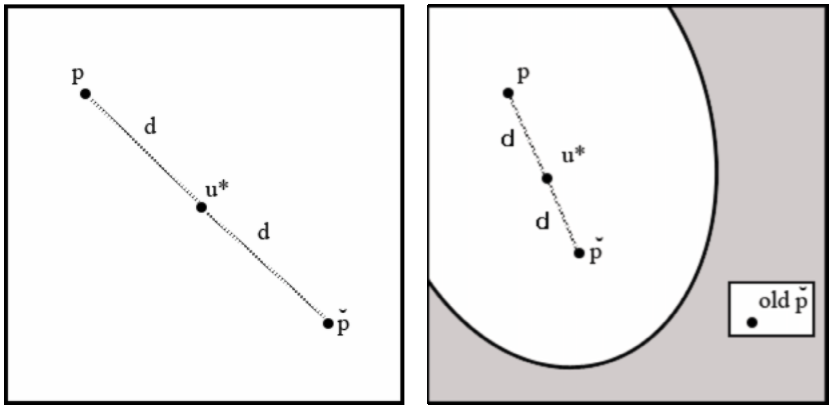
[36] B. Yuan and M. Gallagher. On the importance of diversity maintenance in estimation of distribution algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 719–726, 2005.

[37] Q. Zhang, T. Wu, and Liu B. A population-based incremental learning algorithm with elitist strategy. In *Third International Conference on*

Natural Computation, pages 583–587, 2007.

[38] K. Zhu and Z. Liu. Empirical Study of Population Diversity in Permutation-Based Genetic Algorithm. *Lecture Notes in Computer Science*, 3201:537–547, 2004.

List of Figures



(a) (b)

Fig. 1. (a) If u^* lies in the center of the search space then every point has an opposite and (b) if u^* is translated then points p, $\breve{p}$ can lie closer together. The shaded area represents elements which have no notion of opposite, and are essentially removed from consideration. In both cases we are considering opposites with respect to point p.

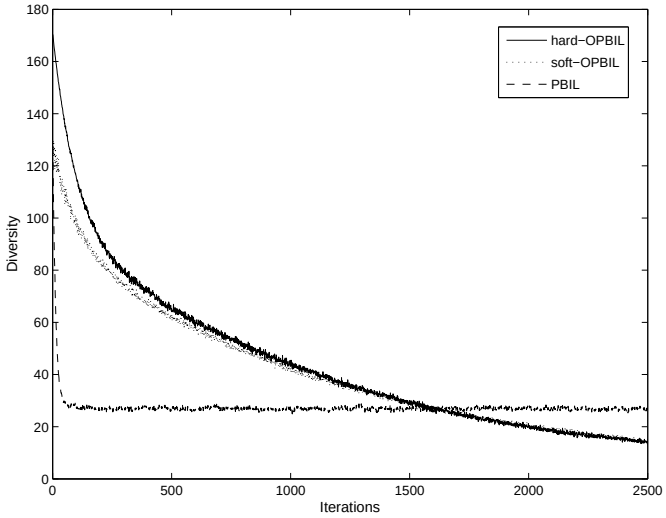


Fig. 2. All-pairs-diversity for the Goldberg deceptive function with 100 dimensions and 4 samples per iteration. As it can be seen the diversity for both versions of OPBIL maintain a much higher degree of diversity for 1900 of the 2500 iterations.

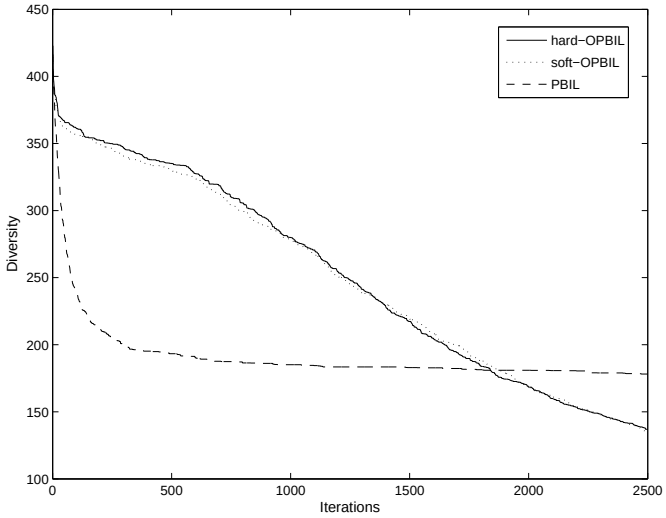


Fig. 3. The convergence curve of results obtained by each algorithm on Goldberg's function with 100 dimensions and 4 samples.

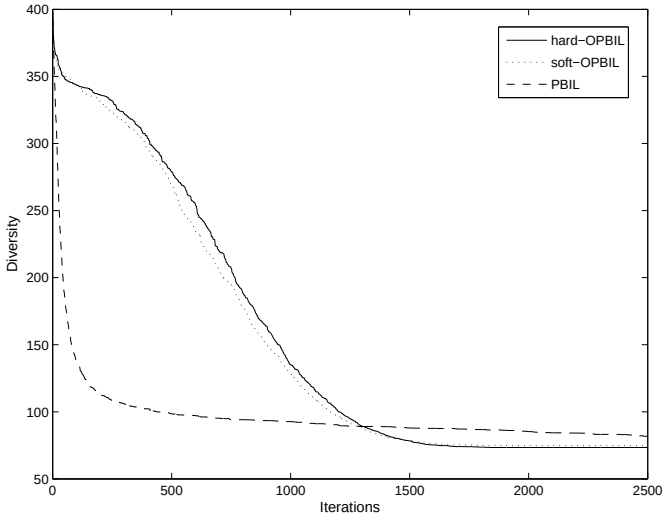


Fig. 4. The convergence curve of results obtained by each algorithm on Goldberg's function with 100 dimensions and 20 samples.

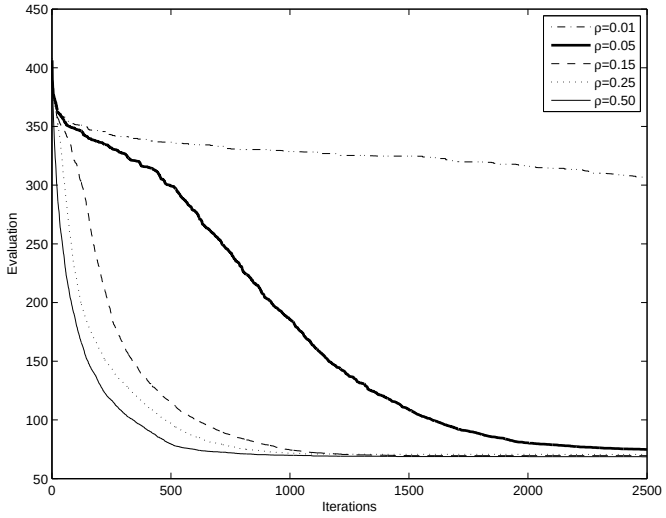


Fig. 5. The effect of varying ρ on OPBIL results for the Goldberg 3-bit deceptive problem with 100 dimensions. For all experiments $\tau = 0.0005$.

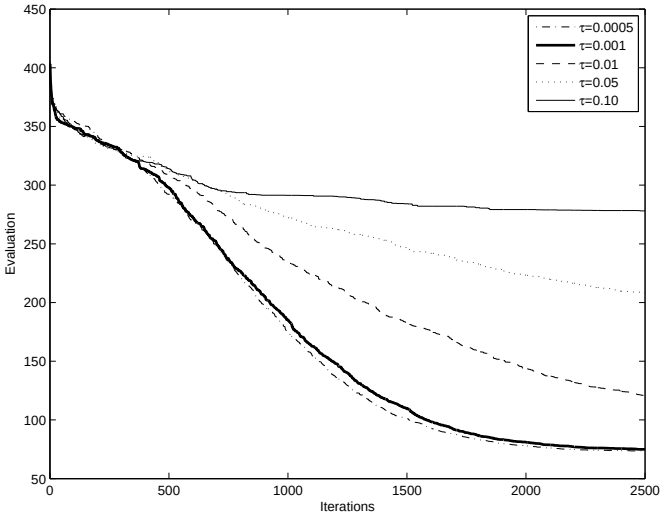


Fig. 6. The effect of varying τ on OPBIL results for the Goldberg 3-bit deceptive problem with 100 dimensions. For all experiments $\rho = 0.05$.

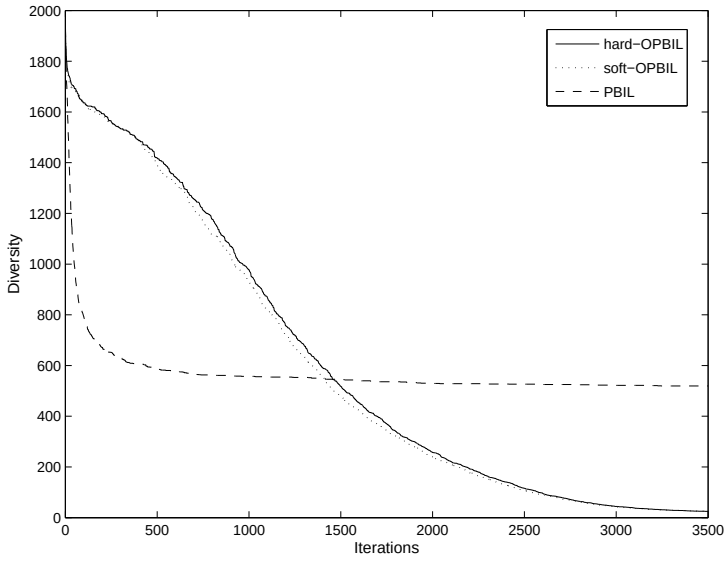


Fig. 7. A comparison of the convergence of PBIL and the two OPBIL algorithms on Whitley's 4-bit attractor problem where $D = 200$ and $S = 20$.

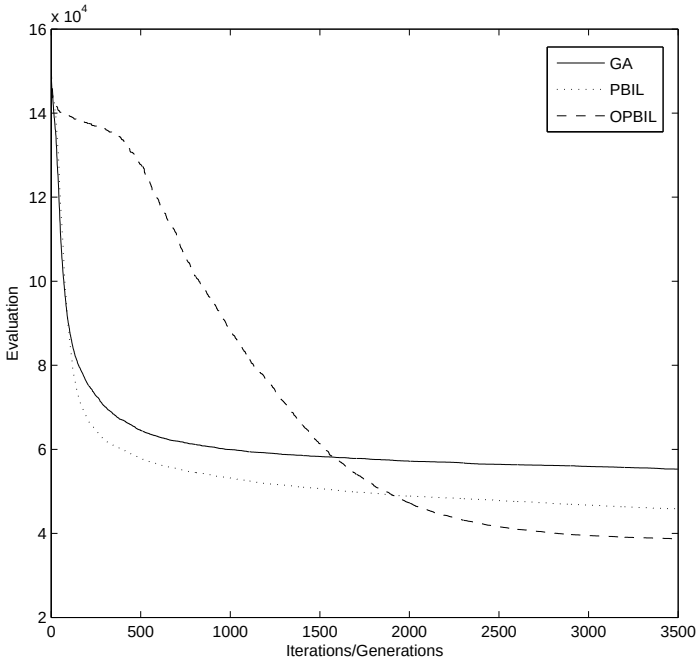


Fig. 8. Comparing OPBIL vs. GA vs. PBIL for the kroA100 TSP instance.

List of Tables

Table 1
Goldberg's 3-bit deceptive Evaluation Values

String	$f(x)$	**String**	$f(x)$
000	1	100	5
001	3	101	8
010	3	110	8
011	8	111	0

Table 2
Whitley's 3-bit attractor Values

String	$f(x)$	**String**	$f(x)$
000	28	100	26
001	22	101	14
010	0	110	0
011	0	111	30

Table 3
Whitley's 4-bit attractor Evaluation Values

String	$f(x)$	**String**	$f(x)$
0000	10	1000	28
0001	25	1001	5
0010	26	1010	5
0011	5	1011	0
0100	27	1100	5
0101	5	1101	0
0110	5	1110	0
0111	0	1111	30

Table 4
Whitley's 4-bit deceptive Evaluation Values

String	$f(x)$	String	$f(x)$
0000	2	1000	10
0001	4	1001	18
0010	6	1010	20
0011	12	1011	28
0100	8	1100	22
0101	14	1101	26
0110	16	1110	24
0111	30	1111	0

Table 5
Whitley's 3-bit attractor results for a minimization problem. In total 5/12 results are statistically favorable for OPBIL, 3/12 for PBIL (including italicized values) and 4/12 show no statistical significance. Bolded values are statistically significant.

S	PBIL		soft-OPBIL		hard-OPBIL	
	μ	σ	μ	σ	μ	σ
Dimensions = 50						
4	*5.867*	8.303	16.333	18.043	24.667	20.594
10	0.4667	2.556	1.400	4.272	0.4667	2.556
20	0.000	0.000	0.000	0.000	0.000	0.000
30	0.000	0.000	0.000	0.000	0.000	0.000
Dimensions = 100						
4	215.000	30.885	**141.400**	54.773	**125.733**	39.799
10	*15.667*	8.535	24.733	19.691	18.667	16.981
20	**0.4667**	2.556	9.333	11.819	10.733	11.441
30	0.000	0.000	2.800	8.544	5.600	10.136
Dimensions = 200						
4	1012.867	65.457	**575.333**	105.790	**541.933**	100.974
10	463.133	45.682	**145.533**	47.322	**132.067**	40.222
20	223.067	31.054	**78.133**	34.329	**90.333**	35.785
30	136.333	24.495	**52.733**	27.435	**52.667**	33.800

Table 6
Goldberg's 3-bit deceptive results: 7/12 results are statistically significant for OPBIL, 3/12 for PBIL and the remaining 2/12 results show no significance. Bolded values are statistically significant.

S	PBIL		soft-OPBIL		hard-OPBIL	
	μ	σ	μ	σ	μ	σ
Dimensions = 50						
4	53.433	4.470	56.833	7.715	56.933	7.172
10	**31.633**	3.864	36.967	2.942	38.267	3.051
20	**31.033**	3.783	37.233	3.170	39.167	2.276
30	**31.667**	3.220	36.900	2.496	36.433	3.014
Dimensions = 100						
4	178.133	8.307	**135.233**	14.301	**136.933**	7.983
10	111.933	5.065	**72.667**	4.551	**72.833**	4.308
20	81.800	5.410	**74.767**	4.329	**73.400**	4.182
30	74.167	5.867	73.533	3.181	73.500	4.329
Dimensions = 200						
4	490.633	11.775	**354.167**	23.343	**351.567**	24.605
10	370.600	10.088	**164.733**	10.116	**167.467**	7.505
20	301.900	7.397	**145.300**	6.889	**147.233**	6.511
30	268.567	7.789	**145.600**	6.273	**145.700**	6.993

Table 7
Whitley's 4-bit attractor results. In total 12/12 results are statistically favorable for OPBIL. Bolded values are significant.

S	PBIL		soft-OPBIL		hard-OPBIL	
	μ	σ	μ	σ	μ	σ
Dimensions = 100						
4	195.533	14.908	**132.000**	14.574	**129.533**	13.903
10	113.133	7.0795	**94.600**	2.358	**96.067**	1.929
20	90.733	3.657	**86.133**	1.961	**85.800**	3.377
30	85.400	4.182	**75.267**	2.651	**76.333**	3.241
Dimensions = 100						
4	621.733	29.812	**347.933**	25.685	**365.800**	34.971
10	411.000	14.429	**188.333**	4.901	**188.600**	4.461
20	305.800	10.121	**187.200**	4.859	**188.467**	4.946
30	263.867	10.372	**187.933**	4.5024	**188.533**	4.3607
Dimensions = 200						
4	1684.533	29.905	**966.333**	62.288	**997.667**	56.256
10	1288.200	31.759	**418.133**	14.799	**418.800**	11.775
20	1069.800	24.849	**366.667**	8.294	**368.333**	6.604
30	955.733	23.1441	**367.200**	7.155	**368.00**	8.485

Table 8
Whitley's 4-bit deceptive results. In total 11/12 results obtained with OPBIL are statistically significant and 1/12 are insignificant. Bolded values are statistically significant.

S	PBIL		soft-OPBIL		hard-OPBIL	
	μ	σ	μ	σ	μ	σ
Dimensions = 50						
4	77.000	8.469	**47.667**	14.003	**56.500**	19.920
10	28.500	6.585	**0.333**	1.826	**0.000**	0.000
20	6.167	3.395	**0.3333**	1.269	**0.000**	0.000
30	0.333	1.269	0.000	0.000	0.000	0.000
Dimensions = 100						
4	293.333	21.600	**172.233**	28.333	**188.633**	27.521
10	173.667	14.478	**11.833**	9.513	**12.00**	12.839
20	118.833	11.423	**2.333**	4.866	**1.500**	3.511
30	93.500	7.673	**1.500**	3.5111	**1.000**	2.842
Dimensions = 200						
4	929.267	31.228	**578.700**	50.813	**608.000**	48.984
10	660.033	29.100	**131.167**	36.547	**131.333**	25.049
20	519.600	22.443	**25.833**	16.033	**24.667**	17.117
30	459.200	22.895	**12.000**	9.966	**16.333**	15.025

Table 9
The average improvement in results over all test instances. Positive values are results where OPBIL outperformed PBIL. In total 37/48 values > 0, 9/48 are < 0, 2/48 are equal to 0

D	S=4	10	20	30
Whitley 3-bit Attractor				
50	-0.641	-0.667	0.000	0.000
100	0.342	-0.367	-0.950	$-\infty$
200	0.432	0.686	0.650	0.613
Goldberg 3-bit Deceptive				
50	-0.060	-0.144	-0.167	-0.142
100	0.241	0.351	0.0860	0.009
200	0.278	0.556	0.519	0.458
Whitley 4-bit Deceptive				
50	0.325	0.164	0.051	0.119
100	0.440	0.542	0.388	0.288
200	0.426	0.675	0.657	0.616
Whitley 4-bit Attractor				
50	0.381	0.988	0.946	∞
100	0.413	0.932	0.980	0.984
200	0.377	0.801	0.950	0.974

Table 10
Results for the 10 TSP problem instances. Bolded values are statistically significant.

	PBIL		GA		OPBIL	
Instance	μ	σ	μ	σ	μ	σ
eil51	590.17	30.60	687.93	40.87	**546.03**	20.94
berlin52	10676.56	812.60	12294.14	859.18	**9792.80**	548.73
eil76	846.94	57.71	986.99	74.75	**754.23**	44.49
kroA100	45871.62	3769.32	55285.64	5290.82	**38726.17**	2880.93
kroB100	45775.77	4352.58	56804.89	5324.88	**38300.59**	2782.29
kroC100	45758.08	3216.85	55560.71	4637.72	**37351.60**	2653.49
kroD100	44964.25	3324.76	56120.53	4833.36	**37229.58**	2327.22
kroE100	46057.64	3396.09	55657.73	4583.16	**38876.37**	2637.26
eil101	1112.97	60.10	1323.52	77.86	**960.99**	59.91
ch130	14400.64	933.67	16566.87	1573.22	**11638.47**	922.89