Symmetry Induction in Computational Intelligence

by

Mario Ventresca

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Systems Design Engineering

Waterloo, Ontario, Canada, 2009

Abstract

Symmetry has been a very useful tool to researchers in various scientific fields. At its most basic, symmetry refers to the invariance of an object to some transformation, or set of transformations. Usually one searches for, and uses information concerning an existing symmetry within given data, structure or concept to somehow improve algorithm performance or compress the search space.

This thesis examines the effects of imposing or inducing symmetry on a search space. That is, the question being asked is whether only existing symmetries can be useful, or whether changing reference to an intuition-based definition of symmetry over the evaluation function can also be of use. Within the context of optimization, symmetry induction as defined in this thesis will have the effect of equating the evaluation of a set of given objects.

Group theory is employed to explore possible symmetrical structures inherent in a search space. Additionally, conditions when the search space can have a symmetry induced on it are examined. The idea of a neighborhood structure then leads to the idea of opposition-based computing which aims to induce a symmetry of the evaluation function. In this context, the search space can be seen as having a symmetry imposed on it. To be useful, it is shown that an opposite map must be defined such that it equates elements of the search space which have a relatively large difference in their respective evaluations. Using this idea a general framework for employing opposition-based ideas is proposed. To show the efficacy of these ideas, the framework is applied to popular computational intelligence algorithms within the areas of Monte Carlo optimization, estimation of distribution and neural network learning.

The first example application focuses on simulated annealing, a popular Monte Carlo optimization algorithm. At a given iteration, symmetry is induced on the system by considering opposite neighbors. Using this technique, a temporary symmetry over the neighborhood region is induced. This simple algorithm is benchmarked using common real optimization problems and compared against traditional simulated annealing as well as a randomized version. The results highlight improvements in accuracy, reliability and convergence rate. An application to image thresholding further confirms the results.

Another example application, population-based incremental learning, is rooted in estimation of distribution algorithms. A major problem with these techniques is a rapid loss of diversity within the samples after a relatively low number of iterations. The opposite sample is introduced as a remedy to this problem. After proving an increased diversity, a new probability update procedure is designed. This opposition-based version of the algorithm is benchmarked using common binary optimization problems which have characteristics of deceptivity and attractive basins characteristic of difficult real world problems. Experiments reveal improvements in diversity, accuracy, reliability and convergence rate over the traditional approach. Ten instances of the traveling salesman problem and six image thresholding problems are used to further highlight the improvements.

Finally, gradient-based learning for feedforward neural networks is improved using opposition-based ideas. The opposite transfer function is presented as a simple adaptive neuron which easily allows for efficiently jumping in weight space. It is shown that each possible opposite network represents a unique input-output mapping, each having an associated effect on the numerical conditioning of the network. Experiments confirm the potential of opposite networks during pre- and early training stages. A heuristic for efficiently selecting one opposite network per epoch is presented. Benchmarking focuses on common classification problems and reveals improvements in accuracy, reliability, convergence rate and generalization ability over common backpropagation variants. To further show the potential, the heuristic is applied to resilient propagation where similar improvements are also found.

Acknowledgments

I cannot overstate my gratitude to my supervisor Dr. Hamid Tizhoosh. This dissertation began as an idea we discussed in September 2005 when I first joined the University of Waterloo. He allowed me a great deal of independence about the direction and goals of the thesis and was always supportive and ensured I kept on track with my ideas. Throughout these four years his encouragement, sound advice, great ideas and company have made this an excellent experience. The emphasis he places on quality publications and exposure to conferences has also greatly contributed to furthering my academic career. I look forward to further research and interactions with him.

The decision of Dr. Beatrice Ombuki-Berman to hire an undergraduate student who knocked on her office door one afternoon has proven one of the most influential in my career. From that day, I have learned so much and hope that in the future I will give the same opportunity to my own students. I am very happy that we have developed a friendship over these many years.

A big thank you must be sent to all my teachers and professors over the many, many years prior to joining the University of Waterloo. Among the many, special acknowledgment to Professor Brian Ross of Brock University whom has provided many letters of reference since my undergraduate days.

I would also like to extend an appreciation to my external examiner Dr. Weber, who flew from Chile to provide his knowledge and insight on this thesis. My committee members from Waterloo; Drs. Eliasmith, Fieguth and Karray additionally provided their valuable comments during my comprehensive examination. Each brought constructive criticism and without doubt this thesis is much better for their expertise.

A grateful thank you to the Natural Sciences and Engineering Research Council of Canada (NSERC), the Ontario Graduate Scholarship Program (OGSST) and the University of Waterloo for providing funding for my research is also in order. This support allowed me to concentrate on my research without the added stress of financial issues.

I proudly and with great sincerity thank my family (Dad, Babbo, Grandma, Cat, Steph, Matt, Jenna, Aunt Rose, Uncle Joe and Alicia) for their constant support and understanding. Working on my PhD has meant that I was not able to be with them nearly as much as I would like, but I always kept them in my thoughts.

My friends from before I could count, those who knew me before I could drive and those whom I am very grateful to have met throughout my university education all deserve a huge thanks for their patience. I wasn't available to hang out often, especially over the past year, so thanks for understanding.

For too many things to list, I must lastly but certainly not anywhere near least, thank Dagmara; my travel buddy, my goofball, my partner in crime, my love.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Numerous biological, natural or man-made structures and concepts exhibit symmetries as a fundamental design principle or as an essential aspect of their function [69]. Whether by evolution or design, symmetry implies potential structural efficiencies that make it universally appealing. Much of our understanding of the world is based on the perception and recognition of shared or repeated structures [53, 54]. As will be discused, the use of symmetry information has recently attracted the attention of many researchers in machine learning and computational intelligence.

In this thesis the idea of inducing symmetry via opposition-based computing concepts is explored with respect to computational intelligence algorithms. Symmetry in the search space is implicit in the proposed methods. Hence, a discussion of group theory is essential for providing an in-depth understanding. The actual mathematics of group theory are not required for chapters focusing on experimental verification, however, these symmetric structures are present nonetheless.

1.1 Symmetry

Symmetry is often used to convey two main lines of thought. Aristotle spoke of a sense or harmonious or esthetically pleasing balance reflecting beauty or perfection [69]. Unfortunately, this is a very imprecise definition which cannot be directly employed computationally. A more specific meaning refers to *pattern self-similarity* which can be demonstrated or proven using formal rules [148]. Depending on the context, these rules of symmetry will have different natures. For example, with respect to geometric transformations, spatial relationships or as an aspect of more abstract concepts such as language or music.

Symmetry has proven very useful for simplifying the mathematical description of physical phenomena [119]. At its simplest, symmetry occurs at the geometric level where objects exhibit symmetry when they are invariant to a transformation such as rotation or reflection. In many cases the symmetry being tested or employed is biased by our perceptions of reality. But, in actuality there may exist an infinite number of possible relationships that fit the definition of symmetry.

Computational intelligence and machine learning algorithms employ a solution representation (for example, a feature vector) to represent the problem under consideration. While these representations are abstractions of reality they nonetheless capture the essence of a physical system. To be more precise, a classical physical system [119]:

1. is identified by all possible configurations $\{S_\gamma\}$, for γ running over the coordinates or parameters,
2. has a time evolution described as
$$\alpha^t : S_\gamma \to \alpha^t S_\gamma \equiv S_{\gamma(t)}. \tag{1.1}$$

and a symmetry of the system is a transformation of the coordinates/parameters $g : \gamma \to g\gamma$ that:

1. induces an invertible map
$$g : S_\gamma \to gS_\gamma \equiv S_{g\gamma}, \tag{1.2}$$
2. does not change the dynamical behavior
$$\alpha^t g S_\gamma = \alpha^t S_{g\gamma} \equiv S_{(g\gamma)(t)} = S_{gy(t)} = g\alpha^t S_\gamma. \tag{1.3}$$

Group theory is the standard mathematical language used to describe symmetry. The requirement of an object to remain invariant with respect to a transformation group can impose restrictions on the form of the system. This thesis will use this property throughout, as will be shown in subsequent chapters. Group theory will be discussed further in Chapter 2.

1.1.1 Symmetry and Computational Intelligence

Recently, the application of symmetry-based methods to computational intelligence and machine learning has been attracting much research attention (see below). These methods can be used in a variety of ways to simplify and understand data or to adjust the behavior of the learning algorithm. One of the key challenges to turning the concept of symmetry into a computationally useful tool is to determine and adapt methods to the noisy, but nearly regular real world.

Vetter et al. [143] describe that general a priori knowledge of the world allows humans to recognize 3D objects significantly easier when they exhibit some degree of symmetry. They explain how symmetry-induced virtual views lead to the observed results. More specifically, humans use their intuitive knowledge to recognize shapes by inducing virtual views of the object. In [117] the facial symmetries of humans is analyzed using group theory. An algorithm is proposed which employs this information for face recognition by symmetry sensing from photographs.

The relatively new field of computational symmetry focuses on the practice of representing, detecting and reasoning about symmetry [64]. The reasons to study computational symmetry include:

1. symmetry seems to exist everywhere,
2. symmetry implies a structure which can be either helpful or harmful,
3. computation of symmetry is challenging,
4. few computational tools exist for dealing with real-world symmetries.

Using a variety of applications including robotics, periodic pattern perception and neuroradiology the power of group-theoretic-based methods was shown [64].

In [124], a probabilistic technique is proposed which reconstructs 3D surfaces from range images. The algorithm exploits the fact that most shapes possess symmetries which can be deduced even from the partial view gathered from the limited range images. More specifically, the method searches a restricted space of possible symmetries for the transformation which will most probably yield the true 3D shape.

Ramakrishna and Mow [90] analyze the group structure of the 2-dimensional sidelobe-invariant transformations for binary arrays. Using this characterization an efficient and exhaustive backtracking search algorithm is proposed which is able to reduce the search space. As a consequence, all optimal binary arrays with minimum peak sidelobe levels consisting of up to 49 elements are obtained and tabulated.

The application of group theoretic methods to metaheuristic local search for partitioning problems was examined in [9, 23, 55]. Group theory is used to characterize the underlying inter- and intra-orbit structures. The resulting theory is shown successful by extending the Tabu search algorithm and testing it on 65 benchmark instances of the unicost set covering problem.

General purpose graph symmetry problems require a quadratic runtime with respect to the number of symmetries and are of limited use on very large, sparse, symmetric graphs. Darga et al. [25] propose a faster symmetry discovery algorithm which further exploits the present symmetries. The results improved on the previous approaches runtime from multiple days to less than a second.

Exploitation of symmetries in real-time dynamic programming was examined in [76]. The approach finds that real world problems exhibit a high degree of symmetry which leads to a high amount of redundancy in the Markov decision process. The authors propose an efficient algorithm capable of exploiting these symmetries which is able to significantly improve overall execution time.

In [98] the role of search space symmetries on the design of evolutionary operators for the simple genetic algorithm is discussed. The authors use group theory to describe the concept of a mixing matrix which they use to determine properties of crossover and mutation methods. Situations when the operator commutes with the search space are paid special attention as they do not constrict the search.

In [57], Kondor concentrates on (a) learning on domains which have a non-trivial algebraic structure, and (b) learning in the presence of invariance. Drawing on many aspects of group theory a general framework for incorporating the ideas into support vector machines and Gaussian kernels is proposed. The efficacy of the approach is highlighted using ranking/matching and machine vision problems, although the framework is general enough to be applied to other problems and algorithms.

Most recently, evolutionary symmetric modular neural networks were proposed [130]. The paper utilizes group theory to represent symmetry and systematically search for it which improves the system's evolvability. The efficacy of the approach is shown for a quadruped robot in physically realistic systems. Evolved controllers are faster than those designed by hand and versus random mutations (versus symmetric).

1.1.2 Inducing Symmetry

The process of inducing symmetry commonly refers to the application of a symmetry transformation to some object [148]. That is, the transformation will cause a symmetry where one may not have been previously present. The typical process of determining which symmetries are valid for the object involves inducing various transformations and selecting the most probable outcome based on prior knowledge or according to some evaluation method. Example 1.1 shows the effect of inducing a symmetry on a given evaluation function.

Example 1.1 (Inducing symmetry.) *Consider a real-valued system and a symmetry transformation* $\Phi\colon \mathbb{R} \to \mathbb{R}$ *which takes real valued input and maps it to a single real value. The original function*

$$f(x) = \frac{1}{(x-3)^2 + .01} + \frac{1}{((x-9)^2 + .04} - 6 \tag{1.4}$$

is transformed using $\Phi(x) = \min(f(x), f(2-x))$*, where* $x \in [0, 2]$*. The resulting impact on* $f(x)$ *can be graphically seen in Figure 1.1.*

If an optimization or learning algorithm was employed to discover the minimum of a function $f(x)$, it may find it simpler to instead consider the transformed function. Here, it is assumed that x is bounded and $f(x)$ cannot have a limit at $\pm\infty$. In subsequent chapters this concept will be formalized and conditions on its success will be given.

1.1.3 Symmetry and Opposition

As discussed above, determining and exploiting regularities in an environment can have various benefits. However, in many cases it may not be possible to efficiently determine the most useful search space invariance unless prior information is available. Should a symmetry be found to exist the common approach is to compress the invariant states, which consequently reduces the search space size. That is, the search space $\mathcal{S}$ is partitioned into equivalence classes and search is conducted on the compressed space. This approach has recently been shown to have high potential in developmental and curiosity-driven robotics [107, 108].

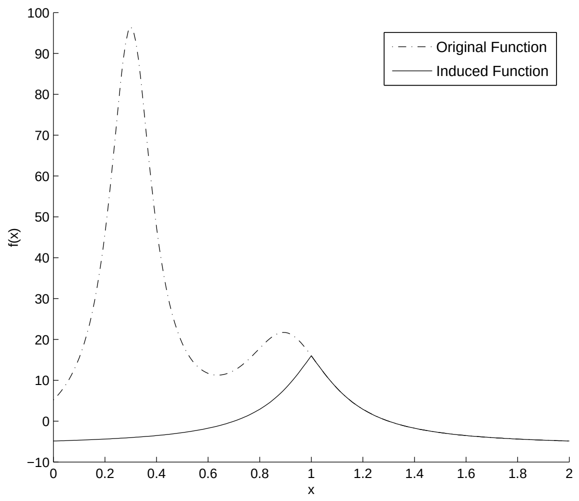

Figure 1.1: Symmetry inducing effect on a function, where symmetry is induced by $\Phi(x) = 2 - x$.

Very large search spaces tend to be inefficient to search completely. As such, various stochastic and concrete algorithms have been proposed which approximate the desired goal. Using symmetries to compress the search space will reduce the number of states. However, if the evaluation of symmetric elements is identical then there is no change in the respective probability distribution. Of course, a reduced $\mathcal{S}$ nevertheless should have a positive influence on a given algorithm.

The alternative view examined in this thesis is to induce a symmetry on non-equal elements of the evaluation[1] function, such as that presented in Example 1.1. Converse to the traditional approach, this idea effectively acts as a filter over possible evaluations (and hence the associated solutions in the search space) such that the probability distribution over fitness values favors the desired goal[2].

Opposition, more specifically opposition-based computing [127], has the effect of inducing a symmetry such that the probability distribution of evaluations is skewed as much as possible to reflect the goal(s) of the algorithm. This is accomplished by defining a neighborhood structure between elements of low and high evaluation. By simultaneously considering these alternatives, with respect to the problem (min/maximization) the theoretically most skewed distribution will be attained which will have the greatest chance of positively influencing the search. Another important consequence is that a reduction in search space is also attained. These ideas are highlighted by Figure 1.1 for a minimization problem.

There exists a large amount of research focusing on the integration of a priori and expert knowledge

[1]Depending on the algorithm being employed this will be the error surface, fitness landscape, etc.

[2]For example, consider minimization. A more desirable probability distribution is one which is skewed towards lower evaluations being more probable.

into computational intelligence (for examples see [2, 34, 62, 75, 93, 106, 120]). One may then ask whether opposition should be distinguished from other forms of this knowledge. In a given context, oppositional knowledge is very special as its processing bears virtually no uncertainty. For instance, let $x_1, \breve{x}_1 \in \mathcal{S}$. If x_1 is known to have a large evaluation then its opposite $\breve{x}_1$ must have a low evaluation (even without actually observing it) and one of these can be quickly dismissed. By recognizing oppositional elements and concepts this special type of a priori knowledge is recovered.

1.2 Overview of Results

In this thesis the relationship between symmetry and opposition-based computing is explored within the context of computational intelligence. Firstly, group theory is used to describe the existence of symmetry in $\mathcal{S}$ and conditions when symmetry can be induced. The next theoretical step involves understanding the role of opposition-based computing and when it should succeed. Experimental evidence using Monte Carlo optimization, estimation of distribution and gradient-based learning algorithms is provided in the second part of the thesis.

1.2.1 Part 1 - Motivations and Theory

Chapter 2: Group Theory and Search Space Symmetry This chapter will introduce group theory and some basic definitions and notations as well as provide a formalization of the symmetry concept. The relationship between the symmetry and permutation groups is described, as is their ability to partition the search space into equivalence classes.

The case where a group action G is isomorphic to $\mathcal{S}$ is given special attention. This formulation focuses on the issue of the search space itself being a group structure. Many popular combinatorial optimization problems have search spaces of this type. For example, the traveling salesman, job shop scheduling or vehicle routing problems. Discretized real function optimization problems can also have group structured search spaces. In this case discretization is required to ensure that $\mathcal{S}$ is countable as it is an assumption for all proofs. Alternatively, topological or Lie groups [45] could have been employed for real-valued search spaces. However, to maintain cohesiveness in the thesis only countable search spaces are assumed.

It is also interesting to determine when $\mathcal{S}$ can be given a group structure compatible with G. Here, compatability refers to the ability to map all group actions of G to an element of $\mathcal{S}$. Three cases were discerned:

1. There exists at least one element of the search space where different group actions result in different outcomes.
2. Only the identity leaves elements unchanged.
3. The only permutation of the search space which is commutative with the group action is the identity.

The neighborhood structure assigns to each element a neighbor element, and is given as a possible source for any symmetries present in the search space. Furthermore, the idea of defining the neighborhood structure on $\mathcal{S}$ is outlined as a means for inducing symmetry on the search space. Taking these symmetries into account, a simple algorithm which searches the space for a target value is also given.

Chapter 3: Opposition-Based Computing Here, the idea of neighborhood structures to induce symmetries on a search space is expanded as an early motivation for opposition-based computing (OBC). Central to this idea is the concept of an opposite map that defines pairs of elements to be considered opposite. As the previous chapter describes, this causes a partitioning of the search space into equivalence classes and hence is a permutation group structure. Examples are given to highlight the potential of a proposed naive algorithm (which forms the basis for a general OBC framework).

Requirements for a good opposite map are explored. The necessary condition is that the probability of paired opposite guesses yielding a more desirable result is greater than randomly choosing two guesses. From this, the optimal choice of opposite map is implied to be that which maximizes the distance between the original expected value and that of the transformed/symmetrized search space. The notions of degree of opposition, degree of competition and opposition mining are defined.

Computational issues and overhead of employing the opposite map are investigated as well. The two main issues focus on computing the most desirable element of a partition and balancing the convergence rate with sample diversity (i.e. exploration versus exploitation). With reference to the latter concern, algorithm $\mathcal{A}$ has an inherent convergence behavior which induces stronger bias as the search progresses. Increasing sample diversity will tend to slow this rate but will allow for greater exploration of the search space. Assuming $\mathcal{A}$ tends towards a relatively good optima the usefulness of diversity will diminish with the number of iterations. Hence, employing opposite strategies will not only require defining an opposite map, but a diversity-convergence control mechanism may also be needed.

To show the uniqueness of opposition a comparison to existing approaches is given. Antithetic variates are often used in Monte Carlo simulation as a variance reduction technique. The central tenet is to develop a mapping which is used to select elements (X_i, X_j) which are negatively correlated. In this way variance reduction over $N > 1$ samples X is accomplished since

$$var(\sum_{i=1}^{N} X_i) = \sum_{i=1}^{N} var(X_i) + 2\sum_{i<j} cov(X_i, X_j), \tag{1.5}$$

will have a negative covariance term. This differs from OBC with respect to construction of a monotonic function where (X_i, X_j) are sampled from. Further but not discussed in this thesis, is the fact that OBC can be used to capture more abstract concepts such as those expressed linguistically.

A comparison to low-discrepancy sequences is also performed. These techniques attempt to yield a uniform sampling of a given space. There is no relationship to evaluation of solutions which exist in that space. Additionally, uniform sampling is not likely to yield the best opposite map, although heuristically using a low-sample version of this idea may be beneficial in some situations. The final comparison is with respect to dis/similarity functions, particularly in the context of statistical learning. These methods attempt to learn the target concept given a matrix of similarity between all pairs of features. Using a similar argument as for low-discrepancy sequences, the lack of consideration of evaluation distinguishes OBC.

1.2.2 Part 2 - Example Applications

Chapter 4: Monte Carlo Optimization Monte Carlo optimization refers to simulation-based approaches to determining the minimum or maximum of some unknown function. Due to the lack of information regarding the evaluation function analytical techniques tend to play an insignificant role, resulting in the loss of a guaranteed outcome. Hence, efficient simulation methods for improving accuracy and precision have become very important.

One popular Monte Carlo optimization method is simulated annealing. Using this technique as an example, the concept of an opposite neighbor is examined. In this case, elements are considered opposite if they lie on different sides of the current solution being examined. That is, if solution matrix X_1 is opposite of matrix X_2, given solution R if $X_1 = R \pm X_2$ for all row values. The diversity-control mechanism employed during algorithm convergence reduces the number of rows where $X_1 \neq X_2$ according to an exponentially decreasing function. Improvements in accuracy, reliability and convergence time will be shown, especially as the problem dimensionality increases.

Chapter 5: Estimation of Distribution Techniques which aim to determine the probability distribution for each parameter of a representation are termed estimation of distribution algorithms. These methods operate via continually resampling from and updating the respective distributions until some termination criteria have been met. Many of these approaches have seen excellent success in a variety of application areas.

This second application of opposition-based computing will focus on population-based incremental learning. This algorithm uses a reinforcement learning-like update of the estimated distributions and is primarily focused on solving problems of the form $\{0,1\}^n \rightarrow \mathbb{R}$. A major issue with this technique (and estimation of distribution algorithms in general) is maintaining diversity between samplings, resulting in an oft observed premature convergence.

Expanding on the concept of the opposite neighbor, the idea of an opposite sample is introduced. Its improvement with respect to random sampling and all-pairs diversity using the Hamming distance is proven. In concert with the opposite population, an alternative update rule is proposed as the diversity-convergence mechanism. Results will show improvements in diversity, accuracy, reliability and convergence time, especially as the problem dimensionality increases.

Chapter 6: Gradient-Based Learning The final application of opposition-based techniques considers gradient-based learning in feedforward neural networks. The most popular method to accomplish learning in these neural models is backpropagation. Unfortunately, this algorithm tends to be much slower than alternatives and often converges to local optima. Various strategies have been developed to improve the results, including incorporation of second-order information, regularization, adaptive transfer functions, and heuristic methods.

In this chapter, a heuristic method based on restricted adaptive transfer functions is proposed. Specifically, an opposite transfer function is introduced as a means to efficiently alter the input/output mapping represented by the network. Due to existing symmetries in the neural network structure and error surface it is proven that considering opposite networks (the set of all permutations of opposite transfer function state) under the assumption of minimality will be composed of unique networks. That is, each opposite network represents a unique input-output mapping. A further consequence is an effect on the Jacobian and Hessian matrices, which have a direct impact on learning accuracy, generalization and convergence rate. It is possible to escape local optima and areas of the error surface where learning progresses at a relatively slow rate by considering opposite networks.

Theoretically, the proposed framework can be used in association with any learning algorithm. The requirements mainly focus on deciding the heuristic for efficiently selecting a single network from the set of opposite networks. Function call overhead, as determined by the number of weight updates/epochs is a major concern since a gradient-based method is essentially guaranteed to lead to an improvement in network performance. Hence, the heuristic must be determined such that

$$Pr\left(\min(Er(x^t), Er(\breve{x}^t)) < \min(Er(x^t), Er(x^{t+1}))\right) > 0.5 \tag{1.6}$$

for error function $Er(\cdot)$ where t represents an epoch and $x \in \mathbb{R}^n$ is a matrix of n network parameters (i.e. weights and biases). That is, the $(x^t, \breve{x}^t)$ pair at a given iteration must outperform the (x^t, x^{t+1}) to be successful. Backpropagation variants and resilient propagation are shown to yield statistically significant improvements in accuracy and generalization ability. Moreover, these results magnify as the problem complexity increases.

1.2.3 Summary of Main Contributions

The main contributions of this thesis are summarized as follows:

- **Search space symmetry:** Describing symmetry of the search space, without regard to the evaluation method using group theory. The foundations for Opposition-Based computing are explained using existing concepts.

- **Formal outline of opposition:** Providing a mathematical description of the concept of opposition within the context of optimization. Opposition is described as a symmetry inducing transformation over the evaluation function which often results in a more desirable function.

- **Necessary conditions for opposition:** Provides conditions where OBC should outperform random sampling and the optimal choice of opposite map.
- **Opposition-based simulated annealing:** Propose the use of opposite neighbors and use them to improve accuracy, reliability and convergence rate of simulated annealing.
- **Oppositional population-based incremental learning:** Propose the use of opposite samples and use them to improve diversity, accuracy, reliability and convergence rate of population-based incremental learning.
- **Opposite transfer functions:** Propose the use of opposite transfer functions and use them to improve accuracy, reliability, convergence rate and generalization ability of gradient-based learning for feedforward neural networks.

Part I

Motivations and Theory

Chapter 2

Group Theory and Search Space Symmetry

Group theory is a well developed branch of mathematics which studies abstract structures. An enormous amount of this literature focuses on the concept of symmetry and it has broad applications in a variety of scientific fields [43, 81, 82, 96, 97, 148]. This chapter provides an introduction to the required concepts with respect to the thesis goals. The main texts and theoretical content on which this chapter is based are Rose [95], Zassenhaus [158], Weyl [148] and Rowe et al. [98, 99].

This chapter introduces the concept of symmetry induction from a group-theoretic standpoint. Conditions which must exist for the search space itself to be a group structure are shown. Additionally, the conditions when a search space is compatible with a group structure will be analyzed. Using the concept of a neighborhood structure, an algorithm is derived which partitions and reduces the search space size. These results are important to outline as they are fundamental to the proposed use of opposition-based computing in Chapter 3.

2.1 A Short Exploration of Group Theory

Although at first glance the concept of a group seems simplistic, group theory is actually one of the cornerstones of modern mathematics, and is the preferred language for describing symmetry. A group is composed of a set G and an associated binary operation $\cdot \colon G \times G \to G$ which satisfies the following specific axioms:

1. For all $a, b, \in G$, $a \cdot b \in G$ (closure).

2. For all $a, b, c \in G$, $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ (associativity).

3. There exists an $e \in G$ such that for all $a \in G$, $a \cdot e = a = e \cdot a$ (identity).

4. For each $a \in G$ there exists an element a^{-1} such that $a \cdot a^{-1} = a^{-1} \cdot a = e$ (inverse).

For readability, it is common that the $\cdot$ is omitted such that $a \cdot b$ is simply denoted ab. In some circumstances this can have a detrimental effect on readability, but for consistency this notation will be adopted throughout. The following properties hold for all groups:

1. The identity is unique.

2. For each $a \in G$, the inverse a^{-1} is unique.

3. For all $a, b \in G$, equations $ax = b$ and $ya = b$ have unique solutions, $x = a^{-1}b$ and $y = ba^{-1}$, respectively.

4. For all $a \in G$, $ax = ax' \implies x = x'$, and $ya = y'a \implies y = y'$.

5. For all $a, b \in G$, $(a^{-1})^{-1} = a$ and $(ab)^1 = b^1a^1$.

where x, y are elements being acted on.

Group theory is composed of specialized areas of study which depend on whether G is finite or infinite, discrete or continuous. Finite groups have a finite number of elements. The cardinality (or order) of such a group is commonly denoted $\#G$ or $|G|$. Infinite, but countable groups have been widely researched, for example the integers $\mathbb{Z}$ using arithmetic addition as the binary operation.

Uncountable groups have also been studied. In this case, G is associated with a topology[1]. The topology is defined such that the inverse and binary operation of G are continuous maps: $f(a) = a^{-1}$ for $f\colon G \to G$ and $g(a, b) = ab$ for $g\colon G \times G \to G$, respectively. The real numbers form a topological group using its usual topology[2] and addition as a binary operation. If a group also satisfies the axiom of commutativity (for all $a, b \in G$, $ab = ba$) it is termed Abelian and are also known as commutative groups. The integers $\mathbb{Z}$ under addition represent an example of an Abelian group.

2.1.1 Basic Concepts

Definition 1 (Conjugate) *Given group G, elements $a, b \in G$ are termed conjugate if there is some $g \in G$ such that $g^{-1}ag = b$.*

Being an equivalency relation, conjugacy partitions G into equivalence classes. That is, every element of G will belong to only one equivalence class. An element $a \in G$ belongs to a conjugacy class Cl determined by

[1]A topology on a set X is a system T of subsets of X such that $\emptyset \in T$ and $X \in T$, and T is closed with respect to arbitrary unions and finite intersections [74].

[2]The standard topology T on $\mathbb{R}$ is defined according to $U \subseteq \mathbb{R} \in T$ if for every point $x \in U$, there is some $\varepsilon > 0$ such that $(x - \varepsilon, x + \varepsilon) \subseteq U$.

$$Cl(a) = \{g^{-1}ag | g \in G\}. \tag{2.1}$$

Definition 2 (Isomorphic) *Two groups G and H are isomorphic, denoted $G \cong H$, if there exists a one-to-one mapping $\phi\colon G \to H$ such that $\phi(a) * \phi(b) = \phi(a \cdot b)$ for all $a, b \in G$. Where $*, \cdot$ are the respective binary operations of G and H.*

Due to the focus on structure versus group elements, isomorphic groups are considered and referred to as the same group.

Definition 3 (Homomorphism) *Given groups G and H a homomorphism $h\colon G \to H$ is such that for all $a, b \in G$, $h(ab) = h(a)h(b)$. The group operation on the left hand and right hand side are of group G and H, respectively.*

Homomorphisms typically are many-to-one functions which preserve the identity and inverse mapping. An isomorphism is a homomorphism that has an inverse (that is also a homomorphism).

Definition 4 (Automorphism) *An automorphism is a homomorphism from a group to itself, $h\colon G \to G$.*

Definition 5 (Subgroup) *A subset $H \subseteq G$ is a subgroup if it is a group under the operation of G, denoted as $H \leq G$. H must be closed under $\cdot$, contain the identity element e as well as the inverse $g^{-1} \in H$, $\forall g \in H$.*

Definition 6 (Group Action) *Let G be a group and X a nonempty set. If there exists a map $\phi\colon G \times X \to X$ such that*

1. *$\phi(e, x) = x$.*
2. *$\phi(g, \phi(h, x)) = \phi(gh, x)$ for all $g, h \in G$.*

then ϕ is called a group action.

Often the group action will be abbreviated as $G(X)$, implying the group G acts on set X. A group action will partition X into disjoint orbits.

Definition 7 (Orbit) *Let $g \in G$ and $x \in X$. The subset of X obtained by the action of all operations in G on x is called the orbit of x, i.e.*

$$O(x, G) = \{\phi(g, x) | g \in G\}.$$

If $x_1, x_2 \in X$ are in the same orbit then there exists $g \in G$ such that $\phi(g, x_1) = x_2$. That is, the orbit is the set of conjugates of an element $x \in X$.

Definition 8 (Compatible Group Structure) *Let G be a permutation group acting on a countable set X. Then, X has a group structure compatible with $G(X)$ if there exists a binary operation with identity element defined on X and a function $\rho\colon G \to X$ such that $\forall g \in G$ and $w \in X$,*

$$g(w) = \rho(g)w. \tag{2.2}$$

2.1.2 The Symmetry and Permutation Groups

A prime example of a group action (Definition 6) is when $X = \{1, \ldots, n\}$ is the set being acted on. In this situation we are considering all $n!$ permutations (a bijection of X onto itself). Under composition of mappings, the set of all permutations forms the symmetric group S_n of degree n. For $\sigma_1, \sigma_2 \in S_n$, $\sigma_2\sigma_1$ represents the operations of permuting X according to σ_1 followed by σ_2. Cayley's Theorem [158] states that every group G is isomorphic to a subgroup of S_n.

There exists different methods to represent elements of S_n. A two-line notation employs a $2 \times n$ matrix where the first row corresponds to the $X = \{1, \ldots, n\}$ elements and the second row is its image under the permutation operation, as seen in Example 2.1.

Example 2.1 (Two-line representation) *A permutation operation on $X = \{1, \ldots, n\}$ is according to $\alpha \mapsto ((3\alpha) \mod 5 + 1) \mod 5$. Then, its two-line representation is given as*

$$\sigma = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 \\ 1 & 4 & 2 & 0 & 3 \end{pmatrix},$$

where $\sigma \in S_5$. This represents the relationship $\sigma(0) = 1$, $\sigma(1) = 4$, $\sigma(2) = 2$, $\sigma(3) = 0$ and $\sigma(4) = 3$.

A more compact method of specifying permutations utilizes the fact that any permutation can be written as a combination of cycles (an ordered subset $s_1, \ldots, s_k$ of $X = \{1, \ldots, n\}$ such that $\sigma(s_i)=s_{i+1}$ for $i < k$ and $\sigma(s_k) = s_1$). Cycles of length one need not be explicitly noted as it is understood that the element is fixed. A list of the lengths of cycles making up σ is the cycle type.

Example 2.2 (Cycle representation) *Using the same permutation of Example 2.1. This permutation can be written as a product of disjoint cycles. The resulting product of cycles representation is*

$$\sigma = (0143)(2) = (0143)$$

as (2) *is understood to be fixed.*

Example 2.2 can also be expressed as an element of the factorized group $S_5 = S_4 \times S_1$.

2.2 Search Space Symmetry

Let G be a group which acts on search space $\mathcal{S}$. Then, there exists a mapping $G \times \mathcal{S} \to \mathcal{S}$ such that for all $x, y \in G$ and $w \in \mathcal{S}$,

$$(xy)(w) = x(y(w)), \text{ and } (x^{-1}x)(w) = (xx^{-1})(w) = w, \tag{2.3}$$

where the inverse of x is denoted x^{-1}. Each element of G represents a bijection of $\mathcal{S}$ and thus G is a permutation group. The group action on $\mathcal{S}$ will be denoted $G(\mathcal{S})$ and it is assumed to be transitive[3].

Definition 9 (Reduced) *A transitive group action $G(\mathcal{S})$ will be called reduced if $\{x \in G : \forall w \in \mathcal{S},\ x(w) = w\} = 0$, where 0 denotes the identity.*

That is, all group actions leave the identity unchanged. Unless otherwise noted it will be assumed that $G(\mathcal{S})$ is reduced. Example 2.3 highlights the idea of search space symmetry for fixed length binary strings of length $\ell = 6$ under bitwise addition modulo 2.

Example 2.3 (Search Space Symmetry) *Let $\mathcal{S}$ be the space of fixed length binary strings of length $\ell = 6$ and let $\cdot$ be the operation of bitwise addition modulo 2 (i.e. XOR). In this case $\mathcal{S}$ has a group structure and the group action on itself will be composition of group actions, $u(v) = uv$ for $u, v \in \mathcal{S}$.*

Let the identity be 000000 *and take $u = 101101$ and $v = 100010$, then*

$$u(v) = 101101(100010) = 101101 \oplus 100010 = 001111. \tag{2.4}$$

The following theorem shows that $G(\mathcal{S})$ and $0 \in \mathcal{S}$ determine the isomorphism ρ (Definition 8) and the group operation on $\mathcal{S}$. To show an isomorphism ρ must be shown to be one-to-one and onto as well as satisfy the main condition of Definition 2.

Theorem 2.2.1 *Let $G(\mathcal{S})$ be reduced and let $\mathcal{S}$ have a group structure compatible with it. Then, $G \cong \mathcal{S}$ by $\rho \colon G \to \mathcal{S}$ [99].*

Proof Assume that ρ is not onto. Then, there must exist some $w \in \mathcal{S}$ such that for all $x \in G$

$$w \neq \rho(x) = \rho(x)0 = x(0). \tag{2.5}$$

But this contradicts the assumption that G is transitive. Hence, ρ is onto.

Further, $\rho(x) = \rho(x')$ implies that for all $w \in \mathcal{S}$,

$$x(w) = \rho(x)w = \rho(x')w = x'(w). \tag{2.6}$$

[3]That is, for every $u, v \in \mathcal{S}$ there exists an $x \in G$ such that $x(u) = v$.

Thus $x = x'$ and ρ is also one-to-one. Furthermore,

$$\begin{aligned} \rho(xx') &= \rho(xx')0 \\ &= (xx')(0) \\ &= x(x'(0)) \\ &= \rho(x)x'(0) \\ &= \rho(x)(\rho(x')0) \\ &= \rho(x)\rho(x') \end{aligned} \tag{2.7}$$

Therefore, ρ is a group isomorphism.

■

The only possibility is for $\rho(x) = \rho(x)0 = x(0)$. Additionally, if $G(\mathcal{S})$ is reduced then the group operation on $\mathcal{S}$ is completely determined by the compatible group structure (as stated in Definition 8) and as is highlighted in the following example.

Example 2.4 (Determining ρ) *Consider the following table representing a hypothetical search space, where the five rows of numbers each correspond to an ordering of $\mathcal{S} = \{A, B, C, D\}$:*

A	B	C	D
1	2	3	4
3	2	1	4
2	4	1	3
3	1	2	4
4	1	3	2

Such a situation is common in optimization. For example, the Traveling Salesman Problem where each row represents the visitation order of the salesman to cities A through D. Row 2 corresponds to the salesman traveling from $C \to B \to A \to D$, and implicitly $D \to C$.

The group G which acts on $\mathcal{S}$ can also be written in a similar style:

A	B	C	D
A	B	C	D
C	B	A	D
B	D	A	C
C	A	B	D
D	A	C	B

Examining G results in defining $\rho\colon G \to \mathcal{S}$ according to $A \mapsto 1, B \mapsto 2, C \mapsto 3, D \mapsto 4$. This gives $\rho(g) = g(0)$, for the identity $1 \to A, 2 \to B, 3 \to C, 4 \to D$. Then, the group operation on $\mathcal{S}$ will be $ab = \rho^{-1}(a)(b)$, which is isomorphic to G.

2.2.1 Conditions for Compatibility of $\mathcal{S}$ and G

This theorem describes when the search space can be given a group structure compatible with G, as described in the previous section. Identifying whether this can be done without having to explicitly examine $\mathcal{S}$ and G in a manner similar to Example 2.4 can be very time saving. It is necessary to first define a set R of permutations of $\mathcal{S}$ such that for all $x \in G$,

$$g \in R \Longleftrightarrow gx = xg. \tag{2.8}$$

So, R contains only elements which commute with G. In this case it is not necessary for a permutation $\sigma \in \mathcal{S}$ to also be an element of G.

In total three conditions for compatibility of $\mathcal{S}$ and G have been identified. All focus on the ability of ρ to leave the identity unchanged or to allow for navigation in the search space. The first states that there exists at least one element of the search space where different group actions will produce different results. That is, there is some neighborhood which exists in the search space (neighborhoods will be discussed in the next section). The second equivalence states that only the identity leaves elements unchanged. And, the final statement dictates that $R(\mathcal{S})$ be reduced. Essentially, the only permutation of $\mathcal{S}$ which is commutative with G is the identity.

Theorem 2.2.2 *If $G(\mathcal{S})$ is reduced then the following statements are equivalent [99]:*

1. *$\mathcal{S}$ has a group structure compatible with $G(\mathcal{S})$.*
2. *There exists an identity element $0 \in \mathcal{S}$ such that for all $x, x' \in G$, $x(0) = x'(0)$ implies that $x = x'$.*
3. *For all $w \in \mathcal{S}$ the set $\{x \in G(\mathcal{S})\colon x(w) = w\}$ contains only the identity 0.*
4. *$R(\mathcal{S})$ is reduced.*

Proof ($1 \Longleftrightarrow 2$): ρ will be a one-to-one function if $\mathcal{S}$ has a group structure that is compatible with $G(\mathcal{S})$. Then, $x(0) = x'(0)$ implies

$$\rho(x) = \rho(x') \Longrightarrow x = x' \tag{2.9}$$

where $x \in G$. Conversely, let the identity be defined as $0 \in \mathcal{S}$ such that for all $x, x' \in G$ results in $x(0) = x'(0) \Longrightarrow x = x'$.

Define $\rho\colon G \to \mathcal{S}$ as $\phi(x) = x(0)$. Since G acts transitively, ρ will be onto. The definition of the binary operation follows from Definition 8 as, $\rho(x)w = x(w)$. Thus, the binary operation is well-defined since

$$\rho(x) = \rho(x') \Longrightarrow x(0) = x'(0) \Longrightarrow x = x'. \tag{2.10}$$

($2 \Longleftrightarrow 3$): From the problem statement,

$$xh(0) = x'h(0) \Longrightarrow xh = x'h \Longrightarrow x = x', \tag{2.11}$$

where $h(0)$ is an arbitrary element of $\mathcal{S}$. Therefore, for all $w \in \mathcal{S}$,

$$x(w) = x'(w) \Longrightarrow x = x' \tag{2.12}$$

which is equivalent to

$$x^{-1}x'(w) = w \Longrightarrow x^{-1}x' = 0 \tag{2.13}$$

which is in turn equivalent to

$$x^{-1}x' \neq 0 \Longrightarrow x^{-1}x'(w) \neq w \tag{2.14}$$

($4 \Longrightarrow 3$): Since R is a permutation group on $\mathcal{S}$ it will be reduced iff it is transitive. So, suppose that R is transitive and that $x(w) = w$. Then, for all $g \in R$,

$$x(g(w)) = g(x(w)) = g(w). \tag{2.15}$$

A consequence of the transitivity of R, is that $g(w)$ can be any arbitrary element of $\mathcal{S}$ and so x must be the identity.
($1 \Longrightarrow 4$): The hypothesis and Theorem 2.2.1 demand that $\mathcal{S}$ be a group. Given arbitrary $b \in \mathcal{S}$, define the permutation g_b as

$$g_b(z) = zb \tag{2.16}$$

where $g_b \in R$ since $\forall x \in G$,

$$\begin{aligned} g_b(x(w)) &= g_b(\rho(x)w) \\ &= \rho(x)wb \\ &= x(wb) \\ &= x(g_b(w)) \end{aligned} . \tag{2.17}$$

Given any $u, v \in \mathcal{S}$ and let $b = u^{-1}v$. Then, $g_b(u) = uu^{-1}v = v$. Therefore, $R(\mathcal{S})$ must be transitive and therefore is reduced.

■

If all conditions hold, then a corollary to this exists. Particularly, $R(\mathcal{S})$ can be chosen such that the group operation is the reverse of the operation induced on G.

Corollary 1 *Assume all conditions of Theorem 2.2.2 hold. Then, the group operation of $R(\mathcal{S})$ can be selected such that it is the reverse of the group operation of G. Moreover, G is antiisomorphic to R [99].*

Proof If $G(\mathcal{S})$ is reduced and conditions 1 through 4 hold then condition 2 will also hold for R instead of G, since for all $k \in R$,

$$g(0) = g'(0) \Rightarrow kg(0) = kg'(0) \Rightarrow g(k(0)) = g'(k(0)) \Rightarrow g = g' \tag{2.18}$$

Therefore, the search space $\mathcal{S}$ has a group operation $\cdot'$ and isomorphism $\rho' \colon R \to \mathcal{S}$ induced on it such that for all $k \in R$ and $w \in \mathcal{S}$,

$$k(w) = \rho'(k) \cdot' w. \tag{2.19}$$

The argument given in ($1 \Longrightarrow 4$) shows that for every $w \in \mathcal{S}$, the permutation g_w defined by $g_w(z) = zw$ will be an element of R. If the identity elements of $(\mathcal{S}, \cdot)$ and $(\mathcal{S}, \cdot')$ are the same then $\rho'(g_w) = g_w(0) = 0w = w$. However, since ($1 \Longleftrightarrow 2$) and ($2 \Longleftrightarrow 3$) show that the choice of identity is arbitrary, this can be easily accomplished. Then, it follows that

$$\begin{aligned} w \cdot' z &= \rho'(\rho'^{-1}(w)) \cdot' z \\ &= (\rho'^{-1}(w))(z) \\ &= (\rho'^{-1}(\rho'(g_w)))(z) \\ &= (g_w)(z) \\ &= zw \end{aligned} \tag{2.20}$$

Finally, the map $x \mapsto g_x$ represents an antimorphism,

$$g_{xx'}(z) = zxx' = (g_x(z))x' = g_{x'}(g_x(z)) = g_{x'}g_x(z) \tag{2.21}$$

■

Thus, conditions for the existence of search space symmetries have been given. The next section discusses a possible manner in which symmetries may arise or be imposed on a search space.

2.2.2 Neighborhood Structures

The previous sections formalized the concept of search space symmetry within the mathematics of group theory. Conditions on when the search space is itself a group were also given. However, this does not describe a possible source for the symmetries. This section will use the concept of a neighborhood structure [100] to describe this phenomenon.

Definition 10 (Neighborhood Function) *A neighborhood function is a mapping $\eta\colon \mathcal{S} \to \mathcal{S}$ which assigns a neighbor $\eta(w)$ to each $w \in \mathcal{S}$.*

It is possible for $w \in \mathcal{S}$ to have more than one neighbor. If there exists a set $\mathcal{O}$ of neighborhood functions which generate a set,

$$N(w) = \{\eta(w)\colon \eta \in \mathcal{O}\} \subset \mathcal{S} \tag{2.22}$$

then every $w \in \mathcal{S}$ has a set of neighbors. This structure captures symmetries on the search space. Moreover, this structure represents an automorphism since if $u, w \in \mathcal{S}$ are related by a neighborhood function then any permutation of these elements will form a symmetry group. Hence, for $N_{i=1\ldots m}$ neighborhoods, $N_i \cap N_j = \emptyset$ and

$$\mathcal{S} = \bigcup_{i=1}^{m} N_i. \tag{2.23}$$

That is, each neighborhood represents a cycle of a permutation of $\mathcal{S}$. As a corollary, for finite $\mathcal{S}$,

$$|\mathcal{S}| = \sum_{i=1}^{m} |N_i|. \tag{2.24}$$

The group acting on $\mathcal{S}_n$ is factorized as $S_{n_1} \times \cdots \times S_{n_m}$, where

$$\sum_{i=1}^{m} n_i = n. \tag{2.25}$$

Example 2.5 highlights this concept where $\mathcal{S}$ is composed of three symmetry groups.

Example 2.5 *If the problem representation is of size $n = 8$ and $n_1 = 3, n_2 = 3, n_3 = 2$ and $m = 3$ then a possible solution could be $x = (210|101|43)$. The group acting on x is $S_3 \times S_3 \times S_2$. The orbit to which x is a member of contains all solutions which are composed of each partition having elements $2, 1, 0$ and $1, 0, 1$ and $4, 3$, respectively.*

Therefore, if a search space has existing neighborhood structures it will contain symmetry. Importantly, there is no restriction on whether the symmetries pre-exist or whether the search space is transformed such that they will exist. In either case, the following section describes a naive search algorithm which accounts for symmetries in the search space.

2.3 Implied Symmetry Reduction Algorithm

Assume there exists a search space $\mathcal{S}$ with imposed group structure, as defined above, with a neighborhood structure represented as an element of the permutation group. Further, assume $\mathcal{S}$ has m cycles (where each is an equivalence relation) defined on it. Since each element of the cycle shares the same neighborhood structure and the group action generating the symmetry is known, only a single element need be stored. This compression yields a reduction in storage space requirements; it also implies a reduction in the state space. This compressed space $\mathcal{S}'$ will form subgroup, $\mathcal{S}' \leq \mathcal{S}$.

The framework implied by the above is given in Algorithm 1, under the assumption that symmetry exists in $\mathcal{S}$. In line 1 the transformation which defines search space symmetry is searched for, and assigned to F. This mapping represents the partitioning of $\mathcal{S}$, which is some element of the permutation group. In lines 3-7, m iterations of the sampling procedure are conducted. This process iterates through all cycles to search for the desired element.

Algorithm 1 Implied sampling framework.

```
1: $F$ = Detect symmetry of $\mathcal{S}$
2: $B^*$ = current best solution.
3: for all cycle $c_i \in F$ do
4:     if $c_i$ is more desirable than $B^*$ then
5:         Let $B^* = c_i$.
6:     end if
7: end for
```

Traditionally, the symmetric nature of the space is unknown a priori and detection methods are required in order to exploit it [28, 29, 79]. However, this thesis proposes the a priori induction of symmetry over $\mathcal{S}$ with respect to the evaluation function, therefore foregoing the often computationally expensive detection process. The next chapter discusses the concept of Opposition-Based Computing and its relationship to symmetry induction with respect to the context of computational intelligence.

Chapter 3

Opposition-Based Computing

The previous chapter described the existence of symmetry in the search space and provided conditions when $\mathcal{S}$ is itself a group. Using neighborhood structures it was shown that the search space can be partitioned and a simple iterative algorithm for searching the reduced search space was presented. This chapter will focus on the neighborhood function and its implications within the context of optimization.

The goal of any optimization algorithm is to discover the minimum or maximum of some function $f\colon \mathcal{S} \to \mathbb{R}$ acting on search space $\mathcal{S}$. Without loss of generality, the following will assume a minimization problem. That is,

$$x^* = \arg\min_{x \in \mathcal{S}} f(x). \tag{3.1}$$

The previous chapter described an algorithm for partitioning and searching $\mathcal{S}$. Extending that situation to arbitrary neighborhood structures c_i leads to the following formulation:

$$\begin{aligned} x^* &= \arg\min_{x \in \mathcal{S}} f(x) \\ &= \arg\min(\arg\min_{x \in c_1} f(x), \ldots, \arg\min_{x \in c_r} f(x)), \end{aligned} \tag{3.2}$$

where each c_i represent a cycle or neighborhood structure. Although, there exists an associated computational overhead to compute the $\arg\min$ function. For relatively large search spaces, such as those characteristic of $\mathcal{NP}$-hard problems, possible benefits will be shown to arise.

Theoretically, this concept can be reapplied until only the globally optimal element(s) remain. That is, $\mathcal{S}$ can be transformed as above and compressed by retaining only a single element from each c_i. This more compact space can be continuously compressed in a similar fashion until only the optima are present. Unfortunately, the computational cost associated with such an operation is very large as it is equivalent to an exhaustive search.

3.1 Opposition

The search space for $\mathcal{NP}$-hard or sufficiently large problems becomes infeasible to efficiently search. Consequently, countless concrete and stochastic optimization algorithms have been proposed (see [109] for a good review). As the problem size increases and concrete solutions become infeasible the need increases for efficient, accurate and reliable stochastic optimization methods.

Algorithm 1 of Chapter 2 can be easily restated to accommodate the formulation of Equation (3.2), as is shown in Algorithm 2. However, there remains an open question regarding the choice of each c_i. In general, this decision could involve analytical techniques, prior knowledge, intuition or online learning. However, $\mathcal{NP}$-hard problems render analytical techniques infeasible, and no assumptions are made about prior knowledge. Of the remaining options, this thesis explores an intuitive approach and leaves online learning for future work.

Algorithm 2 Implied search procedure.

```
1: Determine $c_i$.
2: $B^*$ = current best solution.
3: for all cycles $c_i$ do
4:     if $\arg\min_{x \in c_i} f(x)$ is more desirable than $B^*$ then
5:         Let $B^* = \arg\min_{x \in c_i} f(x)$.
6:     end if
7: end for
```

3.1.1 The Opposition Map

According to the American Heritage Dictionary *opposite* is defined as, "being the other of two complementary or mutually exclusive things" and *oppositional* as "placement opposite to or in contrast with another" [1]. The latter implies simultaneous consideration of the opposite entities. Opposition-Based Computing (OBC) uses this as motivation to determine a partition/neighborhood structure of $\mathcal{S}$ by imposing a symmetry on $f(\mathcal{S})$ [125, 127].

Definition 11 (Opposition Map) *Given search space $\mathcal{S}$. Any bijective function $\Phi\colon \mathcal{S} \to \mathcal{S}$ can serve as an opposition map if for $x_1, x_2 \in \mathcal{S}$ then $\Phi(x_1) = x_2 \implies \Phi(x_2) = x_1$. The elements $x_1 \neq x_2$ are considered opposites.*

The accepted notation for opposites is $\breve{x}$, which represents $\breve{x} = \Phi(x)$. It has been shown in [88] that a possible choice for the opposition map is

$$\Phi(x) = a + b - x, \tag{3.3}$$

where $x_1 \in [a,b]$ and $a < b \in \mathbb{R}$. Assuming there exists an optimal solution $x^* \in [a,b]$, a sampling strategy according to $(x_1, \breve{x}_1)$ was shown to yield a higher probability of being closer to x^* than selecting a random (x_1, x_2), where $x_2 \in [a,b]$. Specifically it was proven that when samples are drawn according to a uniform distribution that,

$$Pr\left(\min(|x_1 - x^*|, |\breve{x}_1 - x_*|) < \min(|x_1 - x^*|, |x_2 - x_*|)\right) = 0.5833, \tag{3.4}$$

where $|\cdot|$ denoted the Euclidean distance measure. However, Equation (3.3) is not the only mapping where the probability of Equation (3.4) > 0.5. Another possibility could be,

$$\Phi(x) = \left(x + \frac{b-a}{2}\right) \mod (b-a). \tag{3.5}$$

3.1.2 Examples

Sampling at random has been long acknowledged as a poor strategy in many situations [65]. This is because of the susceptibility to sampling error due to randomness of the selection process and the poor reflection on the actual population. A minimization of this effect can be achieved by increasing the sample size, however this may not always be computationally or practically feasible. Thus, more powerful techniques have been proposed [65].

Within the context of optimization in computational intelligence purely random sampling remains the standard method[1] as usually either computational, analytical or a priori knowledge is required for more advanced sampling strategies. That is, computational intelligence algorithms often aim to minimize the user-required information in favor of flexibility in the framework. Many methods aim to efficiently minimize the effects of random sampling. For example, Tabu Search [37] proposes to store recent solutions in a list. If an element of the list is encountered during the search it is automatically rejected. In this way it is hoped that attractive basins on the evaluation surface are not re-searched by the algorithm.

Table 3.1 compares a uniform random sampling strategy to those implied in Equations (3.3) and (3.5), denoted as Φ_a and Φ_b, respectively. The simulation compares the strategies over 2, 5 and 10 dimensional problems containing 1, 2 and 5 optimal solutions which are distributed i.i.d. over the interval $[0,1]$, per dimension. Results are presented as a triple $< A, B, C >$ where

- **A**: $Pr\left(\min(x_1, x_2) = \min(x_1, \breve{x}_1)\right)$,
- **B**: $Pr\left(\min(x_1, x_2) > \min(x_1, \breve{x}_1)\right)$,

[1]As a consequence examples in this section focus on a comparison to random sampling.

- **B**: $Pr\left(\min(x_1, x_2) < \min(x_1, \breve{x}_1)\right)$.

Each experiment was run for $1,000$ trials, each containing $10,000$ samples. The values presented represent the averaged results. Utilizing the paired sampling strategies yield a higher probability of being closer to an optimal solution, as seen in Table 3.1. Although, this benefit decreases with the number of optimal solutions. For this example Φ_b seems to be a more desirable mapping, but both methods show improvements over basic random sampling.

Table 3.1: Comparing Φ_a and Φ_b paired sampling strategies against a random sampler. In both situations the random sampler is outperformed.

	Number of Optimal Solutions		
Method	1	2	5
	2 Dimensions		
Φ_a	$< 0.358, 0.358, 0.283 >$	$< 0.344, 0.343, 0.313 >$	$< 0.338, 0.338, 0.321 >$
Φ_b	$< 0.364, 0.364, 0.273 >$	$< 0.355, 0.355, 0.291 >$	$< 0.344, 0.344, 0.312 >$
	5 Dimensions		
Φ_a	$< 0.361, 0.361, 0.278 >$	$< 0.346, 0.346, 0.309 >$	$< 0.334, 0.334, 0.332 >$
Φ_b	$< 0.359, 0.359, 0.282 >$	$< 0.353, 0.354, 0.293 >$	$< 0.347, 0.347, 0.307 >$
	10 Dimensions		
Φ_a	$< 0.361, 0.362, 0.277 >$	$< 0.345, 0.345, 0.310 >$	$< 0.332, 0.332, 0.336 >$
Φ_b	$< 0.344, 0.360, 0.296 >$	$< 0.343, 0.358, 0.299 >$	$< 0.340, 0.356, 0.304 >$

Figure 3.1 extends the simulation of Table 3.1 to two continuous functions (termed Bowl and Humps, respectively) where it is assumed the optimal is unknown. In this situation measuring the distance to optimality is impossible and only a comparison of the evaluation is made. The opposition mappings Φ_a and Φ_b are applied for each row, hence the opposite of the plot is determined by a transformation in the x-axis. The statistics in Table 3.2 show the effect of the respective opposite maps. Using the results from the 10,000,000 samples it becomes very clear that an improvement in mean and standard deviation is observed, assuming a maximization problem.

Table 3.2: The effect of Φ_a and Φ_b on the Bowl and Humps functions. All values were determined over 10,000,000 sampling iterations.

Function	Mean	Std. Dev.	Min	Max
$random(Bowl)$	0.126	0.708	-1.000	1.000
$\Phi_a(Bowl)$	0.173	0.699	-0.999	1.000
$\Phi_b(Bowl)$	0.519	0.545	-0.999	1.000
$random(Humps)$	0.356	1.891	-6.539	8.100
$\Phi_a(Humps)$	0.852	1.728	-5.856	8.100
$\Phi_b(Humps)$	1.436	1.758	-0.194	8.100

To further examine whether the transformed spaces could be more effective than paired random sampling on the original space a comparison between the probabilities of success are determined, similar to

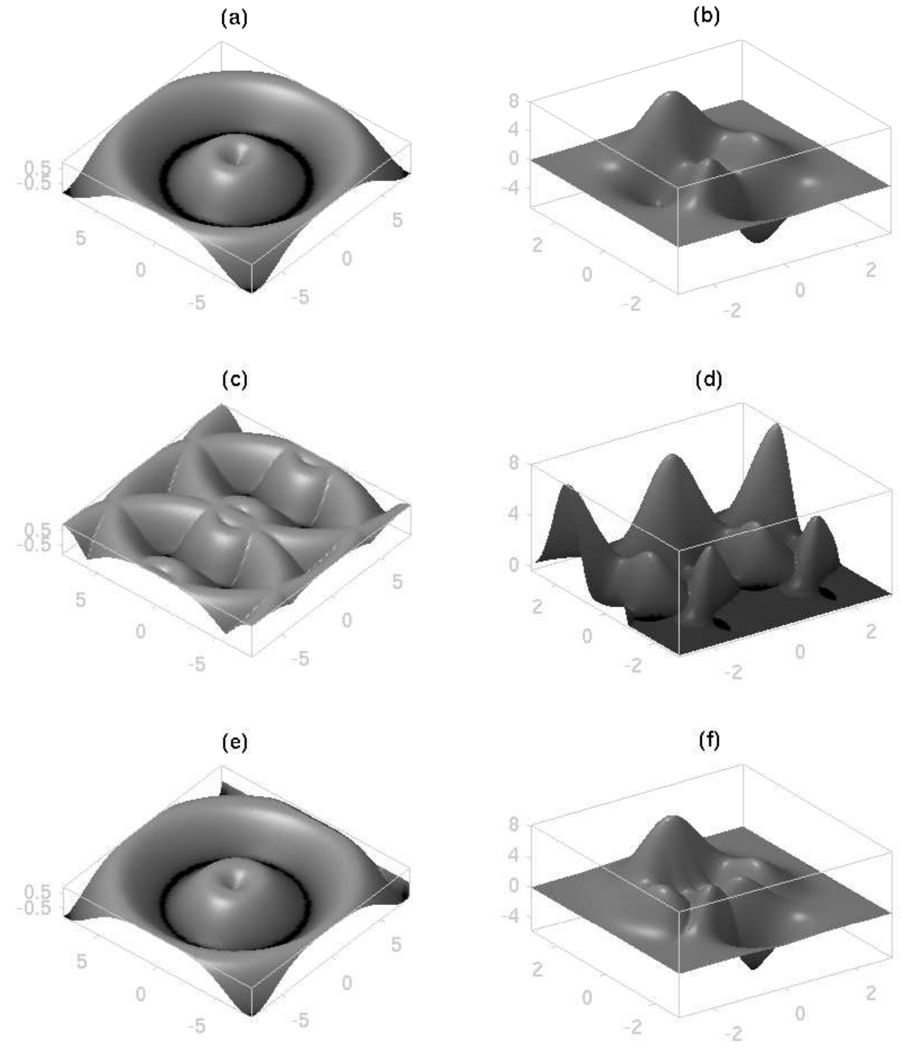

Figure 3.1: The effect of Φ_a and Φ_b on two hypothetical evaluation functions. Figure (a) and (b) are the original functions, (c) and (d) are transformed using Φ_b and (e) and (f) use Φ_a, respectively. The evaluation functions are represented by $m \times n$ matrices and both opposite maps act on a given row (each $i = 1, ..., m$ row has the opposite map applied to it).

those in Table 3.1. In this instance it is found that for the Bowl and Humps functions the probability of paired random sampling outperforming the opposition-based method is less than 0.005 for either problem. The probability of one of the oppositionally mapped evaluation functions yielding a more desirable value is greater than 0.76 in both cases. This is strong evidence supporting the use of oppositional concepts. However, the next section will provide a more rigorous explanation of the conditions which should be met in order for such a transformation to yield success.

Table 3.3: Comparing the probabilities of different sampling strategies on the Bowl and Humps functions. Using traditional random sampling is significantly worse for this situation where a maximization problem is under consideration.

Function	Bowl	Humps
Φ_a	$< 0.168, 0.000, 0.831 >$	$< 0.155, 0.000, 0.844 >$
Φ_b	$< 0.235, 0.003, 0.761 >$	$< 0.187, 0.004, 0.807 >$

3.1.3 Conditions for a Good Opposition Map

The above examples highlight the possible benefit of opposition-based ideas compared to random sampling. In this section general conditions where one should expect an improvement in performance (with respect to evaluation) when comparing paired i.i.d. random sampling versus the dependant sampling of opposition. Without loss of generality, a minimization problem is assumed. Typically, an improvement can be expected if the opposition map Φ is designed such that for n samples,

$$Pr\left(\min(d(x_1, x^*), \ldots, d(x_{n/2}, x^*), d(\breve{x}_1, x^*), \ldots, d(\breve{x}_{n/2}, x^*)) < \min(d(x_1, x^*), \ldots, d(x_n, x^*))\right) > 0.5 \quad (3.6)$$

for distance metric $d(\cdot, \cdot)$ and all optimal solutions $x^* \in \mathcal{S}$ [88, 137]. However, in the context of optimization the distance metric is inapplicable as the optimal solutions are often unknown. Disregarding the distance to an optimal solution in lieu of only the evaluation leads to,

$$Pr\left(\min(f(x_1), \ldots, f(x_{n/2}), f(\breve{x}_1), \ldots, f(\breve{x}_{n/2})) < \min(f(x_1), \ldots, f(x_n))\right) > 0.5, \quad (3.7)$$

where $f(\cdot)$ is the evaluation function. Assuming Equation 3.7 is satisfied, it implies that

$$\mathbb{E}[\min(f(x_1), \ldots, f(x_{n/2}), f(\breve{x}_1), \ldots, f(\breve{x}_{n/2}))] < \mathbb{E}[\min(f(x_1), \ldots, f(x_n))]. \quad (3.8)$$

Hence, the most desirable choice of the opposition map Φ^* will maximize this difference,

$$\Phi^* = \arg\max_{\Phi} \mathbb{E}[\min(f(x_1), \ldots, f(x_{n/2}), f(\breve{x}_1), \ldots, f(\breve{x}_{n/2}))] - \mathbb{E}[\min(f(x_1), \ldots, f(x_n))], \quad (3.9)$$

where $\varPhi$ is implied in the breve notation. That is, the opposition mapping which has the greatest impact on the expected outcome. Since there exists infinitely possible opposite maps solving Equation (3.9) is impossible without sufficient prior information. In practice, an intuition-based function has been found to yield benefits (see Section 3.4).

3.2 Computational Issues

In this section, computational issues related to the aforementioned motivations and theory are presented and a general algorithmic framework is described.

3.2.1 Partition Generation and Evaluation

There remain some issues related to computational overhead concerning the opposition map. One particular question arises as a result of the assumed search space size. It is enormous and therefore becomes impractical to physically transform entirely. Indeed, uncountable search spaces would be impossible to transform unless the exact form of the space in known. Instead, the transformation will be applied as needed during the search process. That is, computing the $\arg\min$ value for a given partition will be accomplished when any element of that partition is examined.

It follows that a major concern of opposition map design is to ensure that partition sizes do not cause excessive overhead, rendering the method too slow. Of course, as the number of elements per partition increases there will likely be a diminishing return of Equation 3.8, which is also an important issue. In the context of learning (versus sampling) this becomes an especially important issue because for a given learning iteration $t > 0$, it should be that

$$Pr\left(\min(f(x^t), f(\breve{x}^t)) < \min(f(x^t), f(x^{t+1}))\right) > 0.5. \tag{3.10}$$

At any given t the opposition-based technique should be more likely to have a lower evaluation. For the terminal iteration this should definitely hold, otherwise the opposition-based technique will not be useful.

Another, more subtle computational cost occurs during the generation of opposites. Specifically, the execution of $\varPhi$ may involve algorithms to select a single element from the partition or generating opposite elements may be costly. This is an important consideration since it directly affects the running time of the algorithm. As a design principle, it is suggested to keep the sample and opposite generation methods as close as possible with respect to computational cost. If such a situation arises that additional computational cost is required, the final outcome must warrant the cost. That is, the ends must justify the means.

The choice of opposition map is crucial to the amount of success one can expect for opposition-based techniques. This thesis investigates an intuitive/heuristic approach, however it is also possible to incorporate a strategy which has the ability to mine the search space in order to determine the entities with maximum evaluation difference. This opposition mining can be accomplished either a priori or online. In either case, efficiency will be a major issue to be addressed.

Definition 12 (Opposition Mining) *Opposition mining refers to the offline or online discovery of $\varPhi^*$. The approximate opposition map is represented as $\varUpsilon$.*

3.2.2 Uses of Opposition

One may also desire to relate concepts which are known not to be opposite, but perhaps there is only a small logical difference which leads to a nearly opposite relationship. The concept of a degree of opposition can be used to measure this situation. Example 3.1 highlights one of its possible uses in comparing linguistic terminology.

Definition 13 (Degree of Opposition) *Let $\varPhi$ be an opposition map and $x_1, x_2 \in \mathcal{S}$ be selected such that $x_1 \neq \varPhi(x_2)$. Then, the relationship between x_1 and x_2 is not opposite, but can be described as partial opposition, determined by a function $\tau\colon f(x_1), f(x_2) \to [0,1]$, where as $\tau \to 1$, x_1, x_2 become closer to being opposite given evaluation function $f\colon \mathcal{S} \to \mathbb{R}$.*

Example 3.1 (Ordering according to τ) *Consider the following statements:*

$$\begin{aligned} s_1 &= \text{“Everybody likes apples”} \\ s_2 &= \text{“Nobody likes apples”} \\ s_3 &= \text{“Some people like apples”} \\ s_4 &= \text{“Some people do not like apples”} \end{aligned}$$

and let the frame of reference be s_1. Then, the logical differences to statements $s_2 - s_4$ can be ascertained and even used to order the four statements with respect to s_1. The resultant ordering will be $\tau(s_1, s_4) < \tau(s_1, s_3) < \tau(s_1, s_2)$. In this case $\tau(s_1, s_2) = 1$.

The application of opposition-based concepts to optimization implies a zero-sum situation having a single winner (the best solution found in a partition) and many losers (the other elements). This is a highly competitive situation. However the possibility exists for more cooperative uses of opposites. In these situations the grouped elements will combine according to some user-defined rule. An example which will be discussed below is antithetic sampling where the goal is to lower the variance of mean estimation. The degree of competition quantifies this competitiveness of elements within a partition.

Definition 14 (Degree of Competition) *Let $x_1, x_2 \in \mathcal{S}$ be arbitrarily selected such that $x_1 = \varPhi(x_2)$. With inherent respect to the algorithm employing opposition, $\varSigma$, the degree at which x_1, x_2 compete for* *use in Σ is given by the function $\zeta\colon x_1, x_2|\Sigma \to [0,1]$. Values of ζ closer to 1 indicate a more competitive situation.*

The following simple examples describe competitive and cooperative opposition. More complicated forms may arise when many opposites are considered and some compete, while others cooperate. Alternatively, one of the competing solutions could be more heavily weighted than the others.

Example 3.2 (Ordering according to ζ) *Using Example 3.1, a competitive use for opposition would be searching for the statement which conveys the minimum number of people that like apples. If the opposites were defined arbitrarily according to $s_1 = \Phi(s_3)$ and $s_2 = \Phi(s_4)$ then s_3 and s_2 are mordesirable. This situation has $\zeta = 1$.*

Conversely, let the goal be to approximate "about half of people like apples". Then, each pairing will aim to return a reasonable solution. Perhaps the evaluation function will be defined as:

$$\begin{aligned} (s_1, s_3) &= \text{"At least one person likes apples, and one person dislikes them"} \\ (s_2, s_4) &= \text{"A group of people like apples, and a group do not"} \end{aligned}$$

The arguments have been revised to try and satisfy the original statements, resulting in different approximations to the target statement. Here $\zeta = 0$, as both of the competing opposites contribute an equal amount of their knowledge to the new evaluation.

3.2.3 Convergence

Thus far opposition-based ideas have focused on a random sampling-based assumption. However, in practical and efficient optimization methods, such as those in computational intelligence, this sampling is rarely unbiased. Specifically, each algorithm will introduce bias towards different optima in its own way, thus providing a specific solution to the diversity-convergence (exploration/exploitation) problem.

The basic framework which most computational intelligence algorithms can be described as is given in Algorithm 3. Line 1 refers to any preprocessing of the data, parameters, search space, etc that may be required. For instance, neural network learning may perform standardization of the data and principal component analysis in addition to initial weight generation. The main loop is iterated through $t = 1...n$ times, and the three steps within it may occur in any permutation. In line 3, at least one element from $\mathcal{S}$ is selected according to some selection mechanism, including random sampling. A genetic algorithm will perform the operations of crossover and mutation during this phase. The final two stages of the main loop (lines 4 and 5) represent some selection mechanism where the best solution(s) found are retained and any algorithm parameters are accordingly adjusted. Bayesian Optimization Algorithms [80] will update beliefs at this stage. The final step performs any postprocessing such as evaluation with respect to test data or transformation to the original space.

Algorithm 3 Basic framework of most computational intelligence algorithms.

```
1: Preprocessing.
2: for all $t = 1...n$ do
3:     Sample at least one element from the search space.
4:     Comparison of element(s) with best solution(s).
5:     Update algorithm parameters.
6: end for
7: Postprocessing.
```

Sample selection and algorithm parameters usually have a great influence on each other and guide the search towards optima. This bias will impact the influence of opposite elements. Assume that algorithm $\mathcal{A}$ considers a subset or window, $\mathcal{W} \subset \mathcal{S}$ at each iteration. Bias will reduce the size of $\mathcal{W}$ as $t \rightarrow \infty$. Assuming that $\mathcal{W}$ contains a reasonable solution it would be most desirable to ensure most of the sampled elements lie within its bounds. For diversity maintenance a small proportion of samples could exist outside $\mathcal{W}$, however the chances of yielding a desirable outcome are likely minimal.

Let $\mathcal{R}^d \subset \mathbb{R}^d$ be a d-dimensional problem representation and $r_1 \in \mathcal{R}^d$ be arbitrarily selected such that $r_1 \in \mathcal{W}$. The opposite will initially be computed with reference to the window $\mathcal{W} = \mathcal{S}$, i.e. the entire search space. However, as $\mathcal{A}$ decreases the size of the window, the likelihood of $\breve{r}_1 \notin \mathcal{W}$ increases. Hence, there exists three possibilities:

1. Consider opposites only during early stages of the search, when $\mathcal{W}$ is relatively large.

2. Allow the definition of Φ to vary with respect to the size of $\mathcal{W}$.

3. Allow Φ to vary with respect to the iteration t.

The first case only requires defining some constant $t' > 0$ such that for $t > t'$ opposites are no longer considered. Alternatively, redefining Φ is not so straightforward. Three possible directions can be distinguished:

1. Determine the bounds $[a_i, b_i]$ for $\mathcal{W}$ at each dimension $i = 1...d$. Then, generate opposite solutions with respect to these bounds.

2. Select $0 < c \leq d$ dimensions and consider the full range of $\mathcal{S}$ for determining the opposite of each c. That is, there will exist $d - c$ values where $r_1 = \breve{r}_1$.

3. A combination of the above two methods which will consider a subset of dimensions and generate the opposite solution with respect to the bounds of $\mathcal{W}$ or $\mathcal{S}$.

The strategy which will yield the best results can vary with each optimization algorithm and problem instance being addressed. At this point user experimentation and intuition is the swiftest course of action.

Essentially, opposition provides a means for jumping to more desirable solutions, however, the manner in which this feedback is employed is based on the parent algorithm. Therefore, the convergence rate or behaviour is very unlikely to be significantly impacted. Due to this, proof of convergence for each of the algorithms has not been explored. Although, the number of iterations required to achieve a target evaluation will be shown to be improved.

3.2.4 Summary of the Implied Framework

The computational issues of partition evaluation, use of opposition and convergence imply a general framework which opposition-based optimization (as a subset of opposition-based computing for the particular case of optimization) can be described under. The outline follows that of Algorithm 3 but includes opposition-based requirements and is presented in Algorithm 4.

Line 2 performs opposition mining to determine some approximation Υ to the optimal opposite map Φ^*. If no mining is employed, it is assumed that some function Φ has been defined. The generation of opposites is performed in line 5 according to one of the strategies described in the previous section. Finally, line 8 updates any parameters Φ requires to generate opposite solutions.

Algorithm 4 Basic framework of opposition-based computational intelligence algorithms.

```
1:  Preprocessing.
2:  Opposition mining.
3:  for all $t = 1...n$ do
4:      Sample at least one element from the search space.
5:      Generate opposite sample.
6:      Comparison of element with best known.
7:      Update algorithm parameters.
8:      Update opposition-based parameters.
9:  end for
10: Postprocessing.
```

3.3 Comparison to Existing Methods

Intuition concerning opposites may sometimes involve the application of existing techniques. However, they are distinguishable from the opposition-based computing concept. The most notable techniques are antithetic variates, low-discrepancy/quasi-random sequences and dis/similarity functions.

3.3.1 Antithetic Variates

Suppose we desire to estimate $\xi = \mathbb{E}[f] = \mathbb{E}[Y_1, Y_2]$ with an unbiased estimator

$$\hat{\xi} = \frac{Y_1 + Y_2}{2}. \tag{3.11}$$

If Y_1, Y_2 are i.i.d. then $var(\hat{\xi}) = var(Y_1 + Y_2)/2$. However, if $cov(Y_1, Y_2) < 0$ then the variance can be further reduced.

One method to accomplish this is through the use of a monotonic function h. First, generate Y_1 as an i.i.d. value as before, but utilizing h the two variables become $h(Y_1)$ and $h(1 - Y_1)$, which are monotonic over interval [0,1]. Then,

$$\hat{\xi} = \frac{h(Y_1) + h(Y_2)}{2} \tag{3.12}$$

will be an unbiased estimator of $\mathbb{E}[f]$.

Opposition is similar in its selection of elements with different evaluations. However, antithetic variates require a predesigned monotonic function. Although [101] shows that one exists, no guidelines are provided. It is assumed prior knowledge, or left as an analytical exercise to derive should the form of $\mathcal{S}$ be known. Therefore, a key difference is that opposition-based computing allows arbitrary definition of opposites or online learning of the relationship. Moreover, using the OBC techniques one effectively compresses the search space by grouping elements into equivalence classes and giving a single representative value. No such compression is present when using antithetic variates.

Further, opposition extends beyond the generation of solutions in random sampling-based algorithms. It can also be applied to algorithm behavior and can be used to relate concepts expressed linguistically, where no evidence has been found that antithetic variates have such applications.

3.3.2 Quasi-Randomness and Low-Discrepancy Sequences

These methods aim to combine the randomness from pseudorandom generators which select values i.i.d. with the advantages of generating points distributed in a grid-like fashion. The goal is to uniformly distribute values over the search space by achieving a low-discrepancy. This is achieved at the cost of statistical independence.

The discrepancy of a sequence is a measure of its uniformity and can be calculated via [41]:

$$D_N(X) = \sup_{I \in J} \left| \frac{|B \cap X|}{N} - \lambda_d(B) \right| \tag{3.13}$$

where λ_d is the d-dimensional Lebesque measure, $|B \cap X|$ is the number of points of $X = (x_1, ..., x_N)$ that fall into interval I, and J is the set of d-dimensional intervals defined as:

$$\prod_{i=1}^{d}[a_i, b_i) = \{x \in \Re^d \colon a_i \leq x_i \leq b_i\} \tag{3.14}$$

for $0 \leq a_i < b_i \leq 1$. That is, the actual number of points within each interval for a given sample is close to the statistically expected number. Such sequences have been widely studied in quasi-Monte Carlo methods [61].

Opposition may utilize low-discrepancy sequences in some situations. Though in general, these sequences are simply a means for attaining uniform distribution and are static once determined, which is a major contrast with opposites (see Section 3.2.3). Further, opposition-based techniques simultaneously consider a low number of points (usually 2) in order to induce a symmetric evaluation function and improve performance of the sampling algorithm, whereas quasi-random sequences often are concerned with many more points (usually in the 100s or 1000s).

Quasi-randomness has been employed to incorporate diversity in a sampling procedure. These methods have been applied to evolutionary algorithms where it was found that by a performance study of the different sampling methods such as Uniform, Normal, Halton, Sobol and Faure that low-discrepancy is valuable only for low-dimensional ($d < 10$) and thus non-highly-sparse populations [67].

3.3.3 Similarity Functions

Learning, especially in a statistical context, has been recently examined using the concept of similarity and dissimilarity functions [6, 144, 145]. Instead of an algorithm being presented with feature vectors, it is presented with a matrix which quantifies the similarity or dissimilarity between all pairs of data items. It is assumed that K is some distance metric between points of $\mathcal{S}$.

Definition 15 (Similarity Function) *A similarity function over $\mathcal{S}$ is any pairwise function $K \colon \mathcal{S} \times \mathcal{S} \to [-1, 1]$. If K is symmetric then $K(x, x') = K(x', x)$ for all $x, x' \in \mathcal{S}$.*

As stated above, an opposite map aims to group items which have a very different evaluation. Sometimes this is attained by grouping dissimilar solution representations. However, one very common assumption of statistical learning is that similarly featured samples yield a similar result (the smoothness assumption [17]). Dis/similarity functions are often employed in unsupervised or semi-supervised learning situations where the problem assumptions differ from those of opposition. Specifically, opposition is concerned with grouping elements of the search space with respect to the evaluation function, whereas dis/similarity functions focus on grouping similar items with respect to features.

3.4 Looking Forward

The remaining chapters explore the use of intuitive knowledge concerning the opposition map design. Heuristic methods for updating the definition of Φ will be given and rigorously tested. Being intuitive, there is only strong circumstantial evidence to support the heuristic choice, but according to Equation 3.7 many such opposite mappings could be successful. All the algorithms in subsequent chapters fit within the framework of Algorithm 4.

It is important to note that opposition-based ideas have been previously applied with success. Reinforcement learning defining opposite states and actions was one of the first examples [84, 126]. It has subsequently been improved [114–116] and applied to various problems with promising outcomes [68, 105].

Another example, improving differential evolution has been explored in [85–87]. The idea uses a specific example of opposite map (given in Equation (3.3)) and has been shown to outperform differential evolution for many benchmark problems. This thesis will append simulated annealing (Chapter 4), population-based incremental learning (Chapter 5), backpropgation and resilient propagation variants (Chapter 6) to this list.

Each subsequent chapter will focus on improving traditional versions of the aforementioned algorithms. It is acknowledged that various flavors of each method exist, however, it was not feasible to examine each version. Thus, it was decided to focus on the common framework (i.e. the traditional algorithm) shared by each family of computational intelligence algorithm explored in this thesis. Consequently, the outcomes of the algorithms will probably not be to the quality of a state-of-the-art approach. However, each chapter will briefly discuss state-of-the-art implementations which can be investigated for future work. The choice to investigate traditional versions of several algorithms as opposed to an in-depth analysis focusing on a particular one was made to highlight the generality of the proposed framework discussed in Section 3.2.4.

Part II

Example Applications

Chapter 4

Monte Carlo Optimization

This chapter will discuss the application of opposition-based computing to a popular Monte Carlo optimization algorithm, simulated annealing. The opposite neighbor is introduced as a basic strategy to highlight the potential of the ideas considered in the previous chapters. Despite its simplicity, improvements in accuracy, precision and convergence rate will be achieved. Various benchmarking tests are conducted to validate the approach. An application to image thresholding is also presented. Most of the results of this chapter appear in [140].

4.1 Background

Without loss of generality, assume a maximization problem. Monte Carlo optimization methods take a simulation-based approach to determining

$$\theta^* = \arg\max_{\theta \in \Theta} f(\theta) \tag{4.1}$$

for some evaluation function $f(\cdot)$ and parameter space Θ. In this simulation-based approach analytical properties play a lesser role than many alternative numerical optimization methods. The use of Monte Carlo simulation methods have increased in popularity recently due to their relaxed constraints on Θ and $f(\cdot)$. However, the tradeoff is a lack of guarantee in computed result (within feasible time constraints). Hence, methods which are capable of improving the accuracy, precision and convergence rate are of importance.

4.1.1 Simulated Annealing

Simulated Annealing (SA) is a Monte Carlo optimization technique which takes motivation from the annealing process of cooling molten substances, for example to harden steel [14, 56]. An effect of careful cooling of the substance is the condensing of matter into a crystalline solid. This annealing process can be regarded as an algorithm which optimizes the stability of the final crystalline solid. Whether the state of minimum free energy is actually reached will depend on the rate of temperature decrease.

The probability for the system to be in state $s \in \mathcal{S}$ at a certain temperature T with free energy $E(s)$ is described by the Boltzmann distribution

$$Pr_T(s) = \frac{1}{Z} e^{\frac{-E(s)}{kT}} \tag{4.2}$$

where

$$Z = \sum_{s \in \mathcal{S}} e^{\frac{-E(s)}{kT}} \tag{4.3}$$

is a normalization constant and k is the Boltzmann constant. Given two states $s_1, s_2 \in \mathcal{S}$ a possible probability of moving from s_1 to s_2 is calculated using

$$Pr_T(s) = \max(1, e^{\frac{-(E(s_2)-E(s_1))}{kT}}). \tag{4.4}$$

The simulated annealing algorithm will employ Equation (4.4) to probabilistically accept a candidate solution.

Algorithm 5 shows the SA algorithm and presupposes the existence of an energy function $E(s)$ which computes the quality of a given solution. In Line 7 a neighbor s_n of s is generated via a user-defined procedure and is evaluated using the energy function in Line 8. Lines 9 - 12 monitor whether s_n represents a new best solution to the optimization process.

The approach then uses Equation (4.4) as a rule to accept s_n as the new search focus, where $\mathcal{U}(0,1)$ represents a randomly generated value from a uniform distribution over interval $(0,1)$. To simulate the slow annealing process of its inspiration a user determined temperature update function must be provided (Line 17). It has been shown that an annealing schedule like

$$T = \frac{a}{b + log(i)} \tag{4.5}$$

for $a, b > 0$ will yield the globally optimal solution, given infinite time [44]. Unfortunately, the available computation time is finite. Therefore, alternative cooling methods have been proposed [78]. The two main cooling schedules are

- **Arithmetic:**

$$T = a - bi$$

  where a is the initial temperature and b is the temperature decrease decrement. Normally, the initial temperature is problem dependant and $b \in [0.01, 0.2]$.

Algorithm 5 Simulated Annealing

```
Require: $k :=$ constant
Require: $T :=$ initial temperature
1: $s = s_0$
2: $e^s = E(s)$
3: $s^* = s$
4: $e^* = e$
5: $n = 0$
6: while termination criteria not satisfied do
7:     $s_n = neighbor(s)$
8:     $e_n = E(s_n)$
9:     if $e_n \leq e^*$ then
10:        $s^* = s_n$
11:        $e^* = e_n$
12:    end if
13:    if $\mathcal{U}(0,1) \leq exp(-(e^s - e_n)/kT)$ then
14:        $s = s_n$
15:        $e^s = e_n$
16:    end if
17:    $T = update(T)$
18:    $n = n + 1$
19: end while
```

- **Exponential:**

$$T = ab^i$$

  where a is an initial temperature and $b \in [0.8, 0.999]$ is the cooling factor.

Algorithm 5 represents the standard framework for simulated annealing, however, many advances have been proposed. Adaptive simulated annealing [50] allows the algorithm parameters associated with the cooling schedule and neighbor selection to be adjusted according to the progress of the algorithm. The resulting approach is more efficient and less sensitive to parameter selection of SA.

The application of chaos to simulated annealing has also been investigated [70], resulting in Chaotic Simulated Annealing. Chaotic systems are used during the initialization and neighborhood generation procedures. Benchmark results show an improvement in convergence and efficiency while also being easy to implement.

Ill-shaped energy functions can slow the convergence of simulated annealing. To minimizae this effect stochastic tunneling algorithms were proposed [146]. The basic idea is to transform the energy function such that the size of undesirable "energy wells" are reduced. Generally, the transformed function is defined beforehand and often requires a priori knowledge.

Sample-sort simulated annealing represents another advanced approach [123]. This method maintains an array of samplers, each operating at a static temperature. At each iteration a subset of the samplers' solutions is accepted and traditional simulated annealing is applied to them. Another large sample-based algorithm based on SA is coupled simulated annealing. In these approaches each sampler can make improvements is based on the other samplers. Both of these algorithms have shown improvements in solution quality over traditional simulated annealing.

Parallel simulated annealing algorithms have also been investigated [89] where two general parallel algorithms were proposed. Using job-shop scheduling and the traveling salesman problems it was shown that superlinear speedups can be attained.

Recently, the quantum annealing method has attracted attention for optimization of discrete combinatorial optimization problems [4, 26]. This approach has a very similar framework to simulated annealing although the temperature parameter of SA is replaced by a "tunneling field strength" value. One of the consequences of the field strength is to determine the search radius for neighborhood generation, similarly to adaptive simulated annealing.

4.2 Opposition-Based Simulated Annealing

This section will describe the concept of an opposite neighbor and its influence on the target evaluation function. The manner in which the diversity-convergence issue is handled is discussed and an algorithm is proposed.

4.2.1 Opposite Neighbors

Section 3.2.3 of Chapter 3 described three directions which can be followed when determining the dynamics of the opposite map, Φ. Here, opposites are determined with respect to the bounds of the current search window, $\mathcal{W} \subset \mathcal{S}$. Given $s \in \mathcal{S}$ a neighbor x will be generated, its parameter values determine the bounds of the search window. Then, an opposite will exist on the opposite side of the search window.

Consider $s \in \mathbb{R}^d$, for $d > 0$. Let a neighbor x be generated according to the following rule

$$x_i = s_i + \mathcal{U}(-\Delta_i, \Delta_i). \tag{4.6}$$

The search window $\mathcal{W}$ is defined such that $\pm\Delta$ defines the boundary around the current solution, s_i, i.e. $\mathcal{W} = [s_i - \Delta, s_i + \Delta]$ for all $i = 1 \dots d$. An opposite will be defined as the element located at the opposite side of the window with respect to s. For example if $\mathcal{S} = [-5, 5]^3$ and $s =< 2.5, 1.3, 0.3 >$ and the randomly generated neighbor $x =< 2.3, 0.5, 1.1 >$ then $\breve{x} =< 2.7, 2.1, -0.5 >$. Evaluation is then performed by simultaneously considering x and $\breve{x}$.

Also, as described in Chapter 3, the usefulness of considering opposite solutions will diminish with respect to the convergence of the parent algorithm. The heuristic decision used in this instance has been chosen to be based on a monotonically decreasing function. Specifically,

$$Pr(\text{consider opposite}) \leq e^{\frac{-n}{C}} \tag{4.7}$$

for $C > 0$ a constant and n is the current iteration of SA.

4.2.2 The Algorithm

The opposition-based simulated annealing algorithm serves as an example of how even a simplistic definition of opposite map and diversity-convergence control mechanism can be employed to improve computational intelligence algorithms.

Algorithm 6 presents the opposition-based simulated annealing (OSA) approach. As highlighted in Chapter 3, the parent framework remains as it was with opposition-based concepts added. In this case, the additional steps are found in lines 8 - 10. The function $opposite(i, s_i)$ will generate a guess that is opposite to neighbor s_i with respect to the present search focus s, according to rules in the previous section. The best of the neighbors and its corresponding evaluation e_i are retained.

The probability of employing the opposite guess is not obvious from Algorithm 6. It is implemented in the $opposite(i, s_i)$ function according to the exponential decay explained above (Equation 4.7). In this case, no update of opposition parameters is needed since only the present iteration is required by the diversity control mechanism.

4.3 Benchmarking

In this section the performance of OSA is benchmarked versus traditional SA with regard to problem dimensionality, neighborhood size and randomness (versus opposition). The effect of arithmetic and exponential cooling schedules is also examined.

The selected six benchmark functions are commonly available in literature and popular for benchmarking new procedures. The results have been averaged over 250 runs of length 5000. For experiments using the OSA algorithm, the constant $C = 500$ was empirically decided, but not tuned to optimality. This resulted in an additional 500 calls to the $evaluate(s_g)$ function, which will be taken into consideration when comparing the algorithms.

4.3.1 The Test Functions

Six common real optimization functions, which are to be minimized, have been utilized to benchmark the OSA algorithm.

1. The sphere problem [27]:

$$\text{sphere}(\mathbf{x}) = \sum_{i=1}^{n} {x_i}^2 \tag{4.8}$$

Algorithm 6 Opposition-Based Simulated Annealing

```
Require: $k :=$ constant
Require: $C :=$ constant
Require: $T :=$ initial temperature
1: $s = s_0$
2: $e^s = E(s)$
3: $s^* = s$
4: $e^* = e$
5: $i = 0$
6: while termination criteria not satisfied do
7:     $s_g = neighbor(s)$
8:     $s_o = opposite(i, s_i)$
9:     $s_i = \arg\min(E(s_g), E(s_o))$
10:    $e_i = E(s_i)$
11:    if $e_i \leq e^*$ then
12:        $s^* = s_i$
13:        $e^* = e_i$
14:    end if
15:    if $\mathcal{U}(0,1) \leq exp(-(e^s - e_i)/kT)$ then
16:        $s = s_i$
17:        $e^s = e_i$
18:    end if
19:    $T = update(T)$
20:    $i = i + 1$
21: end while
```

where $-5.12 \leq x_i \leq 5.12$. The minimum of this function is 0 and occurs at $\mathbf{x} = 0, ..., 0$.

2. Rosenbrock's Valley function [27]:

$$\text{rosenbrock}(\mathbf{x}) = \sum_{i=1}^{n-1}[100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2] \tag{4.9}$$

   where $-2 \leq x_i \leq 2$. This function has a minimum at $\mathbf{x} = 1, ...1$, with a corresponding value of 0.

3. A solution to Rastrigin's function [73] is defined over $-5.12 \leq x_i \leq 5.12$. The optimal value of this function is rastrigin$(0, ..., 0) = 0$.

$$\text{rastrigin}(\mathbf{x}) = 10n + \sum_{i=1}^{n}(x_i^2 - 10\cos(2\pi x_i)) \tag{4.10}$$

4. Schwefel function [111], which is bounded according to $-10 \leq x_i \leq 10$. The function has a minimum

at schwefel$(0, ..., 0) = 0$.

$$\text{schwefel}(\mathbf{x}) = \sum_{i=1}^{n} |x_i| + \prod_{i=1}^{n} |x_i| \tag{4.11}$$

5. Alpine function:

$$\text{alpine}(\mathbf{x}) = \sum_{i=1}^{n} |x_i \sin(x_i) + 0.1 x_i| \tag{4.12}$$

   where $-10 \leq x_i \leq 10$. The minimum of this function occurs at alpine$(0, ..., 0) = 0$.

6. De Jong's noiseless function #4 [27]:

$$\text{dejong}(\mathbf{x}) = \sum_{i=1}^{n} i {x_i}^4 \tag{4.13}$$

   where $-1.28 \leq x_i \leq 1.28$. The corresponding minimum value is also found at dejong$(0, ..., 0) = 0$.

4.3.2 Experimental Setup

This section will describe the behavior of the functions needed for OSA to operate, as described in Section 4.2. Specifically, it outlines the *neighbor*(), *update*() and *opposite*() functions. Unless otherwise noted, these functions and any parameters will remain constant.

Neighborhood Function A neighbor of some current guess $s \in \mathcal{S}$ is generated using the *neighbor*(s) function. Specifically, select m of the n variables in guess s and add a uniform random value $\pm\Delta$ where $\Delta = (L+H)/15$, where L and H represent the lower and upper bound of the current problem, respectively. This operator is not optimal, but performs well on each test function.

Temperature Update Fundamental to the operation of simulated annealing is the temperature update function, *update*(T). It has been empirically decided to update the temperature according to an exponential rule

$$T^i = \alpha \cdot T^{i-1} \tag{4.14}$$

for iteration i. Initially, set the decay constant $\alpha = 0.95$.

Determining Opposites As mentioned in Section 4.2, a dynamic reference point in determining opposites, which is accomplished by the *opposite*(i, s_g) function is utilized. Here, s_g is a neighbor of s as generated by the *neighbor*(s) function. An opposite of s_g with respect to s is a point $\breve{s}_o$ directly opposite to s with respect to $\mathcal{W}$. This can be summarized,

$$\breve{s}_o = s_{g,i} \pm \breve{\Delta} \tag{4.15}$$

where $\breve{\Delta}$ represents the opposite value added to the i^{th} variable of s_g to arrive at s_n by the *neighbor* function.

4.3.3 Problem Dimensionality

Here, the dimensionality of each benchmark function is varied in order to determine the sensitivity of OSA to various sized problems. The results will be compared to those found with traditional simulated annealing. The neighbor function will alter only 1 variable for these experiments. The results for varying the problem dimensionality size of $\{10, 25, 50, 100\}$ are presented in Table 4.1.

Table 4.1: Results for experiments on dimensionality with fixed variable change=1. The bold values represent a statistically significant results at 0.95 confidence. The final column reports Cohen's d-statistic.

		SA		OSA		
	dim	μ	σ	μ	σ	Effect
sphere	10	0.000	0.000	0.000	0.000	0.00
	25	0.001	0.000	0.001	0.000	0.00
	50	0.007	0.002	**0.005**	0.002	0.452,m
	100	0.269	0.045	**0.151**	0.049	0.781,ℓ
rosenbrock	10	7.678	0.689	7.674	0.682	0.003
	25	38.218	2.578	**35.465**	2.391	0.484,ℓ
	50	89.842	4.211	**87.762**	4.167	0.241,s
	100	263.658	6.638	**240.794**	6.087	0.874,ℓ
rastrigin	10	82.371	24.925	**63.258**	22.654	0.372,m
	25	215.611	31.620	**182.938**	22.001	0.514,ℓ
	50	422.267	46.710	**390.629**	34.155	0.361,m
	100	849.204	57.428	**776.422**	48.165	0.566,ℓ
schwefel	10	0.028	0.009	**0.025**	0.009	0.164,s
	25	0.178	0.036	**0.159**	0.033	0.265,s
	50	0.793	0.114	**0.692**	0.093	0.437,m
	100	4.633	1.144	**3.550**	0.572	0.514,ℓ
alpine	10	0.128	0.056	**0.099**	0.047	0.270,s
	25	0.798	0.225	**0.682**	0.225	0.250,s
	50	3.080	0.614	**2.809**	0.512	0.233,s
	100	12.789	1.935	**11.139**	1.674	0.415,m
dejong	10	0.000	0.000	0.000	0.000	0.000
	25	0.000	0.000	0.000	0.000	0.000
	50	0.000	0.000	0.000	0.000	0.000
	100	0.138	0.991	**0.019**	0.143	0.084

The results found using OSA show a much lower expected value (μ) and standard deviation (σ) than those found with SA. In each case the results of the OSA approach are more desirable than those found with SA. For the 24 experiments 19 were better using OSA, and 5 were equal (for > 3 decimal places OSA outperforms SA on all the experiments).

The null hypothesis that the two means are equal was tested using a t-test with a confidence value of 0.95. From the results it is found that many are statistically significant. Additionally, the results found by OSA exhibited a lower standard deviation, meaning that the results are more reliable. Overall, OSA is able to at least improve on all solutions.

Using Cohen's d-statistic (See Appendix A) the practical importance of the results is ascertained. That is, whether the results are merely statistically significant or whether the actual differences are practically "small" (s), "medium" (m) or "large" (ℓ). The final column of Table 4.1 provides this information. It is seen that $6/24 = 0.25$ of the results yield a small improvement, $5/24 = 0.21$ are medium and $6/24 = 0.25$ represent a large improvement. Thus, OSA has found practical improvements on $17/24 = 0.71$ of all test functions.

Figure 4.1 shows the influence of dimensionality on convergence of the OSA algorithm for the Schwefel function. Increasing the dimensionality has the typical effect of requiring more computation time for the algorithm to converge. The behavior of the convergence curve is also very similar as the dimensionality increases.

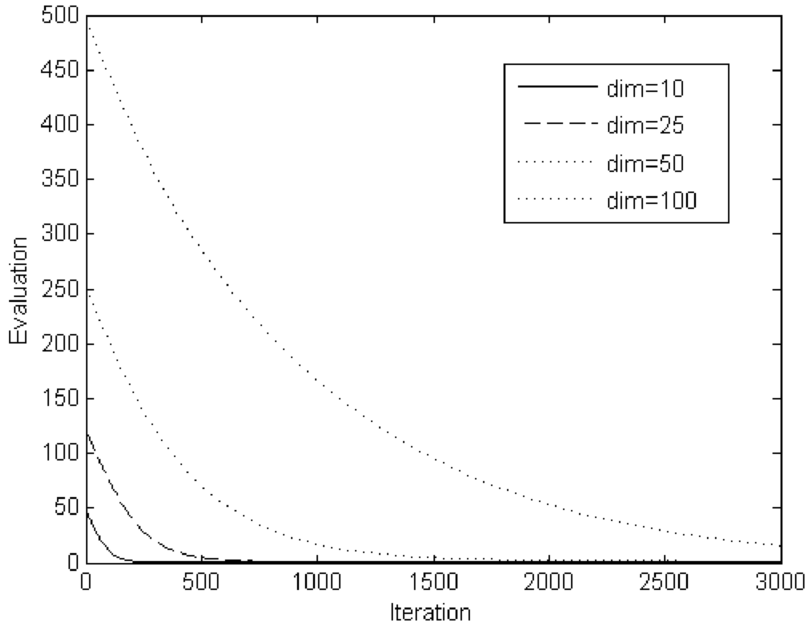

Figure 4.1: Influence of dimensionality on convergence of OSA on the Schwefel function.

A comparison of the convergence of the OSA and SA algorithms on the 25-dimensional De Jong function is presented in Figure 4.2. Although both approaches eventually achieve a similar final result, the OSA approach exhibits a much more rapid convergence. This characteristic curve is can also be seen above in Figure 4.1 for the Schwefel function.

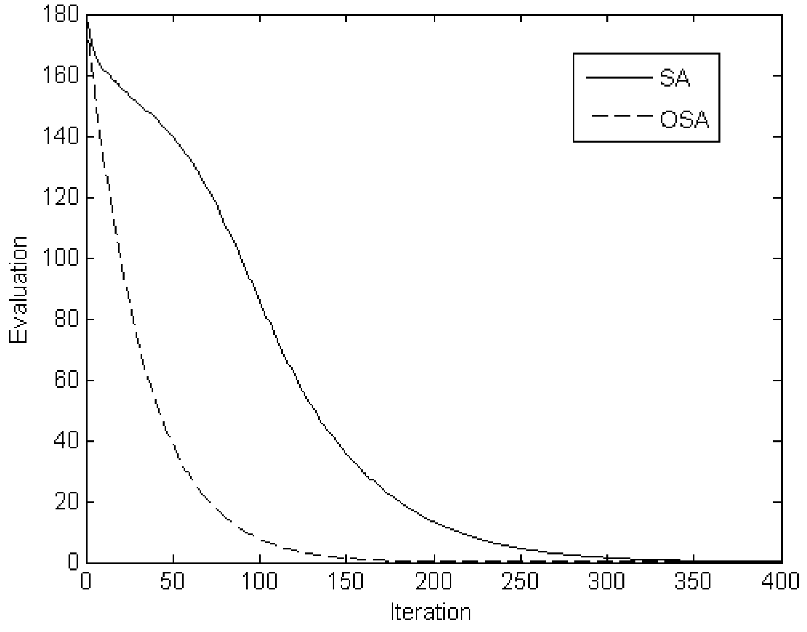

Figure 4.2: Convergence of OSA versus SA on the De Jong function of dimensionality 25.

4.3.4 Neighborhood Function

This experiment is aimed to discover the degree to which the neighborhood function influences the outcome of OSA. All experiments were run for a fixed dimensionality of 100 and the number of variables altered by the *neighbor*(s) function varied from $\{1, 3, 5\}$.

Table 4.2 lists these results, where the items in bold represent statistically significant results. As with the dimensionality experiments, the average OSA outcomes are relatively low compared to those achieved with SA. In fact, all results obtained using the OSA algorithm were more desirable than those reached using SA. A t-test with a 0.9 confidence level confirmed that all the results are indeed statistically significant. Furthermore, the standard deviations of results found via the OSA algorithm are also relatively low, and so the results are more reliable.

The final column of Table 4.2 provides the results of calculating Cohen's d-statistic (Appendix A).

These results show that $3/18 = 0.17$ of the results show a small practical improvement, $9/18 = 0.50$ can be classified as medium and $5/18 = 0.28$ are found to be large. Thus, $17/18 = 0.94$ of the test functions were practically improved using the opposition-based algorithm.

Table 4.2: Results for experiments on neighborhood with fixed dimensionality = 100. The bold values represent a statistically significant results at 0.9 confidence. The final column reports Cohen's d-statistic.

		SA		OSA		
	neigh.	μ	σ	μ	σ	Effect
sphere	1	0.269	0.045	**0.151**	0.048	0.785,ℓ
	3	0.909	0.111	**0.818**	0.094	0.405,m
	5	1.846	0.187	**1.727**	0.177	0.311,s
rosenbrock	1	273.658	46.382	**220.794**	43.877	0.505,ℓ
	3	229.613	42.386	**181.011**	32.081	0.543,ℓ
	5	212.753	36.991	**166.079**	29.565	0.577,ℓ
rastrigin	1	849.204	77.428	**806.422**	76.165	0.268,s
	3	882.186	76.033	**832.605**	79.160	0.304,s
	5	957.445	72.301	**899.624**	73.026	0.370,m
schwefel	1	4.633	1.144	**3.550**	0.572	0.514,ℓ
	3	14.018	1.091	**13.095**	0.991	0.405,m
	5	20.406	1.334	**19.428**	1.318	0.346,m
alpine	1	12.789	1.935	**11.139**	1.674	0.415,m
	3	35.514	4.500	**31.862**	4.192	0.387,m
	5	55.293	6.538	**49.644**	5.989	0.431,m
dejong	1	0.138	0.991	**0.019**	0.143	0.084
	3	0.004	0.001	**0.003**	0.001	0.447,m
	5	0.017	0.003	**0.015**	0.003	0.316,m

The influence of the number of variables altered at each call of the *neighbor*(s) function can be seen in Figure 4.3 for the Alpine function. When the *neighbor* function only perturbs a single variable the algorithm converges more rapidly than when more variables are considered. This is reasonable since the likelihood of escaping a local optima is lower if the operator makes only a small change to the solution. Nevertheless, the behavior of the convergence curve is relatively similar.

4.3.5 Neighborhood Randomness

Without loss of generality, Yao [155] described that increasing the neighborhood size corresponds to a higher probability of arriving at a global minimum. To examine whether the observed improvements shown above are a result of opposition or just increased neighborhood size they are compared against the results obtained by a randomized version of OSA. If the results obtained by OSA are of higher quality, then they are not simply due to a larger neighborhood but rather due to opposition. The randomized version is easily implemented by replacing line 8 in Algorithm 6 with *neighbor*(s). This effectively generates

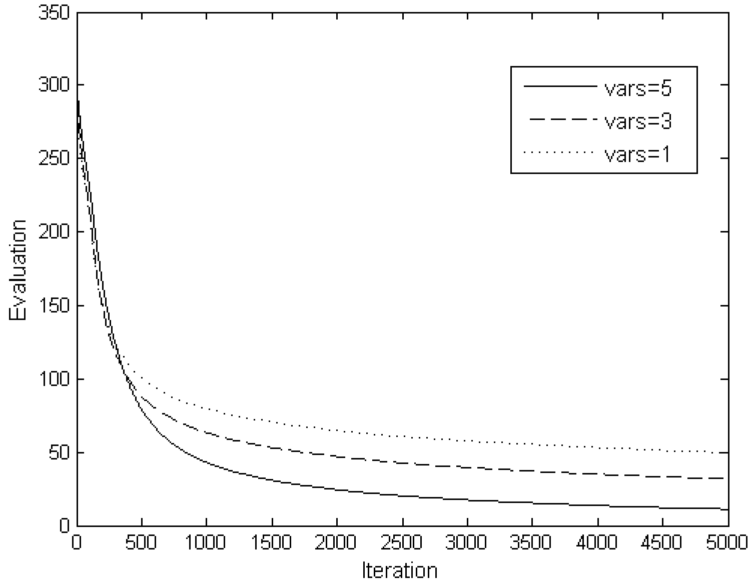

Figure 4.3: Influence of neighbor generation on convergence of 100-dimensional OSA for the Alpine function.

two random, independent neighbors. In the following this randomized simulated annealing algorithm is denoted RSA.

The results for this experiment are presented in Table 4.3 for fixed dimensionality of 100. Also, the number of variables altered by each call to the *neighbor* function is varied, as above. Most of the final results obtained with OSA are relatively lower than those obtained with RSA. In total 14/18=78% results favored OSA, 2/18=11% for RSA and 2/18=11% were equal. The *rosenbrock* experiments yielded a slightly more favorable result for the RSA algorithm. However, according to a t-test at a 0.95 confidence level all but the *dejong*-5 and rosenbrock-3 experiments exhibit a statistically significant difference in μ. Also, as with the previous experiments, the σ values are lower for OSA, indicating more reliable results. The average number of function calls to the $evaluate(Guess)$ function is the same for both techniques.

The practical improvement over SA is determined using Cohen's d-statistic, as is reported in the *Effect* column of Table 4.3. In total $8/18 = 0.44$ of the results have a small improvement favoring OSA. Therefore, the random neighbor-based algorithm performs well for this problem, however the opposite-version still yields many more favorable results.

Table 4.3: Comparing OSA vs RSA, fixed dimension=100. Bold values are more desirable. The final column reports Cohen's d-statistic where the appended s="small", m="medium" and ℓ="large" correspond to practical improvement effects.

		RSA		OSA		
	neigh.	μ	σ	μ	σ	Effect
sphere	1	0.220	0.053	**0.151**	0.047	0.412,s
	3	0.826	0.096	**0.818**	0.094	0.042
	5	1.787	0.185	**1.727**	0.177	0.163,s
rosenbrock	1	**230.556**	63.818	240.794	63.877	-0.080
	3	182.201	54.387	**181.011**	54.081	0.011
	5	**192.794**	49.155	196.079	59.565	-0.030
rastrigin	1	843.953	78.029	**806.422**	76.165	0.236,s
	3	864.800	71.285	**832.605**	79.160	0.209,s
	5	934.826	77.407	**899.624**	73.026	0.228,s
schwefel	1	3.890	0.858	**3.550**	0.572	0.227,s
	3	13.249	0.983	**13.095**	0.991	0.078
	5	19.522	1.365	**19.428**	1.318	0.035
alpine	1	11.508	1.447	**11.139**	1.674	0.117
	3	33.430	4.440	**31.862**	4.192	0.179,s
	5	52.503	6.299	**49.644**	5.989	0.227,s
dejong	1	0.024	0.224	**0.019**	0.143	0.013
	3	0.003	0.001	0.003	0.001	0.000
	5	0.015	0.003	0.015	0.003	0.000

4.3.6 Cooling Schedule Effect

The cooling schedule can affect the behavior of the simulated annealing algorithm. In this section a comparison between the linear and exponential schedules is provided for various values, not including the initial temperature which is fixed to $T_0 = 1000$. The neighborhood size is also fixed to 1. All results presented are averaged over 250 trials, each of 5000 iterations.

Table 4.4 shows the results for various parameters of the linear cooling schedule. Common values for this parameter are $b \in [0.01, 0.20]$, hence the experiment is run for values within this range. With exception of the Alpine function, all results obtained by OSA are more desirable than both SA and RSA. Furthermore, nearly all experiments show a lower standard deviation for OSA obtained results.

Computing Cohen's d-statistic on the best results for RSA and OSA from Table 4.4 reveal that the Sphere (0.710), Schwefel (0.839) and Dejong (0.504) functions are largely improved, and the Rosenbrock (0.255) function has a "small" improvement. The Alpine function showed a small improvement in favor of RSA while the Rastrigin function did not have any favored result.

Table 4.4: Comparing results for traditional simulated annealing (SA), randomized neighbor (RSA) and opposition-based (OSA) for various parameters of the linear cooling schedule. With exception of the Alpine function, OSA yields the most desirable results.

	Cooling Rate Parameter (b=)									
	0.01		0.05		0.10		0.15		0.20	
Algorithm	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
Sphere										
SA	807.86	66.35	801.80	67.89	806.47	68.69	800.15	61.18	800.86	59.90
RSA	303.16	48.47	302.80	47.01	299.41	48.19	302.46	49.41	301.25	47.18
OSA	210.71	39.44	213.03	35.73	212.09	34.98	211.46	36.95	212.62	36.86
Rosenbrock										
SA	21401.13	3110.74	19870.70	2969.00	17470.39	2657.83	13050.22	2154.64	5360.22	1046.52
RSA	7623.73	1557.77	7346.94	1602.38	7028.55	1406.75	6846.97	1319.99	3756.35	623.54
OSA	5418.16	1180.82	5318.3	1135.81	5158.00	1231.53	5038.95	1167.17	3441.56	567.04
Rastrigin										
SA	1631.94	73.08	1636.43	70.02	1632.86	69.98	1631.37	69.92	1552.88	83.57
RSA	1346.33	75.20	1340.22	77.59	1352.48	79.22	1339.48	78.21	1321.26	74.80
OSA	1322.89	67.47	1325.23	70.31	1321.24	72.41	1324.43	72.89	1305.03	65.39
Schwefel										
SA	473.05	24.47	470.88	23.75	474.83	26.73	473.05	24.67	470.55	24.47
RSA	273.74	21.56	272.65	21.39	275.46	22.95	275.66	22.54	272.13	23.88
OSA	203.89	18.61	202.92	19.32	203.80	20.32	203.2	19.01	202.56	21.10
Alpine										
SA	254.30	15.68	254.86	16.32	257.7	15.62	255.29	15.22	252.95	14.94
RSA	171.91	15.33	171.41	15.28	171.31	13.73	171.65	15.18	170.69	15.01
OSA	175.87	14.32	179.16	13.52	178.21	14.83	177.89	15.09	177.37	14.41
Dejong										
SA	2236.50	319.56	2210.26	328.57	2197.88	342.39	2164.47	337.42	1873.44	283.28
RSA	448.67	137.21	456.35	154.11	448.28	152.03	447.12	133.89	455.81	138.77
OSA	323.68	126.16	337.62	124.06	305.52	117.62	321.23	114.52	313.61	123.61

A comparison between SA, RSA and OSA for different cooling parameters of the exponential cooling schedule is presented in Table 4.5. As with results for the linear schedule, all values except those for the Alpine function are in favor of OSA. Moreover, the reliability of the results is higher, as the standard deviation is lower. Cohen's *d*-statistic reveals large improvements in the Sphere (0.728), Rosenbrock (0.564), Schwefel (0.858) and Dejong (0.488) functions, if comparing the best result obtained for each of the RSA and OSA algorithms, respectively. Only the Alpine function had a small improvement of -0.198 favoring RSA (the negative value implies favoring the non-opposition-based algorithm).

Table 4.5: Comparing results for traditional simulated annealing (SA), randomized neighbor (RSA) and opposition-based (OSA) for various parameters of the exponential cooling schedule. With exception of the Alpine function, OSA yields the most desirable results.

	Cooling Rate Parameter (b=)									
	0.80		0.85		0.90		0.95		0.99	
Algorithm	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
					Sphere					
SA	796.38	70.20	811.59	64.02	809.58	65.77	812.6	74.54	810.20	65.63
RSA	301.83	48.91	305.819	52.63	300.83	48.41	302.82	45.51	301.38	47.04
OSA	212.99	37.87	211.50	37.10	215.02	37.43	213.73	39.33	216.87	36.56
					Rosenbrock					
SA	39631.97	5702.29	39395.24	5791.32	39228.55	5487.13	39506.02	5212.44	39068.18	5275.09
RSA	9945.66	2468.65	9929.62	2506.93	9676.62	2288.42	9861.94	2423.88	9692.72	2375.25
OSA	6855.96	1965.73	6925.56	1725.91	6785.24	1849.06	6864.85	1837.63	6922	1854.42
					Rastrigin					
SA	1635.90	73.36	1644.00	71.80	1640.24	75.16	1643.56	73.68	1640.8	70.84
RSA	1344.71	81.03	1338.04	78.14	1346.75	77.23	1337.45	76.26	1339.95	85.17
OSA	1325.12	66.15	1326.19	70.40	1330.28	71.24	1326.71	75.05	1330.24	69.93
					Schwefel					
SA	469.67	24.60	471.90	24.97	471.32	25.62	475.3	24.67	474.95	24.71
RSA	273.3	23.14	272.82	22.91	274.16	22.09	272.64	22.17	272.7	22.27
OSA	203.73	22.3	203.09	17.04	203.28	18.54	202.61	19.68	204.79	19.06
					Alpine					
SA	253.70	16.48	255.79	15.02	256.15	16.45	254.36	16.05	255.87	13.95
RSA	172.9	15.13	1171.25	15.52	172.38	13.99	173.4	14.86	170.99	15.56
OSA	177.15	14.93	178.94	15.59	179.63	14.64	178.02	14.02	177.64	15.15
					Dejong					
SA	2295.39	322.48	2339.67	331.18	2297.2	326.4	2314.56	329.45	2296.43	319.22
RSA	482.32	145.88	468.80	142.66	489.23	143.85	472.69	148.91	463.73	135.91
OSA	337.19	133.52	337.89	123.04	347.88	128.72	318.1	124.36	340.87	134.1

4.4 Image Thresholding

Image segmentation involves partitioning an image I into a set of segments with the goal of locating objects in the image which are sufficiently similar. Thresholding is a subset problem of image segmentation, with only 2 classes defined by whether a given pixel is above or below a specific threshold value ω. This task has numerous applications and several general segmentation algorithms have been proposed [156]. However, due to the variety of image types there does not exist a single algorithm for segmenting all images optimally. Further, numerous methods for evaluating the quality of a segmentation have also been put forth [159]. In this chapter, a simple method which aims to minimize the discrepancy between the original $M \times N$ gray-level image I and its thresholded image T [147] is used:

$$\sum_{i=1}^{M}\sum_{j=1}^{N}|I_{i,j} - T_{i,j}| \tag{4.16}$$

where $|\cdot|$ represents the absolute value operation. Using different evaluations will change the outcome of the algorithm, however, the problem of segmentation in this manner nonetheless remains computationally expensive and decreasing the number of evaluations is thus an important task.

Figure 4.4 shows the images used to evaluate the algorithms. The first column represents the original image, then the gold and the third column corresponds to the approximate target image given Equation (4.16). The gold image is shown for completeness, it is not required in the experiments. Table 4.6 provides the value-to-reach (VTR) target evaluation which corresponds to the approximate optimal target image with respect to Equation (4.16)

Table 4.6: Value-to-reach (VTR) for comparing RSA and OSA.

Image	Value-to-reach
1	19850
2	4925
3	7175
4	19850
5	19700
6	22700

4.4.1 Results

The goal of this experiment is to compare the required number of iterations needed by RSA and OSA to reach the VTR value given in Table 4.6. To increase the problem difficulty, the original image I is split into 4×4 equal sized regions and the algorithms will determine a ω for each of these 16 regions. Hence, the problem representation will be a vector V of length 16 where each $(v \in V) \in [0, 255]$. The initial temperature has been tuned to $T_0 = 1000$ and the neighborhood function generates a random number between $(-17, 17)$, as per the same strategy used in benchmarking. The cooling schedule is exponential with parameter $b = 0.98$.

Table 4.7 presents the results for the image thresholding task, averaged over 30 trials. Each iteration required two function calls, as two candidate solutions are generated (either two random or one random and its opposite for RSA and OSA, respectively). Although the increased problem difficulty led to high standard deviations, there is an improvement in expected function calls to reach the respective targets. For all six instances OSA required a lower number of iterations to achieve the desired result. The overall expected improvement is 466 iterations. Averaged, this yields approximately 78 iterations less, per image. This is an improvement factor of $2158/1694 = 1.27$.

The final column of Table 4.7 presents Cohen's *d*-statistic (nonoverlap), as described in Appendix A. Using this measure Images 5 and 6 are found to show small practical improvements, Images 1 an 2 "medium" whereas Images 3 and 4 show large improvements. Thus, in all cases a useful advantage for using OSA over RSA is revealed.

Table 4.7: Summary results for RSA vs. OSA with respect to the required number of iterations needed to reach the VTR. μ and σ correspond to the mean and standard deviation of the subscripted algorithm, respectively. Bold values are statistically significant at a 0.95 confidence level using a t-test. The final column reports Cohen's *d*-statistic.

Image	μ_{RSA}	σ_{RSA}	μ_{OSA}	σ_{OSA}	Improvement	Effect
1	419	134	324	112	**95**	0.359,m
2	415	135	304	107	**111**	0.416,m
3	303	85	196	60	**109**	0.588,ℓ
4	251	48	180	36	**71**	0.642,ℓ
5	348	120	299	80	**49**	0.234,s
6	422	108	391	88	31	0.155,s

4.5 Summary

The purpose of this chapter was to show that even a basic opposite map definition can lead to significant improvements in the behavior of simulated annealing. The proposed approach followed directly from the framework outlined in the previous chapters. Benchmarking the new algorithm focused on ascertaining the sensitivity of OSA with respect to the neighborhood function and problem size. The cooling schedule was also examined using different forms of the exponential and linear methods. After revealing various improvements the method was employed to the practical problem of image thresholding, which further confirmed its efficacy with respect to lowering the needed number of function calls to achieve a desired target thresholding. An important observation was that improvements were more drastic as the problem dimensionality increased.

The next chapter expands on ideas presented here to a population-based algorithm. The goal will be to improve diversity via opposites. As a consequence, other desirable improvements are obtained.

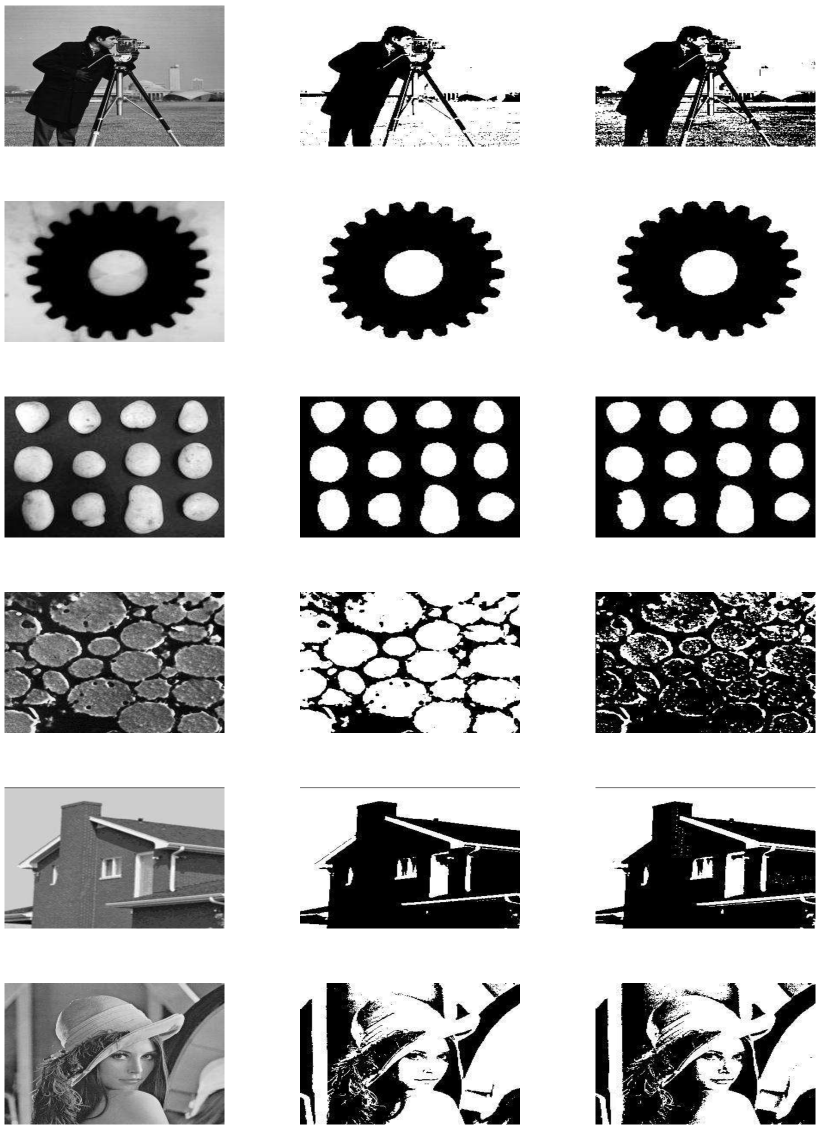

Figure 4.4: The images used to benchmark the algorithms. The first column is the original gray-level image, the second is the gold and the third colum is the target image of the optimization within the required function calls.

Chapter 5

Estimation of Distribution Algorithms

In this chapter a new probability update rule and sample generation procedure for population-based incremental learning are presented. Given an arbitrary sample, its corresponding opposite is generated by taking the complement of the initial binary solution. This complement is dynamic in that the entire solution complement is not always considered, 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 size difference between opposite samples is decreased in order to allow the algorithm to 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. To highlight the improvements, benchmark binary optimization problems are employed. The traveling salesman problem and image thresholding tasks are also used to compare against a traditional version of the algorithm. The majority of this work appears in [136, 141].

5.1 Background

Estimation of Distribution (EDA) algorithms represent a powerful class of stochastic optimization techniques based on evolutionary computation and machine learning [59]. In EDAs there are no operations of crossover and mutation which are found in traditional evolutionary-based algorithms. Furthermore, the population is not explicitly stored between successive generations. Instead, EDAs maintain and continuously modify an estimated probability distribution over possible solutions. As a result, the joint distribution explicitly represents the interrelations between variables which are implicitly present in evolutionary algorithms via the linkage phenomena. While very useful, estimation of this joint distribution is a computational bottleneck for EDAs. Furthermore, in lieu of a selection mechanism the probability model represented by the prototype distribution is updated accordingly. The general EDA algorithm is provided in Algorithm 7 for distribution $\mathcal{D}$ over M variables and $x \in \mathcal{S}$.

Algorithm 7 The general EDA algorithm framework.

```
1: $\mathcal{D}_0 \leftarrow$ randomly generate initial population,
2: for $i = 1$ until stopping criteria is met do
3:     $\mathcal{D}_{i-1}^{Se} \leftarrow$ select $N \leq M$ from $\mathcal{D}_{i-1}$.
4:     $p_i(\mathbf{x}) = p(\mathbf{x}|\mathcal{D}_{i-1}^{Se}) \leftarrow$ estimate probability distribution of selecting a given $\mathbf{x}$.
5:     $\mathcal{D}_i \leftarrow$ sample $M$ solutions from $p_i(\mathbf{x})$.
6: end for
```

The EDA used in this chapter is known as Population-based Incremental Learning (PBIL) [7]. 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) [118]. It has been experimentally shown that population-based incremental learning has the capacity to outperform traditional evolutionary algorithms [40].

PBIL's implementation ease and robustness have allowed it to be employed for solving a variety of real-world problems. In [154] 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 [33]. It was found that PBIL designed controllers are able to adequately stabilize the system over the varying conditions. Additionally, for large disturbances in conditions PBIL is shown to outperform a genetic algorithm approach. An initial investigation into the use of PBIL for evolutionary robotics was given in [118]. 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 [133]. 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 [51]. And, in [83] PBIL was employed to provide a measure of the uncertainty in parameters for reservoir models which quantify the amount of risk associated with alternative production scenarios.

5.1.1 Population-Based Incremental Learning

Population-Based Incremental Learning (PBIL) is a population-based stochastic search which combines elements from evolutionary computation (EC) and reinforcement learning (RL) [7]. 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 each $b_{i,j} = 1$.

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 [7]:

$$\mathbf{M}_t = (1-\alpha)\mathbf{M}_{t-1} + \alpha \mathbf{B}^*, \tag{5.1}$$

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 8. 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.

Initially, in line 2 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, 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 [48, 91, 92]. PBIL algorithms for continuous spaces have also been explored (for examples see [102, 112]), although only the discrete binary case is considered in this chapter.

The Double Learning PBIL algorithm, based on an elitist strategy using both current best and global best solutions for updating is studied in [160], where improvements in convergence speed and accuracy are

Algorithm 8 Population-Based Incremental Learning [7]

```
1: {Initialize probabilities}
2: $\mathbf{M}_0 := (m_{i,j}) = 0.5$
3: for $t = 1$ to $\omega$ do
4:     {Generate samples}
5:     $G_1$ = generate_samples($k$,$\mathbf{M}_{t-1}$)

6:     {Find best sample}
7:     $\mathbf{B}^*$ = select_best($\{\mathbf{B}^*\} \cup G_1$)

8:     {Update $\mathbf{M}$}
9:     $\mathbf{M}_t = (1-\alpha)\mathbf{M}_{t-1} + \alpha\mathbf{B}^*$

10:    {Mutate probability vector}
11:    for $i = 0...d$ and $j = 0...c$ do
12:        if $random(0,1) < \beta$ then
13:            $m_{i,j} = (1-\gamma)m_{i,j} + \gamma \cdot random(0\ or\ 1)$
14:        end if
15:    end for
16: end for
```

observed. Additionally, the usefulness of PBIL for dynamic problems has also been investigated [153].

Expanding PBIL to multi-objective problems has been examined in [13]. Two different probability vector updating schemes were proposed and the resulting approach is compared to other population-based methods. Using 8 bi-objective test problems it is found that multi-objective PBIL is similar in convergence rate, but has improved diversity over the other algorithms.

While EDAs have been quite successful, they (as well as evolutionary algorithms) have a tendency to converge to local optima [11, 113, 157]. This premature convergence is due to various algorithmic properties, but a major reason is due to a lack of diversity in the sample. Typically, diversity is highest at the onset of the algorithm when all solutions have been randomly generated. As the algorithm progresses diversity normally decreases rapidly due to sample generating bias, therefore increasing the difficulty of escaping a local optima.

5.2 Oppositional Population-Based Incremental Learning

As a basis for the proposed approach, it is proven that a higher diversity can be achieved during the sampling stages of PBIL using opposition. A new update rule made possible because of the properties of opposition is also presented. This rule requires a new procedure to mutate the probability matrix $\mathbf{M}$.

5.2.1 Improved Diversity

Diversity is a measure of a sample or population, independent of the evaluation function [150]. 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.

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 [66, 150].

Since the Hamming distance (d_{HAM}) is symmetric it is only needed 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 [150]

$$V(\mathcal{P}) = \sum_{i=1}^{n}\sum_{j=1}^{i-1} d_{HAM}(p_i, p_j), \tag{5.2}$$

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

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

Definition 16 (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:*

$$\breve{g}_i = \begin{cases} 1, \textit{if } g_i = 0, \\ 0, \textit{otherwise} \end{cases} \quad \textit{for } i = 0...l \tag{5.3}$$

Definition 17 (Comparison Set) *Given some universal set $\mathcal{U}$ and subsets $A \subset \mathcal{U}$ and $B \subset \mathcal{U}$, define a* comparison set *S as the Cartesian product $A \times B = \{< a, b > \mid a \in A \text{ and } b \in B\}$. This set is denoted as $S = A \rightarrow B$, inferring set A is being compared to set B.*

Definition 18 (Opposite Set) *Given some universal set $\mathcal{U}$ and subset $A \subset \mathcal{U}$, 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 19 (Diversity of Binary-valued Opposite set) *Given some universal set $\mathcal{U}$ and opposite* *set $A^\circ \subset \mathcal{U}$ of size m, calculate the all-pairs-diversity of this opposite set as*

$$V(A^\circ) = \sum_{i=1}^{m} d_{HAM}(p_i, \breve{p}_i) \tag{5.4}$$

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

Using these criteria, it is now proven 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).

Additionally, assume the samples are generated according to a probability matrix $\mathbf{M}$ (as is the case with PBIL) and 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,

$$V(R^\circ) = \sum_{i=1}^{m} d_{HAM}(p_i, \breve{p}_i) \tag{5.5}$$

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, 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 5.2.1 (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 $V(R_1 \rightarrow \breve{R}_1) \geq V(R_1 \rightarrow R_2)$.*

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

$$S^\circ = (R_1 \rightarrow \breve{R}_1) \setminus S_2 \tag{5.6}$$

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

It is also possible to write $R_1 \to R_2$ as the composition of two subsets,

$$R_1 \to R_2 = R_a \cup R_b \tag{5.7}$$

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 it is assumed that in general $V(S_2) = V(R_b)$ since S_2 and R_b are essentially composed of random samples, then,

$$\begin{aligned} V(R_1 \to \breve{R}_1) - V(R_1 \to R_2) =& V(S^{\circ}) + V(S_2) - (V(R_a) + V(R_b)) \\ =& V(S^{\circ}) - V(R_a) \\ \geq& 0 \end{aligned} \tag{5.8}$$

■

Now, extend Theorem 5.2.1 to the complete situation where the all-pairs-diversity of an entire set of guesses is considered. It is proven that using opposite guesses yields a higher diversity than traditional independent random sampling.

Theorem 5.2.2 (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 Begin by decomposing $V(G_1)$ and $V(G_2)$ into

$$V(G_1) = V(R_1) + V(R_2) + V(R_1 \to R_2), \tag{5.9}$$

$$V(G_2) = V(R_1) + V(\breve{R}_1) + V(R_1 \to \breve{R}_1). \tag{5.10}$$

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, the assumption that in general $V(\breve{R}_1) = V(R_2)$ is made. So, by Theorem 5.2.1,

$$V(G_2) - V(G_1) = V(R_1 \to \breve{R}_1) - V(R_1 \to R_2) \geq 0 \tag{5.11}$$

■

Also, the following property concerning the convergence of diversity between generating samples by opposition versus solely independent random guessing can be proven.

Corollary 2 (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,*

$$\lim_{V(R_1 \to R_2) \to 0} V(R_1 \to \breve{R}_1) - V(R_1 \to R_2) = l \cdot \frac{k(k+1)}{2}, \tag{5.12}$$

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 it is assumed that 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

$$\lim_{t \to \infty} V(R_1^t \to R_2^t) = 0 \tag{5.13}$$

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 $d_{HAM}(a, b) = 0 \; \forall a \in R_1$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

$$\begin{aligned} \lim_{V(R_1 \to R_2) \to 0} V(R_1 \to \breve{R}_1) - V(R_1 \to R_2) &= \lim_{V(R_1 \to R_2) \to 0} V(R_1 \to \breve{R}_1) \\ &= l \cdot \frac{k(k+1)}{2}. \end{aligned} \tag{5.14}$$

■

From this corollary diversity is infused into a system utilizing the 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 [161]. 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.

5.2.2 Proposed Use for Opposite Guessing Strategy

Thus far, it has been shown 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 [161]. Thus, in order for opposition to be useful the definition of opposite guesses must adapt to this scenario. This is due to the property of a search algorithm to discover high

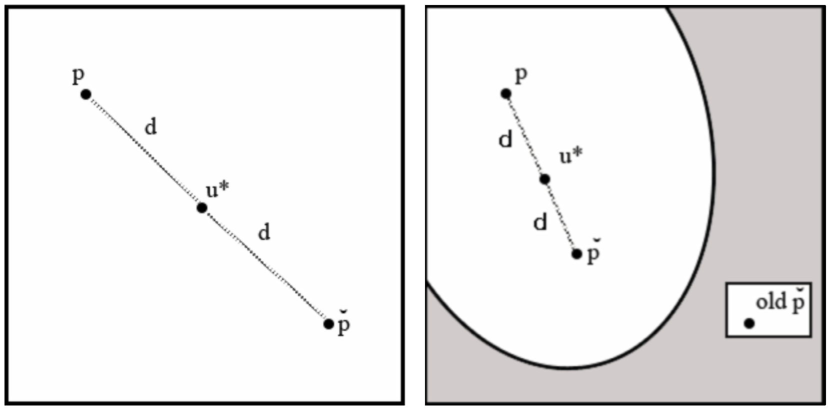

Figure 5.1: (a) If u^* lies in the center of the search space then every point has an opposite and (b) if u^* is translated then points p, $\breve{p}$ can lie closer together. The shaded area represents elements which have no notion of opposite, and are essentially removed from consideration. In both cases considering opposites with respect to point p.

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^* the magnitude of the distance between two points can be relaxed. Moving u^* to some location other than the center of $\mathcal{U}$ yields the following consequences (which are also shown in Figure 5.1):

- 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.

The translation of u^* need not be explicit. In the case of a binary domain (as is the case in this chapter) it 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 a mechanism to control the distance between opposite points is given it is possible to control the amount of diversity infused into the search algorithm by opposite guesses.

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 can be modeled. For binary problems this is used to reflect the Hamming distance. While a variety of decision functions are possible, an exponentially decaying function in the flavor of

$$\xi(t) = le^{(ct)}, \tag{5.15}$$

is used, where l represents the maximum number of bits in a guess and $c < 0$ is a user defined constant. Then, take the complement of $\xi(t)$ by randomly selecting 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.

5.2.3 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, opt for a more advanced mutation rule which adjusts probabilities of $\mathbf{M}$ through decay or amplification functions, with respect to either the current sample best or current 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 stabilize this situation he following logic is employed:

- Whenever a new solution $\mathbf{B}^*$ 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 quality solutions.
- 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, two parameters are introduced; $0 < \tau < 1$ and $0 < \rho < 1$, respectively. The above three logics are implemented as described in the following.

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

$$m_{i,j} = m_{i,j} \cdot (1 - \rho) + \mathbf{B}^*_{i,j} \cdot \rho. \tag{5.16}$$

Also apply this rule to exploit 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 used in this chapter that represents this situation is

$$p_{amp}(\Delta) = 1 - e^{-b\Delta} \tag{5.17}$$

where $b > 0$ is a user defined constant. Use the sample best solution η instead of the global best $\mathbf{B}^*$ (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, but experimentation has led to the selection of this one. No claim is made that it is the optimal choice.

If no amplification of the probabilities, then a probabilistic decay allows for exploration. This probability is controlled by (as with Equation (5.17), no claim to its optimality is made)

$$p_{decay}(\Delta; f(\mathbf{B}^*), f(\eta)) = \frac{1 - \frac{f(\mathbf{B}^*) - f(\eta)}{f(\mathbf{B}^*)}}{\sqrt{\Delta + 1}} \tag{5.18}$$

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 (5.18) for the case where $\eta_{i,j} = \mathbf{B}^*_{i,j}$ is:

$$m^{t+1}_{i,j} = m^t_{i,j} \cdot \begin{cases} 1 - \tau \cdot random(0,1), & \text{if } \eta_{i,j} = 1, \\ 1 + \tau \cdot random(0,1), & \text{otherwise} \end{cases} \tag{5.19}$$

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

$$m^{t+1}_{i,j} = m^t_{i,j} \cdot \begin{cases} 1 + \tau \cdot random(0,1), & \text{if } \eta_{i,j} = 1, \\ 1 - \tau \cdot random(0,1), & \text{otherwise} \end{cases} \tag{5.20}$$

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 employ the global best solution in lieu of η in Equation (5.19). This can also be done in a probabilistic or heuristic manner. For simplicity the probability of Equation (5.18) is chosen here.

5.2.4 Summary of the Proposed Algorithm

In this subsection an outline the OPBIL algorithm is given. Specifically, the probability update rule and sample generation procedure are altered from PBIL and the mutation rule is removed. The pseudocode for OPBIL is presented in Algorithm 9.

In line 2, initialize the probability matrix to reflect the amount of prior knowledge 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) create an opposite for each of the $k/2$ elements (in line 6), represented by the set $\breve{R}_1$ in accordance with Section 5.2.2. In lines 8 and 9 the sample best η and global best $\mathbf{B}^*$ solutions 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, 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 $b = 0.01$). 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 vicinity. The update itself is performed in line 16.

If the first test fails, it can 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.

5.3 Benchmarking

This section is composed of three sets of experiments. The first is concerned with testing and comparing the results of OPBIL parameters, followed by a comparison to PBIL using standard binary optimization benchmark functions which have characteristics of deception and multiple attraction regions. The second set of experiments compares OPBIL and PBIL using 10 instances of the Traveling Salesman Problem

Algorithm 9 Pseudocode for the OPBIL algorithm

```
Require: Maximum iterations, $\omega$
Require: Number of samples per iteration, $k$
 1: {Initialize probabilities}
 2: $\mathbf{M}_0 = m_{i..l} = 0.5$

 3: for $t = 1$ to $\omega$ do
 4:     {Generate samples}
 5:     $R_1$ = generate_samples($k/2$,$\mathbf{M}$)
 6:     $\breve{R}_1$ = generate_opposites($R_1$)

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

10:     {Compute probabilities}
11:     $p_{amp}(\Delta) = 1 - e^{-b\Delta}$
12:     $p_{decay}(\Delta; f(\mathbf{B}^*), f(\eta)) = \frac{1 - \frac{f(\mathbf{B}^*) - f(\eta)}{f(\mathbf{B}^*)}}{\sqrt{\Delta + 1}}$
13:     {Update $\mathbf{M}$}
14:     if $\eta = \mathbf{B}^*$ OR $random(0,1) < p_{amp}$ then
15:         $\Delta = 0$
16:         $\mathbf{M}_t = (1 - \rho)\mathbf{M}_{t-1} + \rho\eta$
17:     else if $random(0,1) < p_{decay}$ then
18:         if $random(0,1) < p_{decay}$ then
19:             use $\mathbf{B}^*$ in line 23 instead of $\eta$
20:         end if
21:         for all $i, j$ each with probability $< p_{decay}$ do
22:             if $\eta_{i,j} = \mathbf{B}^*_{i,j}$ then
23:                 $m_{i,j} = m_{i,j} \cdot \begin{cases} 1 - \tau \cdot random(0,1), & \text{if } \eta_{i,j} = 1, \\ 1 + \tau \cdot random(0,1), & \text{otherwise} \end{cases}$
24:             else
25:                 $m_{i,j} = m_{i,j} \cdot \begin{cases} 1 + \tau \cdot random(0,1), & \text{if } \eta_{i,j} = 1, \\ 1 - \tau \cdot random(0,1), & \text{otherwise} \end{cases}$
26:             end if
27:         end for
28:     end if
29:     $\Delta = \Delta + 1$
30: end for
```

(TSP). The ability to achieve a lower (more desirable) evaluation with higher stability is highlighted. The final experiment uses 5 common image thresholding problems to show the ability to achieve a desired value-to-reach accuracy with a lower number of fitness evaluations. Unless otherwise noted, 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, set the amplification $\rho = 0.05$, decay $\tau = 0.0005$. The function used to determine the distance between opposite guesses is

$$\xi(t) = \max(1, e^{-0.01t}) \tag{5.21}$$

where t is the current iteration of the algorithm. The range of possible functions for $\xi(t)$ was not explored, but of those attempted Equation (5.21) 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.

Additionally, also examined were 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.

5.3.1 The Test Functions

In order to evaluate the efficacy of the OPBIL algorithms four common binary optimization problems were utilized. 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 optima [149]. 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 $f\colon (\{0,1\}^m)^n \to \mathbb{Z}^*$, where n is the number of dimensions of size m in a solution representation.

Goldberg's 3-bit Deceptive Function Goldberg's multi-modal 3-bit deceptive function [39] utilizes 3 bits per each of the n dimensions and is evaluated according to Equation (5.22). This is a multi-modal function, meaning there exist more than one local optima, but only a single global optimum.

$$f(x) = \sum_{i=1}^{n} h(x_i) \tag{5.22}$$

Each 3-bit pattern is evaluated by the $h(\cdot)$ function which is summarized in Table 5.1. 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 5.1: 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

Whitley's Functions Whitley's 3 and 4-bit deceptive functions are similar to Goldberg's, but are considered more difficult to solve [149]. 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 5.2, 5.3 and 5.4. 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 5.2: 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

5.3.2 Diversity

In this section a comparison of the amount of diversity for PBIL and the OPBIL algorithms is given. Only a single result is shown because the diversity of OPBIL is controlled through equation (5.21) and therefore will always appear similar for different problems. In order to compare the expected behavior of diversity the results from Goldberg's 3-bit deceptive function with 100 dimensions and a sample size of 4 is arbitrarily selected. The results are the average of the best solution found over 30 trials of the algorithms.

Table 5.3: 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 5.4: 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

Figure 5.2 shows the amount of all-pairs-diversity as calculated by equation (5.2). As expected, the hard-OPBIL diversity is greatest. 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 65% 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.

In order to provide insight into the relationship between the diversity and the actual results yielded by each algorithm examine Figure 5.3. 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.

Figure 5.4 shows the results of increasing the number of samples from 4 to 20. Comparing with the results in figure 5.3, 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

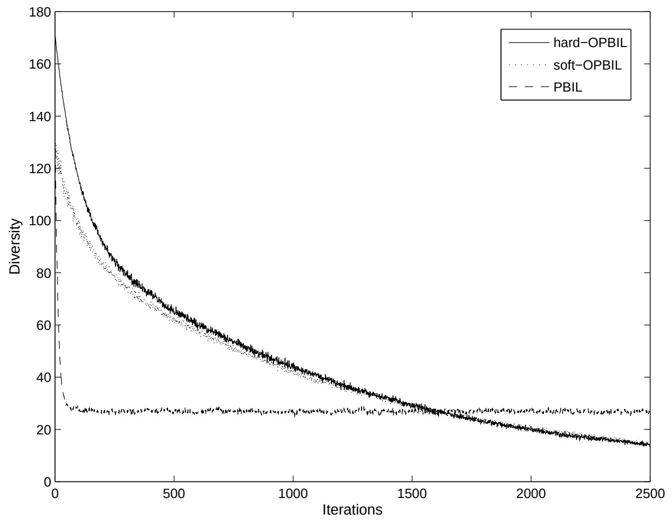

Figure 5.2: All-pairs-diversity for the Goldberg deceptive function with 100 dimensions and 4 samples per iteration. As it can be seen the diversity for both versions of OPBIL maintain a much higher degree of diversity for 1600 of the 2500 iterations.

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.

5.3.3 Parameter Control

In this section the effect of the amplification (ρ) and decay (τ) parameters on the outcome of OPBIL is examined. To accomplish this, fix the dimensionality of the problem to 100 and select Goldberg's 3-bit deceptive problem as a case study. Additionally, only consider the soft-OPBIL algorithm (both hard- and soft-OPBIL behave similarly) using a total of 10 samples at each iteration. 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 5.5 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 it was found that a setting of $\rho = 0.05$ has the tendancy to yield the best final result as larger values

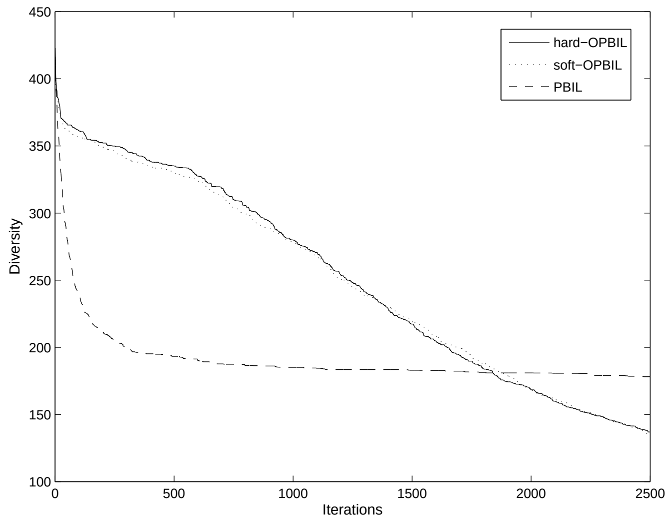

Figure 5.3: The convergence curve of results obtained by each algorithm on Goldberg's function with 100 dimensions and 4 samples.

tended to converge too rapidly (at least for the benchmark functions and TSP data used in this chapter). Nevertheless, these experiments clearly show the effect of ρ on OPBIL.

Figure 5.6 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 it was found that $\tau = 0.0005$ usually yields the most desirable final result.

These results have provided insight into the control of exploration/exploitation ability of the soft-OPBIL algorithm (the hard-OPBIL behaves similarly). 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.

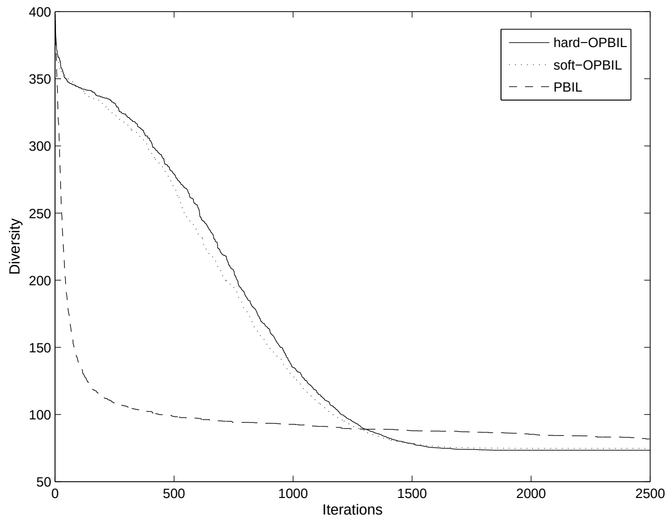

Figure 5.4: The convergence curve of results obtained by each algorithm on Goldberg's function with 100 dimensions and 20 samples.

5.3.4 Accuracy

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

To examine whether the results are statistically significant, a two sample Kolmogorov-Smirnov test at a 0.95 confidence level is employed. In the following tables, bold values represent statistically significant results. If PBIL is bold 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 bold 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 5.5. 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. In this case, increasing the amplification factor of OPBIL could yield better results, however, the parameter values used were selected to typically yield good results. When the dimensionality is increased to 100, the smaller sample size both OPBIL algorithm results are significantly lower than PBIL for a sample size of 4. For sample sizes of 10 and 20, the results favor PBIL, and the

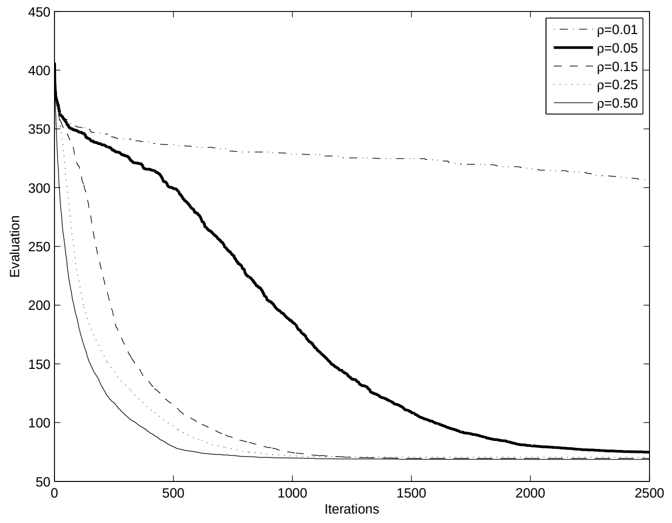

Figure 5.5: The effect of varying ρ on OPBIL results for the Goldberg 3-bit deceptive problem with 100 dimensions. For all experiments $\tau = 0.0005$.

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 5.5 also shows the effect size results as computed by Cohen's d-statistic (actually, it's equivalent nonoverlap value). These values in the final column shows that 1 result shows a "small" and 1 result has a "medium" favoring for PBIL. Two of the values strongly favor PBIL. However, 5/12 results show a large practical improvement for OPBIL. It is important to note that PBIL was compared to the least desirable results found by either hard- or soft OPBIL. Therefore, these are pessimistic interpretations.

Table 5.6 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.

The final column displays the nonoverlap results of Cohen's d-statistic which describes the practical difference between the results of PBIL and OPBIL. For this experiment, for $D = 50$ the improvements favor PBIL. However, for larger $D = 100, 200$ OPBIL's results are much more desirable as their nonoverlap

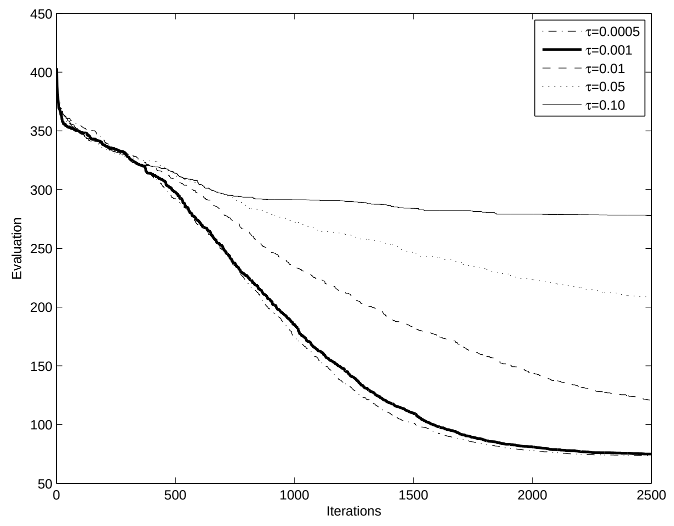

Figure 5.6: The effect of varying τ on OPBIL results for the Goldberg 3-bit deceptive problem with 100 dimensions. For all experiments $\rho = 0.05$.

percentiles are greater than 0.50 for most of these problems.

Table 5.7 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. Also, for most of the results the standard deviation of OPBIL results is relatively low compared to PBIL. The final column of this table presents the effect size, as an overlap percentile. For these problems all 12 results are practically in favor of OPBIL.

Figure 5.7 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 is 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.

The results from the 4-bit deceptive Whitley function are given in Table 5.8. 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

Table 5.5: 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. Bold values are statistically significant. The final column reports Cohen's d-statistic comparing PBIL with the least desirable of hard- and soft-OPBIL.

	PBIL		soft-OPBIL		hard-OPBIL		
S	μ	σ	μ	σ	μ	σ	Effect
Dimensions = 50							
4	*5.867*	8.303	16.333	18.043	24.667	20.594	-0.514,ℓ
10	0.4667	2.556	1.400	4.272	0.4667	2.556	-0.131
20	0.000	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	0.000
Dimensions = 100							
4	215.000	30.885	**141.400**	54.773	**125.733**	39.799	0.638,ℓ
10	*15.667*	8.535	24.733	19.691	18.667	16.981	-0.286,s
20	**0.4667**	2.556	9.333	11.819	10.733	11.441	-0.526,ℓ
30	0.000	0.000	2.800	8.544	5.600	10.136	-0.364,m
Dimensions = 200							
4	1012.867	65.457	**575.333**	105.790	**541.933**	100.974	0.928,ℓ
10	463.133	45.682	**145.533**	47.322	**132.067**	40.222	0.960,ℓ
20	223.067	31.054	**78.133**	34.329	**90.333**	35.785	0.893,ℓ
30	136.333	24.495	**52.733**	27.435	**52.667**	33.800	0.849,ℓ

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. The effect size calculations reveal that all values are practically favorable for OPBIL.

5.3.5 Summary of Results

Firstly, a representative example of the increased diversity induced by opposition into the OPBIL algorithms by comparing the all-pairs-diversity was provided. It was shown 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 result obtained at each iteration it was observed that as the sample size increased, OPBIL was typically able to make a more effective use of the diversity.

Also, a comparison of the results obtained with the hard and soft versions of the OPBIL algorithm to those discovered by PBIL was conducted. To summarize the above results:

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

Table 5.6: 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. Bold values are statistically significant. The final column reports Cohen's *d*-statistic comparing PBIL with the least desirable of hard- and soft-OPBIL.

	PBIL		soft-OPBIL		hard-OPBIL		
S	μ	σ	μ	σ	μ	σ	Effect
			Dimensions = 50				
4	53.433	4.470	56.833	7.715	56.933	7.172	-0.281,s
10	**31.633**	3.864	36.967	2.942	38.267	3.051	-0.690,ℓ
20	**31.033**	3.783	37.233	3.170	39.167	2.276	-0.793,ℓ
30	**31.667**	3.220	36.900	2.496	36.433	3.014	-0.673,ℓ
			Dimensions = 100				
4	178.133	8.307	**135.233**	14.301	**136.933**	7.983	0.923,ℓ
10	111.933	5.065	**72.667**	4.551	**72.833**	4.308	0.972,ℓ
20	81.800	5.410	**74.767**	4.329	**73.400**	4.182	0.583,ℓ
30	74.167	5.867	73.533	3.181	73.500	4.329	0.065
			Dimensions = 200				
4	490.633	11.775	**354.167**	23.343	**351.567**	24.605	0.965,ℓ
10	370.600	10.088	**164.733**	10.116	**167.467**	7.505	0.996,ℓ
20	301.900	7.397	**145.300**	6.889	**147.233**	6.511	0.996,ℓ
30	268.567	7.789	**145.600**	6.273	**145.700**	6.993	0.993,ℓ

- 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 OPBIL 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 calculate the average percentage improvement as

$$P_{i,j} = \frac{PBIL_{i,j} - OPBIL_{i,j}}{\max(PBIL_{i,j}, OPBIL_{i,j})} \tag{5.23}$$

where a substitution of 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 ($S = 4, 10, 20, 30$). 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, 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 5.9 shows the average improvement of results using soft-OPBIL versus PBIL over all four test problems (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

Table 5.7: Whitley's 4-bit attractor results. In total 12/12 results are statistically favorable for OPBIL. Bold values are significant.The final column reports Cohen's d-statistic comparing PBIL with the least desirable of hard- and soft-OPBIL.

	PBIL		soft-OPBIL		hard-OPBIL		
S	μ	σ	μ	σ	μ	σ	Effect
Dimensions = 100							
4	195.533	14.908	**132.000**	14.574	**129.533**	13.903	0.907,ℓ
10	113.133	7.080	**94.600**	2.358	**96.067**	1.929	0.854,ℓ
20	90.733	3.657	**86.133**	1.961	**85.800**	3.377	0.617,ℓ
30	85.400	4.182	**75.267**	2.651	**76.333**	3.241	0.771,ℓ
Dimensions = 100							
4	621.733	29.812	**347.933**	25.685	**365.800**	34.971	0.969,ℓ
10	411.000	14.429	**188.333**	4.901	**188.600**	4.461	0.995,ℓ
20	305.800	10.121	**187.200**	4.859	**188.467**	4.946	0.991,ℓ
30	263.867	10.372	**187.933**	4.5024	**188.533**	4.361	0.978,ℓ
Dimensions = 200							
4	1684.533	29.905	**966.333**	62.288	**997.667**	56.256	0.992,ℓ
10	1288.200	31.759	**418.133**	14.799	**418.800**	11.775	0.998,ℓ
20	1069.800	24.849	**366.667**	8.294	**368.333**	6.604	0.999,ℓ
30	955.733	23.144	**367.200**	7.155	**368.000**	8.485	0.998,ℓ

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.

It was 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 the PBIL algorithm.

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.

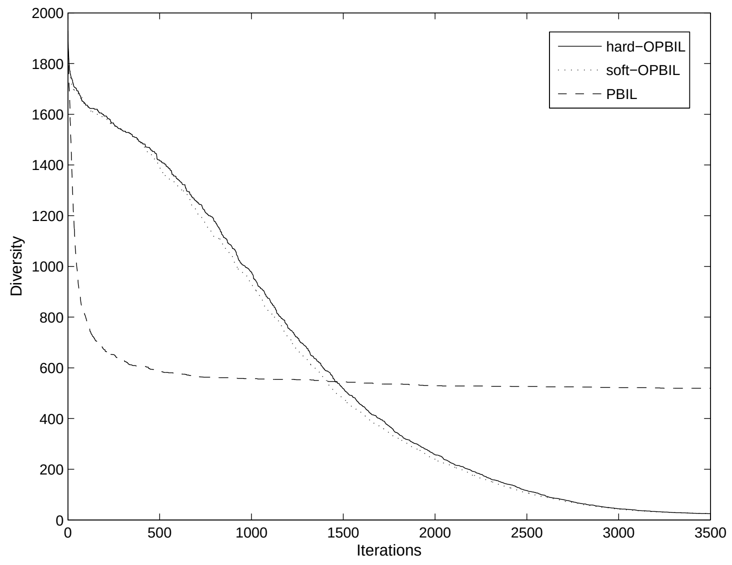

Figure 5.7: A comparison of the convergence of PBIL and the two OPBIL algorithms on Whitley's 4-bit attractor problem where $D = 200$ and $S = 20$.

5.4 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 [129] to further examine the ability of OPBIL. Its performance is tested 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, a binary representation is adopted [7] 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 [7] this representation is not ideal, and many others are in existence which have led to better results.

Table 5.8: Whitley's 4-bit deceptive results. In total 11/12 results obtained with OPBIL are statistically significant and 1/12 are insignificant. Bold values are statistically significant. The final column reports Cohen's d-statistic comparing PBIL with the least desirable of hard- and soft-OPBIL.

	PBIL		soft-OPBIL		hard-OPBIL		
S	μ	σ	μ	σ	μ	σ	Effect
			Dimensions = 50				
4	77.000	8.469	**47.667**	14.003	**56.500**	19.920	0.556,ℓ
10	28.500	6.585	**0.333**	1.826	**0.000**	0.000	0.946,ℓ
20	6.167	3.395	**0.333**	1.269	**0.000**	0.000	0.751,ℓ
30	0.333	1.269	0.000	0.000	0.000	0.000	0.182,s
			Dimensions = 100				
4	293.333	21.600	**172.233**	28.333	**188.633**	27.521	0.904,ℓ
10	173.667	14.478	**11.833**	9.513	**12.00**	12.839	0.986,ℓ
20	118.833	11.423	**2.333**	4.866	**1.500**	3.511	0.988,ℓ
30	93.500	7.673	**1.500**	3.511	**1.000**	2.842	0.992,ℓ
			Dimensions = 200				
4	929.267	31.228	**578.700**	50.813	**608.000**	48.984	0.969,ℓ
10	660.033	29.100	**131.167**	36.547	**131.333**	25.049	0.995,ℓ
20	519.600	22.443	**25.833**	16.033	**24.667**	17.117	0.997,ℓ
30	459.200	22.895	**12.000**	9.966	**16.333**	15.025	0.996,ℓ

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, also included are results obtained by a standard genetic algorithm (GA). During the reproduction phase use an n-point crossover [71] with probability 0.75. Alternatively, with probability 0.25 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 [71]. The parameter settings for OPBIL are the same as above ($\tau = 0.0005$, $\rho = 0.05$), $\xi(t)$ is as shown in Equation (5.21), and the remainder of the OPBIL algorithm is the same as listed in Algorithm 9. For PBIL, 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 ($|V|$).

The same method of generating opposites for the benchmark problems was also adopted here. Unfortunately, this definition of opposite does not allow for an easy interpretation when considering the graph structure or path being acted upon. Generally, the concept of an "opposite graph" or "opposite path" does not seem to have an intuitive definition.

Table 5.10 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, it was found that the average of all final results (μ) are statistically in favor of OPBIL. Also, for all experiments

Table 5.9: 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

the standard deviation σ was lower for OPBIL than for PBIL or the GA, implying more reliable results. Cohen's d-statistic values show that the effect size for all problems is practically favorable for OPBIL. The comparison is between OPBIL and PBIL, the GA results were not considered (although OPBIL is also favorable in comparison).

In Figure 5.8 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 chapter, and allows for improved exploration during early stages of the algorithm.

5.5 Image Thresholding

The image thresholding example used in Chapter 4 is also used here to compare PBIL and OPBIL. As a summary, the purpose of image thresholding is to classify each pixel in an $M \times N$ image I into one of two classes $\{important, unimportant\}$ signified by a binary output image T. Each class is defined as being above or below a specific threshold value ω, respectively. The variety of image types has led to the design of numerous threshold quality metrics. This chapter utilizes a simple measure which aims to minimize the

Table 5.10: Results for the 10 TSP problem instances. Bold values are statistically significant. The final column reports Cohen's *d*-statistic comparing PBIL with the least desirable of hard- and soft-OPBIL.

	PBIL		GA		OPBIL		
Instance	μ	σ	μ	σ	μ	σ	Effect
eil51	590.17	30.60	687.93	40.87	**546.03**	20.94	0.644,ℓ
berlin52	10676.56	812.60	12294.14	859.18	**9792.80**	548.73	0.537,ℓ
eil76	846.94	57.71	986.99	74.75	**754.23**	44.49	0.669,ℓ
kroA100	45871.62	3769.32	55285.64	5290.82	**38726.17**	2880.93	0.729,ℓ
kroB100	45775.77	4352.58	56804.89	5324.88	**38300.59**	2782.29	0.715,ℓ
kroC100	45758.08	3216.85	55560.71	4637.72	**37351.60**	2653.49	0.819,ℓ
kroD100	44964.25	3324.76	56120.53	4833.36	**37229.58**	2327.22	0.803,ℓ
kroE100	46057.64	3396.09	55657.73	4583.16	**38876.37**	2637.26	0.763,ℓ
eil101	1112.97	60.10	1323.52	77.86	**960.99**	59.91	0.785,ℓ
ch130	14400.64	933.67	16566.87	1573.22	**11638.47**	922.89	0.830,ℓ

discrepancy between the original gray-level image I and its thresholded image T [147]:

$$\sum_{i=1}^{M}\sum_{j=1}^{N} |I_{i,j} - T_{i,j}|. \tag{5.24}$$

Each evaluation function will change the outcome of the algorithm. Table 5.11 presents the value-to-reach targets for each of the 6 test images (see Figure 4.4 of Chapter 4 for the corresponding image).

Table 5.11: Value-to-reach (VTR) for comparing PBIL and OPBIL.

Image	Value-to-reach
1	19850
2	4925
3	7175
4	19850
5	19700
6	22700

5.5.1 Parameter Settings

As stated above, OPBIL requires a binary solution representation. However, thresholding aims to discover an integer value $0 \leq T \leq 255$, to perform the segmentation operation of $I > T$. Additionally, use an approach of splitting I into subimages $I_{1,\ldots,16}$ where each I_i is an equal sized square region of the original image.

Encoding was determined to be a matrix $\mathbf{R} := (r_{i,j})_{16\times 8}$ which corresponds to 16 subimages having a gray-level value $< 2^8 = 256$. Each row of $\mathbf{R}$ is converted to an integer which is used to segment the

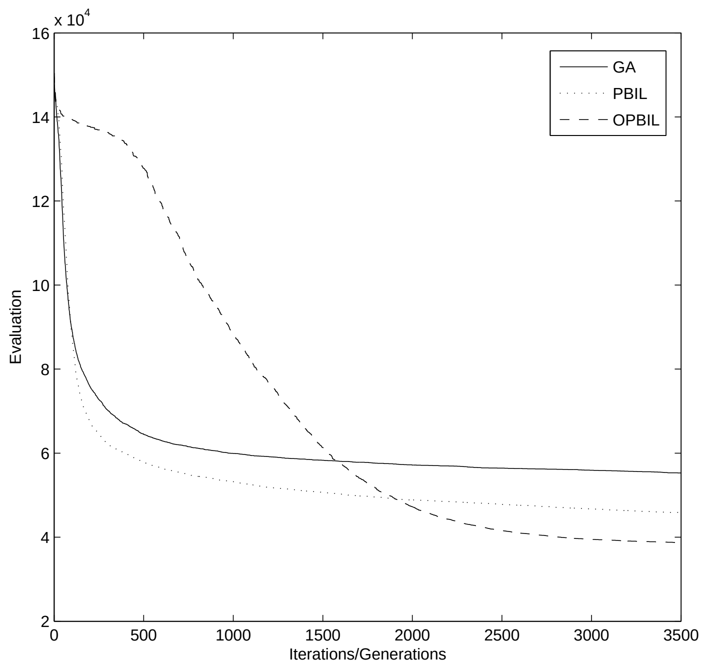

Figure 5.8: Comparing OPBIL vs. GA vs. PBIL for the kroA100 TSP instance.

respective region of I. The extra regions increase problem difficulty as they result in more deceptive and multimodal problems. Parameter settings for PBIL and OPBIL are as shown in Table 5.12.

5.5.2 Results

Table 5.13 shows the expected number of iterations (each iteration has 24 function calls) to attain the value-to-reach given in Table 5.11. In all cases OPBIL reaches its goal in fewer iterations that PBIL, where results for images 2,5,6 are found to be statistically significant using a t-test at a 0.95 confidence level. Additionally, in all cases a lower standard deviation indicating a more reliable behavior for OPBIL is found.

Overall, 446-347=97 saved iterations using OPBIL, which is an average of 16*24=384 function calls per image. The approximate savings is $446/347 \approx 1.28$ which is about a 28% improvement in required iterations. Interestingly, this is similar to the 1.28 improvement factor found for the opposition-based simulated annealing approach described in Chapter 4. The effect size values show that Images 1 and 4 have a practical improvement, Image 2 has a medium increase and Images 5 and 6 show very large improvements.

Table 5.12: Parameter settings for image thresholding experiments.

Parameter	Value
Maximum iterations	$t = 150$
Sample size	$k = 24$
PBIL Only	
Learning rate	$\alpha = 0.35$
Mutation probability	$\beta = 0.15$
Mutation degree	$\gamma = 0.25$
OPBIL Only	
Update frequency control	$b = 0.1$
Learning rate	$\rho = 0.25$
Probability decay	$\tau = 0.0005$

Table 5.13: Summary results for PBIL vs. OPBIL with respect to required iterations calls. μ and σ correspond to the mean and standard deviation of the subscripted algorithm, respectively. The final column reports Cohen's d-statistic comparing PBIL with OPBIL.

Image	μ_{PBIL}	σ_{PBIL}	μ_{OPBIL}	σ_{OPBIL}	Improvement	Effect
1	62	19	53	12	9	0.272, s
2	80	25	**65**	9	15	0.370,m
3	61	12	60	5	1	0.054
4	47	14	40	10	7	0.276,s
5	68	13	**53**	9	15	0.557,ℓ
6	128	21	**76**	14	52	0.824,ℓ

5.6 Summary

In this chapter the use of opposition-based computing concepts to improve population-based incremental learning was proposed. The method is based on increasing diversity in the generated samples. Theoretical evidence to support the claim of increased diversity when considering opposition as opposed to independent random sampling was provided. 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, it was found that there was generally no statistical difference between their results.

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, it was observed that $35/48 = 72.9\%$ of the results from OPBIL were statistically significant and only $6/48 = 12.5\%$ results from PBIL were favored. Moreover, results obtained using OPBIL for problems of high dimensionality were found to be far superior to PBIL. It was shown 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 in those instances.

A comparison of the two approaches on 10 TSP instances composed of between 50 and 130 cities. In all situations OPBIL was able to achieve a statistically significant result over PBIL and a GA. A simple and not very robust, yet common, problem representation and no heuristic information was used. Incorporating more advanced options in these two areas can greatly improve the results. However, for the purpose of this chapter the comparison between OPBIL, a GA and PBIL does not require such additions. Moreover, an increased performance with respect to the required number of iterations to achieve a target goal was observed for benchmark image thresholding problems.

One generally observed phenomenon was that the benefit of using the opposition-based algorithm was more drastic as the problem dimensionality increased. This is very important because small problems can likely be solved exactly using standard methods, which fail for very large problems.

Chapter 6

Gradient-Based Learning

This chapter proposes a new learning heuristic for improving gradient-based learning methods in feedforward neural networks. The essential process involves an efficient strategy for adaptive transfer functions which allows for jumps along the error surface. The new network will represent a more desirable input/output mapping as well as a possible improvement in numerical conditioning. The proposed method is robust, in that it can easily be adjusted for various learning algorithms. In order to show the benefits of this approach, variants of backpropagation and resilient propagation are modified. Experimental results focus on numerical conditioning, accuracy, generalization ability and convergence rate for a variety of benchmark problems. Theory and results of this chapter are mainly found in [134, 135, 138, 139, 142].

6.1 Background

Artificial neural networks have proven to be very useful for a wide variety of tasks. However, for complex or high-dimensional problems the traditional first-order or gradient-based training method of backpropagation [12, 103] often encounters difficulties. In these cases, the associated error surface is complex and may include numerous local optima or even relatively flat areas [36]. By preprocessing the data via a whitening or similar transformation on the data [46], jittering [10] and/or regularization [24, 77, 110] it is possible to alleviate some difficulties. Alternatively, other first-order methods have been proposed, including resilient-propagation (rProp) [94], quick propagation [32] and a variety of conjugate gradient techniques [18, 72] which typically outperform backpropagation.

The rate of error surface curvature can be deduced by computing the Hessian[1] matrix. By incorporating this information, second-order learning algorithms such as Levenberg-Marquardt [46] are often capable of yielding superior results. Unfortunately, the associated computational cost is very high, thus making

[1]The matrix of second derivatives of the error function w.r.t. network weights.

second-order methods infeasible for large networks. Hence, the focus here is on improving first-order learning algorithms, using backpropagation as the benchmark.

6.1.1 Notations and Definitions

Consider an l-layer feedforward network $\mathcal{N}^l$ with a set

$$W = \{\mathbf{V}^1, \ldots, \mathbf{V}^{l-1}\} \tag{6.1}$$

of weight matrices $\mathbf{V}^i$ of weights (and biases) from layer $i-1$ to layer i.

Let the m, d-dimsensional input data be $\mathbf{X} := (x_{i,k})_{m \times d} \in \mathbb{R}$ with associated r-dimensional target vector $\mathbf{T} := (t_{i,j})_{m \times r} \in \mathbb{R}$. Thus, the network is presented the pair (x_i, t_i), corresponding to the i^{th} respective row. Output of the network is given by $\mathbf{Y} := (y_{i,j})_{m \times r}$ where the error at each j^{th} output layer neuron (which has $size(V^m) = r$, where $size(\cdot)$ returns the number of nodes in the given layer) is given by

$$e_j = t_{i,j} - y_{i,j} \tag{6.2}$$

and total network error is given by the mean squared error (MSE)

$$Er(\mathbf{X}) = \frac{1}{m} \sum_{x \in \mathbf{X}} \sum_{j=1}^{r} e_j^2. \tag{6.3}$$

Each hidden neuron employs a specific transfer function ϕ. The transfer function of each neuron in layer $\mathbf{V}^i$ is denoted by $\Phi_i = \langle \phi_j \rangle$ for $j = 1, \ldots, size(\mathbf{V}^i)$ and the set $\Theta = \{\Phi_1, \ldots, \Phi_{l-1}\}$ represents the transfer functions of the entire network. To show the last epoch which a transfer function was changed to/from its opposite $\mathbf{L}$ is used, which has all elements positive, and the set of all values for the network is denoted

$$E = \{\mathbf{L}_1, \ldots, \mathbf{L}_{l-1}\}. \tag{6.4}$$

It will also prove useful to define the average input at the g^{th} epoch for each i^{th} neuron of layer l as

$$\bar{I}^g_{l,i} = \frac{1}{m} \sum_{m} \sum_{j=1}^{|V^{l-1}|-1} v^{l-1}_{j,i} \phi_{l-1,j}(I_{l-1,j}) \tag{6.5}$$

over all m input patterns. Similarly, it will be required to denote the average output of the neuron,

$$\bar{O}^g_{l,i} = \frac{1}{m} \sum_{m} \phi(I_{l,i}). \tag{6.6}$$

6.1.2 Symmetric Structural Transformations

A neural network represents a mapping $\Psi : \mathbb{R}^d \mapsto \mathbb{R}^r$. Symmetry in neural networks typically refers to a transformation T in the network structure or free parameters which does not affect the input-output mapping. That is, Ψ is invariant to the transformation such that $T(\Psi) = \Psi$. Two networks $\mathcal{N}_1$ and $\mathcal{N}_2$ representing the same mapping are denoted as $\mathcal{N}_1 \sim \mathcal{N}_2$ ($\nsim$ is used for different mappings). This work concentrates on structural symmetry as it is concerned with transfer functions, but a more thorough examination can be found in [8].

Structural symmetries can arise as a result of permuting neurons within a hidden layer or by a sign inversion of a transfer function. Permutation is possible by exchanging all the input and output connections within a set of neurons from a specific layer. This permutation transformation does not affect the output of the network, and thus is a symmetric transformation. Given n neurons, a total of $n!$ permutations are possible [20].

The other coherent transformation operates directly on the transfer function and is known as a sign transformation. Given some transfer function having odd symmetry (i.e. $\phi(x) = -\phi(-x)$), multiplying all input and output weights by -1 will result in an invariant input/output mapping [20]. It has been shown that this specific symmetry is valid for any infinitely differentiable function where each successively differentiated function evaluates to zero [3].

On the other hand, if the transfer function exhibits even symmetry (i.e. $\phi(x) = \phi(-x)$) then multiplying all input connections by -1 also leaves $\Psi(\mathcal{N})$ unchanged. This symmetry is also valid for an infinitely differentiable function [3], of which the most common is the radial-basis transfer function. For either even or odd transfer functions, given a layer of n non-input neurons there exists 2^n possible sign-symmetric transformations. The set of all equi-output transformations on the weight space $\mathcal{W}$ forms a non-Abelian group G of order $\#G$, where

$$\#G = \prod_{l=2}^{L-1} (m_l!)(2^{m_l}) \tag{6.7}$$

where L is the number of non-input layers and m_l is the number of neurons in layer l [20].

Each of these strutural transformations defines a symmetry in weight space consisting of equivalent parts. By taking these symmetries into consideration it is possible to reduce the size of the weight space [3, 8, 19, 20, 52, 121]. The underlying idea behind this search space compression has been discussed in Chapters 2 and 3.

6.1.3 Adaptive Transfer Functions

Although in general, the Universal Approximation Theorem [35] does not favor a particular transfer function, for a given data set the choice can be very influential on training time and accuracy. The output range, shape and rate of curvature of the function all play an important role during learning. Furthermore, different function combinations may fit the target function more robustly. Two main methods for employing different transfer functions exist. The first option is to determine the non-standard transfer functions a priori, however the manner in which this decision is made is not obvious [30, 31]. Second, is to choose functions with adaptable parameters which affect the slope, gain, etc. The tradeoff is an increased search space.

The use of adaptive transfer functions has been previously investigated. In [15] it was hypothesized that for a particular data set there may be a preferred transfer function. Unfortunately, the task of determining the optimal function is very complex, especially for large real world data. Hence, they proposed that adaptive transfer functions provide flexibility and may be better suited for the given problem. Similar results were found by [152] and [49] who additionally observed a decrease in the number of hidden neurons required to adequately perform the given task.

The case of adaptive sigmoid functions was investigated in [16] where they were shown to outperform traditional backpropagation by up to an order of magnitude. A gain parameter was proposed in [58] and the resulting algorithm yielded an increase in network performance. Fixed sigmoid functions were also outperformed w.r.t. convergence rate and accuracy by adaptive functions in [122].

In [21], a comparison between single hidden layer networks with and without adaptive transfer functions for static and dynamic patterns was performed. By utilizing adaptive transfer functions, the networks were able to model the data more accurately. Improved generalization and robustness were also observed using adaptive functions in multiresolution learning [63]. An increase in convergence rate was also observed in [128], where they allow for the learning of transfer function amplitude. By finding better areas within the search space (and so error surface) a better generalization ability was also shown. An extension of this work to real-time recurrent learning exhibited similar improvements [38].

Vecci *et al.* investigate a special class of neural networks which are based on cubic splines [132]. The method adapts the control points of a Catmull-Rom spline. It is shown that this model has several advantages including lower training times and reduced complexity. Higher-order adaptive transfer functions were considered in [151]. They reported an increased flexibility, reduced network size, faster learning and lower approximation errors.

Hoffmann [47] proposes universal basis functions (UBFs) which are parameterized to allow for a smooth transition between different functional forms. The UBFs can be utilized for both bounded and unbounded subspaces. Hoffmann's experimental results show an improvement over radial basis functions for some common benchmark data sets.

These works show that adaptive transfer functions can lead to improvements in accuracy, robustness and generalization of neural networks. Adjusting the parameters of the neural network structure represents a structural transformation. It is possible that some transformations do not alter the network input/output mapping, making them irrelevant.

6.2 Opposite Transfer Functions

The proposed algorithm is based on the concept of an opposite transfer function (OTF). These functions have the potential to rapidly change the input-output (IO) representation of the network with minimal computational effort.

Definition 20 (Opposite Transfer Function) *Given some odd-symmetric transfer function* $\phi\colon \mathbb{R} \rightarrow [\alpha, \beta]$*, for* $(\alpha < \beta | \alpha, \beta \in \mathbb{R})$*, the corresponding opposite transfer function is* $\breve{\phi}(x) = \phi(-x)$*. The breve notation indicates the opposite state of* ϕ*.*

Consider a network employing a single neuron in the hidden layer which uses a sigmoid-shaped transfer function, such as the hyperbolic tangent. For large enough input I the gradient of $\phi(I)$ is considered diminished, which has the consequence of slow weight update and thus elongated convergence times or lower accuracy. The latter occurs when the learning algorithm terminates before the weights have been adjusted into regions of higher curvature and therefore faster weight update to more desirable locations on the error surface.

Consider the situation where the target weights yielding optimal results exist on the opposite side of $\phi(0)$ and I is such that the neuron is at or near saturation. In this case, a gradient-based technique could require a significant amount of iterations before it enters a high gradient area, thus allowing it to quickly update its weights. Likely this will occur during the onset of training. Moreover, if the network is trapped at a local optima this strategy could additionally aid in tunneling towards a region of more desirable optima.

In order to be useful in backpropagation-like learning algorithms the following characteristics of an OTF should be present:

1. Both $\varphi(x)$ and $\breve{\varphi}(x)$ are continuous and differentiable.
2. For derivatives, $\frac{d\breve{\varphi}(x)}{dx} = -\frac{d\varphi(x)}{dx}$.

Since most applications require more than a single hidden neuron, OTFs must be incorporated into the network architecture. This introduces the combinatorial problem of ascertaining the best network at each epoch amongst the many options.

Definition 21 (Opposite Network) *Given some minimal feedforward network* $\mathcal{N}$*, the corresponding set of opposite network(s)* $\Gamma(\mathcal{N})$ *is defined as the associated set of all permutations of network function states. In practice, an opposite network is any of* $\breve{\mathcal{N}} \in \Gamma(\mathcal{N}) \setminus \{\mathcal{N}\}$*.*

Assume an arbitrary initial weight configuration $\mathbf{W}^*$ for network $\mathcal{N}$ which has a single hidden layer with transfer function set Φ. If $\mathcal{N}$ is minimal then there exist $2^n - 1$ possible opposite networks, where n is the number of neurons of $\mathcal{N}$ which can exist in an opposite state. In general, the number of possible opposite alternative networks for L-layered architectures can be calculated as

$$\prod_{l=1}^{L-1} 2^{size(\Phi_l)} - 1. \tag{6.8}$$

It is assumed that on average it is expected that a randomly generated network will yield the average expected error of the evaluation function,

$$\mathbb{E}[Er_{\mathcal{N}}] \approx \mathbb{E}[Er]. \tag{6.9}$$

As $\mathbf{W}^*$ is updated by a given learning algorithm, the error also decreases, hence it becomes increasingly unlikely that an opposite network yields a lower error than the currently best known network, i.e.

$$\lim_{g\to\infty} Pr^g \left(\min\{Er_\Phi | \Phi \in \Theta, \Phi \neq \Phi_{\mathcal{N}}\} < Er_{\mathcal{N}}\right) \to 0 \tag{6.10}$$

where $g > 0$ is the training epoch and Er_Φ is the evaluation under consideration of transfer function set Φ. As a corollary, the rate opposite transfer functions will remain useful diminishes with respect to the quality of learning algorithm or initial conditioning of the network. That is, the rate at which (6.10) converges will differ for each learning algorithm as well as the initial random weight vector. This issue makes effectively employing OTFs a much more difficult task compared to random sampling-based methods, such as those described in Chapters 4 and 5.

6.2.1 Network Irreducibility

Aside from symmetrical transformations, it is possible for $\mathcal{N}_1 \sim \mathcal{N}_2$ if one network can be reduced to the other [121]. For example, if there exists some neuron $\eta_a \in \mathcal{N}$ which has all outgoing weights equal to zero. Then, the removal of η_a does not affect Ψ. A formal definition of minimality (or equivalently irreducibility) has been given in [121]:

Definition 22 (Irreducibility) *A feedforward neural network with d input nodes and one hidden layer of n hidden neurons can be called irreducible if none of the following is true:*

1. *One of the $v_{j,k} \in \mathbf{V}^2$ vanishes.*
2. *There exists two indices $j_1, j_2 \in \{1, ..., n\}$ where $j_1 \neq j_2$ such that the functionals Ψ_{j_1}, Ψ_{j_2} are sign-equivalent*[2]*.*
3. *One of the functionals Ψ_j is constant.*

An important consequence of minimality is that every minimal network represents a unique input-output mapping [3, 121].

[2]Two functions $f_1(x), f_2(x)$ are sign-equivalent if $f_1(x) = f_2(x)$ or $f_1(x) = -f_2(x)$ $\forall x \in \mathbb{R}^d$ where d is the dimensionality of the space.

It can now proven that each $\gamma \in \Gamma(\mathcal{N})$ represents a unique mapping if $\mathcal{N}$ is irreducible. The theorem trivially follows from the following three basic lemmas.

Lemma 2 *A vanishing $v_{j,k} \in \mathbf{V}^j$ only exists in some $\gamma_g \in \Gamma(\mathcal{N})$ if and only if it exists in $\mathcal{N}$.*

Proof Given that $\breve{\phi}(I_{j,k}) = \phi(-I_{j,k})$ is a symmetric transformation on $I_{j,k}$ then each $v_{j,k} = -v_{j,k}$ for the relevant neurons in γ_g. It follows trivially that the only instance where $-v_{j,k}$ vanishes is if $v_{j,k} = 0$.

■

Lemma 3 *Sign-equivalency can only exist in some $\gamma_g \in \Gamma(\mathcal{N})$ if and only if exists in $\mathcal{N}$.*

Proof Following a similar argument as in Lemma 2, an OTF maps $I_{j,k} = -I_{j,k}$. Sign-equivalency exists if $|\phi_{j_1}(I_{j_1,k})| = |\phi_{j_2}(I_{j_2,k})|$. Substituting $|-\phi_{j_1}(I_{j_1,k})| = |\phi_{j_2}(I_{j_2,k})|$. However, this can only occur if ϕ_{j_1} and ϕ_{j_2} were already sign-equivalent.

■

Lemma 4 *For $\gamma_g \in \Gamma(\mathcal{N})$, the associated $I_g =$ constant if and only if it is constant in $\mathcal{N}$.*

Proof This follows directly from Definition 20, which states that $I_g = -I_g$.

■

Theorem 6.2.1 (Opposite Network Irreducibility) *Given an irreducible network $\mathcal{N}$ all $\gamma_g \in \Gamma(\mathcal{N})$ are also minimal.*

Proof If every γ_g is minimal then it must obey the constraints outlined in Definition 22. Each of the requirements is proven in Lemmas 2-4.

■

Let $S = \{\mathcal{N}\} \cup \Gamma(\mathcal{N})$, then from Theorem 6.2.1 every $s \in S$ represents a unique input-output mapping, denoted $\Psi(s)$. The following will show that each has the possibility of being the most desirable (i.e. have the lowest error) given a random starting position on the error surface.

Theorem 6.2.2 (Equi-Probable Input-Output Mapping) *Let $\mathcal{N}$ be a minimal neural network where $\mathbf{W} \in \mathcal{U}(-\alpha, \alpha)$ (α, such that transfer functions avoid saturation) and with opposite networks $S = \{\mathcal{N}\} \cup \Gamma(\mathcal{N})$. Without a-priori knowledge concerning $\mathbf{X}$ and for some $s^* \in S$,*

$$P\left(s^* = \arg\min_{s \in S}(f(s))\right) = \frac{1}{|S|}$$

where,

$$|S| = \prod_{l \in L} 2^{m_l},$$

where L corresponds to the number of layers which can utilize opposite transfer functions, each having m_l neurons.

Proof Two common assumptions are made (the second makes the proof trivial):

1. The size and underlying distribution of $\mathbf{X}$ is unknown but is bounded by known finite bounds, which are scaled to [-1,1].
2. For $\mathcal{N}_1$ and $\mathcal{N}_2$ both minimal, having the same number of input, hidden and output neurons all using the same respective transfer functions, then

$$P(Er_{\mathcal{N}_1}(\mathbf{X}) \leq Er_{\mathcal{N}_2}(\mathbf{X})) = P(Er_{\mathcal{N}_2}(\mathbf{X}) \leq Er_{\mathcal{N}_1}(\mathbf{X})) = 0.5 \tag{6.11}$$

   Section 6.4.2 provides experimental evidence to further support this assumption.

From Theorem 6.2.1 and Definition 22

$$s_a \nsim s_b \; \forall s_a \neq s_b \in S. \tag{6.12}$$

Each transfer function can be in either $\varphi(\cdot)$ or $\breve{\varphi}(\cdot)$ state. The number of combinations is calculated by

$$|S| = \prod_{l \in L} 2_l^k. \tag{6.13}$$

Using Definition 20, each $s \in S$ represents a unique point in weight space such that

$$\mathbf{W^a} \neq \mathbf{W}^b, \forall s_a \neq s_b \in S, \tag{6.14}$$

where the superscript identifies which network the weight matrices belong to. From the assumptions each s is equally likely to yield the lowest error on $\mathbf{X}$ since they each exist within the same weight space. So,

$$P\left(s^* = \arg\min_{s \in S}(f(s))\right) = \frac{1}{|S|} \tag{6.15}$$

■

This result has been experimentally confirmed (see Section 6.4). While this holds for a random location in weight space, the rate at which this probability changes for each network during learning has not be determined analytically. Experimental evidence will be shown to support an exponential increase in probability.

6.2.2 Changes in the Jacobian and Hessian

Numerical condition is a very important and fundamental concept which affects the speed and accuracy of neural network learning algorithms [10]. Essentially, numerical condition refers to the sensitivity of the network output to changes in its weights and biases. If a network is ill-conditioned it may require long training times or converge to a poor solution. Changes in the Jacobian and Hessian matrices will impact the learning convergence rate.

The Jacobian matrix $\mathbf{J}$ is composed of first partial derivatives of the residual error for each pattern with respect to W. For a network with one output neuron $\mathbf{J}$ is computed as

$$\mathbf{J} = \left[\frac{\partial e(\mathbf{X})}{\partial w_i}\right]_{i=0\ldots,|W|}. \tag{6.16}$$

Recall, that each $s \in S$ represents a unique mapping (i.e. $s_1 \nsim s_2$) implying their Jacobian matrices $\mathbf{J}(s)$ are different. So, for

$$\mathbf{\Delta}^J = \mathbf{J}(s_1) - \mathbf{J}(s_2) \tag{6.17}$$

it follows that $|\delta^J_{i,j} \neq 0| \geq 1$ for $\delta^j_{i,j} \in \mathbf{\Delta}^J$. Due to this property rank($\mathbf{\Delta}^J$) > 0, where the rank of an $m \times n$ matrix $\mathbf{A}$ represents the number of linearly independent rows or columns. Rank deficiency occurs when rank($\mathbf{A}$) $< \min(m, n)$.

Rank deficiency of the Jacobian is related to the concept of ill-conditioning [104, 131]. For backpropagation-like algorithms, a rank deficient Jacobian implies that only partial information of possible search directions is known, which can lead to longer training times. Furthermore, many optimization algorithms such as steepest decent, conjugate gradient, Newton, Gauss-Newton, Quasi-Newton and Levenberg-Marquardt directly utilize the Jacobian to determine search direction [46, 104].

The Hessian matrix $\mathbf{H}$ represents the second derivative of $Er(\mathbf{X})$ with respect W,

$$\mathbf{H} = \left[\frac{\partial^2 Er(\mathbf{X})}{\partial w_i w_j}\right]_{i,j=0\ldots,|W|}. \tag{6.18}$$

The Hessian is very important to nonlinear optimization as it reveals the nature of error surface curvature. Specifically, the eigenvalues of $\mathbf{H}$ have a large impact on the learning dynamics of backpropagation-like algorithms. It is also employed by second-order learning algorithms [10], and the inverse $\mathbf{H}^{-1}$ can be used for network pruning strategies [60]. For neural networks using squared-error based functions it is common to compute

$$\mathbf{H} = \mathbf{J}^T\mathbf{J}. \tag{6.19}$$

As shown in Equation 6.17, utilizing some $\gamma_g \in \Gamma(\mathcal{N})$ will result in a change in $\mathbf{J}$. From (6.19) also expect a change in $\mathbf{H}$,

$$\boldsymbol{\Delta}^H = \mathbf{H}(s_1) - \mathbf{H}(s_2) \tag{6.20}$$

where there exists some $\delta^H_{i,j} \in \boldsymbol{\Delta}$ such that $\delta^H_{i,j} \neq 0$. Depending on the number and magnitude of the $\delta^H_{i,j} \neq 0$, the difference between the two positions in weight space could be significant enough to warrant moving the search to that location. This could possibly be used as either a restart method or during learning.

The conditioning of $\mathbf{H}$ has a profound impact on the learning time and accuracy of the training algorithm. The most common method to measure the condition of $\mathbf{H}$ is through the condition number,

$$\kappa = \frac{\lambda_{max}}{\lambda_{min}} \tag{6.21}$$

where λ_{max} and λ_{min} are the largest and smallest nonzero eigenvalues of $\mathbf{H}$, respectively. The larger this ratio, the more ill-conditioned the network is.

6.3 Opposition-Based Gradient Learning

A framework is proposed which is based on opposite transfer functions. Being very general it can be easily adapted to many existing algorithms, although experiments will be required to adjust free parameters.

6.3.1 An Exhaustive Approach

For the sake of completeness, consider an exhaustive enumerative approach to determining the most desirable transfer function configuration in Φ by selecting

$$\Phi_* = \arg\min_{\varphi \in \Phi} Er_\varphi(\mathbf{X}) \tag{6.22}$$

at each epoch. Define a learning trajectory for k iterations as $\pi_* = \langle Er^g_{\Phi_*} \rangle$ for $g = 0, \ldots, k$ (as opposed to the trajectory of only the original configuration $\pi = \langle Er^g_{\mathcal{N}} \rangle$). The following cases for the naive approach to converge to the same value as the parent learning algorithm can be distinguished:

1. There does not exist a transfer function set which yields a lower error than $\mathcal{N}$ at any point during the trajectories. Hence, $\pi = \pi_*$.

2. Let $1 \leq g_1 < g_2$ be epochs. Assume that at g_1 there exists some Φ_* where $Er^{g_1}_{\Phi_*} < Er^{g_1}_{\mathcal{N}}$ and that both solutions reside in the same attractive basin of the error surface. There may also exists some g_2 resulting in $Er^{\geq g_2}_{\Theta_*} - Er^{\geq g_2}_{\mathcal{N}} = 0$. Hence, $\pi(\geq g2) = \pi_*(\geq g2)$.

3. If case (2) repeats but the networks eventually settle in the same basin. That is, there exists some g_i where for $1 \leq j < i$ case (2) repeats but eventually the two solutions arrive at the same error, maintaining it until the termination criteria is arrived.

In the last two situations the convergence time for each algorithm will likely differ. The experimental simulations will show that the probability of convergence to the same error is relatively low.

6.3.2 The Proposed Framework

The number of combinations of transfer function state grows exponentially with a linear increase in the amount of neurons. Therefore in practice, a naive approach of searching all possible transfer function states, as presented in Equation (6.22) becomes infeasible for networks with many hidden neurons. Even for small networks the computational overhead associated with testing each possible network state is likely not worth the effort. However, a possible heuristic is proposed for selecting one of these combinations in a computationally efficient manner, resulting in a feasible learning procedure. The heuristic filters out likely undesirable network states by examining properties of the input-output representation, output gradient and weight update gradient.

Algorithm 10 presents the modified main loop of the new framework. The motivation behind each of lines 5-13 will be described in detail. Line 16 provides the functionality of pattern presentation and weight update with an appropriate learning algorithm. Hence, the method is essentially independent of the learning method chosen.

Equation (6.10) expresses the probabilistic limits on the usefulness of opposite transfer functions, but leaves to interpretation the specific probability decay rate. Based on experiments, it has been chosen to let $f: \mathbb{R} \to \mathbb{R}$ be a decreasing monotonic function such that

$$\lim_{g \to \infty} f(g, \mathbf{\Theta}_g) = 0 \tag{6.23}$$

where $\mathbf{\Theta}_g$ are parameters of f at iteration $g \geq 0$. If vector $P_0 := (0.5)_n$ represents the transition probability of each transfer function at iteration $g = 0$. Then, $P_g = f(g, \mathbf{\Theta}_g)P_{g-1}$ and so

$$\lim_{g \to \infty} f(g, \mathbf{\Theta}_g)P_g = \mathbf{0}. \tag{6.24}$$

Thus, there exists $g > 1, \varepsilon > 0$ such that $|P_g - P_{g-1}| < \varepsilon$. Hence, under a monotonically decreasing probability the system of transfer function transitions is stable and will converge. In practice, set $f(g, \mathbf{\Theta}_g) = 0$

when it falls below a small, predetermined threshold. After convergence of P the learning procedure convergence behavior is governed by the original learning algorithm being employed. This probabilistic aspect is considered in line 5.

Alternating between a transfer function and its opposite is more likely to greatly impact learning when the input signal lies within a large gradient region than at saturation or when close to the transfer function inflection point. This does not imply that improvements cannot be gained from considering the full output region of the function. Relatively flat regions of the error surface have small error gradients and thus weight update procedures using this information will be slow. Hence, the heuristic assumes the situation where saturation is desired. The impact is geared to achieving a more desirable input-output network representation in this case and not necessarily on numerical conditioning. Line 6 determines whether the neuron output is far from $\phi(0)$. This is computed via the difference between the average neuron output $\bar{O}_l$ and the output at zero input, $\phi(0)$, i.e.

$$\mathbf{D} = (|\bar{O}_l - \phi(0)|) > \alpha \tag{6.25}$$

for $\alpha > 0$ is a threshold which determines the minimum output difference point. It is also possible to provide an upper bound on the output difference to avoid considering saturated regions.

In some instances alternating the transfer function state can be in opposition to the direction of weight update. This situation is undesirable as it would impede the convergence rate. To limit this canceling-out effect on weight updates, only consider cases where the result of the alteration is towards the same direction as weight update. In line 7 this is accomplished by first considering the weight update direction,

$$\mathbf{Q} = (V^l - V^{l-1}) < 0. \tag{6.26}$$

The second consideration required is if the altered function $\breve{\phi}$ adheres to the weight update. The opposite transfer function output is computed in line 8,

$$\mathbf{F} = \phi(0) < \breve{\phi}(\bar{I}_l). \tag{6.27}$$

Determining whether Equations (6.26) and (6.27) are indeed not in contradiction is computed in lines 9 and 10 for movement in the negative and positive weight direction, respectively. Since $\mathbf{Q}$ and $\mathbf{F}$ are binary matrices, computing the Hadamard product (denoted via the $\odot$ operation) will result also in a binary matrix whose result depicts the truthfulness of the above heuristics.

To allow an altered transfer function the opportunity to be successfully integrated into the network learning dynamics and a minimum number of epochs must pass before it can serve as a candidate again. In line 11 compute

$$\mathbf{E} = (t - \mathbf{L}_l) > \varepsilon_t, \tag{6.28}$$

where $\varepsilon_t > 0$ is the epoch threshold.

The heuristic is finally culminated in line 12 where updates in the same direction are considered, either positive or negative (via the logical OR operation $\oplus$) in addition to considering **E**. Based on these binary operations, update the input mask accordingly in line 13 which reflects the opposite state.

Algorithm 10 The main algorithm framework for employing OTFs.

```
1:  $g = 0$
2:  while termination criteria not satisfied do
3:      if $g > 1$ then
4:          for $l = 1, \ldots, layers$ do
5:              $\mathbf{U} = f(t, \mathbf{\Theta})$
6:              $\mathbf{D} = (|\bar{O}_l - \phi(0)|) > \alpha$
7:              $\mathbf{Q} = (V^l - V^{l-1}) < 0$
8:              $\mathbf{F} = \phi(0) < \breve{\phi}(\bar{I}_l)$
9:              $\mathbf{C}_1 = \mathbf{F} \odot \mathbf{Q}$
10:             $\mathbf{C}_2 = (1 - \mathbf{F}) \odot (1 - \mathbf{Q})$
11:             $\mathbf{E} = (t - \mathbf{L}_l) > \varepsilon_t$
12:             $\mathbf{C} = (C_1 \oplus C_2) \odot \mathbf{E} \odot \mathbf{U}$
13:             $\mathbf{M}^l = -2\mathbf{C} + 1$
14:         end for
15:     end if
16:     $updateWeights(\mathcal{N}, \mathcal{A})$
17: end while
```

The computational overhead associated with algorithm 10 must be addressed. While many considerations are being investigated in lines 5 - 12 and finally applied in line 13, these can be accomplished in a single iteration over the hidden neurons. In this case, each hidden neuron is checked against each constant time (i.e. $\mathcal{O}(1)$) operation. Therefore, determining which neurons to change transfer function to an opposite is at most an $\mathcal{O}(size(V))$ algorithm, where $size(V)$ returns the total number of hidden nodes. Since line 5 only applies the procedure if it is successful, the computation time will be practically at most linear with respect to the number of hidden neurons. Furthermore, since $f(t, \mathbf{\Theta})$ is a monotonically decreasing function this overhead will be highest during early training and negligible as t increases. Moreover, the number of weights will be larger than the number of hidden neurons and so this cost will be (much) less than is required for a weight update.

6.4 Pre- and Early Training

In this section results for two main experiments are provided. Firstly, the $Er(\mathbf{X})$, rank($\mathbf{J}$) and κ at random points in the error surface for all combinations of transfer functions are analyzed. The second experiment is aimed to examine the changes in the same three measures during early learning of a conjugate gradient algorithm.

6.4.1 Experimental Setup

Unless otherwise noted the following experiments all use a single hidden layer feedforward architecture with hyperbolic tangent transfer functions for the hidden and output neurons, respectively. Additionally, initial weights and biases are uniformly generated over $[-1, 1]$.

To evaluate the networks three common benchmark problems from the UCI-ML database [5] and two versions of the parity problem are used. These data sets, along with the number of hidden layers in the networks are presented in Table 6.1. To avoid errors and bias in Jacobian and Hessian calculations the input data are standardized and normalized. For simplicity, records with missing data (very small percentage of the entire data set) are also removed. Output values are binary and do not require adjustment.

Table 6.1: The benchmark data and the number of hidden layers used.

Dataset	# Patterns	Inputs	Hidden Size	$\|W\|$
3-bit parity	8	3	3	16
6-bit parity	64	6	6	49
Pima diabetes	768	8	5	51
Wisconsin breast cancer[3]	783	9	5	56
ionosphere	351	33	5	176

The first experiment begins by generating a single neural network with random weights. Then, keeping W constant, evaluate $Er(\mathbf{X})$, rank$(\mathbf{J})$ and κ for each $s \in S$. The process repeats for 4000 randomly generated networks.

The second experiment focuses on the early stages of learning. Similar to the first experiment, generate a random network $\mathcal{N}$. However, in this case train the network using Fletcher-Reeves Conjugate Gradient [46] for 10 epochs using scale factors $\alpha = 0.001$ and $\beta = 0.01$. At each epoch record the $Er(\mathbf{X})$, rank$(\mathbf{J})$ and κ for each $s \in S$, although only train with respect to $\mathcal{N}$. In this manner the relationship between opposite networks, and how learning could benefit from considering opposite networks can be shown. The use of a second order training method in this experiment is due to its generally more rapid convergence rate than first order methods, thus achieving a pessimistic estimate for the use of OTFs in learning.

In both experiments, a minimum threshold value is employed when calculating the rank and condition number. By doing this very small values which may skew the calculations are ignored. Only singular values of the Jacobian greater than 0.02, and eigenvalues of the Hessian greater than 0.02 are considered.

6.4.2 Before Training

The first experiment is aimed at comparing the conditioning of every transfer function combination before training begins in order to assess whether each transfer function combination is equally likely to yield the minimum error for a given weight configuration (providing evidence for Theorem 6.2.2). For m hidden nodes there are 2^m combinations represented by an m-bit mask detailing the opposite state of the associated transfer function. For example if $m = 2$, four combinations are possible $C = \{(00), (01), (10), (11)\}$, where a 0 or a 1 indicates whether the opposite transfer function is "off" or "on", respectively.

Figure 6.1 shows the result of random sampling using four of the benchmark data sets. As mentioned above, 4000 random locations in weight space were generated and all 2^m combinations of transfer function were evaluated. After 4000 samples these results provide experimental evidence to support Theorem 6.2.2.

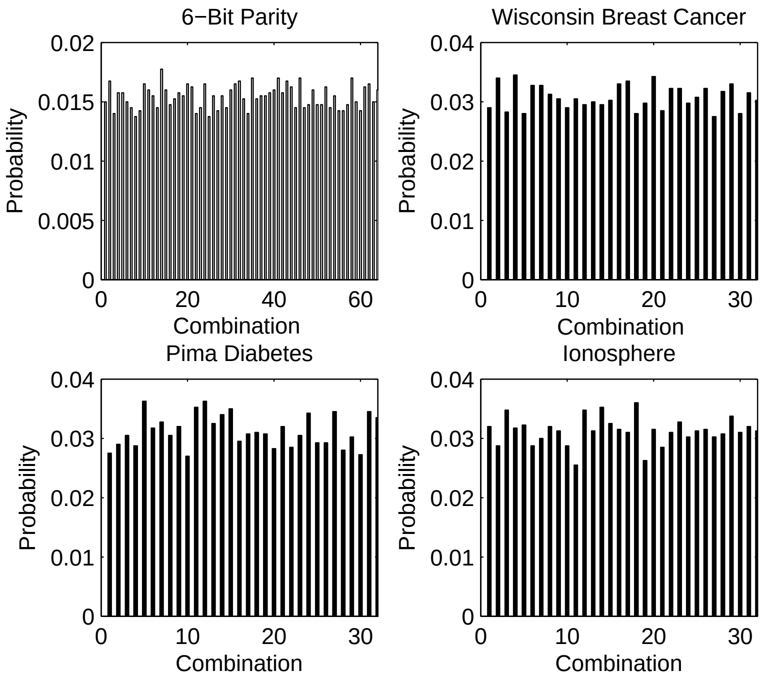

Figure 6.1: Random sampling results for four benchmark problems. The probability of each combination yielding the lowest error is approximately uniform for each problem.

Let $z_{i=1...4000}$ be the i^{th} randomly sampled location of the weight space. Then the sample mean is,

$$\mu = \frac{1}{4000} \sum_{i=1}^{4000} \min(Er_{S_{z_i}}(\mathbf{X})) \tag{6.29}$$

and σ is its standard deviation. Table 6.2 presents μ and σ of these 4000 random samples. Also computed are,

$$min = \min(\min(Er_{S_{z_i}}(\mathbf{X}))) \tag{6.30}$$

and

$$max = \max(\min(Er_{S_{z_i}}(\mathbf{X}))) \tag{6.31}$$

which represent the lower and upper bound defined by the best $s \in S$ at each z_i. Using these measures a better understanding of how the initial Er can vary, versus simply examining the standard deviation. So, considering every $s \in S$ the Er range for each of the five benchmark problems is relatively large compared to $\mu \pm \sigma$. This is very representative of a network that is prone to ill-conditioning.

The final two comparison measures μ_{diff} and σ_{diff} represent the mean and standard deviation of $\max(Er_{S_{z_i}}(\mathbf{X})) - \min(Er_{S_{z_i}}(\mathbf{X}))$. Using these values, and then computing μ_{diff}/μ for each problem leads to the ratios 0.28, 0.26, 2.36, 0.65 and 1.05, respectively, and reveals the spread with respect to the mean. These results imply a large spread in random network performance, and simply examining the opposite networks can lead to greatly improved starting positions (w.r.t. the MSE).

Table 6.2: A comparison of $Er(\mathbf{X})$ for each problem.

Dataset	μ	σ	min	max	μ_{diff}	σ_{diff}	μ_{diff}/μ
3-Bit	0.25	0.02	0.20	0.31	0.07	0.04	0.28
6-Bit	0.27	0.02	0.23	0.36	0.07	0.02	0.26
cancer	0.14	0.04	0.07	0.33	0.33	0.11	2.36
diabetes	0.23	0.03	0.19	0.35	0.15	0.05	0.65
ionosphere	0.21	0.03	0.14	0.34	0.22	0.07	1.05

Using the methodology and measures described above, the rank($\mathbf{J}$) is explored in Table 6.3. Additionally, all five problems have μ within 88.0% of the respective maximum rank, and so $\mathbf{J}$ will tend be rank deficient but not to a high degree. Comparing μ_{diff}/μ for each problem yields 0.03, 0.06, 0.01, 0.01 and 0.04, respectively. Thus, given a location in weight space the difference between the network with the highest and lowest rank($\mathbf{J}$) will be approximately 3% with respect to the mean rank for that specific problem.

Table 6.3: A comparison of rank($\mathbf{J}$) for each problem.

Dataset	μ	σ	min	max	μ_{diff}	σ_{diff}	μ_{diff}/μ
3-Bit	7.6	0.57	4.00	8.00	0.24	0.43	0.03
6-Bit	41.71	2.93	21.00	47.00	2.56	1.23	0.06
cancer	54.25	3.36	34.00	56.00	0.49	1.00	0.01
diabetes	50.04	2.37	33.00	51.00	0.23	0.64	0.01
ionosphere	157.80	13.21	79.00	176.00	6.05	3.64	0.04

Table 6.4 presents the results when comparing the impact of OTFs on the condition κ of $\mathbf{H}$. The key comparison is the ratio μ_{diff}/μ for each of the five problems, resulting in values: 1.24, 0.85, 1.24, 0.97 and

0.88, respectively. These values represent significant differences between the conditions of each $s \in S$ at a given weight configuration and further highlight the impact of OTFs before learning begins.

Table 6.4: A comparison of κ for each problem.

Dataset	μ	σ	min	max	μ_{diff}	σ_{diff}	μ_{diff}/μ
3-Bit	9.0	4.5	2.1	27.5	11.2	7.2	1.24
6-Bit	146.9	21.6	46.3	264.9	125.0	32.8	0.85
cancer	3061.5	1030.1	869.9	8568.1	3798.0	1346.9	1.24
diabetes	1753.1	441.0	527.7	3979.0	1695.6	631.0	0.97
ionosphere	2284.3	687.2	846.6	6047.4	2004.1	978.4	0.88

6.4.3 During Early Training

This subsection provides insight into the impact of a learning algorithm on the usefulness of OTFs during the first 10 epochs of learning. The focus on early learning is based on the fact that the usefulness of OTFs will decay as learning progresses.

In Figure 6.2 the probability a specific opposite network will yield the lowest $Er(\mathbf{X})$ assuming only training occurs on the original network structure is examined. For this comparison, only the 3-bit parity problem is considered because $|S|$ is small and the resulting plot is more readable. After the second epoch, 3 of the 8 networks show a non-zero probability of yielding the lowest error. By the fourth epoch the probability the original network $\mathcal{N}$ has the lowest error is 1.0, keeping in mind the simplicity of the problem. The main purpose of this graph is to show an example of the behavior for each opposite network during training.

Figure 6.3 presents the probability that $\mathcal{N}$ with transfer function combination (00...0) yields the lowest error when compared to $\Gamma(\mathcal{N})$. Except for the 3-bit parity problem, all probabilities are between 0.80 and 0.95 by the tenth epoch, where the behavior of probability increase is similar for each problem. Therefore, considering opposite networks even while training has the potential to yield a lower performance measure. Furthermore, a learning algorithm which selects one of these network at each epoch, and continues training with it could potentially result in a lower final training error and/or a possible increase in convergence rate.

Next, compute the difference $\Delta^{err} = Er_{\mathcal{N}} - \min(Er_{\Gamma(\mathcal{N})})$, and plot the results in Figure 6.4. The 3-bit parity problem shows a substantial difference between $\mathcal{N}$ and its opposite networks and the 6-bit version of the problem shows a smaller, yet increasing difference. However, the remaining three problems show very little difference between the network being trained and its opposites. For these latter three problems an opposite network is likely of about equal quality to the trained network. However, the error surface characteristics (especially local curvature) may be more desirable and thus switching away from the current network to the opposite will be beneficial. An appropriate method must be incorporated into

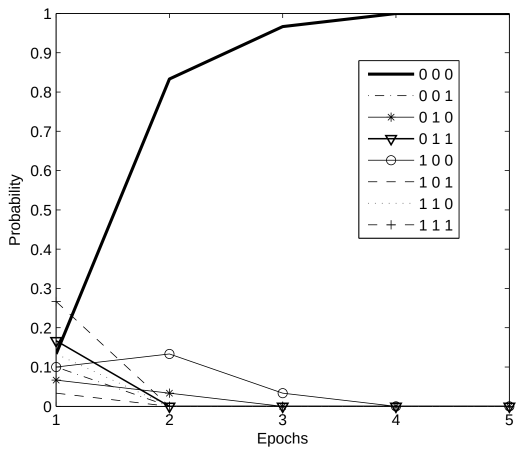

Figure 6.2: A comparison of the probability a transfer function combination will yield the minimum error at the given epoch, if the network is trained only on combination (000) for the 3-bit parity problem.

the framework in order to determine this.

and it may be beneficial to consider switching the network being trained to the relevant opposite.

To examine the effect of training on the difference in rank of $\mathbf{J}$, calculate

$$\Delta^{rank} = \frac{\text{rank}(\mathbf{J}(\mathcal{N})) - \min(\text{rank}(\mathbf{J}(\Gamma(\mathcal{N}))))}{\min(|\mathbf{Z}|, |\mathbf{X}|)} \tag{6.32}$$

where $\min(|\mathbf{Z}|, |\mathbf{X}|)$ represents the maximum[4] possible rank of the Jacobian matrix for each problem, respectively. The results are presented in Figure 6.5. The 6-bit parity and the Wisconsin breast cancer problems show a more rapid increase in the difference between the trained network, and the best opposite network with respect to rank. By the 10th epoch the other three problems show only a 2.0% difference in rank. So, for these latter three problems considering an opposite network (using only a rank criterion) is less likely to show an improvement than the former two problems.

The final experiment will compare the difference in the mean condition κ of $\mathbf{H}$ over the 30 trials. To determine this compute,

$$\Delta^{\kappa} = \kappa(\mathcal{N}) - \min(\kappa(\Gamma(\mathcal{N}))), \tag{6.33}$$

where Figure 6.6 plots these results. The Pima Diabetes and Ionosphere problems show a similar behavior, but the curve for the Pima data has most of its values > 0 which means that the trained network is not the best conditioned network. Since the Ionosphere results are mainly < 0 it is better conditioned than its

[4]See Table 6.1 for the respective values.

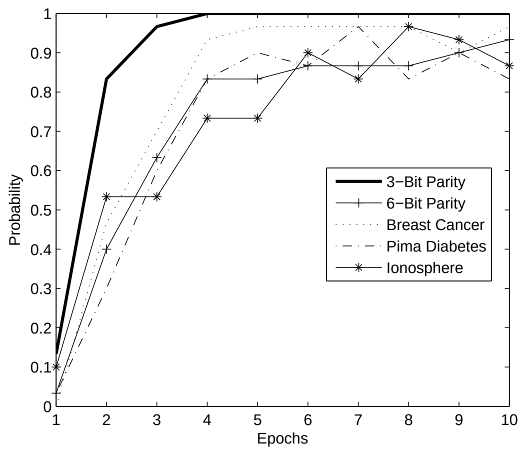

Figure 6.3: Comparing the probability of transfer function combination (00...0) of yielding the lowest error for the five benchmark problems.

opposite networks. The 6-bit parity and Wisconsin breast cancer data are both relatively close to having no difference between their trained and opposite networks, respectively. The 3-bit parity problem shows nearly no change in Δ^{κ}. These results show that even during training there are cases when it may be desirable to consider training an opposite network, especially if the training error shows little improvement.

6.5 Comparing Variants of Backpropagation

There exists various adaptations of the standard backpropagation algorithm. To show that the proposed framework can be widely applied it is employed to gradient descent with momentum (GDM) and with momentum and adaptive learning rates (GDX). A subsequent section will also provide an extension to resilient propagation. Improving variants of conjugate gradient were also attempted, however due to the implications of the restart procedure these were unsuccessful. Nevertheless, it is still very possible that a successful opposition-based conjugate gradient method could be developed.

6.5.1 Experimental Setup

For these experiments, each network architecture and learning parameters (i.e. momentum or learning rate value) has been tuned to yield the best results. Unless stated otherwise all outcomes represent the averaged results over 30 trials, each of 100 epochs with parameter settings as follows:

- **GDM:** Learning rate = 0.15, momentum = 0.90.

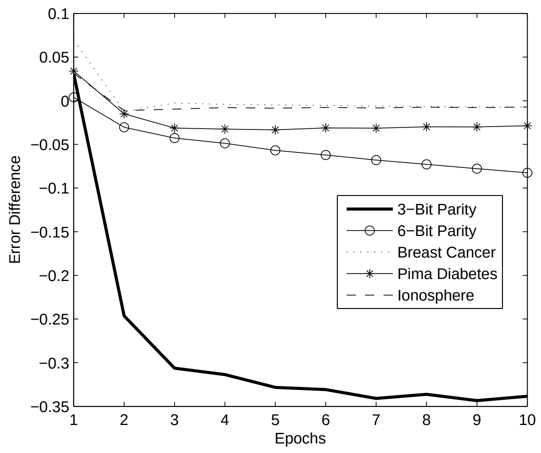

Figure 6.4: Comparing the difference in error between the trained network and the opposite network with minimum error.

- **GDX:** Initial learning rate = 0.01, momentum = 0.90.

The probability function used in Line 5 of Algorithm 10 is:

$$f(t, C) = \max(0.05, e^{-t/C}) \tag{6.34}$$

where for GDX, $C = 25$ and for GDM, $C = 50$. All experiments use a single hidden layer with the hyperbolic tangent transfer function.

The 10 data sets used in these experiments were attained from the UCI machine learning repository and are shown in Table 6.5. Unless testing data is given with the data set, 80% of the training data is randomly selected per trial for testing purposes (the arcene and madelon data sets used the validation set for testing). These problems were selected because they provide variety in the number of training patterns, features and classes.

6.5.2 Training

Table 6.6 presents a summary of the results comparing GDM and its opposite counterpart OGDM. Firsly, observe that 5 of the 10 problems yield a statistically significant result according to a two sample Kolmogorov-Smirnov significance test with 0.95 confidence. Other values are not significant at this level, however OGDM consistently yields a lower mean training error μ and standard deviation σ.

The column $\mathbb{E}[acc]$ shows the average number of epochs where a transfer function change was accepted. With the exception of the madelon data set, it is seen that only a few (1 to 6) transfer changes can have an

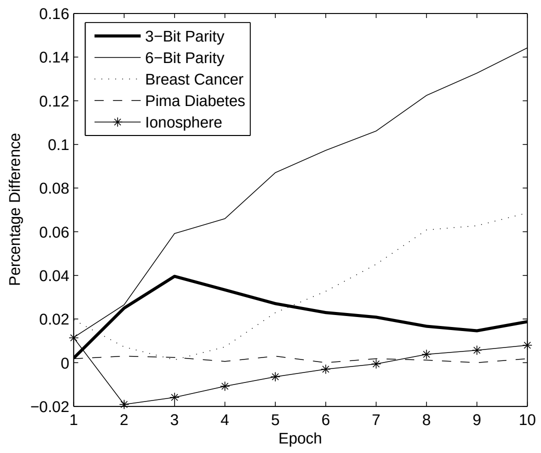

Figure 6.5: Comparing the difference in rank between the trained network and opposite networks.

important impact on the outcome of training. Although, the results for the madelon data set were among those found to be statistically significant.

The second last column compares the final results of GDM and OGDM, given as a triple $< OGDM < GDM, OGDM > GDM, OGDM = GDM >$. In all cases OGDM finds more results with a lower training error than GDM. Only 23 of the 300 trials (7.6%) of the results are found to be equal. In total, $179/300 \approx 0.61$ of the results for OGDM are more desirable than those with OGDM. The remaining 31% of experiments represent GDM outperforming OGDM. The final column computes the effect size via Cohen's d-statistic. As is observed three instances have a "small" improvement and two and four instances show a medium and large improvement, respectively. In 9/10 instances there is an improvement when using the opposition-based algorithm.

A comparison of training results for GDX versus OGDX is given in Table 6.7. Of these results 4 of 10 are statistically significant. But, an improvement in accuracy and reliability is observed in all problems. The average number of transfer function changes that were accepted as improving the network is between 2 and 8, which highlights the impact of OTFs. Further, $190/300 \approx 0.63$ of experiments showed an improvement when using OTFs, whereas only 33% of the results favored GDM. The remaining 3% of experiments yielded equal outcomes. Comparing the effect size of GDX versus OGDX shows an improvement in 9/10 instances; 2 are "small", 3 are "medium" and 4 are "large".

Figures 6.7 and 6.8 show the characteristic behavior of the application of the OTF framework to GDM and GDX learning, respectively. In both instances a more rapid convergence is observed, especially during the early training stages of learning. These results represent the average convergence over the 30 trials.

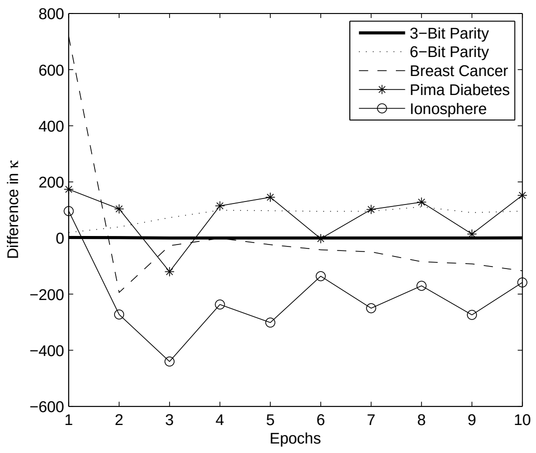

Figure 6.6: A comparison of the difference in condition between the trained network and its opposite networks.

6.5.3 Generalization Ability

Tables 6.8 and 6.9 show the generalization results for GDM versus OGDM and GDX versus OGDX, respectively. In both cases 5 of the 10 data sets show a statistically significant improvement when using opposite transfer functions. As with the training results, an improvement is observed with all problem sets. Even a small improvement in generalization can have drastic effects on the application. For example, in a large population where medical diagnosis of a condition is costly, a small improvement in accuracy or a lowering of false results could potentially save a significant amount of resources.

Table 6.8 shows a small improvement for the thyroid problem by using OTFs and a moderate improvement for the digits problem. The iris, lp4, madelon and arcene instances all show large bias towards OGDM. Similar results are shown in Table 6.9 where 1 small, 2 moderate and 4 large improvements are attained.

6.6 Training Over-Sized Networks

In this section results of experiments to test the behavior of the proposed method on over-sized networks, as well as compare it to two variants of backpropagation are presented. The network hidden layer architecture for all experiments is Inputs-50-25-25-Outputs, except for experiments examining the effects of hidden layer size. These over-sized networks will give insight into the ability of the proposed method to determine weights when the target concept is very simple but the network has many free parameters which can lead to local optima or very slow convergence.

Figure 6.7: Example convergence curves for GDM and OGDM for the cancer problem.

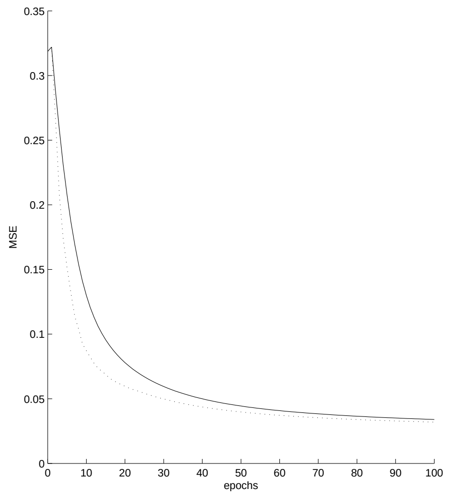

Figure 6.8: Example convergence curve for GDX and OGDX for the iris problem.

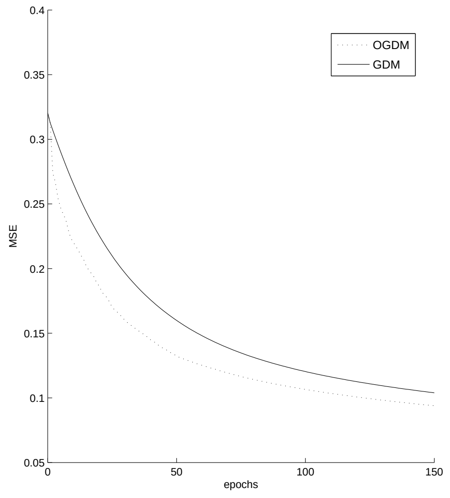

Table 6.5: The data used for comparing variants of backpropagation. The input and output sizes, in addition to the number of patterns is presented.

Data Set	# Training Patterns	# Features	# Hidden Neurons	# Classes
cancer	559	9	10	2
thyroid	5760	21	10	3
wine	142	13	10	3
iris	120	4	20	3
glass	171	9	15	2
digits	3823	62	25	10
lp4	93	90	25	3
ionosphere	280	33	25	2
madelon	2000	500	35	2
arcene	100	9920	50	2

Five common benchmark problems from the UCI Machine Learning Repository [5] were selected. For all experiments inputs are standardized and normalized and the results are averaged over 30 trials and network performance is measured using the mean-squared error measure (MSE). In total, the number of weights to optimize is shown in Table 6.10, unless otherwise noted.

6.6.1 Function Call Overhead Analysis

Here, the behavior of the method with respect to function calls is analyzed. A hidden network architecture of 50-25-25 hidden neurons using $\varphi(\cdot) = tanh(\cdot)$ being trained for 250 epochs using backpropagation with momentum and an adaptive learning rate was used.

The probabilistic component of the decision rule for each neuron (i.e. line 5 of Algorithm 10) will have a large impact on the number of function calls which decide whether to attempt to use an OTF. The $U(i)$ used in these experiments dominate the behavior of the algorithm, therefore the results are presented without loss of generality for one benchmark problem, unless otherwise noted.

Figure 6.9 shows the average number of neurons which undergo the opposite transformation per epoch for the cancer dataset. As described above, the behavior is essentially governed by line 5 of the proposed approach. As such, a similar behavior was seen for all datasets. The initial iterations allow for more neurons to undergo the transformation and as the epochs progress only few neurons will change.

Table 6.11 presents the average number of attempted opposite transformations and the number transformation attempts which did not improve the network error. The corresponding probability of a successful transformation is also given. Due to $\mathbf{U}(i)$ there is a high rate of attempted transformations, although the acceptance rate is around 0.04. The decision to perform a transformation was designed for computational efficiency based on fast heuristic and stochasticity and therefore a large number of non-accepted transformation is expected. As more complex decision function is developed this rate is expected to greatly increase.

Table 6.6: Comparing GDM with and without opposite transfer functions (OGDM) using the mean (μ), standard deviation (σ), expected number of accepted transformations ($\mathbb{E}[acc]$) and the number of times ($OGDM < GDM, OGDM > GDM, OGDM = GDM$). Bold values are statistically significant at 0.95 confidence using a Kolmogorov-Smirnov test. The final column reports Cohen's d-statistic to determine the effect size.

	GDM		OGDM				
Data Set	μ	σ	μ	σ	$\mathbb{E}[acc]$	$(<,>,=)$	Effect
cancer	0.034	0.006	0.030	0.003	3.8	(19,10,1)	0.389,m
thyroid	0.068	0.008	**0.055**	0.007	3.8	(18,11,1)	0.654,ℓ
wine	0.047	0.008	0.042	0.009	1.4	(18,3,9)	0.282,s
iris	0.104	0.036	0.091	0.012	2.6	(18,12,0)	0.235,s
digits	0.089	0.015	**0.071**	0.007	2.8	(24,6,0)	0.610,ℓ
glass	0.046	0.008	**0.035**	0.007	2.8	(19,10,1)	0.591,ℓ
lp4	0.102	0.010	0.094	0.008	3.2	(17,13,0)	0.404,m
ionosphere	0.072	0.010	0.072	0.009	2.3	(10,9,11)	0.000
madelon	0.126	0.009	**0.094**	0.005	16.7	(17,13,0)	0.910,ℓ
arcene	0.013	0.023	**0.005**	0.005	6.4	(24,6,0)	0.234,s

6.6.2 Expected Improvements

Although the probability of using a proposed set of transformations is approximately 0.04, the average improvement and impact on learning is well worth the overhead. It has already been shown above that before learning commences the probability a given transfer function combination has the lowest error, for a given weight configuration, is approximately uniformly distributed. This experiment is to examine the improvement with respect to error during learning. More specifically,

$$rel\_error = \frac{eval^{t-1} - eval^{t}}{eval^{t-1}}. \tag{6.35}$$

where $eval$ is computed according to the MSE and $t > 0$ is the epoch, also $eval^{t-1} > eval^{t}$ (as a consequence of MATLAB training only the best network with respect to training error over the learning period is retained).

Figure 6.10 plots the relative improvement in cases where a transfer function transformation has taken place and the transformation yielded an improvement over the current error. As hypothesized, relatively large improvements tend to be found during early stages (first 100 epochs) of training due to ill-conditioning. Changing the transfer function has a relatively large impact and often improves the error by more than 15% at a given epoch. The largest observed average improvement was above 60% (epoch 50 of the cancer data set). Improvements are also observed at epochs > 100, however due to the relatively rapid convergence rate the expected improvement is usually < 0.05.

Table 6.7: Comparing GDX with and without opposite transfer functions (OGDX) using the mean (μ), standard deviation (σ), expected number of accepted transformations ($\mathbb{E}[acc]$) and the number of times ($OGDX < GDX, OGDX > GDX, OGDX = GDX$). Bold values are statistically significant at 0.95 confidence using a Kolmogorov-Smirnov test. The final column reports Cohen's d-statistic to determine the effect size.

	GDX		OGDX				
Data Set	μ	σ	μ	σ	$\mathbb{E}[acc]$	$(<,>,=)$	Effect
cancer	0.021	0.003	0.019	0.003	7.7	(21,8,1)	0.316,s
thyroid	0.031	0.002	**0.028**	0.001	3.6	(17,13,1)	0.688,ℓ
wine	0.003	0.001	0.003	0.001	3.3	(16,14,0)	0.000
iris	0.018	0.004	0.015	0.002	3.4	(22,8,0)	0.429,m
digits	0.019	0.001	**0.010**	0.001	1.6	(20,10,0)	0.976,ℓ
glass	0.012	0.005	0.009	0.003	3.1	(20,10,0)	0.342,m
lp4	0.055	0.012	0.048	0.007	3.2	(23,7,0)	0.336,m
ionosphere	0.074	0.015	0.070	0.008	2.6	(11,11,8)	0.164,s
madelon	0.019	0.010	**0.007**	0.004	6.2	(19,11,0)	0.619,ℓ
arcene	0.010	0.003	**0.003**	0.002	7.7	(21,8,1)	0.808,ℓ

Table 6.8: Comparing the generalization ability of GDM without and with opposite transfer functions (OGDM) using the mean (μ) and standard deviation (σ) on test data. Bold values are statistically significant at 0.95 confidence using a Kolmogorov-Smirnov test. The final column reports Cohen's d-statistic to determine the effect size.

	GDM		OGDM		
Data Set	μ	σ	μ	σ	Effect
cancer	0.036	0.012	0.034	0.009	0.094
thyroid	0.070	0.009	0.065	0.009	0.268,s
wine	0.058	0.016	0.055	0.012	0.105
iris	0.117	0.022	**0.085**	0.018	0.623,ℓ
digits	0.089	0.014	**0.080**	0.006	0.386,m
glass	0.062	0.021	0.057	0.012	0.145
lp4	0.154	0.039	**0.101**	0.030	0.606,ℓ
ionosphere	0.099	0.016	0.095	0.023	0.100
madelon	0.301	0.013	**0.283**	0.012	0.584,ℓ
arcene	0.237	0.036	**0.182**	0.032	0.628,ℓ

Table 6.9: Comparing the generalization ability of GDX without and with opposite transfer functions (OGDX) using the mean (μ) and standard deviation (σ) on test data. Bold values are statistically significant at 0.95 confidence using a Kolmogorov-Smirnov test. The final column reports Cohen's d-statistic to determine the effect size.

Data Set	GDX		OGDX		
	μ	σ	μ	σ	Effect
cancer	0.024	0.013	0.023	0.012	0.040
thyroid	0.032	0.003	**0.021**	0.001	0.926,ℓ
wine	0.012	0.008	0.012	0.007	0.000
iris	0.028	0.015	0.021	0.009	0.272,s
digits	0.022	0.001	0.022	0.001	0.000
glass	0.053	0.024	**0.037**	0.016	0.365,m
lp4	0.154	0.039	**0.101**	0.030	0.606,ℓ
ionosphere	0.113	0.034	0.109	0.027	0.065
madelon	0.331	0.012	**0.267**	0.010	0.945,ℓ
arcene	0.212	0.045	**0.165**	0.032	0.516,ℓ

Table 6.10: Comparing data sets and network sizes.

Data Set	Inputs	Outputs	Patterns	Weights + Biases
cancer	9	2	699	2475
wine	13	3	178	2700
thyroid	21	3	7200	3100
iris	4	3	150	2250
glass	9	2	214	2475

Table 6.11: The average number of attempted opposite transformations (Trans), average extra epochs and probability a flip led to an improvement in network error.

Data set	avg(Trans)	avg(Extra evaluations)	P(improvement)
cancer	235	222	0.055
wine	236	226	0.042
thyroid	237	224	0.055
iris	236	230	0.025
glass	237	228	0.038

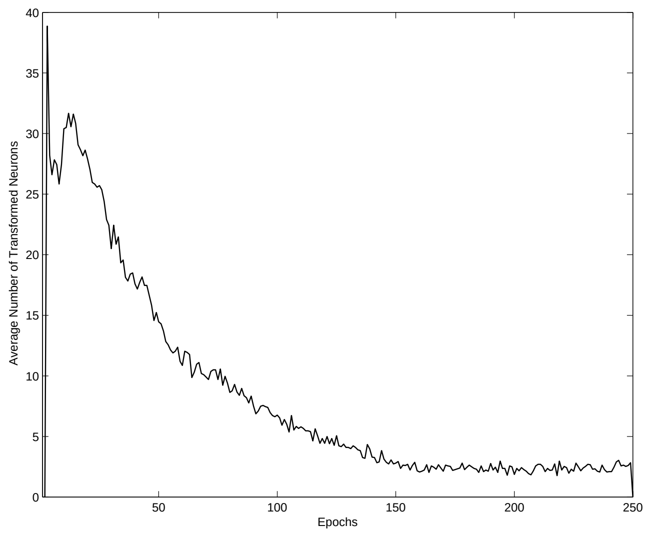

Figure 6.9: The average number of neurons which undergo the opposite transformation per epoch for the cancer dataset. Similar behavior was observed for all data sets.

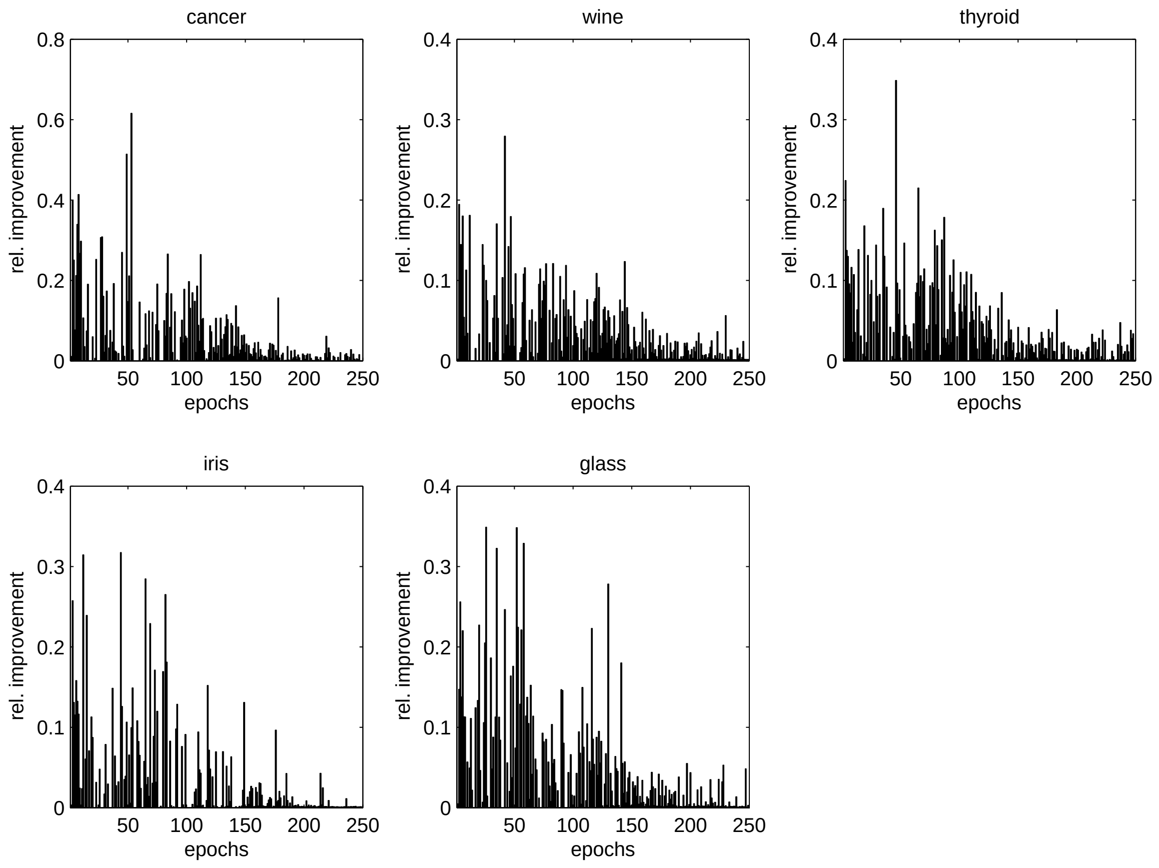

Figure 6.10: The average relative improvement over the previous network performance (MSE). Only cases where a transfer flip was successful are considered.

6.6.3 Layer Size and Number

In this section the impact of layer size and the number of layers on the performance of the proposed method is examined. For all experiments the cancer data set is used, but similar behavior was observed on all problems. Opposite networks are trained for 250 epochs, original/normal networks are trained for 500 epochs to accommodate the function call overhead. Although, a weight update is much more expensive than determining whether a neuron should switch to its opposite. Hence, this is also a pessimistic estimate of OTF performance.

Table 6.12 shows the results of varying the hidden layer size and number. In all cases the average (μ) and standard deviation (σ) values are more favorable for the proposed method. A Kolmogorov-Smirnov significance test at 0.99 confidence level was conducted and all mean values are indeed significant.

Additionally, calculate the improvement factor,

$$\frac{\mu_{orig}}{\mu_{otf}} \tag{6.36}$$

where the subscripts otf and $orig$ correspond to the proposed and original methods, respectively. The minimum improvement in error is 154% and the maximum observed average improvement is 337%. When the number of hidden layers is fixed and only its size varies the improvement factor also increases. This is a very significant result, showing that the proposed method is particularly well suited to training large networks. Furthermore, this improvement is more prominent as the network complexity increases.

The final column of Table 6.12 measures the effect size using Cohen's d-statistic's equivalent, the nonoverlap percentile. All cases have a measurable improvement when using OTFs; 1 is of the small variety, 3 and 5 are moderate and large, respectively.

Table 6.12: Summary of final error results when varying layer size. Bold values are statistically significant at a 0.95 confidence level using a t-test. The final column reports Cohen's d-statistic (nonoverlap percentile) to determine the effect size.

Size	μ_{orig}	σ_{orig}	μ_{otf}	σ_{otf}	Improvement	Effect
25	0.075	0.033	**0.045**	0.016	1.671	0.501,ℓ
50	0.092	0.102	**0.031**	0.005	2.938	0.389,m
100	0.093	0.106	**0.027**	0.004	3.371	0.403,m
25-25	0.060	0.011	**0.037**	0.009	1.602	0.753,ℓ
50-50	0.059	0.012	**0.031**	0.004	1.895	0.843,ℓ
100-100	0.081	0.078	**0.031**	0.004	2.592	0.412,m
50-25-25	0.057	0.010	**0.037**	0.009	1.541	0.725,ℓ
100-50-25	0.060	0.011	**0.037**	0.008	1.602	0.767,ℓ
100-100-100	0.073	0.074	**0.039**	0.006	1.852	0.308,s

6.6.4 Backpropagation Variants

Here, variants of backpropagation are used to train the networks. The two variants chosen are; BP with adaptive learning rate [42] (GDA) and BP with adaptive learning rate and momentum (GDX).

Figure 6.11 compares the performances of GDA and GDX as the training method being improved. The proposed method only trains for 250 epochs, and remains constant for the remaining 250 epochs of training for the original algorithms. In all cases the proposed method has a much quicker rate of convergence relative to the original training procedure. Furthermore, the final result achieved is also found to be statistically significant at a confidence level of 0.99 using a Kolmogorov-Smirnov test.

Focusing on the first 50 epochs of training, poorly initialized networks are quickly be improved via the proposed transfer function transformation. This is evident due to the very quick drop in MSE measure. In fact, the MSE of the proposed method is statistically significant using the same test as above for all epochs > 3.

6.6.5 Generalization Ability

To show an avoidance of the over-fitting phenomena, re-train the networks above with a randomly chosen 80% of the data, leaving the other 20% as the test set. The training and testing data were reselected at each of the 30 runs in order to give a better estimate on the performance. A comparison to GDA training and GDX results were very similar.

Table 6.13 summarizes the testing and training results for the 5 problems. The Training column presents final mean training error and standard deviation. The confusion matrix column gives results for gda/otf for the problems, respectively. Note the class labels are not shown but assumed to be 0,1 horizontal and vertical. The otf based algorithm shows more desirable results for all problems.

6.7 Resilient Propagation

Resilient propagation [94] (RP) is a popular alternative method to training feedforward neural networks. Since only the sign of the gradient is used many consider it a first order training method. The underlying motivation for rProp is to eliminate the harmful influence of the size of the partial error derivative on the weight update step of learning. Its popularity is due to both its simplicity, fast learning rate and quality results.

Weight update is performed in two stages. First, for each weight there is a parameter γ_{ij} between neurons i, j which is updated according to

Table 6.13: Confusion matrix results (nrm/otf). The training error is given in parentheses beside the data set name.

Training	Confusion matrix (0,1)	
Cancer		
gda: 0.041 ± 0.009	86.9/87.7	3.9/3.1
otf: 0.028 ± 0.005	4.2/2.6	44.9/46.6
Wine		
gda: 0.036 ± 0.011	10.9/10.8	1.1/1.2
otf: 0.028 ± 0.010	3.1/2.1	21.0/21.9
Thyroid		
gda: 0.043 ± 0.018	9.7/11.9	23.4/21.1
otf: 0.026 ± 0.009	52.3/23.8	1354.7/1383.2
Iris		
gda: 0.010 ± 0.010	10.5/10.5	0.1/0.0
otf: 0.002 ± 0.002	0.5/0.1	19.0/19.4
Glass		
gda: 0.054 ± 0.014	8.1/8.6	2.9/2.4
otf: 0.042 ± 0.009	2.2/1.7	29.8/30.2

$$\gamma_{ij}^{t+1} = \begin{cases} \min(\gamma_{ij}^{t}\eta^{+}, \gamma_{max}), & \text{if } \nabla_{ij}Er^{t}\nabla_{ij}Er^{t-1} > 0, \\ \max(\gamma_{ij}^{t}\eta^{-}, \gamma_{min}), & \text{if } \nabla_{ij}Er^{t}\nabla_{ij}Er^{t-1} < 0, \\ \gamma_{ij}^{t}, & \text{otherwise.} \end{cases} \tag{6.37}$$

where Er is the network error, t is the iteration and η^{+}, η^{-} are step sizes and $\gamma_{min}, \gamma_{max}$ determine boundaries. Commonly, $0 < \eta^{+} < \eta^{-} < 10$ and γ_{min} is very close to 0 and γ_{max} is between 25 and 75.

The second step performs the actual weight update according to

$$w_{ij}^{t+1} = w_{ij}^{t} - \gamma_{ij}^{t} sgn(\nabla_{ij}Er^{t}). \tag{6.38}$$

This section will compare the training of rProp with and without opposite transfer functions, under the proposed heuristic. Specific settings for learning parameters are $\eta^{-} = 0.5$, $\eta^{+} = 1.2$, $\gamma_{min} = 0.07$ and $\gamma_{max} = 50.0$. These values were experimentally determined to yield good results on all problems. Unless noted otherwise all networks are trained for 100 epochs, and the presented results represent the average over 30 trials. The data presented in Table 6.5 is also employed for these experiments.

6.7.1 Training

Table 6.14 presents the results of the training phase for resilient propagation with and without opposite transfer functions (ORP and RP, respectively). Resilient propagation often yields more desirable results than standard backpropagation learning. These findings show that only 2 of the 10 test problems resulted in statistically significant differences between RP and ORP. However, there is still an improvement over RP in all data sets. Importantly, the results for the larger madelon and arcene problems were statistically significant at a 0.95 confidence level.

The expected number of accepted transfer function changes is between 3 and 7 for all problems, further supporting the potential of OTFs. The number of instances where ORP outperformed RP is $196/300 \approx 0.65$, whereas RP yielded more desirable outcomes in only $96/300 \approx 0.32$ meaning 2.6% of the results are equal. Of these, one is small, three are moderate and three are large, as shown by the effect size column.

Table 6.14: Comparing RP with and without opposite transfer functions (ORP) using the mean (μ), standard deviation (σ), expected number of accepted transformations ($\mathbb{E}[acc]$) and the number of times ($ORP < RP, ORP > RP, ORP = RP$). Bold values are statistically significant at 0.95 confidence using a Kolmogorov-Smirnov test. The final column reports Cohen's d-statistic (nonoverlap percentile) to determine the effect size.

	RP		ORP				
Data Set	μ	σ	μ	σ	$\mathbb{E}[acc]$	$(<,>,=)$	Effect
cancer	0.004	0.002	0.004	0.002	3.6	(18,12,0)	0.000
thyroid	0.004	0.001	0.003	0.000	7.1	(15,10,5)	0.577,ℓ
wine	0.003	0.001	0.003	0.001	3.3	(16,14,0)	0.000
iris	0.007	0.003	0.005	0.002	3.2	(20,8,2)	0.365,m
digits	0.002	$\ll 0.001$	0.001	$\ll 0.001$	3.1	(20,10,0)	0.447,m
glass	0.002	0.002	0.001	0.001	4.9	(18,12,0)	0.302,s
lp4	0.002	0.013	$\ll 0.001$	0.001	4.1	(20,10,0)	0.054
ionosphere	0.006	0.003	0.004	0.001	3.7	(20,9,1)	0.408,m
madelon	0.012	0.007	**0.004**	0.004	6.2	(26,4,0)	0.574,ℓ
arcene	0.113	0.031	**0.042**	0.011	3.3	(23,7,0)	0.836,ℓ

Figure 6.12 presents an example convergence curve between RP and ORP for the arcene problem. The rate of convergence is similar between the two algorithms, however the opposite transfer function version is able to discover a lower error rate, making it much more desirable. It can be interpreted that the convergence rate for ORP is improved since RP is not able to converge to the same value within the same number of epochs. Other problems exhibit a similar convergence graph.

6.7.2 Generalization Ability

The results focusing on generalization ability (with respect to the MSE on the testing data) are presented in Table 6.15. Only 2 of the 10 results are statistically significant, although an improvement with respect to both μ and σ are observed in all instances. Even a small improvement in generalization ability can have a deep impact on the application requesting the results. Being said, the effect size reveals that the iris and lp4 problems show a small practical improvement whereas the arcene and madelon outcomes represent somewhat large improvements.

Table 6.15: Comparing the generalization ability of RP without and with opposite transfer functions (ORP) using the mean (μ) and standard deviation (σ) on test data. Bold values are statistically significant at 0.95 confidence using a Kolmogorov-Smirnov test. The final column reports Cohen's d-statistic to determine the effect size.

Data Set	RP		ORP		
	μ	σ	μ	σ	Effect
cancer	0.036	0.012	0.034	0.009	0.094
thyroid	0.007	0.001	0.007	0.001	0.000
wine	$\ll 0.001$	0.000	$\ll 0.001$	0.000	0.000
iris	0.025	0.014	0.020	0.010	0.201,s
digits	0.010	0.001	0.010	0.001	0.000
glass	0.057	0.034	0.051	0.029	0.095
lp4	0.123	0.036	0.101	0.040	0.278,s
ionosphere	0.086	0.035	0.083	0.022	0.051
madelon	0.373	0.015	**0.355**	0.013	0.540,ℓ
arcene	0.222	0.045	**0.157**	0.040	0.607,ℓ

6.8 Summary

This chapter explored the symmetry of neural networks and concentrated on adaptive transfer functions. Specifically, the opposite transfer function and opposite neural network concepts were introduced. Properties of irreducibility and numerical conditioning were discussed and some basic theorems were proven. Using these results a proposed heuristic was presented which efficiently selects a single network from the set of opposite networks.

Experimental results focused on the potential of opposite networks to improve the initial conditioning of the neural networks. Further, the early stages of training were examined as they are the most likely to see opposite transfer functions yielding an improved input-output mapping. Ten benchmark data sets were also used to compare different gradient-based learning methods for reasonably-sized networks. Finally, very large networks were used to test whether the proposed method scales to complex networks. In all cases opposite transfer functions were shown to have much potential for improving accuracy, reliability, generalization and convergence rate. An application of OTFs to resilient propagation also revealed desirable benefits. As with opposition-based simulated annealing and oppositional population-based incremental learning observed improvements were more drastic for larger, more complex problems.

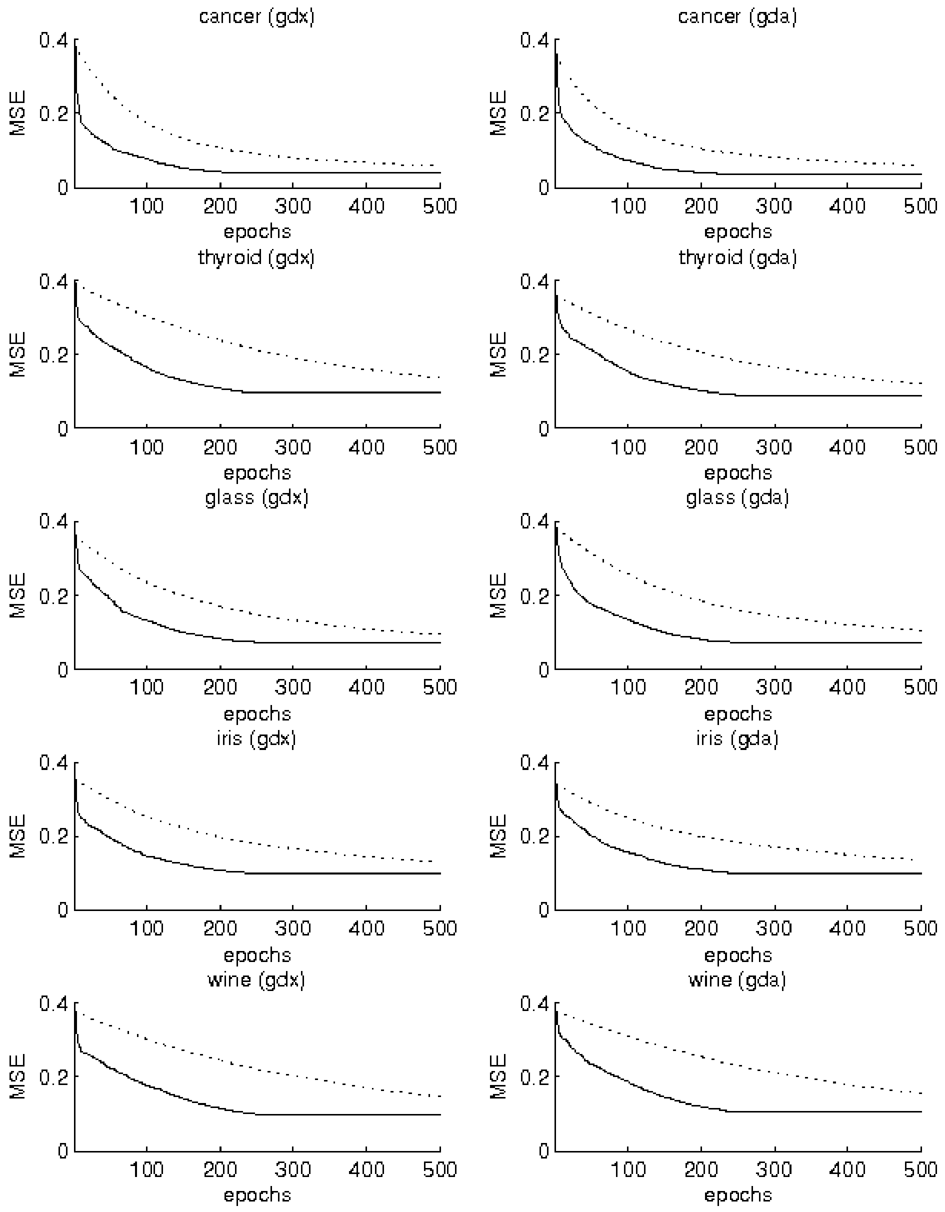

Figure 6.11: Comparing the proposed method performance for networks trained with gradient descent using adaptive learning rate (GDA) and adaptive learning rate with momentum (GDX). The dotted line refers to the original training procedure, solid line is the proposed method.

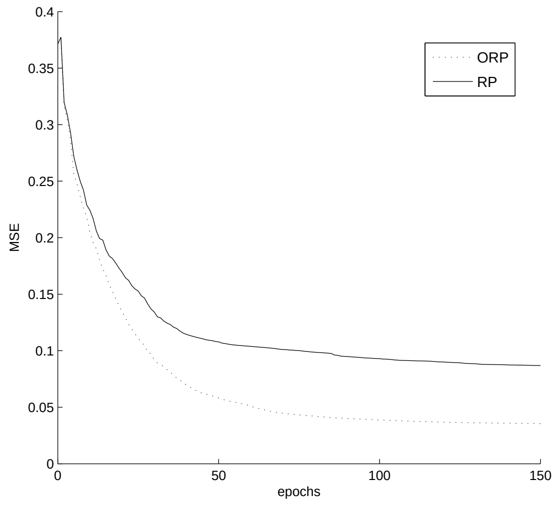

Figure 6.12: Comparing the convergence of RP and ORP for the arcene problem.

Chapter 7

Conclusions

Symmetry is a fundamental aspect of nature that has been exploited in various scientific fields. This thesis has explored a possible framework for inducing symmetry on a search space with respect to the evaluation function. The framework was applied to simulated annealing, population-based incremental learning and gradient-based learning in neural networks with very promising results in all domains.

The main contributions of this thesis can be summarized as follows:

1. **Mathematical Motivation:** Providing a mathematical description of the concept of opposition within the context of optimization. Opposition is described as a symmetry inducing transformation which yields a more desirable evaluation function, with respect to the problem at hand. Initially, symmetry was discussed within the context of group theory and the conditions which a given search space may have symmetry induced on it, with respect to its representation were discussed. While computational group theory concepts were not directly employed, the fundamental basis for the proposed approach are rooted in this field of abstract mathematics. Using this as a basis, the neighborhood structure was discussed as a means by which symmetry may be imposed and described.

2. **Symmetry and Opposition:** Expanding on the concept of the neighborhood structure, the opposition map was introduced as a means to induce symmetry over the evaluation function. In this way it becomes possible to define a symmetric search space without regard to its actual structure. As a consequence, a general framework whereby an existing optimization algorithm can employ opposition was derived and forms the basis for the example applications. Concepts related to opposition-based computing were also presented and discussed.

3. **Opposition-Based Simulated Annealing:** The concept of an opposite neighbor was proposed as a simple means to improve the simulated annealing algorithm. Benchmarking results using standard real optimization functions found improvements in accuracy and reliability for the proposed method.

The influence of various parameter settings were also tested. Using the real-world problem of image thresholding, the OSA algorithm was also shown to have potential in decreasing the required number of function calls to attain a desired target image, i.e. an improvement in convergence rate.

4. **Oppositional Population-Based Incremental Learning:** Expanding on the idea of an opposite neighbor, an opposite set was proposed as a possible concept to improve population-based searches, particularly population-based incremental learning. The improvement in sample diversity over random sampling was first proven and a new probability update method was then given. The resulting algorithm was compared to its predecessor using common, but difficult, binary optimization benchmark problems. The results yielded improvements in diversity, expected result, reliability and convergence rate. The traveling salesman and image thresholding problems were employed to provide further support to its abilities.

5. **Opposition-Based Gradient Learning:** The third example application focused on gradient-based learning in feedforward neural networks. The simple concept of an opposite transfer function was proposed as a means to jump within the search space during learning. These functions were shown to yield unique networks under a minimality assumption, which had consequences on the numerical conditioning. Experimental results confirmed an influence before and during early training stages. An efficient heuristic method for selecting an opposite network was provided as a means to improve backpropagation variants. Experimental results confirm the improvement in accuracy, reliability, convergence rate and often in generalization ability. The heuristic was also employed to resilient propagation, where valuable improvements were also observed.

An important finding generally observed for all three proposed algorithms was an increased benefit over the traditional method as the problem complexity increased. Deeper insight in this direction is another very exciting research direction, as is the sensitivity to noisy and missing data.

While this thesis has given theoretical motivation and experimental evidence supporting the use of opposition-based computing, much work must still be done. From a theoretical standpoint further statistical evidence and deeper insight through group theory may lead to a more advanced understanding of the subject and lead to improved heuristics, or exact approaches. More abstract mathematics could be investigated as a means to comprehending opposition in a more general form.

The methods presented in this thesis considered basic forms of simulated annealing, population-based incremental learning and gradient-based learning. However, there exists more advanced versions of these approaches and further experimentation using opposition on these is required. Additionally, there exists a wide number of algorithms in machine learning and computational intelligence that remain unexamined through the lense of opposition. Investigations in these areas are an exciting direction of future work as well.

Appendix A

Computing Effect Size

Statistical significance tests determine whether observed differences between sample populations are due to randomness. However, these measures do not quantify the strength of the relationship between the samples. Effect-size measures accomplish this goal, thus allowing the possibility to distinguish whether observed differences are practically “small”, “medium” or “large”. A popular method of calculating this is via Cohen’s d-statistic [22]:

$$d = \frac{\mu_1 - \mu_2}{\sigma^*} \tag{A.1}$$

where μ_1 and μ_2 are the means of sample 1 and 2, respectively. The s^* value is the pooled standard deviation:

$$\sigma^* = \sqrt{\frac{\sigma_1^2 + \sigma_2^2}{2}} \tag{A.2}$$

where σ_1 and σ_2 are the standard deviations for samples 1 and 2, respectively. Note, the sample size is irrelevant. Different interpretations for the d statistic can be found, however the traditional interpretation is “small”>0.2, “medium”>0.5 and “large”>0.8.

A common equivalent (and interchangeable) measure is to report the percent of nonoverlap. This measure quantifies the percentage of nonoverlap between the two samples. An overlap of 0.0 indicates the samples completely overlap. The equivalent interpretations are 0.147 < “small” < 0.330 < “medium” < 0.474 < “large”. This thesis utilizes this measure as it is bounded between 0 and 1, whereas the d-statistic can take larger values. It should be reinforced that the two measures are equivalent [22].

Bibliography

[1] *The American Heritage Dictionary of the English Language.* Houghton Mifflin, 2000.

[2] J. Abonyi and B. Feil. Computational intelligence in data mining. *Informatica*, 29:2–12, 2005.

[3] F. Albertini and E. Sontag. For neural networks, function determines form. *Neural Networks*, 6:975–990, 1993.

[4] B. Apolloni, C. Caravalho, and D. De Falco. Quantum stochastic optimization. *Stochastic Processes and Their Applications*, 33:233–244, 1989.

[5] A. Asuncion and D. J. Newman. UCI machine learning repository, 2007.

[6] N. Balcan, A. Blum, and N. Srebro. A theory of learning with similarity functions. *Machine Learning Journal*, 72(1-2):89–112, 2008.

[7] 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.

[8] E. I. Barakova and L. Spaanenburg. Symmetry: Between indecision and equality of choice. *Biological and Artificial Computation: From Neuroscience to Technology*, 1997.

[9] J. Barnes, B. Colletti, and D. Neuway. Using group theory and transition matrices to study a class of metaheuristic neighborhoods. *European Journal of Operational Research*, 138(3):531–544, 2001.

[10] C. Bishop. *Pattern Recognition and Machine Learning.* Springer, 2007.

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

[12] A. Bryson and Y. Ho. *Applied Optimal Control: Optimization, Estimation, and Control.* Blaisdell Publishing Company, 1969.

[13] 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.

[14] V. Cerney. A thermodynamical approach to the travelling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45:41–51, 1985.

[15] P. Chandra and Y. Singh. An activation function adapting training algorithm for sigmoidal feedforward networks. *Neurocomputing*, 61, 2004.

[16] P. Chandra and Y. Singh. A case for the self-adaptation of activation functions in FFANNs. *Neurocomputing*, 56:447–545, 2004.

[17] O. Chapelle, B. Schlopf, and A. Zien. *Semi-Supervised Learning*. MIT Press, 2006.

[18] C. Charalambous. Conjugate gradient algorithm for efficient training of artificial neural network. In *Proc. IEEE Circuits, Devices and Systems*, volume 139, pages 301–310, 1992.

[19] A. Chen and R. Hecht-Nielsen. On the geometry of feedforward neural network weight spaces. In *Proc. Second International Conference on Artificial Neural Networks*, pages 1–4, 1991.

[20] A. M. Chen, H. Lu, , and R. Hecht-Nielsen. On the geometry of feedforward neural networks error surfaces. *Neural Computation*, 5(6):910–927, 1993.

[21] C. T. Chen and W. D Chang. A feedforward neural network with function shape autotuning. *Neural Networks*, 9, 1996.

[22] J. Cohen. *Statistical Power Analysis for the Behavioral Sciences*. Lawrence Earlbaum Associates, 1988.

[23] B. Colletti and J. Barnes. Using group theory to construct and characterize metaheuristic search neighborhoods. In C. Rego and B. Alidaee, editors, *Metaheuristic Optimization via Memory and Evolution Tabu Search and Scatter Search*, pages 303–328. 2005.

[24] V. Corradi and H. White. Regularized neural networks: Some convergence rate results. *Neural Computation*, 7(6):1225–1244, 1995.

[25] P. Darga, K. Sakallah, and I. Markov. Faster symmetry discovery using sparsity of symmetries. In *Proc. Annual ACM IEEE Design Automation Conference*, pages 149–154, 2008.

[26] A. Das and B. K. Chakrabarti, editors. *Quantum Annealing and Related Optimization Methods*, volume 679 of *Lecture Note in Physics*. Springer, 2005.

[27] K. A. De Jong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan, 1975.

[28] A. Donaldson and A. Miller. A computational group theoretic symmetry reduction package for the spin model checker. In *Proc. Algebraic Methodology and Software Technology*, pages 374–380, 2006.

[29] A. Donaldson and A. Miller. Exact and approximate strategies for symmetry reduction in model checking. In *Proc. Formal Methods*, pages 541–556, 2006.

[30] W. Duch and N. Jankowski. Optimal transfer function neural networks. In *Proc. 9th European Symposium on Artificial Neural Networks*, pages 101–106, 2001.

[31] W. Duch and N. Jankowski. Transfer functions: Hidden possibilities for better neural networks. In *Proc. 9th European Symposium on Artificial Neural Networks*, pages 81–94, 2001.

[32] S. E. Fahlman. Faster-learning variations on backpropagation: An empirical study. Morgan Kaufmann, 1988.

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

[34] P. Frasconi, M. Gori, M. Maggini, and G. Soda. Unified integration of explicit knowledge and learning by example in recurrent networks. *IEEE Transactions on Knowledge and Data Engineering*, 7(2):340–346, 1995.

[35] K. Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2(3):183–193, 1989.

[36] M. Gallagher. *Multi-Layer Perceptron Error Surfaces: Visualization, Structure and Modelling*. PhD thesis, University of Queensland, 2000.

[37] F. Glover and M. Laguna. *Tabu Search*. Kluwer, 1997.

[38] S. L. Goh and D. Mandic. Recurrent neural networks with trainable amplitude of activation functions. *Neural Networks*, 16, 2003.

[39] D. Goldberg, B. Korb, and K. Deb. Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems*, 3:493–530, 1989.

[40] T. Gosling, J. Nanlin, and E. Tsang. Population based incremental learning versus genetic algorithms: Iterated prisoners dilemma. Technical report, University of Essex, 2004.

[41] Niederreiter H. *Random Number Generation and Quasi-Monte Carlo Methods*. Society for Industrial and Applied Mathematics, 1992.

[42] M. T. Hagan, H. B. Demuth, and M. H. Beale. *Neural Network Design*. PWS Publishing, 1996.

[43] W. Hahn. *Symmetry As A Developmental Principle In Nature And Art*. World Scientific, 1998.

[44] B. Hajek. Cooling schedules for optimal annealing. *Mathematics of Operations Research*, 13(2):311–329, 1988.

[45] B. Hall. *Lie Groups, Lie Algebras, and Representations: An Elementary Introduction*. Springer, 2004.

[46] S. Haykin. *Neural Networks: A Comprehensive Foundation (2nd Edition)*. Prentice Hall, 1998.

[47] G. A. Hoffmann. Adaptive transfer functions in radial basis function (RBF) networks. In *Lecture Notes in Computer Science*, volume 3037, pages 682–686. 2004.

[48] M. Hohfeld and G. Rudolph. Towards a theory of population-based incremental learning. In *Proc. IEEE Conference on Evolutionary Computation*, pages 1–5, 1997.

[49] Z. Hu and H. Shao. The study of neural network adaptive control systems. *Control and Decision*, 7, 1992.

[50] L. Ingber. Adaptive simulated annealing (ASA): Lessons learned. *Control and Cybernetics*, 25(1):33–54, 1996.

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

[52] F. Jordan and G. Clement. Using the symmetries of multilayered network to reduce the weight space. In *Proc. IEEE Second International Joint Conference on Neural Networks*, pages 391–396, 1991.

[53] J. A. S. Kelso. *Uncertainty and Surprise*, volume 3 of *Springer Series in Understanding Complex Systems*, chapter The Complementary Nature of Coordination Dynamics: Toward a Science of the in-Between. Springer-Verlag, 2005.

[54] J. A. S. Kelso and D. A. Engstrm. *The Complementary Nature*. MIT Press, 2006.

[55] G. W. Kinney Jr. *A Group Theoretic Approach to Metaheuristic Local Search for Partitioning Problems*. PhD thesis, Univerity of Texas at Austin, 2005.

[56] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[57] R. Kondor. *Group Theoretical Methods in Machine Learning*. PhD thesis, Columbia University, 2008.

[58] N. Lagaros and M. Papadrakakis. Learning improvement of neural networks used in structural optimization. *Advances in Engineering Software*, 35(1):9–25, 2004.

[59] P. Larranaga and J. Lozano, editors. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Springer, 2002.

[60] Y. Le Cun, J. S. Denker, and S. A. Solla. Optimal brain damage. In *Proc. Advances in Neural Information Processing Systems*, pages 598–605, 1990.

[61] C. Lemieux. *Monte Carlo and Quasi-Monte Carlo Sampling*. Springer, 2009.

[62] Y. Li, D. de Ritter, R. Duin, and M. Reinders. Integration of prior knowledge of measurement noise in kernel density classification. *Pattern Recognition*, 41(1):320–330, 2008.

[63] Y. Liang. Adaptive neural activation functions in multiresolution learning. In *Proc. IEEE International Conference on Systems, Man, and Cybernetics*, volume 4, pages 2601–2606, 2000.

[64] Y. Liu. Computational symmetry. In *Proc. of Symmetry*, pages 231 – 245, 2002.

[65] S. Lohr. *Sampling: Design and Analysis*. Duxbury, 1999.

[66] 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.

[67] H. Maaranen, K. Miettinen, and A. Penttinen. On intitial populations of genetic algorithms for continuous optimization problems. *Journal of Global Optimization*, 37(3):405–436, 2007.

[68] M. Mahootchi, Tizhoosh H. R., and K. Ponnambalam. Opposition-based reinforcement learning in the management of water resources. In *Proc. IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pages 217–224, 2007.

[69] K. Mainzer. *Symmetries of Nature: A Handbook for Philosophy of Nature and Science*. Walter De Gruyter Inc, 1996.

[70] J. Mingjun and T. Huanwen. Application of chaos in simulated annealing. *Chaos, Solitons and Fractals*, 21(4):933–941, 2003.

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

[72] M. F. Moller. A scaled conjugate gradient algorithm for fast supervised learning. 6:525–533, 1993.

[73] D. Muhlenbein, H. Schomisch, and J. Born. The parallel genetic algorithm as function optimizer. *Parallel Computing*, pages 619–632, 1991.

[74] J. Munkres. *Topology.* Prentice Hall, 2000.

[75] K. Murrary. Learning as knowledge integration. Technical report, University of Texas at Austin, 1995.

[76] S. R. Narayanamurthy and B. Ravindran. Efficiently exploiting symmetries in real time dynamic programming. In *Proc. International Joint Conference on Artificial Intelligence*, pages 2556–2561, 2007.

[77] D. Nguyen and B. Widrow. Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights. In *Proc. IEEE International Joint Conference on Neural Netowrks*, volume 3, pages 21–26, 1990.

[78] Y. Nourani and B. Andresen. A comparison of simulated annealing cooling strategies. *Journal of Physics A: Mathematical and General*, 31(41):8373–8385, 1998.

[79] W. D. Obal, M. G. McQuinn, and W. H. Sanders. Detecting and exploiting symmetry in discrete-state markov models. *IEEE Transactions on Reliability*, 56(4):643–654, 2007.

[80] M. Pelikan, D. E. Goldberg, and E. Cantu-Paz. Linkage problem, distribution estimation, and bayesian networks. *Evolutionary Computation*, 8(3):311–341, 2002.

[81] M. Petitjean. Chirality and symmetry measures: A transdisciplinary review. *Entropy*, 5(3):271–312, 2003.

[82] M. Petitjean. A definition of symmetry. *Symmetry: Culture and Science*, 18(2):99–119, 2007.

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

[84] Tizhoosh H. R. Reinforcement learning based on actions and opposite actions. In *Proc. International Conference on Artificial Intelligence and Machine Learning*, 2005.

[85] S. Rahnamayan, H. R. Tizhoosh, and M. Salama. Opposition-based differential evolution for optimization of noisy problems. In *Proc. IEEE Congress on Evolutionary Computation*, 2006.

[86] S. Rahnamayan, H. R. Tizhoosh, and M. Salama. A novel population initialization method for accelerating evolutionary algorithms. *Computers and Mathematics with Applications*, 53(10):1605–1614, 2007.

[87] S. Rahnamayan, H. R. Tizhoosh, and M. M. A. Salama. Opposition-based differential evolution. *IEEE Transactions on Evolutionary Computation*, 12(1):64–79.

[88] S. Rahnamayan, H. R. Tizhoosh, and M. Salamaa. Opposition versus randomness in soft computing techniques. *Applied Soft Computing*, 8(2):906–918, 2008.

[89] D. Ram, T. Sreenivas, and K. Subramaniam. Parallel simulated annealing algorithms. *Journal of Parallel and Distributed Computing*, 37:207–212, 1996.

[90] G. Ramakrishna and W. Mow. A new search for optimal binary arrays with minimum peak sidelobe levels. In *Proc. Sequences and Their Applications*, pages 355–360, 2005.

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

[92] R. Rastegar, A. Hariri, and M. Mazoochi. A convergence proof for the population-based incremental learning algorithm. In *Proc. IEEE International Conference on Tools with Artificial Intelligence*, pages 387–391, 2005.

[93] S. Reckow and V. Tresp. Proc. integrating ontological prior knowledge into relational learning. In *Neural Information Prosessing Systems Workshop: Structured Input-Structured Output*, 2008.

[94] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Proc. IEEE Conference on Neural Networks*, pages 586–591, 1993.

[95] J. Rose. *A Course on Group Theory*. Dover Publications, 1994.

[96] J. Rosen. *Symmetry in Science: An Introduction to the General Theory*. Springer-Verlag, 1995.

[97] J. Rosen. *Symmetry Rules*. Springer-Verlag, 2008.

[98] J. Rowe, M. Vose, and A. Wright. Structural search spaces and genetic operators. *Evolutionary Computation*, 12(4):461–493.

[99] J. Rowe, M. Vose, and A. Wright. Group properties of crossover and mutation. *Evolutionary Computation*, 10(2):151–184, 2002.

[100] J. Rowe, M. Vose, and A. Wright. Neighborhood graphs and symmetric genetic operators. In *Proc. Foundations of Genetic Algorithms*, pages 110–122, 2007.

[101] R. Rubinstein. *Monte Carlo Optimization, Simulation and Sensitivity of Queueing Networks*. Wiley, 1986.

[102] S. Rudlof and M. Koppen. Stochastic hill climbing with learning by vectors of normal distributions. In *Proc. First Online Workshop on Soft Computing*, 1996.

[103] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[104] S. Saarinen, R. Bramley, and G. Cybenko. Ill-conditioning in neural network training problems. *SIAM Journal on Scientific Computing*, 14(3):693–714, 1993.

[105] F. Sahba, Tizhoosh H. R., and M. A. Salama. Application of opposition-based reinforcement learning in image segmentation. In *Proc. IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pages 246–251, 2007.

[106] R. Schapire, M. Rochery, M. Rahim, and N. Gupta. Incorporating prior knowledge into boosting. In *Proc. International Conference on Machine Learning*, pages 538–545, 2002.

[107] J. Schmidhuber. Developmental robotics, optimal artificial curiosity, creativity, music, and the fine arts. *Connection Science*, 18(2):173–187, 2006.

[108] J. Schmidhuber. Driven by compression progress: A simple principle explains essential aspects of subjective beauty, novelty, surprise, interestingness, attention, curiosity, creativity, art, science, music, jokes. In *Proc. Anticipatory Behavior in Adaptive Learning Systems, from Sensorimotor to Higher-level Cognitive Capabilities*, page to appear, 2009.

[109] J. J. Schneider and S. Kirkpatrick. *Stochastic Optimization*. Springer, 2006.

[110] N. Schraudolph. Centering neural network gradient factors. In G. Orr and K. R. Muller, editors, *Neural Networks: Tricks of the Trade*, pages 207–226. Springer-Verlag, 1998.

[111] J. P. Schwefel. *Evolution and Optimum Seeking*. John Wiley & Sons, 1995.

[112] M. Sebag and A. Ducoulombier. Extending population-based incremental learning to continuous search spaces. *Lecture Notes in Computer Science*, 1498:418–427, 1998.

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

[114] M. Shokri, Tizhoosh H. R., and M. Kamel. Opposition-based q(lambda) algorithm. In *Proc. IEEE International Joint Conference on Neural Networks*, pages 254–261, 2006.

[115] M. Shokri, Tizhoosh H. R., and M. Kamel. Oppositional target domain estimation using grid-based simulation. *Applied Soft Computing*, 9(1):423–430, 2009.

[116] M. Shokri, Tizhoosh H. R., and M. S. Kamel. Opposition-based q(lambda) with non-markovian update. In *Proc. IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pages 288–295, 2007.

[117] P. Sinha. Symmetry sensing through computer vision and a facial image recognition system. *Forensic Science International*, 77(2):27–36, 1995.

[118] F. Southey and F. Karray. Approaching evolutionary robotics through population-based incremental learning. In *Proc. IEEE International Conference on Systems, Man, and Cybernetics*, volume 2, pages 710–715, 1999.

[119] F. Strocchi. *Symmetry Breaking*. Springer, 2007.

[120] Z. Sun, Z. Zhang, and H. Wang. Incorporating prior knowledge into kernel based regression. *Acta Automatica Sinica*, 34:1515–1521, 2008.

[121] H. J. Sussmann. Uniqueness of the weights for minimal feedforward nets with a given input-output map. *Neural Networks*, 5(4):589–593, 1992.

[122] G. Tezel and Y. Özbay. A new neural network with adaptive activation function for classification of ECG arrhythmias. In *Lecture Notes in Computer Science*, volume 4692, pages 1–8. 2007.

[123] D. Thompson and G. Bilbro. Sample-sort simulated annealing. *IEEE Transactions on Systems, Man and Cybernetics B*, 35(3):625–632, 2005.

[124] S. Thrun and B. Wegbreit. Shape from symmetry. In *Proc. Tenth IEEE International Conference on Computer Vision*, pages 1824–1831, 2005.

[125] H. R. Tizhoosh. Opposition-based learning: A new scheme for machine intelligence. In *Proc. International Conference on Computational Intelligence for Modelling, Control and Automation*, pages 695–701, 2005.

[126] H. R. Tizhoosh. Opposition-based reinforcement learning. *Journal of Advanced Computational Intelligence and Intelligent Informatics*, 10(5):578–585, 2006.

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

[128] E. Trentin. Networks with trainable amplitude of activation functions. *Neural Networks*, 14(4):471–493, 2001.

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

[130] V. Valsalam and R. Miikkulainen. Evolving symmetric and modular neural networks for distributed control. In *Proc. of the Genetic and Evolutionary Computation Conference*, pages 731–738, 2009.

[131] P. van der Smagt and G. Hirzinger. Solving the ill-conditioning in neural network learning. In G. Orr and K. R. Muller, editors, *Neural Networks: Tricks of the Trade*, pages 193–206. Springer-Verlag, 1998.

[132] L. Vecci, F. Piazza, and A. Uncini. Learning and approximation capabilities of adaptive spline activation function neural networks. *Neural Networks*, 11:259–270, 1998.

[133] 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.

[134] M. Ventresca and Tizhoosh H. R. Numerical condition of feedforward networks with opposite transfer functions. In *Proc. IEEE International Joint Conference on Neural Networks*, pages 3232–3239, 2008.

[135] M. Ventresca and Tizhoosh H. R. Improving gradient-based learning algorithms for large scale neural networks. In *Proc. IEEE International Joint Conference on Neural Networks*, page to appear, 2009.

[136] M. Ventresca, S. Rahnamayan, and H. R. Tizhoosh. *Computational Intelligence in Optimization-Applications and Implementations*, chapter The Use of Opposition for Decreasing Function Evaluations in Population-Based Search. Springer-Verlag.

[137] M. Ventresca, S. Rahnamayan, and H. R. Tizhoosh. A note on opposition versus randomness in soft computing techniques. *Applied Soft Computing*, 8(2), 2009.

[138] M. Ventresca and H. R. Tizhoosh. Improving the convergence of backpropagation by opposite transfer functions. In *Proc. IEEE International Joint Conference on Neural Networks*, pages 9527–9534, 2006.

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

[140] M. Ventresca and H. R. Tizhoosh. Simulated annealing with opposite neighbors. In *Proc. IEEE Symposium on Foundations of Computational Intelligence*, pages 186–192, 2007.

[141] M. Ventresca and H. R. Tizhoosh. A diversity maintaining population-based incremental learning algorithm. *Information Sciences*, 178(21):4038–4056, 2008.

[142] M. Ventresca and H. R. Tizhoosh. *Oppositional Concepts in Computational Intellligence*, chapter Two Frameworks for Improving Gradient-Based Learning Algorithms. Springer-Verlag, 2008.

[143] T. Vetter, T. Poggio, and H. Bulthoff. The importance of symmetry and virtual views in three-dimensional object recognition. *Current Biology*, 4:18 – 23, 1994.

[144] L. Wang, M. Sugiyama, C. Yang, K. Hatano, and J. Feng. Theory and algorithm for learning with dissimilarity functions. *Neural Computation*, 21(5):1459–1484, 2009.

[145] L. Wang, C. Yang, and J. Feng. On learning with dissimilarity functions. In *Proc. 24th International Conference on Machine Learning*, pages 991–998, 2007.

[146] W. Wenzel and K. Hamacher. A stochastic tunneling approach for global minimization. *Physical Review Letters*, 82(15):3003–3007, 1999.

[147] J. Weszka and A. Rosenfeld. Threshold evaluation techniques. *IEEE Transactions on Systems, Man and Cybernetics*, 8(8):622–629, 1978.

[148] H Weyl. *Symmetry.* Princeton University Press, 1983.

[149] D. Whitley. Fundamental principles of deception in genetic search. In G. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 221–241. Morgan Kaufmann, 1991.

[150] M. Wineberg and F. Oppacher. Metrics for population comparisons in evolutionary computation systems. In *Proc. Intelligent Systems and Control*, 2003.

[151] S. Xu and M. Zhang. Adaptive higher-order feedforward neural networks. In *Proc. IEEE International Joint Conference on Neural Networks*, volume 1, pages 328–332, 1999.

[152] S. Xu and M. Zhang. Justification of a neuron-adaptive activation function. In *Proc. IEEE International Joint Conference on Neural Networks*, volume 3, pages 465–470, 2000.

[153] 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.

[154] 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.

[155] X. Yao. Simulated annealing with extended neighbourhood. *International Journal of Computer Mathematics*, 40:169–189, 1991.

[156] T. Yoo, editor. *Insight into Images: Principles and Practice for Segmentation, Registration, and Image Analysis.* AK Peters, 2004.

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

[158] H. Zassenhaus. *The Theory of Groups*. Dover Publications, 1999.

[159] H. Zhang, J. Fritts, and S. Goldman. Image segmentation evaluation: A survey of unsupervised methods. *Computer Vision and Image Understanding*, 110:260–280, 2008.

[160] Q. Zhang, T. Wu, and Liu B. A population-based incremental learning algorithm with elitist strategy. In *Proc. Third International Conference on Natural Computation*, pages 583–587, 2007.

[161] K. Zhu and Z. Liu. Empirical study of population diversity in permutation-based genetic algorithm. *Lecture Notes in Computer Science*, 3201:537–547, 2004.