USING NEURAL NETWORKS AND GENETIC ALGORITHMS AS HEURISTICS FOR NP-COMPLETE PROBLEMS

by

William McDuff Spears

A Thesis Submitted to the
Faculty of the Graduate School
of
George Mason University
in Partial Fulfillment of
the Requirements for the Degree
of
Masters of Science in
Computer Science

Committee:

_____ Director

_____

_____

_____ Department Chairperson

_____ Dean of the Graduate School

Date: _____

Fall 1989

George Mason University

Fairfax, Virginia

Using Neural Networks and Genetic Algorithms as Heuristics for NP-Complete Problems

A thesis submitted in partial fulfillment of the requirements
for the degree of Master of Science at George Mason University.

By

William McDuff Spears
Bachelor of Arts in Mathematics
Johns Hopkins University, May 1984.

Director: Kenneth A. De Jong
Associate Professor
Department of Computer Science

Fall 1989
George Mason University
Fairfax, Virginia

Acknowledgements

There are a number of people who deserve thanks for making this thesis possible. I especially wish to thank my parents, for encouraging and supporting my education throughout my life; Ken De Jong, for suggesting this project and for his sound advice; my committee members, Henry Hamburger and Eugene Norris, for their time and interest; and my friend Diana Gordon for the numerous hours she spent correcting every aspect of my work. Finally, I wish to thank Frank Pipitone, Dan Hoey and the Machine Learning Group at the Naval Research Laboratory, for many valuable discussions. Any remaining flaws are the sole responsibility of the author.

Table of Contents

List of Tables

List of Figures

Abstract

USING NEURAL NETWORKS AND GENETIC ALGORITHMS
AS HEURISTICS FOR NP-COMPLETE PROBLEMS

William M. Spears, M.S.

George Mason University, 1989

Thesis Director: Dr. Kenneth A. De Jong

Paradigms for using neural networks (NNs) and genetic algorithms (GAs) to heuristically solve boolean satisfiability (SAT) problems are presented. Results are presented for *two-peak* and *false-peak* SAT problems. Since SAT is NP-Complete, any other NP-Complete problem can be transformed into an equivalent SAT problem in polynomial time, and solved via either paradigm. This technique is illustrated for hamiltonian circuit (HC) problems.

INTRODUCTION

One approach to discussing and comparing AI problem solving methods is to categorize them using the terms *strong* or *weak*. Generally, a weak method is one that has the property of wide applicability but, because it makes few assumptions about the problem domain, can suffer from combinatorially explosive solution costs when scaling to larger problems. State space search algorithms and random search are familiar examples of weak methods.

Frequently, scaling problems can be avoided by making sufficiently strong assumptions about the problem domain and exploiting these assumptions in the problem solving method. Many expert systems fall into this category in that they require and use large amounts of domain- and problem-specific knowledge in order to efficiently find solutions in enormously complex spaces. The difficulty with strong methods, of course, is their limited domain of applicability leading, generally, to significant redesign even when applying them to related problems.

These characterizations tend to make one feel trapped in the sense that one has to give up significant performance to achieve generality, and vice versa. However, it is becoming increasingly clear that there are two methodologies that fall in between these two extremes and offer in similar ways the possibility of powerful, yet general problem solving methods. These two methods are neural networks (NNs) and genetic algorithms (GAs).

Neural networks and genetic algorithms are similar in the sense that they achieve both power and generality by *demanding* that problems be mapped into their own particular representation in order to be solved. If a fairly natural mapping exists, impressive robust performance results. On the other hand, if the mapping is awkward and strained, both approaches behave much like the more traditional weak methods, yielding mediocre, unsatisfying results when scaling.

These observations suggest two general issues that deserve further study. First, we need to understand how severe the mapping problem is. Are there large classes of problems for which effective mappings exist? Clearly, if we have to spend a large amount of time and effort constructing a mapping for each new problem, we are not any better off than we would be if we used the more traditional, strong methods. The second major issue involves achieving a better understanding of the relationship between NNs and GAs. Are the representation issues and/or performance characteristics significantly different? Are there classes of problems handled much more effectively by one approach than the other?

This thesis is a first step in exploring these issues. It focuses on the application of GAs and NNs to a large, well-known class of combinatorially explosive problems: NP-complete problems. NP-Complete problems are problems that are not currently solvable in polynomial time. However, they are polynomially equivalent in the sense that any NP-Complete problem can be transformed into any other in polynomial time. Thus, if any NP-Complete problem can be solved in polynomial time, they all can [Garey79]. An example of an NP-Complete problem is the boolean satisfiability (SAT) problem: given an arbitrary boolean expression of n variables, does there exist an assignment to those variables such that the expression is true? Other familiar examples include job shop scheduling, bin packing, and traveling salesman (TSP) problems.

GAs and NNs have been used as heuristics for some NP-Complete problems [Goldberg89, Tagliarini87]. Unfortunately, the results have been mixed because although NP-complete problems are computationally equivalent in the complexity theoretic sense, they do not appear to be equivalent at all with respect to how well they map onto NN or GA representations. The TSP is a classic example of a problem that does not map naturally to either NNs [Gutzmann87] or GAs [De Jong89].

These observations suggest the following intriguing technique. Suppose we are able to identify an NP-complete problem that has an effective representation in the methodology of interest (GAs or NNs) and develop an efficient problem solver for that particular case. Other NP-complete problems that do not have effective representations can then be solved by transforming them into the canonical problem, solving it, and transforming the solution back to the original one.

This thesis outlines GA and NN paradigms that solve SAT problems, and uses hamiltonian circuit (HC) problems to illustrate how either paradigm can be used to solve other NP-Complete problems after they are transformed into equivalent SAT problems.† The remainder of the thesis is divided into four sections. The first section discusses the GA paradigm. The second section discusses the NN paradigm. The third section discusses the technique of solving HC problems using either paradigm after polynomial transformation into equivalent SAT problems. The final section summarizes the thesis.

---

† Note, this thesis does not show that P = NP. For a discussion on P and NP problems, see [Garey79].

1. GENETIC ALGORITHMS

In the book "Adaptation in Natural and Artificial Systems" [Holland75], John Holland lays the groundwork for GAs. GAs are based on a process of nature, namely, Darwinian evolution. In GAs, a population of individuals reproduce according to their *fitness* in an environment. The population of individuals, coupled with stochastic recombination operators, combine to perform an efficient domain-independent search strategy that makes few assumptions about the search space.

This section is divided into three subsections. First, an overview and survey of GAs is presented. Second, the application of GAs to SAT problems is described. The final subsection provides experimental results.

1.1. Overview

GAs consist of a population of individuals competing on a survival-of-the-fittest basis in an environment. The algorithm proceeds in steps called generations. During each generation, a new population of individuals (the *offspring*) is created from the old via application of genetic operators (*crossover*, *mutation*, and *inversion*), and evaluated as solutions to a given problem (the environment). Due to selective pressure, the population adapts to the environment over succeeding generations, evolving better solutions [Goldberg89]. If the environment is a function, GAs can be used for function optimization. In this case, each individual in a population is a sample point in the function space.

Over the years, GAs have been subject to extensive experimentation and theoretical analysis. The following subsections summarize important issues and indicate where future research may lead.

1.1.1. Representation

Historically, an individual in a GA is represented as a bit string of some length n. Each individual thus represents one sample point in a space of size 2^n. Analytical results are also typically based on these assumptions. Furthermore, the bit positions are assumed to be independent and context insensitive. While certain problems map well to such representations, many do not. Current research is exploring strings with non-binary alphabets, variable length strings, violations of independence, and tree representations [Bickel87].

These representations are all single-stranded in the sense that one piece of genetic material represents an individual. Such representations are termed *haploid*.† However, natural genetics makes use of double stranded chromosomes (*diploid*) as well. For example, suppose an individual is represented by two bit strings:

1010001010

0010101001

These double strands can contain different and possibly conflicting information. In nature, *dominance* is the primary mechanism for conflict resolution. Supposing 1 to dominate 0, the individual *phenotype* can be expressed as:

1010101011

Suppose the first bit represents eye color, with a 1 denoting brown eyes and a 0 denoting blue eyes. Then the 0 is a recessive gene, expressed only if both first bits are 0. Goldberg has shown that diploidy and dominance can be used in GAs to improve performance over time varying environments [Goldberg87].

---

† We only use the haploid representation in this thesis.

1.1.2. Genetic Operators

The standard genetic operators are mutation, crossover, and inversion. Mutation operates at the bit level. The population of individuals over generations represents a vast sea of bits that can be mutated at random. As an example, consider the individual:

1010101010

If the first bit is randomly chosen for mutation, the new individual is:

0010101010

Mutation rates are low, generally around one per thousand. Higher mutation rates are usually disruptive.

Crossover operates at the individual level. It swaps portions of genetic material between two individuals. This encourages the formation of genetic *building blocks*. This formation is a key to the power of the GA. As an example of crossover, consider the two individuals:

Individual 1: 1010101010

Individual 2: **1000010000**

Suppose the *crossover point* randomly occurs after the fifth bit.† Then each new individual receives one half of the original individual's genetic material:

Individual 1: 10101**10000**

Individual 2: **10000**01010

Recent work has concentrated on improving the effectiveness of crossover [Booker87]. Schaffer has experimented with adaptive crossover, where the GA itself learns the good crossover points [Schaffer87]. Finally, some conjectures about the best number of crossover points have been made and need to be examined [De Jong85].

---

† This is referred to as *one-point* crossover. *n-point* crossover randomly chooses n crossover points.

Inversion reorders the bits within an individual. Consider the individual:

1011101010

Suppose the positions after the second and sixth bits are randomly picked. Inverting the group of bits between those two positions yields: †

10**0111**1010

Inversion assumes that it is possible to change the physical location of the information on an individual without changing the functional interpretation. Evidence suggests that it is of little use in function optimization contexts since the meaning of each bit is position dependent [De Jong85]. However, in order-independent problems, Whitley has shown it to be useful when combined with crossover and *reproductive evaluation* [Whitley 87].

Genetic operators are tightly coupled with representations. Researchers are currently examining high-level operators to work with high-level list and tree representations. As De Jong points out, however, fundamental theorems need to be reproved in light of the change in underlying assumptions [De Jong85]. To date, little of this work has been done.

1.1.3. Evaluation Function

Each individual in a population has a fitness assigned by a payoff function. This payoff function represents the environment in which the population exists. Traditionally, these environments are relatively simple. However, many complex problems depend on statistical sampling. In this case, the payoff functions are approximations. Grefenstette has explored the relationship between the amount of time spent on individual evaluations and the efficiency of the genetic algorithm. Results show that some experiments benefited from making less accurate

† The inverted group of bits are in bold type.

evaluations and letting the GA run for more generations [Grefenstette85].

It is also traditional to have the payoff function return a scalar value. However, this is not always appropriate if more than one objective needs to be optimized. Schaffer describes a GA that performs multiple objective optimization using vector valued payoff functions [Schaffer85].

Until recently, the payoff functions have always measured the immediate worth of an individual's genetic material. However, Whitley argues that in biological systems, individuals are rated by their reproductive potential [Whitley87]. He claims that a GA using *reproductive evaluation* and inversion on real-valued, order-independent feature spaces, yields better solutions more efficiently.

1.1.4. Selection

During the selection phase of the genetic algorithm, the expected number of offspring that each individual will receive is determined, based on a relative fitness measure. The expected value is a real number indicating an average number of offspring that individual should receive over time. A sampling algorithm is used to convert the real expected values into integer numbers of offspring. It is important to provide consistent, accurate sampling while maintaining a constant population size. Previous sampling algorithms fail to minimize *bias* and *spread*.† Baker outlines a sampling algorithm (stochastic universal sampling) that has zero bias and minimal spread [Baker87].

Despite the improvements in sampling, finite populations still cause stochastic errors to accumulate, resulting in what researchers call *premature convergence*. Premature convergence refers to a decrease in genetic diversity before the

† Bias refers to the absolute difference between the individual's expected value and the sampling probability. Spread refers to the range of possible values for the number of offspring an individual receives [Baker87].

optimal solution is found. This is also referred to as the *exploration vs exploitation* problem. Global search performs exploration. Once the space has been globally sampled, local search can attempt to exploit the information already obtained. The problem is to maintain a good balance between exploration and exploitation. Too much exploration may result in a loss in efficiency. Too much exploitation may cause the system to miss good solutions. Theoretically, GAs strike a good balance between exploration and exploitation. In practice, however, the loss of genetic diversity represents a loss in exploration.

Recent work in GAs involves both predictions of premature convergence and possible solutions to the problem. Baker proposes using *percent involvement* as a predictor. Percent involvement is the percentage of the current population that contributes offspring to the next generation. Sudden drops in the percentage indicate premature convergence [Baker85]. Solutions to the premature convergence problem are similar in that all solutions attempt to maintain genetic diversity. Some proposed solutions use *crowding factors* [De Jong75], subpopulations [Schaffer85], *sharing functions* [Goldberg87], improved crossover [Booker87], selection by rank, and dynamic population size [Baker85].

1.1.5. Analysis

In a standard, fixed-length, binary string representation, each bit position represents a first-order hyperplane in the solution space. Analysis shows that all first-order hyperplanes are being sampled in parallel by the GA population. Furthermore, higher-order hyperplanes are sampled in parallel as well, although to lesser degrees of accuracy. The evaluation of individuals produces differential in payoff that increases sampling in the appropriate hyperplanes. Comparison of these sampling techniques with K-armed bandit problems shows the sampling to be near-optimal [Holland75]. This analysis results in the fundamental theorem of genetic algorithms that indicates a lower bound on the expected number of

representatives of a hyperplane in successive generations.

Recent work attempts to extend GA analysis and to define GA-Hard problems (in the sense that the GA is intentionally mislead). Bridges has given an exact expression for the expected number of representatives of a hyperplane in successive generations, given some simplifying assumptions [Bridges87]. Both Goldberg and Bethke have attempted to construct deliberately misleading problems for GAs [Bethke81, Goldberg87]. Such problems turn out to be hard to construct. Goldberg also extends De Jong's Markov chain analysis of "genetic drift" to include preferential selection (instead of random selection) [Goldberg87].

1.1.6. Applications

Early work in GA applications concentrated on N-dimensional function optimization of numerical parameters [De Jong75]. Such work indicated that parameter optimization was conceptually identical to optimizing parameterized task programs. This has led to the application of GAs to searching program spaces [Smith80]. Also, genetic algorithms have been applied to gas pipeline control [Goldberg85], semiconductor layout [Fourman85], keyboard configuration problems [Glover87], the Prisoner's Dilemma problem [Fujiko87], communication link speed design [Davis87], and battle management systems control [Kuchinski85].

GAs work well when the values for the parameters can be selected independently. This implies that the solution space consists of all combinations of parameter values. Recent applications of GAs to NP-Complete problems (job shop scheduling, bin packing, and the traveling salesman problem) violate the independence assumption. In these cases, the solution space consists of all permutations of parameter values. Such problems are considered GA-Hard in the sense that they do not map well to the standard genetic algorithm paradigm. Without modification, standard GAs perform poorly on permutation spaces. Current

research attempts to improve performance by adding domain knowledge to the genetic algorithm.

1.1.7. Domain Knowledge

Genetic algorithms are applicable to problems where little domain knowledge is known. However, Grefenstette points out that many opportunities exist for incorporating problem specific heuristics into GAs [Grefenstette87]. This knowledge can influence population initialization, evaluation, recombination operators, and local search.

In research, populations are usually initialized randomly. This provides a good test of the algorithm. In applications, however, reasonable solutions are often known. Judicious seeding of the population with good solutions is often advantageous. Care must be taken to ensure that the population is not biased away from even better solutions.

Considerable knowledge can be incorporated into the payoff function. In highly constrained problems, it is common to allow the payoff function to be a heuristic routine for constructing explicit, legal solutions from individuals. [Smith85] provides an example in which a heuristic payoff function produces legal bin packings from individuals that represent a set of objects. In such cases, the GAs are searching a space of constraints.

Recombination operators can also be a good source of problem specific knowledge. For example, a heuristic crossover is used to perform apportionment of credit at the level of genes in the traveling salesman problem [Grefenstette85]. Other examples of heuristic operators include *creep* [Davis87], *scramble*, and *flip* [Smith85].

Finally, although GAs often find good solutions quickly, they are not well suited for local search. Domain knowledge can often be used to improve the

search characteristics of GAs in local domains. As an example, it is known that the optimal tour in a TSP can not cross itself. The addition of a local search heuristic can greatly reduce the probability of a GA becoming stuck on these local minima [Grefenstette87].

1.1.8. Implementation

Until recently, GAs have been implemented on sequential computers. This has limited researchers to small populations, few generations, and simple payoff functions. However, GAs are inherently parallel in the sense that each individual in a population can be independently evaluated. This has led to parallel implementations of GAs on SIMD machines [Robertson87]. Further thought has also indicated that subpopulations may exist on complex processors, with a GA running on each. This has resulted in GA implementations on MIMD machines [Pettey87]. The theory of Punctuated Equilibria provides evolutionary support for the MIMD implementations [Cohoon87]. In either case, nearly linear decreases in execution time can result from the use of parallel architectures.

1.1.9. Connectionism

Recent enthusiasm for neural networks has led many researchers to combine GAs and connectionism in some fashion. Since GAs are evolutionary in nature, and neural networks are cognitive models, it is natural to wonder if GAs can construct good neural networks [Dolan87]. It may also be possible to merge the two paradigms [Ackley85] or to use thermodynamic operators in GAs [Sirag87]. At this time, the work is highly speculative and ad hoc, with little theoretical justification.

1.1.10. Summary

The preceding sections outline the current state of GA research and indicate possible future research interests. The next section discusses the application of GAs to one particular problem domain: boolean satisfiability.

1.2. GAs and SAT

In order to apply GAs to a particular problem, one must select an internal string representation for the solution space and define an external payoff function that assigns payoff to candidate solutions. Both components are critical to the success/failure of the GAs on the problem of interest.

1.2.1. Representation

SAT is a good choice for a canonical NP-complete problem because it appears to have a highly desirable GA string representation. Each individual in the population is a binary string of length N in which the i-th bit represents the truth value of the i-th boolean variable of the N boolean variables present in the boolean expression. It is hard to imagine a representation much better suited for use with GAs: it is fixed-length, binary, and context independent in the sense that the meaning of one bit is unaffected by changing the value of other bits [De Jong85].

1.2.2. Choosing a Payoff Function

After choosing a representation, the next step is to select an appropriate payoff function. The simplest and most natural function assigns a payoff of 1 to a candidate solution (string) if the values specified by that string result in the boolean expression evaluating to *true*, and 0 otherwise. However, for problems of interest, this payoff function would be 0 almost everywhere and would not support the formation of useful intermediate building blocks. Even though in the real

problem domain, partial solutions to SAT are not of much interest, they are critical components of a GA approach.

One approach to providing intermediate feedback would be to transform a given boolean expression into conjunctive normal form (CNF) and define the payoff to be the total number of top level conjuncts that evaluate to true. While this makes some intuitive sense, one cannot in general perform such transformations in polynomial time without introducing a large number of additional boolean variables that, in turn, combinatorially increase the size of the search space.

An alternative would be to assign payoff to individual subexpressions in the original expression and combine them in some way to generate a total payoff value. In this context the most natural approach is to define the value of *true* to be 1, the value of *false* to be 0, and to define the value of simple expressions as follows:

$$value\,(NOT\ expr) = 1 - value\,(expr)$$

$$value\,(AND\ expr_1\ \cdots\ expr_n) = MIN\,(value\,(expr_1)\ \cdots\ value\,(expr_n))$$

$$value\,(OR\ expr_1\ \cdots\ expr_n) = MAX\,(value\,(expr_1)\ \cdots\ value\,(expr_n))$$

Since any boolean expression can be broken down (parsed) into these basic elements, one has a systematic mechanism for assigning payoff. Unfortunately, this mechanism is no better than the original one since it still only assigns payoff values of 0 and 1 to both individual clauses and the entire expression.

However, a minor change to this mechanism can generate differential payoffs, namely:

$$value\,(AND\ expr_1\ \cdots\ expr_n) = AVERAGE\,(value\,(expr_1)\ \cdots\ value\,(expr_n))$$

This suggestion was made first by Smith [Smith79] and intuitively justified by arguing that this would reward ‘‘more nearly true’’ *AND* clauses. So, for

example, solutions to the boolean expression

$$X_1 \; AND \; (X_1 \; OR \; \overline{X_2} \;)$$

would be assigned payoffs as follows:

X_1	X_2	PAYOFF
0	0	(AVERAGE 0 (MAX (0 (1 - 0))) = 0.5
0	1	(AVERAGE 0 (MAX (0 (1 - 1))) = 0.0
1	0	(AVERAGE 1 (MAX (1 (1 - 0))) = 1.0
1	1	(AVERAGE 1 (MAX (1 (1 - 1))) = 1.0

Table 1: Sample Payoff Function

Notice that both of the correct solutions (lines 3 and 4) are assigned a payoff of 1 and, of the incorrect solutions (lines 1 and 2), line 1 gets higher payoff because it got half of the *AND* right.

This approach was used successfully by Smith and was initially adopted in the experiments. However, careful examination of this form of payoff function indicates some potential problems.

The first and fairly obvious property of using *AVERAGE* to evaluate *AND* clauses is that the payoff function is not invariant under standard boolean equivalency transformations. For example, it violates the associativity law:

$$value\,((X_1 \; AND \; X_2) \; AND \; X_3) \neq value\,(X_1 \; AND \; (X_2 \; AND \; X_3))$$

since

$$(AVE \; (AVE \; X_1 \; X_2) \; X_3) \neq (AVE \; X_1 \; (AVE \; X_2 \; X_3))$$

Attempts to construct alternative *differential payoff* functions that have this ideal

property of *payoff invariance* have had no success. However, one could argue that a weaker form of invariance might be adequate for use with GAs, namely, *truth invariance*. In other words, the payoff function should assign the same value (typically 1, but could even be a set of values) to all correct solutions of the given boolean expression, and should map all incorrect solutions into a set of values (typically $0 \leq \text{value} < 1$) that is distinct and lower than the correct ones. Since boolean transformations do not occur *while* the GAs are searching for solutions, the actual values assigned non-solutions would seem to be of much less importance than the fact that they are useful as a differential payoff to support the construction of partial solutions.

Unfortunately, the proposed payoff function does not even guarantee this second and weaker property of truth invariance as the following example shows:

$$X_1 \; OR \; X_2 = \overline{(\overline{X_1} \; AND \; \overline{X_2})} \qquad \textit{by De Morgan}$$

However,

$$(MAX \; X_1 \; X_2) \neq 1 - \frac{((1 - X_1) + (1 - X_2))}{2}$$

as can be seen in the following table:

X_1	X_2	Left side	Right side
0	0	0	0
0	1	1	1/2
1	0	1	1/2
1	1	1	1

Table 2: Violation of Truth Invariance

Notice that lines 2-4 are all solutions, but lines 2 and 3 are assigned a payoff of 1/2 after De Morgan's law has been applied.

In general, it can be shown that, although the payoff does not assign the value of 1 to non-solutions, it frequently assigns values less than 1 to perfectly good solutions and can potentially give higher payoff to non-solutions!

A careful analysis of boolean transformations, however, indicates that these problems *only* arise when De Morgan's laws are involved in *introducing* terms of the form $\overline{(AND \cdots)}$. This suggests a simple fix: preprocess each boolean expression by systematically applying De Morgan's laws to remove such constructs. It also suggests another interesting opportunity. Constructs of the form $\overline{(OR \cdots)}$ are computed correctly, but only take on 0/1 values. By using De Morgan's laws to convert these to *AND* constructs, additional differential payoff is introduced. Converting both forms is equivalent to reducing the scope of all *NOT*s to simple variables. Fortunately, unlike the conversion to CNF, this process has only linear complexity and can be done quickly and efficiently.

In summary, with the addition of this preprocessing step, an effective payoff function for applying GAs to boolean satisfiability problems results. This payoff function has the following properties: 1) it assigns a payoff value of 1 if and only

if the candidate solution is an actual solution; 2) it assigns values in the range $0 \leq$ value < 1 to all non-solutions; and 3) non-solutions receive differential payoff on the basis of how near their *AND* clauses are to being satisfied.

1.2.3. Possible Improvements to the Payoff Function

One way to view the problems discussed in the previous section is to note that many of the undesirable effects are due to the fact that, by choosing to evaluate *AND/OR* clauses with *AVERAGE/MAX*, the natural symmetry between *AND* and *OR* has been broken in the sense that *AND* clauses will have differential payoffs assigned to them while *OR* clauses will only be assigned 0/1. However, suppose that an *AND* node is evaluated by raising *AVERAGE* to some integer power *p*. This operator is still truth preserving (assuming the preprocessing step described above) and has several additional beneficial effects. First, it has the effect of reducing the *AND/OR* asymmetry by reducing the average score assigned to a false *AND* clause. In addition, it increases the differential between the payoff for *AND* clauses with only a few 1s and those that are nearly true.

On the other hand, as *p* approaches infinity, the function AVE^{p} behaves more and more like *MIN*, which means that the differential payoff property has been lost. This behavior suggests an interesting optimization experiment to determine a useful value for *p*. An experiment for determining *p* is described in the next section.

The previous sections describe an effective GA representation for SAT problems. The individual bit string naturally represents the 2^n possible assignments to the boolean variables. The payoff function, after applying De Morgan's laws, reflects the structure of the SAT problem and has appropriate properties. Finally, possible improvements to the payoff function are outlined. The next section presents initial results.

1.3. Results

All of the experiments described in this section have been performed using a Lucid Common Lisp implementation of the GAs. In all cases, the population size has been held fixed at 100, the standard two-point crossover operator has been applied at a 60% rate, the mutation rate is 0.1%, and selection is performed via Baker's SUS algorithm [Baker87].

After formulating SAT as an optimization problem, there appear to be some interesting issues concerning convergence to a solution. First of all, whenever a candidate evaluates to 1, a solution has been found and the search can be terminated. Conversely, there is strong motivation to continue the search until a solution is found (since nearly true expressions are not generally of much interest to the person formulating the problem). The difficulty, of course, is that on any particular run there is no guarantee that a solution will be found in a reasonable amount of time due to the increasing homogeneity (premature convergence) of the population as the search proceeds.

One approach would be to take extra measures to continue exploration by guaranteeing continuing diversity. Such measures as described in the earlier section on selection (See page 9). Unfortunately, these all have additional side effects that would need to be studied and controlled as well. A simpler approach using De Jong's measure of population homogeneity based on allele *convergence* [De Jong75] has been taken. When that measure exceeds 90%, the GA is restarted with a new random population. Consequently, in the experimental data presented in the subsequent sections, the evaluation counts reflect all of the GA restarts. Although this technique might seem a bit drastic, it appears to work quite well in practice.

Since the number of evaluations (trials) required to find a solution can vary quite a bit from one run to the next due to stochastic effects, all of the results presented here represent data averaged over at least 10 independent runs.

The first set of experiments involves constructing two families of boolean expressions for which the size and the difficulty of the problem can be controlled. The first family selected consists of two-peak (TP) expressions of the form:

$$(AND\ X_1\ \cdots\ X_n)\ \ OR\ \ (AND\ \overline{X_1}\ \cdots\ \overline{X_n})$$

that have exactly two solutions (all *false* and all *true*). By varying the number n of boolean variables, one can observe how the GAs perform as the size of the search space increases exponentially while the number of solutions remains fixed. The following table indicates the number of evaluations needed for the GA (using AVE^p, where $p = 1$). Both the mean number of evaluations and the standard deviation is reported. See Appendix 1 for complete data.

n	30	40	50	60	70	80	90
mean	1753	2855	4216	5485	7095	11167	13080
sd	360	676	641	947	1077	4977	5491

Table 3: Performance of GAs on the Two Peak Problems

Figure 1 presents a graph of the results, where the number of variables (bits) is plotted against both the mean number of evaluations (evals) and the log of the mean. It is clear that the differential payoff function is working as intended, and that the GAs can locate solutions to TP problems without much difficulty.

To make things a bit more difficult, the problem was modified slightly by turning one of the solutions into a false peak (FP) as follows:

$$(AND\ X_1\ \cdots\ X_n)\ \ OR\ \ (AND\ X_1\ \overline{X_1}\ \cdots\ \overline{X_n})$$

so that the previous all *false* solution is now almost correct and the only correct solution is that of all *true*. The following table indicates the number of

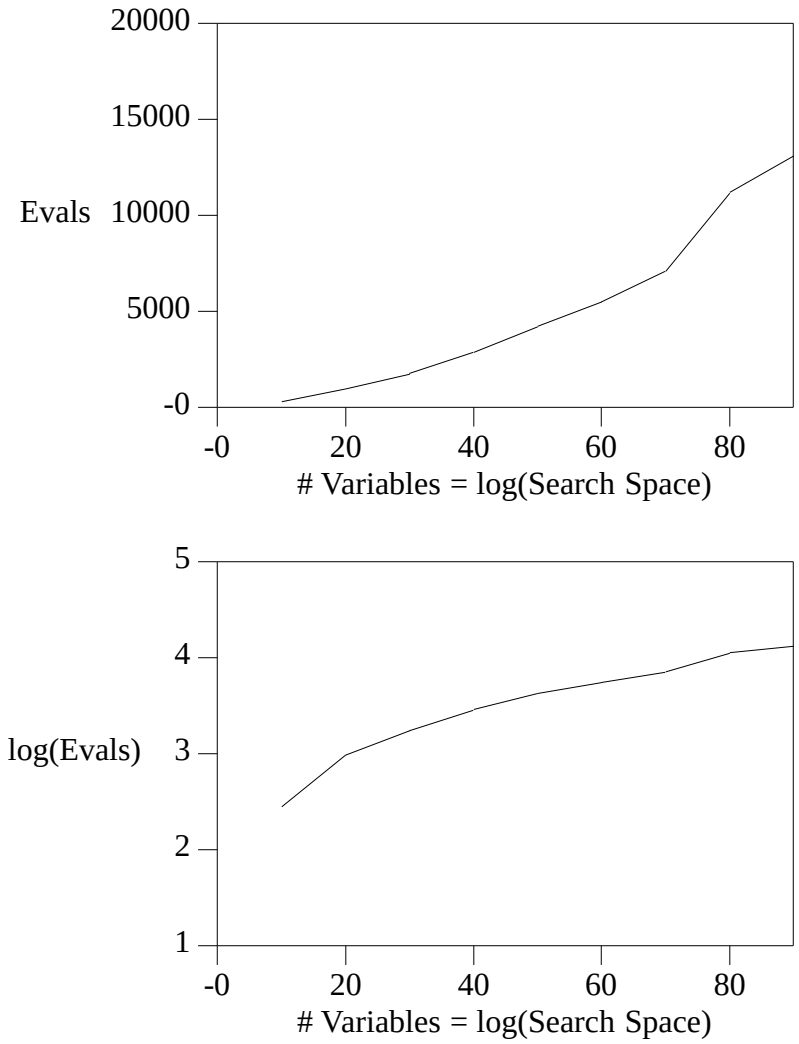

Figure 1: Performance of GAs on the Two Peak (TP) Problems

evaluations needed for the GA (using AVE^p, where $p = 1$).

n	30	40	50	60	70	80	90
mean	4805	8031	12167	18387	15617	18605	35153
sd	5289	8039	12797	21268	13390	16209	31731

Table 4: Performance of GAs on the False Peak Problems

Figure 2 presents a graph of the results of applying GAs to the FP family. As before, the GAs have no difficulty in finding the correct solution even in the presence of false peaks.

Since NP-Complete problems have no known polynomial-time algorithms, the log-log graphs are particularly interesting. Notice that, for both the TP and FP problems, a sub-linear curve is generated, indicating (as expected) a substantial improvement over systematic search. The form that these sub-linear curves take give some indication of the speedup (over systematic exhaustive search) obtained by using GAs. If, for example, these curves are all logarithmic in form, we have a polynomial-time algorithm for SAT! † Additional discussion of these curves occur in section 3.2.

Although the results so far have been satisfying, it is natural to investigate the effects of using $AVE^{\,p}$ in the payoff function for integer values of $p > 1$. The hypothesis is that initial increases in the value of p will improve performance, but that beyond a certain point performance will actually drop off as $AVE^{\,p}$ begins to more closely approximate *MIN*.

† Again, it has not been shown that P = NP.

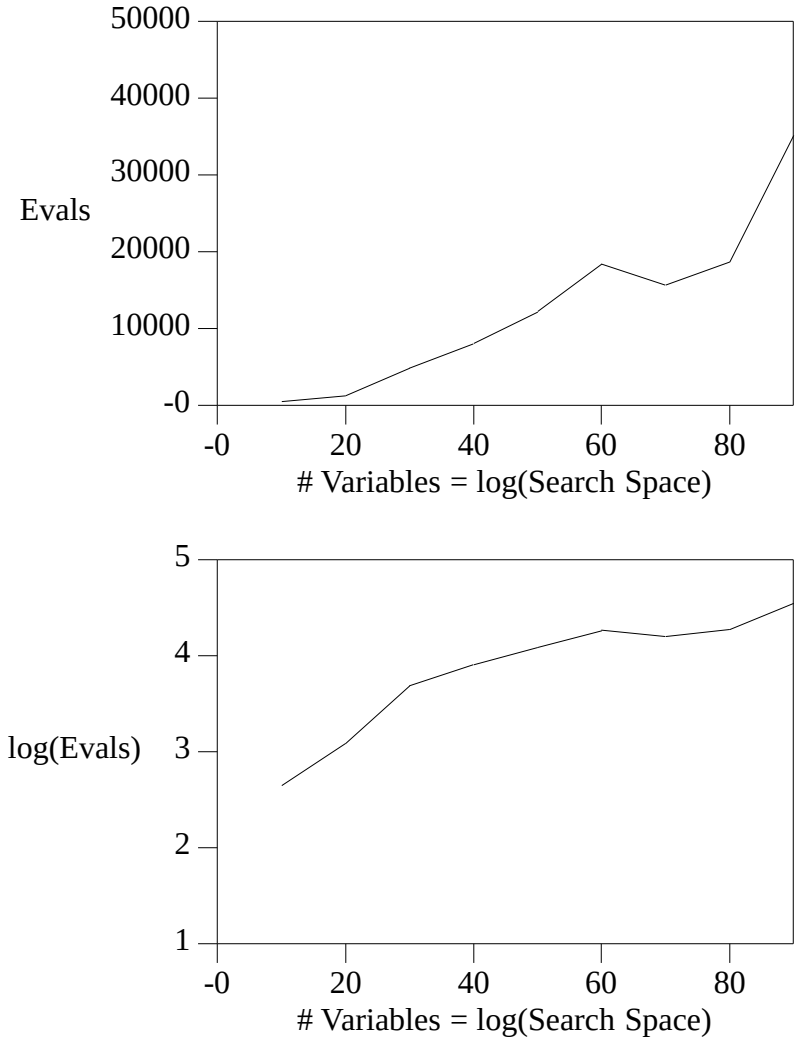

Figure 2: Performance of GAs on the False Peak (FP) Problems

The hypothesis was tested by re-running the GAs on the two families of problems (TP and FP) varying *p* from 2 to 5, and comparing their performance with the original results (*p* = 1). Figure 3 presents the results of the experiments. See Appendix 1 for a complete table of all data. Somewhat surprisingly, an optimum appears at *p* = 2.

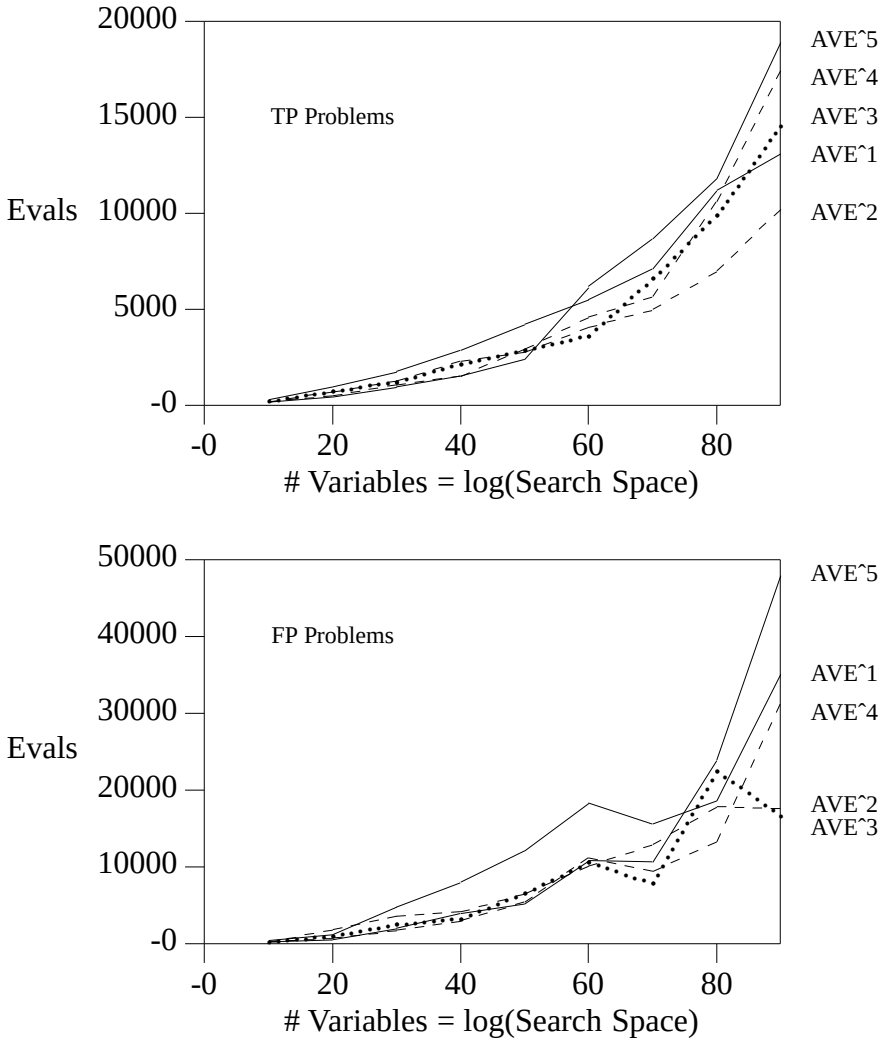

Figure 3: Performance of GAs using AVE^{p}

Figure 4 summarizes the performance of the GAs on the two families of SAT problems using AVE^{2} in the payoff function. As noted earlier, the log-log curves appear to be sub-linear. To get a better feeling for the form of these curves, both linear and quadratic curve fits were attempted. For both of the families of SAT problems, a quadratic form produces a better fit and, by using the coefficients of the quadratic form, the observed speedup can be calculated. The results are as follows:

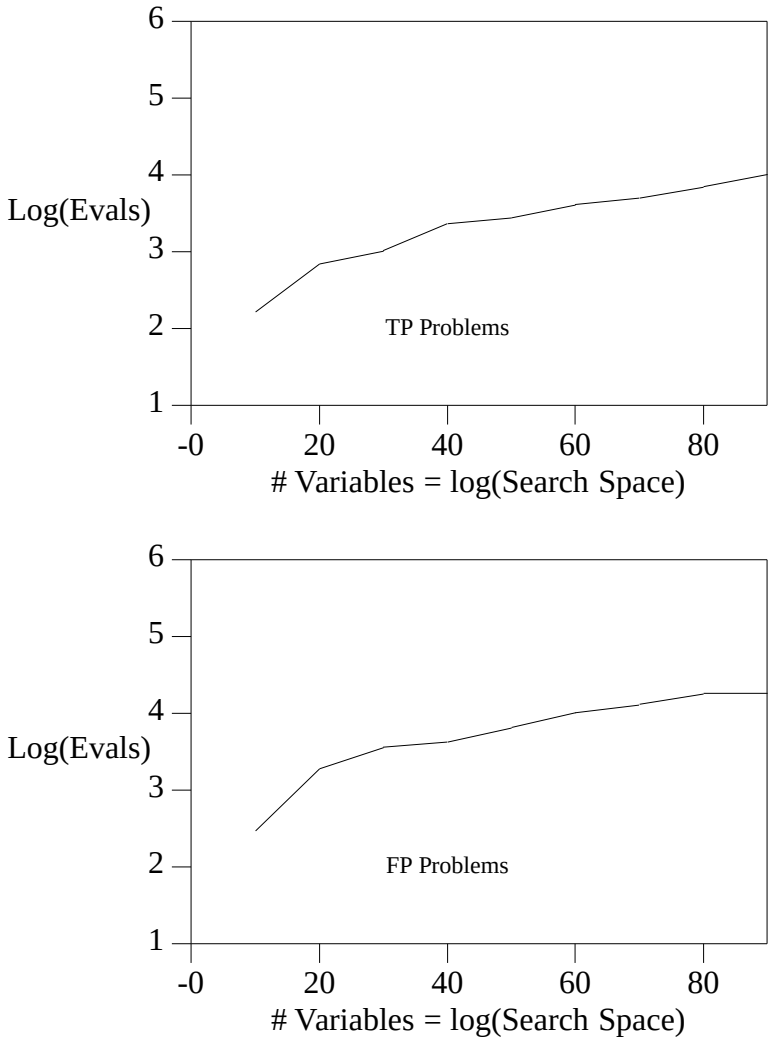

Figure 4: Summary Performance of GAs using AVE^2

TP speedup: $N^{7.28}$

FP speedup: $N^{6.25}$

One of the nice theoretical results in Holland's original analysis of the power of GAs is the *implicit parallelism* theorem that sets a lower bound of an N^3 speedup over systematic exhaustive search [Holland75]. This suggests that, in the worst case, GAs should not have to search more than the cube root of the search

space in order to find a solution and, in general, should do much better. One of the unexpected benefits of the experimental results presented here is substantial empirical evidence of just such speedups on SAT problems. Clearly, on the TP and FP problems the GA is performing better than the theoretical lower bound.

This section on GAs has outlined current GA research and possible future directions. The application of GAs to boolean satisfiability problems is fully described. Finally, experimental results are presented. With these initial encouraging results, it is natural to test the GAs on more naturally arising boolean expressions. The family of hamiltonian circuit problems provide a good source of interesting and hard SAT problems. The details and results of this work will be presented in Section 3.

As mentioned earlier, NNs are also used to heuristically solve NP-Complete problems. The next section describes a NN paradigm for solving boolean satisfiability problems.

2. NEURAL NETWORKS

The class of neural networks (NNs) is a subclass of parallel distributed processing (PDP) models [Rumelhart86]. These models assume that information processing is a result of interactions between simpler processing elements (nodes).

Neural network models consist of:

a) A set of processing nodes.

b) A state of activation for each node.

c) A pattern of connectivity among nodes (a graph).

d) A propagation rule.

e) An activation rule.

Each neural network consists of a number of processing elements (nodes) connected in a graph. Generally, each node represents some feature of the problem space being explored. Each node receives input values (from other nodes), maintains a state of activation, and sends output values (to other nodes). Frequently, the activation of a node is some real numeric quantity (usually ranging over a set of discrete values or taking on any real value within some range). However, sometimes the activation is simply binary.

A rule of propagation determines how the outputs from nodes are combined to form input for other nodes. Usually, this is simply a weighted sum in which the connections between nodes are assigned weights. Positive weights can represent excitatory connections, and negative weights inhibitory connections. Finally, every node has an activation rule which determines a new state of activation for the node, given a set of inputs to that node and its current state of activation.

There exists a large variety of neural networks. This work concentrates only on those that have been used often to solve combinatorial optimization problems: constraint satisfaction networks. This section is divided into three subsections. First, an overview of constraint satisfaction networks is presented. Second, the

application of a constraint satisfaction paradigm to SAT problems is described. The final section provides experimental results.

2.1. Overview

Hinton [Hinton77] has shown that constraint satisfaction networks can be used to find near-optimal solutions to problems with a large set of simultaneous constraints. In such a paradigm, each node represents a hypothesis and each connection a constraint among two hypotheses.

As an example, suppose that the nodes have binary activations (1 or -1) and are connected with symmetric weights. The sign of the weight indicates the polarity constraint between two nodes. For example, a positive weight might indicate that two nodes should have the same state. A negative weight would indicate that two nodes should have opposite states. The magnitude of the weight is proportional to the strength of the constraint.

Hopfield [Hopfield82] views such networks as "computational energy optimizers". In his paradigm (Hopfield networks), the activations of the nodes are binary (1 or -1) and the weights (constraints) are symmetric and real valued. The computational energy is the degree to which the desired constraints are satisfied. If a connection is positive, then the constraint is satisfied if both units are in the same state. If the connection is negative, the constraint is satisfied if both units are in opposite states. One way to express this mathematically is:

$$Energy_i = \sum_j w_{ij}\, a_i\, a_j$$

$$Energy = \sum_i Energy_i$$

The activation of node i is denoted a_i. The weight from node i to node j is denoted w_{ij}. $Energy_i$ reflects the local energy contribution of an individual node. Notice that if the weight is positive, the local energy will be positive only if the

two nodes have the same activations. If the weight is negative, the local energy will be positive only if the two nodes have opposite activations. In this discussion, optimization is equivalent to maximization.

Energy represents the global energy state of the system, and is the combination of local energy contributions from each node. If all constraints are satisfied, then each local energy contribution will be positive, and the total energy of the system will be maximized.

The preceding paragraphs have illustrated how a constraint satisfaction problem can be expressed as a computational energy optimization problem. In this paradigm, the weights are fixed, and only the activation states are allowed to change. In order to actually perform the task of energy optimization, each node must locally decide its own activation, based on neighboring information. One way to see this is to rewrite the above equations:

$$Energy_i = net_i \, a_i$$

$$net_i = \sum_j w_{ij} \, a_j$$

In this formulation, net_i represents the net input to a node from its immediate neighbors. If the net input is positive, then a_i should be 1 in order to have a positive energy contribution. If the net input is negative, then a_i should be -1. In other words, using only local neighbor information, each node can individually decide its own activation state. The combination of all nodes working in parallel leads to global energy optimization.

In summary, constraint satisfaction problems can be viewed as energy optimization problems. From this point of view, violating constraints decreases energy, while obeying constraints increases energy. The goal is to satisfy as many constraints as possible (and maximize energy).

Hopfield networks often become stuck in "local optima". In physics, this has a direct analogy with flaws in crystal formation. Such flaws are often avoided by heating the material and then cooling it slowly. Simulated annealing adds stochastic processing to Hopfield nets in much the same way. In simulated annealing, a system is considered to be a collection of particles with energies determined by the *Boltzmann distribution*:

$$\frac{probability\ (B)}{probability\ (A)} = e^{\frac{(Energy_B - Energy_A)}{T}}$$

Consider two states *A* and *B*. The Boltzmann distribution indicates that the ratio of probabilities of the two states is related to the energy difference between the two states. At high temperatures many possible energy states exist and the kinetic energy of the particles helps them escape from local optima. At very low temperatures the system freezes into one state, sometimes the global optimum. The activation of a node is computed using the net input to the node, the temperature, and the Boltzmann distribution. Mathematically:

$$probability\ (\ a_i\ (t) = 1) = \frac{1}{1+e^{(\frac{-net_i}{T})}}$$

At high temperatures, the probability goes to 1/2, indicating random choice. As the temperature decreases, positive net input yields a probability that approaches 1. Negative net input yields a probability that approaches 0. At very low temperatures the system degenerates into the deterministic Hopfield paradigm outlined earlier.

The system continues until some termination condition is satisfied. Typical conditions include the detection of the solution, low temperature (ie., the material freezes), or a time out.

The key to simulated annealing is the annealing schedule. It has been shown that if the schedule is sufficiently long (ie, the temperature drops extremely slowly with time) the network will find the global optimum [Geman84]. In practice, this is not feasible and experiments are made to determine good schedules that run quickly with reasonable results. However, this is only useful in applications where the same network is used more than once (a common occurrence). Application of simulated annealing to SAT will require a reasonable annealing schedule that is determined on the fly, before the network is executed.

As mentioned earlier, constraint satisfaction neural network approaches have been previously applied to NP-Complete problems. The IEEE First International Conference on Neural Networks (1987) includes a number of articles devoted to the solution of combinatorial optimization problems. Of special interest are the attempts to solve traveling salesman problems [Cervantes87, Gutzmann87]. In both cases, the authors note that the number of valid states is a decreasing fraction of the total number of states as the problem size increases. The problem occurs because the neural network is searching a space of combinations, while the valid states are permutations. Similar problems occur in other work [Tagliarini87, Gunn89].

The previous sections have shown that certain models of neural networks are useful for solving large systems of simultaneous constraints. Such models can also be considered to be function optimizers. The addition of simulated annealing introduces a stochastic element into the model that improves performance. The remainder of this thesis investigates the application of constraint satisfaction and simulated annealing to a canonical NP-Complete problem, boolean satisfiability.

2.2. NNs and SAT

Any application of a constraint satisfaction NN to some problem domain involves a selection of an appropriate graph (representation), as well as a specification of the domain specific constraints. Both components are critical to the success/failure of the NN on the problem of interest.

2.2.1. Representation

In general, choosing a good representation is often difficult. For the specific problem at hand (SAT), however, the previous work in genetic algorithms offers a helpful insight. Recall that a parse tree of the boolean expression is used to create a payoff function for the GA. This parse tree is also a natural NN graph representation that is easily created automatically, and is perfectly matched to the structure of the boolean expression.

2.2.2. System of Constraints

After choosing a representation, the next step is to select an appropriate system of constraints. Figure 5 presents an example parse tree of the simple boolean expression ($x1$ and $x2$) *or* ($x3$ and $x4$):

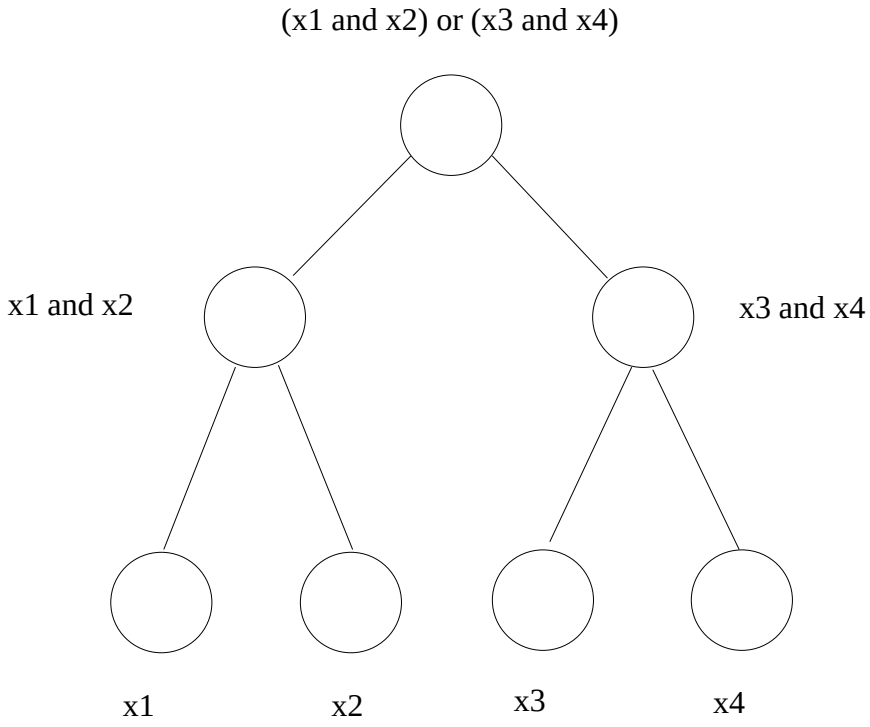

Figure 5: Example Parse Tree

In terms of constraint satisfaction, each node represents the hypothesis that a particular boolean subexpression is *true*. Suppose *true* is represented by a 1 and *false* by a -1. Since the goal is to satisfy the boolean expression, clearly the root node of the parse tree must be 1. The root node, the network topology, and the connection weights constrain the input nodes to be solutions of the original boolean expression.

There are two differences between the proposed network and a Hopfield net. First, since this network is based on a parse tree of a boolean expression it contains *AND*, *OR*, and *NOT* nodes. The nodes in a Hopfield net are of one type. Second, the graph is directed, with all edges directed toward the root (output) node. Each *AND*, *OR*, and *NOT* node has parents and/or children (not just *neighbors*). In a Hopfield net, the links are bi-directional and symmetric.

The asymmetries in the proposed network can be explained by a closer analysis of the constraints inherent in a boolean network. Each node can possibly be influenced by *upstream* constraints and *downstream* constraints. Upstream constraints represent constraints from nodes that are closer to the root, whereas downstream constraints represent constraints from nodes further from the root.

Downstream constraints flow from the children of a node. Suppose that some node is a *NOT* node. Then its activation should be opposite that of its child. An *AND* node should be true if and only if all of its children are true. An *OR* node should be true if and only if any of its children is true.

Upstream constraints flow from the parent of a node. Suppose the parent of a node is a *NOT*. Then the activation of the node should be opposite that of its parent. If the parent is an *AND* and it is true, then the node should be true. However, if the parent is an *AND* and it is false, then the node should be false if all siblings are true. Other situations are possible, but they do not constrain the node. Finally, if the parent is an *OR* and it is false, then the node should be false. However, if the parent is an *OR* and it is true, then the node should be true if all siblings are false. Again, other situations do not constrain the node.

Note from the above that there are two types of constraints implied. In the *NOT* example, nodes are constrained to be different. In the *AND* and *OR* examples, nodes are constrained to be similar. One interpretation of this is that:

> If a connection between two nodes is positive, the constraint is satisfied if both nodes are true or both nodes are false.
>
> If a connection between two nodes is negative, the constraint is satisfied if the nodes are in opposite states.

In other words, a positive connection enforces the idea that two nodes are both true or both false. A negative connection enforces the idea that both nodes are not the same. Note that this latter situation occurs only with *NOT* nodes. In fact, a *NOT* node and a negative connection are equivalent. Only *AND*, *OR*, and input nodes are necessary in this system. The asymmetries in direction of constraint can be formalized in the choice of net energy functions that are used to help compute node activations. The net energy functions are defined in the next section.

The previous paragraphs have shown how to express SAT problems as constraint satisfaction problems. The following subsections show how the constraint satisfaction problem can be viewed in an energy optimization framework. Paradigm I describes a first, but flawed attempt. Paradigm II corrects the flaws in Paradigm I.

2.2.3. Paradigm I

Given the set of boolean constraints outlined above, the task is to show how they can be expressed from an energy optimization viewpoint. Rules for calculating local net input and local energy contributions are presented.

As mentioned above, the neural network graph is based on the parse tree of the boolean expression. It is not a tree, however, because each instance of the same boolean variable is represented by only one node. Each *leaf* node, then, may have multiple parents. The resulting structure is a rooted, directed, acyclic graph. All edges in the graph are directed toward the root node.

Each node has a local energy based on its downstream net input, its upstream net input, and an external bias. Mathematically,

$$Energy_i = net_i \; a_i$$

$$net_i = \left(\sum_j Unet_{ij} \right) + Dnet_i + Bias_i$$

In this formulation, the net input to node *i*, net_i, is broken into three contributions. $Unet_{ij}$ represents the net upstream input of node *i* from parent *j*. Since some nodes may have multiple parents, the contribution from each must be summed. $Dnet_i$ represents the net downstream input of node *i* from its children. $Bias_i$ is an external bias to node *i*. Again, a_i can be 1 or -1, and denotes the activation of node *i*.

There is a one-to-one correspondence between the three net input contributions and the system of constraints that was outlined earlier. The bias is used to force the root node to be 1, since the primary external constraint is one of boolean satisfiability. The downstream/upstream net inputs are based on the downstream/upstream constraints. Before the definitions can be presented, however, some additional notation is necessary.

Denote $<i,j>$ as the directed edge from node *i* to node *j*. *E* is the set of edges in the graph. Let w_{ij} be the weight of that directed edge. We restrict w_{ij} to be 1 or -1. If w_{ij} is -1, this reflects the presence of a *NOT* constraint. If there is no *NOT* constraint, w_{ij} is 1. Furthermore, define $AND(i)$ and $OR(i)$ as predicates that return *true* if node *i* is an *AND* or *OR*. It is now possible to define the downstream constraint, $D(i)$, as:

$$D(i) = [(AND(i) \;\&\; (\forall j)((<j,i> \in E) \rightarrow (a_j w_{ji} = 1))) \text{ v}$$

$$(OR(i) \;\&\; (\exists j)((<j,i> \in E) \rightarrow (a_j w_{ji} = 1)))]$$

$D(i)$ returns *true* if an *AND* has all children *true*, or an *OR* has at least one child *true*. The truth of a child is the product of the activation of the child and the weight. If the weight is -1, a *NOT* constraint is implied, reversing the truth value of the child. Given this definition, the rules for $Dnet_i$ can then be expressed precisely:

$$(\forall i)[(Dnet_i = 1) \leftrightarrow D(i)]$$

$$(\forall i)[(Dnet_i = -1) \leftrightarrow \overline{D(i)}]$$

In a similar fashion, define the upstream constraint $U(i,j)$ of a node *i* from a parent *j*:

$$U(i,j) = [(AND(j)\ \&$$

$$((a_j = 1)\ \text{v}\ [(a_j = -1)\ \&\ (\forall k)(((<k,j> \in E)\&(k \neq i)) \rightarrow (a_k w_{kj} = 1))]))\ \text{v}$$

$$(OR(j)\ \&$$

$$((a_j = -1)\ \text{v}\ [(a_j = 1)\ \&\ (\forall k)(((<k,j> \in E)\&(k \neq i)) \rightarrow (a_k w_{kj} = -1))]))]$$

Given this definition, the rules for $Unet_{ij}$ can then be expressed precisely:

$$(\forall i)(\forall j)[Unet_{ij} = a_j w_{ij} \leftrightarrow U(i,j)]$$

$$(\forall i)(\forall j)[Unet_{ij} = 0 \leftrightarrow \overline{U(i,j)}]$$

Again, these rules correspond with the upstream constraints defined earlier.† A constraint can occur if the parent is an *AND* or an *OR*. If it is an *AND*, and $a_j = 1$, then node *i* should have an activation $a_i = a_j w_{ij} = w_{ij}$. If $a_j = -1$ and all siblings are true ($a_k w_{kj} = 1$), then node *i* should have an activation $a_i = a_j w_{ij} = -w_{ij}$. As before, the truth of children of the parent is a product of the child's activation and the weight. Similar constraints occur when the parent is an *OR*.

It is important to note a difference between the downstream constraint and upstream constraint definitions. If $D(i)$ is *true* or *false*, a real logical constraint is implied. This occurs due to the direction of flow of information. Regardless of the states of the children, the state of the node can be determined unambiguously. However, upstream constraints are dynamic, and only occur in certain circumstances, namely those situations when $U(i,j)$ is *true*. Unfortunately, there are other circumstances when there are no upstream constraints. In this case, $U(i,j)$ is *false*. Given no upstream constraints, then, what is the upstream net input to a

† For a complete explanation, please see the earlier discussion of upstream constraints on page 34.

node? Intuitively, if there is no constraint, there is also no reason to suspect any particular activation for a node. For this reason, $Unet_{ij}$ is 0 when $U(i,j)$ is *false*.

As mentioned earlier, $Bias_i$ is used to force the root node to 1, since the goal is to satisfy the boolean expression. The rules for bias can be expressed simply as:

$$(\forall i)[Root(i) \leftrightarrow (Bias_i = 100)]$$
$$(\forall i)[\overline{Root(i)} \leftrightarrow (Bias_i = 0)]$$

$Root(i)$ is a predicate that returns true if and only if node i is the root node. High bias results in high net input, which in turn implies a high probability that the node will be on ($a_i = 1$). The choice of 100 is completely arbitrary. Any large number will do. More will be said on bias later.

This section has shown how to map SAT problems into an energy optimization paradigm. The next section presents some problems with this approach.

2.2.4. Problems with Paradigm I

An ideal energy definition should possess two qualities. First, solutions should have the highest energy. This energy should be predictable. Second, all non-solutions should have low energy, with *better* non-solutions having progressively higher energy. Unfortunately, the formulation given above does not possess these two qualities, although on the whole a system based on the outlined definitions performs quite well.

Some analysis indicates that several problems exist, all related to the treatment of non-existent upstream constraints. As mentioned above, intuitively it seems reasonable to let $Unet_{ij}$ be 0 when there are no upstream constraints ($U(i,j)$ is *false*), since there is no reason to suspect any particular activation for a node with no constraints. However, suppose a leaf node has no upstream constraints and no external bias. Since it does not have children it has no downstream

constraint. In this case, a node has no net input. Furthermore:

$$Energy_i = net_i \; a_i = 0$$

Although this seems reasonable, it can be seen that the node is contributing no local energy, despite the fact that it is not violating any constraints. The system will only contribute local energy if it satisfies some constraint, but will not contribute local energy if no constraint exists (although either situation is equally desirable). Since the presence of upstream constraints is a dynamic situation, dependent on the activations of neighboring nodes (siblings), it is impossible to predict ahead of time the energy of a solution.

This problem illustrates an even deeper flaw. The outlined paradigm makes a distinction between systems that satisfy all constraints and those that have no constraints. In reality, either situation is equally worthwhile, for both do not violate any constraints. As mentioned earlier, a node that has no constraints contributes no local energy, while a node that satisfies a constraint contributes local energy. Recall that:

$$net_i = (\sum_j Unet_{ij}) + Dnet_i + Bias_i$$

Denote $Unet_i$ as the summation of the $Unet_{ij}$. $Unet_i$ is the only place where it is possible to have non-existent constraints (since they only appear in upstream situations). However, the summation rewards systems that have no constraints differently from systems that have satisfied constraints. In fact, it is possible for systems to achieve higher energy by creating a few more constraints dynamically, violating a few, and satisfying the rest. The result is a non-satisfied system with higher energy than a previous satisfied system. For example, suppose there is an *AND* node with six children (each weight is 1). The following table summarizes the activations of the children, their net inputs, and their local energy contributions.

Node	*Activation*	*Net Input*	*Energy*
Child 1	*1*	*0*	*0*
Child 2	*1*	*0*	*0*
Child 3	*1*	*0*	*0*
Child 4	*1*	*0*	*0*
Child 5	*1*	*0*	*0*
Child 6	*-1*	*-1*	*1*
AND	*-1*	*-1*	*1*

Table 5: Energy of Satisfied System

This system of activations violates no constraints and has a global energy of 2. However, by simply switching the *AND* from -1 to 1, we can increase the global energy.

Node	*Activation*	*Net Input*	*Energy*
Child 1	*1*	*1*	*1*
Child 2	*1*	*1*	*1*
Child 3	*1*	*1*	*1*
Child 4	*1*	*1*	*1*
Child 5	*1*	*1*	*1*
Child 6	*-1*	*1*	*-1*
AND	*1*	*-1*	*-1*

Table 6: Energy of Non-Satisfied System

This system has a global energy of 3. However, this system violates constraints. This example illustrates how dynamically changing constraints can be played off one another to increase energy in a less desirable situation.

Finally, notice that $Unet_i$ can increase as the number of parents to node i increases. This implies that, for any node in some graph, the maximum $Unet_i$ depends on the graph itself. Furthermore, each node is likely to have differing maximums. Recall that:

$$probability\ (\ a_i\ (t) = 1) = \frac{1}{1+e^{(\frac{-net_i}{T})}}$$

A reasonable annealing schedule will start at some maximum temperature, and slowly decrease to some minimum temperature. The maximum temperature should allow for a fair amount of random noise in the system (exploration), while the minimum temperature will signify determinism (exploitation). Ideally, the maximum and minimum temperatures should be chosen to yield reasonable probability distributions.† However, these distributions are also functions of the net input. Since each node may have different maximum net inputs, given some global temperature, the probability distribution varies from node to node. As a result, the search is performed with varying levels of noise across the nodes. This is a source of inefficiency, since it is likely that some nodes are operating with too much noise.

In summary, the outlined paradigm, although reasonable in practice, can lead to anomalies in energy computation. First, it is difficult to predict the energy of a solution. Second, the anomalies create an energy surface that is less than ideal, since non-solutions may have higher energy than solutions. Another difficulty with the paradigm is that, for some global temperature, the probability distribution varies across the nodes. This makes it hard to choose a reasonable annealing schedule. Fortunately, these problems can be fixed with minimal effort.

† A reasonable probability distribution will provide a good balance between exploration and exploitation.

2.2.5. Paradigm II

It is clear from the previous discussion that some changes are needed in the rules for determining net input and local energy contributions. Each problem stems from the manner in which the upstream net input is calculated. These problems can be fixed by monitoring the number of upstream constraints as they occur dynamically, and normalizing the net input based on these constraints.

We define c_i as the number of upstream constraints on node i at some particular time. More precisely:

$$c_i = \mid \{j \mid (<i,j> \in E) \,\&\, U(i,j)\} \mid$$

Given the number of constraints, the rules for computing the total upstream net input $Unet_i$ can be rewritten:

$$(\forall i)[c_i = 0 \rightarrow Unet_i = a_i]$$

$$(\forall i)[c_i \neq 0 \rightarrow Unet_i = \frac{\sum_j Unet_{ij}}{c_i}]$$

Suppose that a leaf node has no upstream constraints ($c_i = 0$). Then $Unet_i = a_i$. We assume that it has no bias. Since it also has no downstream net input, $Energy_i = a_i a_i = 1$. Unlike the previous paradigm, the system is now rewarded for having no upstream constraints by allowing it to contribute local energy. We also justify this change intuitively as a form of Occam's razor. If there are no upstream constraints, there is no need to change the activation.

Note also that $Unet_i$ has been normalized to fall within -1 and 1. In order to fully normalize the net input net_i, the following is rewritten:

$$(\forall i)[(INTERIOR(i) \,\&\, (c_i = 0)) \rightarrow (net_i = Dnet_i + Bias_i)]$$

$$(\forall i)[(INTERIOR(i) \,\&\, (c_i \neq 0)) \rightarrow (net_i = \frac{Dnet_i + Unet_i}{2} + Bias_i)]$$

The predicate $INTERIOR(i)$ is true if i is not the root node or a leaf node. If a node is an interior node, and there are no upstream constraints, we simply consider the downstream constraint and the bias. If, however, there are upstream constraints, they are treated equally with the downstream constraint.

Neglecting the bias, several points can be made. First, net_i has been normalized to lie between -1 and 1. This solves the problem associated with determining a good annealing schedule, since for a given temperature, each node has an identical probability distribution. Second, $Energy_i = a_i a_i = 1$ if and only if either all constraints are satisfied or there are no constraints. If the node violates all constraints, then $E_i = -a_i a_i = -1$. With these changes, it becomes impossible to dynamically create new constraints, violate a few, and increase energy. The system can only achieve maximum energy if it satisfies all constraints. Finally, the energy of a solution can be determined (including bias):

$$Energy = n + \sum_i Bias_i$$

where n is the number of nodes in the system. This solves the energy anomaly problems, since it is now easy to determine the energy of a solution, all solutions have this energy, and all non-solutions have lower energy.

Until this point, very little mention has been made of $Bias_i$. In fact, it is only used to bias the root node to 1. Depending on the format of the boolean expression, however, it is sometimes possible to be sure of the activations of nodes other than the root. For example, since the boolean expression (A *AND* $\overline{(B \text{ OR } C)}$) must be true, it is a trivial task to deduce that A must be true, while B and C are false. The previous rules for bias are inadequate for expressing these new bias constraints.

More precisely, denote $B(i,j)$ as a predicate that is true if and only if there is a bias constraint from parent j to node i.

$$B(i,j) = [(AND(j) \;\&\; (<i,j> \in E) \;\&\; (Bias_j = 100)) \text{ v}$$

$$(OR(j) \;\&\; (<i,j> \in E) \;\&\; (Bias_j = -100))]$$

We further denote $Bias_{ij}$ as the bias value contributed from parent j to node i. We can rewrite the rules for bias as:

$$(\forall i)[Root(i) \rightarrow (Bias_i = 100)]$$

$$(\forall i)(\forall j)[\overline{(Root(i)} \;\&\; B(i,j) \rightarrow (Bias_{ij} = Bias_j w_{ij})]$$

$$(\forall i)(\forall j)[\overline{(Root(i)} \;\&\; \overline{B(i,j)} \leftrightarrow (Bias_{ij} = 0)]$$

The root node still has a bias of 100. The second rule states that a node's bias is influenced by its parent. If the parent is an *AND* with high bias (and $w_{ij} = 1$), then the node should have high bias. If the parent is an *OR* with low bias (and $w_{ij} = 1$), the node should have low bias. The bias is reversed when $w_{ij} = -1$. The third rule states that there is no bias contribution if $B(i,j)$ is false. Here, high bias is denoted with 100 and low bias with −100, although the system is relatively insensitive to the choice of numbers.

Since a node can have multiple parents, a node's bias is influenced by all of its parents. Four possible scenarios can arise. First, if there are no bias constraints ($B(i,j)$ is false for all parents j), then $Bias_i = 0$. Second, if some parents constrain the bias to be high, while no parent constrains the bias to be low, then $Bias_i = 100$. Third, if some parents constrain the bias to be low, while no parent constrains the bias to be high, then $Bias_i = -100$. Finally, if some parents constrain the bias to be high, while others constrain the bias to be low, the boolean expression is unsatisfiable.

The above rules can be applied during the parse of the boolean expression. This information, coupled with the system of constraints, constitutes the maximum information easily derivable from the boolean expression.

In summary, this section has outlined a paradigm that maps SAT problems into an equivalent simulated annealing constraint satisfaction problem. The energy function defined guarantees that solutions have a predictable maximum energy, while non-solutions have lower energy. By normalizing the net input, it is possible to have reasonable probability distributions across all nodes at some temperature. Finally, introducing bias allows the system to make maximum use of all available information. The results of applying this paradigm to the TP and FP problems are described below.

2.3. Results

Paradigm II is fully implemented in C, with Common Lisp as a front end. The Lisp front end takes as input a boolean expression in prefix notation and outputs a description of the graph structure in C. This code is then linked with C simulated annealing code, compiled, and executed.

The program loops through trials until the solution is found. At the start of each trial, the temperature is set to some maximum. The annealing schedule is determined from the starting maximum temperature and a decay rate. The decay rate provides an exponential decay of the temperature, until some minimum (freezing) temperature is reached. Each trial, the decay rate is slightly decreased, allowing the system to cool more slowly. If the minimum temperature is reached within a trial, and the solution is not found, a new trial is started. If the solution is found, the program terminates.

There are two methods for determining whether a solution has occurred. First, since the energy of a solution can be predicted, the system energy can be compared with the predicted solution energy. Second, it is also possible to check the leaf nodes against the original boolean expression. If the expression is satisfied, the problem is solved.

The second method is better for two reasons. First, computing the energy of the system is expensive. Second, it is possible for the leaf nodes to satisfy the original boolean expression, while the interior nodes still violate some constraints. In other words, although the highest energy state is guaranteed to be a solution, useful solutions can be found at lower energy levels. The first method will not detect these redundant solutions, while the second method will.

In summary, Paradigm II is fully implemented, and utilizes all available boolean information. The program loops through trials until the leaf nodes satisfy the boolean expression. During each trial, the temperature decays from some maximum to some minimum according to a decay rate. The decay rate is slightly decreased from trial to trial.

Paradigm II was first tested on the TP problems. For the NN, one evaluation corresponds to updating the activation of each node in the parse tree exactly once. Since both the GA and the NN use the parse tree for evaluation, they have equivalent complexity. The following table indicates the number of evaluations needed for the NN. Both the mean number of evaluations and the standard deviation are reported.

n	10	20	30	40	50	60	70	80	90
mean	6	12	24	35	51	64	78	97	113
sd	4	6	11	14	14	15	15	18	19

Table 7: Performance of NNs on the Two Peak Problems

Figure 6 presents a graph of the results. It is clear that the NN can locate solutions to TP problems without difficulty. For this simple problem, the NN highly outperforms the GA.

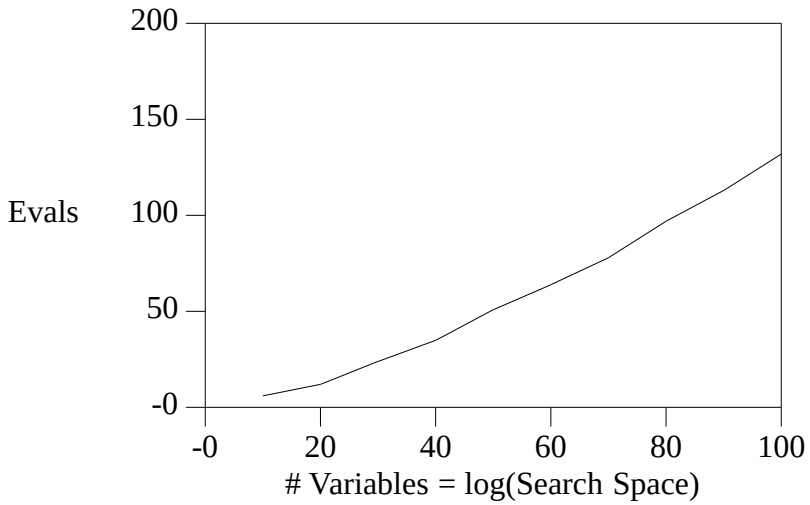

Figure 6: Performance of NN on the Two Peak (TP) Problems

Paradigm II was then tested on the FP problems. The following table indicates the number of evaluations needed for the NN. Both the mean number of evaluations and the standard deviation are reported.

n	10	20	30	40	50	60	70	80	90
mean	49	18	31	44	76	77	90	108	126
sd	106	10	16	18	127	25	26	27	30

Table 8: Performance of NNs on the False Peak Problems

Figure 7 presents a graph of the results. As before, the NN has no difficulty in finding the correct solution even in the presence of false peaks. Again, the GA is highly outperformed.

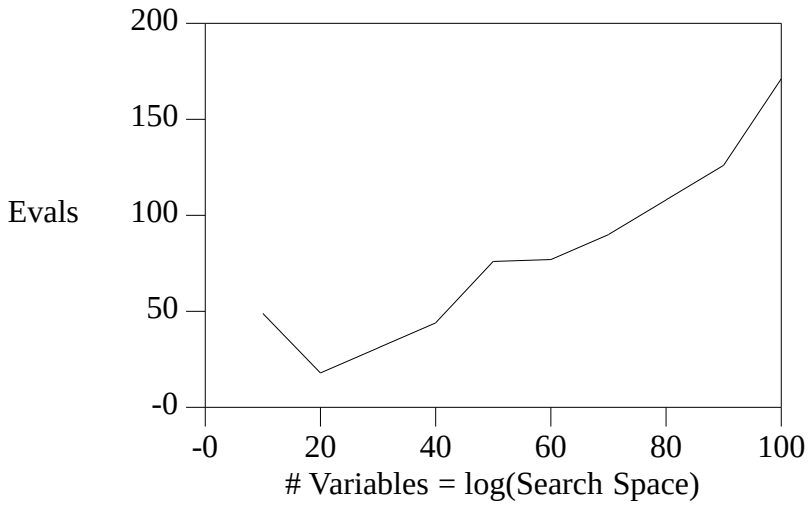

Figure 7: Performance of the NN on the False Peak (FP) Problems

This section on NNs has outlined current NN research on combinatorial optimization problems. An application of a NN to boolean satisfiability problems is fully described. Finally, experimental results are presented. With these initial encouraging results, it is natural to test the NN on more naturally arising boolean expressions. The family of hamiltonian circuit problems provide a good source of interesting and hard SAT problems. The details and results of this work will be presented in the next section.

3. NP-COMPLETENESS

The previous sections outline GA and NN paradigms for heuristically solving SAT problems. Experiments with simple boolean expressions are encouraging. Since SAT is NP-Complete, other NP-Complete problems can be transformed into equivalent SAT problems in polynomial time, and solved via either paradigm. Such problems can also provide more challenging tests for the GA and NN paradigms.

This section concentrates on one family of NP-Complete problem: hamiltonian circuit (HC) problems. This section is divided into two subsections. The first subsection describes hamiltonian circuit problems and the polynomial transformation used to convert these problems into boolean satisfiability problems. The final section discusses the results of applying both the GA and NN paradigms to some HC problems.

3.1. Hamiltonian Circuit Problems

The hamiltonian circuit (HC) problem consists of finding a tour through a directed graph that touches all nodes exactly once. Clearly, if a graph is fully connected, this is an easy task. However, as edges are removed the problem becomes much more difficult, and the general problem is known to be NP-Complete.

Attempting to solve this problem directly with GAs or NNs raises many of the same representation issues as in the case of traveling salesman problems [De Jong85, Grefenstette85]. However, it is possible to construct a polynomial-time transformation from HC problems to SAT problems.

The definition of the HC problem implies that, for any solution, each node must have exactly one input edge and one output edge. If any tour violates this constraint, it cannot be a solution. Therefore, an equivalent boolean expression is

simply the conjunction of terms indicating valid edge combinations for each node.

Figure 8 presents an example hamiltonian circuit problem. Consider node *d*. Node *d* has two output edges and one input edge. The output edge constraints are given by the exclusive-or, ((*db* and $\overline{de}$) *or* ($\overline{db}$ and *de*)). The input edge is described simply by *cd*. The assignments to the edge variables indicate which edges make up a tour, with a value of *true* indicating an edge is included and a value of *false* if it is not. The final boolean expression is a conjunction of similar expressions derived for each node. This transformation is computed in polynomial time, and if a solution to the HC problem exists, then the boolean expression is satisfiable.

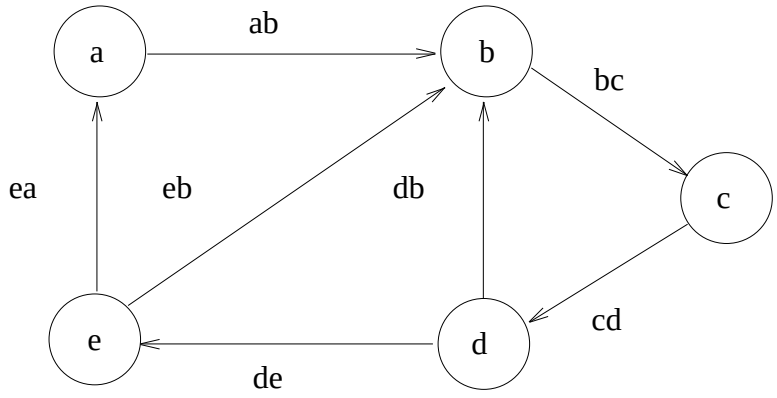

Figure 8: Sample Hamiltonian Circuit Problem

Unfortunately, the above constraints are necessary but not sufficient in the general case. In other words, if the boolean expression is satisfiable there is no guarantee that a solution to the HC problem exists. Figure 9 provides a specific example.

Suppose that all edges in Figure 9 are true, except for *ef* and *bc*. Then each node has one input edge and one output edge. The boolean expression is satisfied,

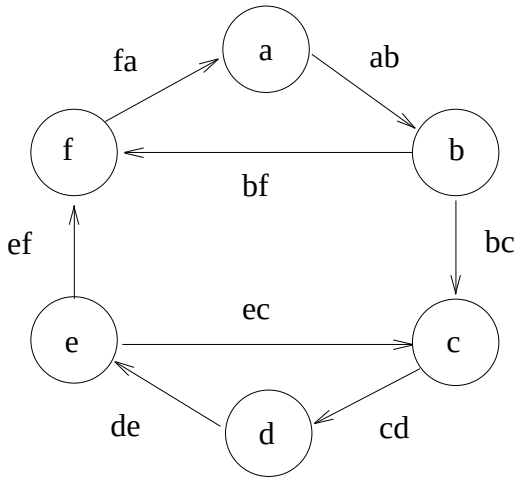

Figure 9: Another Hamiltonian Circuit Problem

yet the assignments do not form a hamiltonian circuit. Fortunately, for the problems described in the next section, the constraints are necessary and sufficient.

3.2. Results

As before, the desire is to systematically study the performance of GAs and NNs on a series of increasingly difficult problems. Clearly, the complexity in this case is a function of both the number of nodes and the number of directed edges. For a given number N of nodes, problems with only a small number of edges ($\leq N$) or nearly fully connected (approximately N^2 edges) are not very interesting. However, problems with approximately $\frac{N^2}{2}$ edges would, in general, present the most difficult problems. In addition, to achieve some degree of uniform difficulty and to allow for a direct comparison with some of the results in the previous section, the problems should have exactly one solution. Consequently, the following family of HC problems have been defined for the experiments.

Consider a graph of N nodes, which are labeled using consecutive integers. Suppose the first node has directed edges to all nodes with larger labels (except for the last node). The next $N-2$ nodes have directed edges to all nodes with larger labels (including the last one). The last node has a directed edge back to the first node. A complete tour consists of following the node labels in increasing order, until you reach the last node. From the last node you travel back to the first. Because the edges are directed, it is clear that this is also the only legal tour.

Intuitively, such instances of HC problems should be difficult. Only one tour exists in each instance. In addition, there are a large number of solutions that are almost complete tours scattered throughout the search space. Figure 10 illustrates the corresponding SAT GA payoff function for the HC problem of this type with 7 nodes. The 100... hyperplane is a first-order hyperplane. All individuals to the left of the hyperplane start with 0. All individuals to the right of the hyperplane start with 1. The other hyperplanes have similar interpretations. From the sampling density it can be seen that the GA concentrated sampling to the right of the 100... hyperplane.† The NN energy function is correspondingly complex.

In summary, the experimental framework consists of varying the number N of nodes in the range $4 \leq N \leq 11$ and, for each value of N, generating a directed graph of the form described above containing approximately $\frac{N^2}{2}$ edges and exactly one solution. Each of these HC problems is transformed into its equivalent SAT problem using the transformation described earlier, generating search space sizes ranging from 2^6 to 2^{55}. GAs and NNs are then used to solve each of the corresponding SAT problems which, in turn, describes a legal HC tour.

---

† Since the solution does in fact start with a 1, the GA is concentrating on the correct subspace.

Figure 10: Graph of HC7 Payoff Function for the GA

The following table indicates the number of evaluations needed for the GA (using AVE^p, where $p = 1$). Both the mean number of evaluations and the standard deviation are reported. The number of edges in the graph (bits) is denoted by n.

n	10	15	21	28	36	45	55
mean	848	1022	5028	21894	70577	259876	838522
sd	2041	1083	2016	36391	49946	227254	123350

Table 9: Performance of GAs on HC Problems

Figure 11 presents a graph of the results. Notice that the number of evaluations required to find a solution is an order of magnitude higher that the earlier TP

and FP problems. However, even with these difficult problems, the log-log plot is still sub-linear.

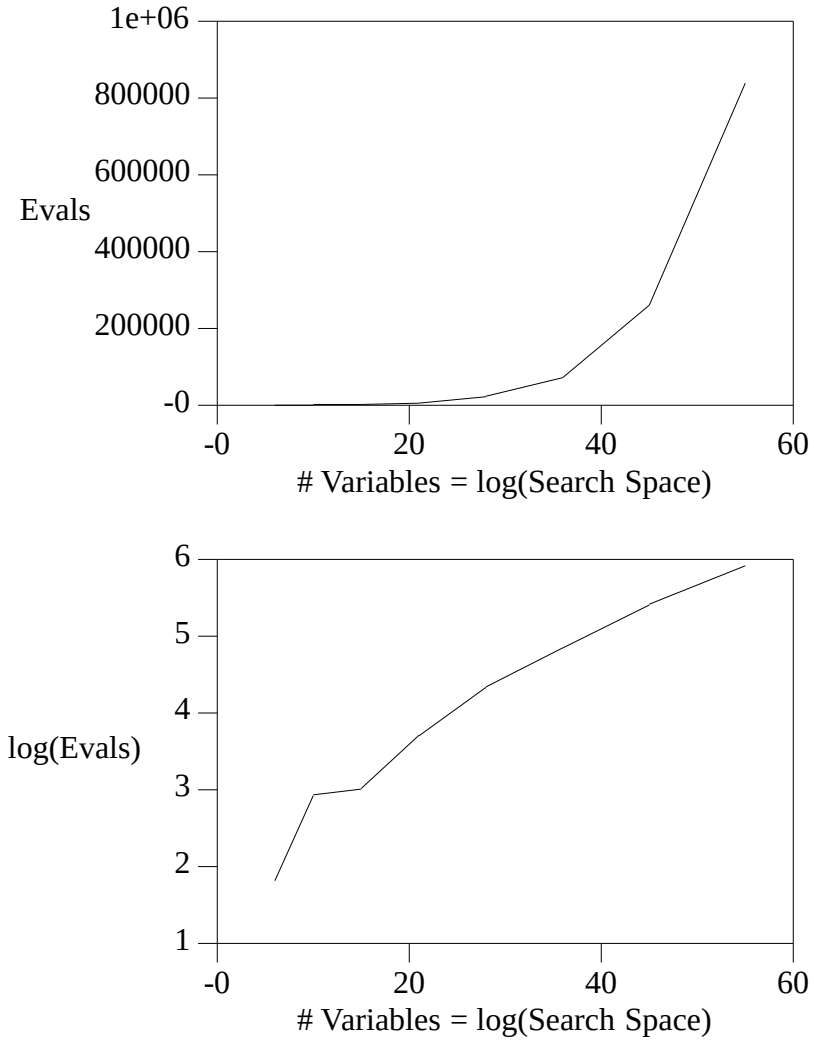

Figure 11: Performance of GAs on the HC Problems

The GAs were then re-run with AVE^p ($p > 1$). Again, an optimum appears at $p = 2$. Figure 12 graphs the results.

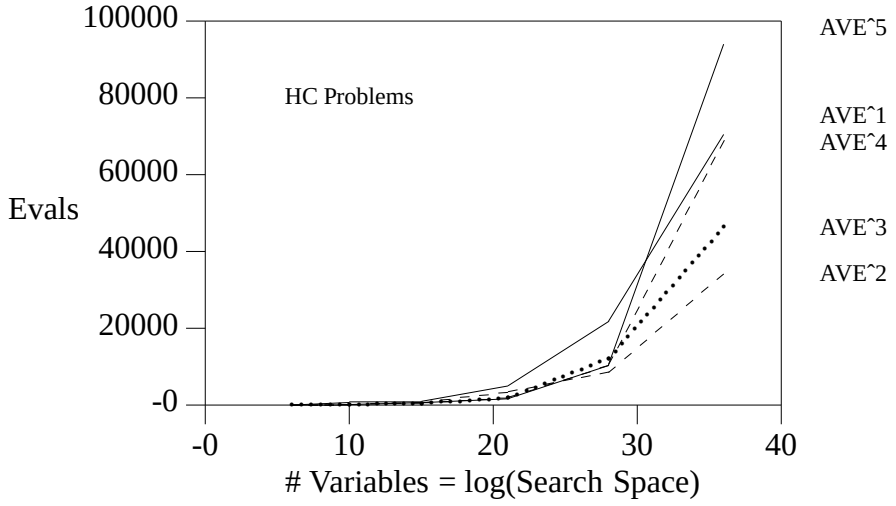

Figure 12: Performance of GAs using AVE^{p}

Finally, the NN paradigm was tested on the HC problems. The following table indicates the number of evaluations needed for the NN. Both the mean number of evaluations and the standard deviation are reported.

n	10	15	21	28	36	45	55
mean	51	169	426	1120	6698	99431	1417388
sd	34	84	265	1058	5927	100052	1341593

Table 10: Performance of NNs on HC Problems

Figure 13 presents a graph of the comparison between the NN and the GA on the HC problems.

Although the NN outperforms the GA on the smaller problems, the GA appears to win on larger, more complex problems. If we again use quadratic fits

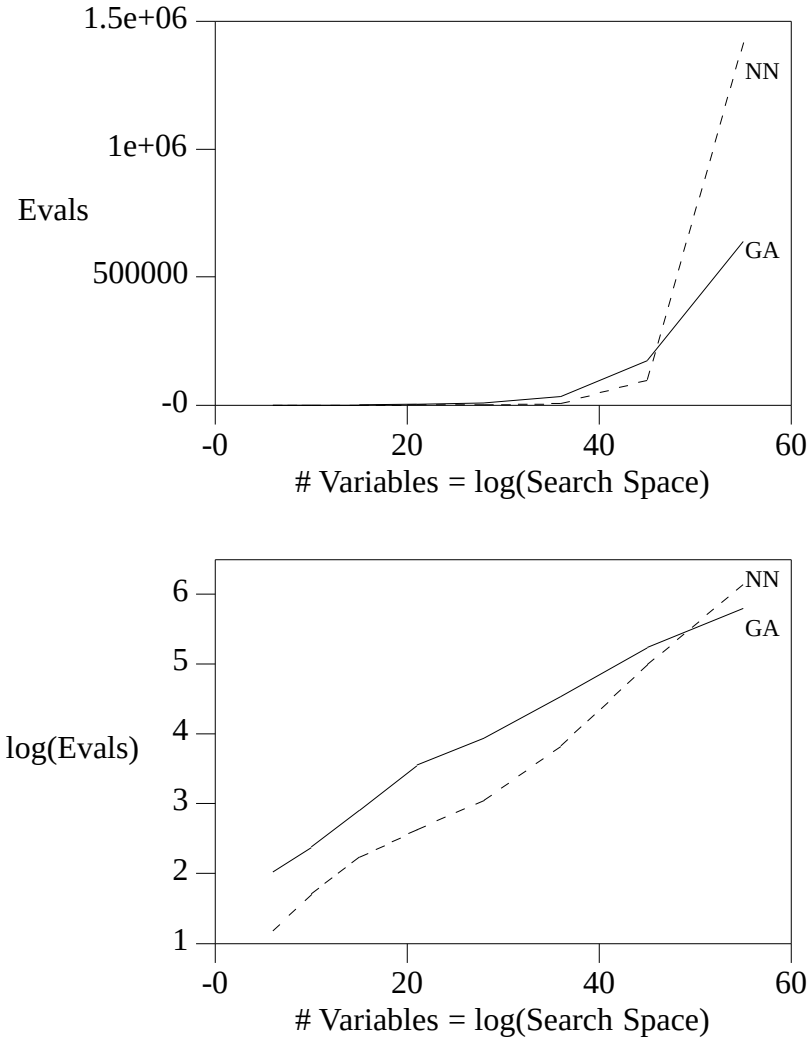

Figure 13: Comparison of GAs and NNs on the HC Problems

to the HC data, the NN speedup is approximately $N^{2.22}$ while the GA speedup is $N^{2.94}$. It is intriguing to note that the GA empirical results match the theoretical N^3 implicit parallelism results for the HC problems, which were deliberately constructed to be very difficult.

This section has introduced hamiltonian circuit problems and described the polynomial transformation used to convert each problem into an equivalent boolean satisfiability problem. The GA and NN paradigms were tested on a series

of increasingly difficult single-solution HC problems, after conversion into boolean expressions. Both paradigms exhibit substantial speedup over systematic sequential search, although the GA appears to perform better for larger and more complex problems.

4. SUMMARY AND FUTURE WORK

This thesis is a first step in exploring two issues. First, are there large classes of problems for which effective GA and NN representations exist? Second, are the representation/performance characteristics of GAs and NNs significantly different?

The thesis addresses the first issue by focusing on the application of GAs and NNs to SAT. NN and GA paradigms for heuristically solving SAT problems are presented. Other NP-Complete problems can be solved via polynomial-time transformation into equivalent SAT problems. This technique is illustrated for HC problems.

It is also possible that other NP-Complete problems map naturally to GA and NN paradigms. Recent work by [Korst89] suggests that NNs can handle NP-Complete graph problems such as *independent set*, *max cut*, and *graph coloring*. Both independent set and max cut [Anand89] appear to have natural GA representations.

These ideas suggest that effective GA and NN representations exist for a set of NP-Complete problems. Other NP-Complete problems may be heuristically solved via polynomial transformation into a problem in that set. Future work should explore both the nature of that set, and the NP-Complete problems that will map well into members of the set.

The second issue seeks to compare the representation and performance characteristics of GAs and NNs. This thesis addresses this issue by applying both techniques to the same problem (SAT). It appears that creating effective representations for GAs and NNs is not trivial. However, it is interesting to note that the parse tree of the boolean expression formed both the GA payoff function and the NN graph representation. A similar relationship may hold for other NP-Complete problems.

From a performance viewpoint, both paradigms perform well on simple two-peak and false-peak problems, with the NN consistently outperforming the GA. Both paradigms exhibit substantial speedup over systematic sequential search on the far more difficult HC problems, with the GA appearing to win on the larger problems.

Another interpretation of the results is that the NN paradigm performs well where local search (exploitation) is appropriate. In contrast, the GA outperforms the NN on global search (exploration). A combination of the two paradigms, coupling NN local search with GA global search, might well outperform either algorithm alone by providing a good mixture of exploration and exploitation.

Future work should explore further the limitations of these paradigms by defining even more difficult classes of SAT problems derived from other NP-Complete problems. An example of such a problem comes from the cryptography community. Most cryptography systems make use of prime numbers and factorization [Rivest78]. Hoey has devised an algorithm for converting a factorization decision problem into an equivalent SAT problem [Hoey89]. For example, a problem of the form:

"Does 689 have a 4 bit factor?"

can be converted to a boolean expression with 22 variables, 105 clauses, and 295 literals. Such problems are of interest to both the cryptographic and complexity theory communities because they are generally highly intractable.

Finally, both paradigms should be compared to existing operations research techniques for solving SAT problems [Davis60, Hammer68, Brown82, Franco86, Van Gelder88].

In summary, this thesis presents paradigms for using neural networks and genetic algorithms to heuristically solve boolean satisfiability problems. Results are presented for *two-peak* and *false-peak* SAT problems. Since SAT is NP-Complete, any other NP-Complete problem can be transformed into an equivalent

SAT problem in polynomial time, and solved via either paradigm. This technique is illustrated for hamiltonian circuit (HC) problems.

List of References

Ackley, David H. (1985). A Connectionist Algorithm for Genetic Search, *Proc. Int'l Conference on Genetic Algorithms and their Applications.*

Aho, Hopcroft, and Ullman (1974). *The Design and Analysis of Computer Algorithms*, Addison-Wesley.

Anand, V. (1989). *The Application of Genetic Algorithms to an NP-Complete Problem*, Unpublished work, Navy Center for Applied Research in Artificial Intelligence.

Antonisse, H. J. and K. S. Keller (1987). Genetic Operators for High-Level Knowledge Representation, *Proc. Int'l Conference on Genetic Algorithms and their Applications.*

Axelrod, Robert (1987). The Evolution of Strategies in the Iterated Prisoner's Dilemma, *Genetic Algorithms and Simulated Annealing,* Lawrence Davis, ed., Morgan Kaufmann Publishers.

Baker, James E. (1985). Adaptive Selection Methods for Genetic Algorithms, *Proc. Int'l Conference on Genetic Algorithms and their Applications.*

Baker, James E. (1987). Reducing Bias and Inefficiency in the Selection Algorithm, *Proc. Int'l Conference on Genetic Algorithms and their Applications.*

Bethke, A. D. (1981). *Genetic Algorithms as Function Optimizers,* Doctoral dissertation, University of Michigan.

Bickel, Arthur S. and Riva Wenig Bickel (1987). Tree Structured Rules in Genetic Algorithms, *Proc. Int'l Conference on Genetic Algorithms and their Applications.*

Booker, Lashon B. (1987). Improving Search in Genetic Algorithms, *Genetic Algorithms and Simulated Annealing,* Lawrence Davis, ed., Morgan Kaufmann Publishers.

Bridges, Clayton L. and David E. Goldberg (1987). An Analysis of Reproduction and Crossover in a Binary-Encoded Genetic Algorithm, *Proc. Int'l Conference on Genetic Algorithms and their Applications.*

Brown, C. and P. Purdom (1982). An Empirical Comparison of Backtracking Algorithms, *IEEE Trans. PAMI* Vol. 4, No. 3, 309-316.

Cervantes, J. H. and Richard Hildebrant (1987). Comparison of Three Neuron-Based Computation Schemes, *IEEE First International Conference on Neural Networks,* pg III-657.

Cohoon, J. P., et. al. (1987). Punctuated Equilibria: a Parallel Genetic Algorithm, *Proc. Int'l Conference on Genetic Algorithms and their Applications.*

Davis, M. and H. Putnam (1960). A Computing Procedure for Quantification Theory, *J. Assoc. Comput. Mach.* 7, 201-215.

Davis, Lawrence (1985). Job Shop Scheduling with Genetic Algorithms, *Proc. Int'l Conference on Genetic Algorithms and their Applications.*

Davis, Lawrence and Susan Coombs (1987). Genetic Algorithms and Communication Link Speed Design, *Proc. Int'l Conference on Genetic Algorithms and their Applications.*

Davis, Lawrence (1987). *Genetic Algorithms and Simulated Annealing,* Morgan Kaufman Publishers, Inc., Los Altos, CA.

De Jong, K. A. (1975). *An Analysis of the Behavior of a Class of Genetic Adaptive Systems,* Doctoral dissertation, Dept. Computer and Communication Sciences, University of Michigan, Ann Arbor.

De Jong, K. A. (1985). Genetic Algorithms: a 10 Year Perspective, *Proc. Int'l Conference on Genetic Algorithms and their Applications.*

De Jong, K. A. (1987). On Using Genetic Algorithms to Search Program Spaces, *Proc. Int'l Conference on Genetic Algorithms and their Applications.*

De Jong, K. A. & William M. Spears (1989). Using Genetic Algorithms to Solve NP-Complete Problems, *Proc. Int'l Conference on Genetic Algorithms and their Applications.*

Dolan, Charles P. and Michael G. Dyer (1987). Toward the Evolution of Symbols, *Proc. Int'l Conference on Genetic Algorithms and their Applications.*

Fourman, Michael P. (1985). Compaction of Symbolic Layout Using Genetic Algorithms, *Proc. Int'l Conference on Genetic Algorithms and their Applications.*

Franco, John (1986). On the Probabilistic Performance of Algorithms for the Satisfiability Problem, *Information Processing Letters* 23, 103-106.

Fujiko, Cory and John Dickinson (1987). Using the Genetic Algorithms to Generate LISP Source Code to Solve the Prisoner's Dilemma, *Proc. Int'l Conference on Genetic Algorithms and their Applications.*

Garey, Michael R. & David S. Johnson (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness,* W. H. Freeman and Company, San

Francisco, CA.

Geman, S. and D. Geman (1984). *Stochastic Relaxation, Gibbs Distributions, and Bayesian Restoration of Images,* IEEE Trans., Vol PAMI-6, No. 6, p. 721, November 1984.

Glover, David E. (1987). Solving a Complex Keyboard Configuration Problem Through Generalized Adaptive Search, *Genetic Algorithms and Simulated Annealing,* Lawrence Davis, ed., Morgan Kaufmann Publishers.

Goldberg, David E. (1985). Genetic Algorithms and Rule Learning in Dynamic Systems Control, *Proc. Int'l Conference on Genetic Algorithms and their Applications.*

Goldberg, David E. and Robert Lingle, Jr. (1985). Alleles, Loci, and the Traveling Salesman Problem, *Proc. Int'l Conference on Genetic Algorithms and their Applications.*

Goldberg, David E. and Philip Segrest (1987). Finite Markov Chain Analysis of Genetic Algorithms, *Proc. Int'l Conference on Genetic Algorithms and their Applications.*

Goldberg, David E. and Jon Richardson (1987). Genetic Algorithms with Sharing for Multimodal Function Optimization, *Proc. Int'l Conference on Genetic Algorithms and their Applications.*

Goldberg, David E. and Robert E. Smith (1987). Nonstationary Function Optimization Using Genetic Algorithms with Dominance and Diploidy, *Proc. Int'l Conference on Genetic Algorithms and their Applications.*

Goldberg, David E. (1987). Simple Genetic Algorithms and the Minimal, Deceptive Problem, *Genetic Algorithms and Simulated Annealing,* Lawrence Davis, ed., Morgan Kaufmann Publishers.

Goldberg, David E. (1989). *Genetic Algorithms in Search, Optimization & Machine Learning,* Addison-Wesley Publishing Company, Inc.

Grefenstette, John J. and J. Michael Fitzpatrick (1985). Genetic Search with Approximate Function Evaluations, *Proc. Int'l Conference on Genetic Algorithms and their Applications.*

Grefenstette, John J., et. al. (1985). Genetic Algorithms for the Traveling Salesman Problem, *Proc. Int'l Conference on Genetic Algorithms and their Applications.*

Grefenstette, John J. (1985). *Proceedings of an International Conference on Genetic Algorithms and their Applications,* Pittsburg, PA.

Grefenstette, John J. (1987). *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms,* Cambridge, MA.

Grefenstette, John J. (1987). Incorporating Problem Specific Knowledge into Genetic Algorithms, *Genetic Algorithms and Simulated Annealing,* Lawrence Davis, ed., Morgan Kaufmann Publishers.

Gunn, Janet P. and Robert B. Weidlich (1989). *A Derivative of the Hopfield- Tank Neural Network Model that Reliably Solves the Traveling Salesman Problem,* IJNNS, Washington, DC., June 1989.

Gutzmann, Kurt M. (1987). Combinatorial Optimization Using a Continuous State Boltzmann Machine, *IEEE First International Conference on Neural Networks,* pg III-721.

Hammer, Peter L. and Sergiu Rudeanu (1968). *Boolean Methods in Operations Research*, Springer-Verlag New York Inc. 1968.

Hoey, Dan J. *Navy Center for Applied Research in Artificial Intelligence.* Private Communication.

Holland, John H. (1975). *Adaptation in Natural and Artificial Systems,* The University of Michigan Press.

Hopfield, J. J. (1982). Neural Networks and Physical Systems with Emergent Collective Computational Abilities, *Proceedings of the National Academy of Sciences,* USA, 79, 2554-2558.

Hopfield, J. J. and D. W. Tank (1985). "Neural" Computation of Decisions in Optimization Problems, *Biological Cybernetics,* 52, 141-152.

Korst, Jan H. M. and Emile H. L. Aarts (1989). Combinatorial Optimization on a Boltzmann Machine, *Journal of Parallel and Distributed Computing,* pg 331.

Kuchinski, M. J. (1985). *Battle Management Systems Control Rule Optimization Using Artificial Intelligence,* Technical Note, Naval Surface Weapons Center, Dahlgren, VA.

Levy, B. C. and Milton Adams (1987). Global Optimization with Stochastic Neural Networks, *IEEE First International Conference on Neural Networks,* pg III-681.

Liepins, G. E., et. al. (1987). Greedy Genetics, *Proc. Int'l Conference on Genetic Algorithms and their Applications.*

McClelland, James L. and David E. Rumelhart (1988). *Explorations in Parallel Distributed Processing,* The MIT Press, Cambridge, MA.

Oliver, I. M., Smith, D. J. and J. R. C. Holland (1987). A Study of Permutation Crossover Operators on the Traveling Salesman Problem, *Proc. Int'l Conference on Genetic Algorithms and their Applications.*

Pettey, Chrisila B., et. al.(1987). A Parallel Genetic Algorithm, *Proc. Int'l Conference on Genetic Algorithms and their Applications.*

Rivest, R. L., et al (1978). A Method for Obtaining Digital Signatures and Public-key Cryptosystems, CACM, 21, 2, 120-6.

Robertson, George G. (1987). Parallel Implementation of Genetic Algorithms in a Classifier System, *Proc. Int'l Conference on Genetic Algorithms and their Applications.*

Rumelhart, David E. and James L. McClelland (1986). *Parallel Distributed Processing*, The MIT Press, Cambridge, MA.

Sannier, Adrian V. II and Erik D. Goodman (1987). Genetic Learning Procedures in Distributed Environments, *Proc. Int'l Conference on Genetic Algorithms and their Applications.*

Schaffer, J. David (1985). Multiple Objective Optimization with Vector Evaluated Genetic Algorithms, *Proc. Int'l Conference on Genetic Algorithms and their Applications.*

Schaffer, J. David and Amy Morishima (1987). An Adaptive Crossover Distribution Mechanism for Genetic Algorithms, *Proc. Int'l Conference on Genetic Algorithms and their Applications.*

Sirag, David J. and Paul T. Weisser (1987). Toward a Unified Thermodynamics Genetic Operator, *Proc. Int'l Conference on Genetic Algorithms and their Applications.*

Smith, Gerald H. (1979). *Adaptive Genetic Algorithms and the Boolean Satisfiability Problem*, Unpublished Work.

Smith, S. F. (1980). *A Learning System Based on Genetic Adaptive Algorithms,* Doctoral Thesis, Department of Computer Science, University of Pittsburg.

Smith, Derek (1985). Bin Packing with Adaptive Search, *Proc. Int'l Conference on Genetic Algorithms and their Applications.*

Suh, Jung Y. and Dirk Van Gucht (1987). Incorporating Heuristic Information into Genetic Search, *Proc. Int'l Conference on Genetic Algorithms and their Applications.*

Tagliarini, Gene A. and Edward W. Page (1987). Solving Constraint Satisfaction Problems with Neural Networks, *IEEE First International Conference on Neural Networks,* pg III-741.

Tanese, Reiko (1987). Parallel Genetic Algorithms for a Hypercube, *Proc. Int'l Conference on Genetic Algorithms and their Applications.*

Van Gelder, Allen (1988). A Satisfiability Tester for Non-clausal Propositional Calculus, *Information and Computation* 79, 1-21.

Whitley, Darrell (1987). Using Reproductive Evaluation to Improve Genetic Search and Heuristic Discovery, *Proc. Int'l Conference on Genetic Algorithms and their Applications.*

Appendix 1

Problem	Vars	Mean	St. Dev	Iterations	Trials
TP1	10	278	132	-	50
TP2	20	963	239	-	50
TP3	30	1753	360	-	50
TP4	40	2855	676	-	50
TP5	50	4216	641	-	50
TP6	60	5485	947	-	50
TP7	70	7095	1077	-	50
TP8	80	11167	4977	11	10
TP9	90	13080	5491	11	10
FP1	10	439	338	12	10
FP2	20	1209	730	11	10
FP3	30	4805	5289	17	10
FP4	40	8031	8039	20	10
FP5	50	12167	12797	20	10
FP6	60	18387	21268	44	20
FP7	70	15617	13390	34	20
FP8	80	18605	16209	32	20
FP9	90	35153	31731	22	10
HC4	6	65	24	-	10
HC5	10	848	2041	-	10
HC6	15	1022	1083	-	10
HC7	21	5028	2016	-	10
HC8	28	21894	36391	20	10
HC9	36	70577	49946	-	10
HC10	45	259876	227254	-	10
HC11	55	838522	123350	-	2

Table 11: GA Performance (AVE^p, $p = 1$)

Appendix 1

Problem	Vars	Mean	St. Dev	Iterations	Trials
TP1	10	164	45	10	10
TP2	20	696	91	10	10
TP3	30	1257	294	10	10
TP4	40	2283	847	11	10
TP5	50	2741	393	10	10
TP6	60	4060	518	10	10
TP7	70	4966	2238	11	10
TP8	80	6973	2516	11	10
TP9	90	10208	7075	13	10
FP1	10	288	157	11	10
FP2	20	1879	2110	18	10
FP3	30	3608	2155	21	10
FP4	40	4219	2065	17	10
FP5	50	6517	3197	19	10
FP6	60	10162	9649	22	10
FP7	70	12929	11687	22	10
FP8	80	17868	14615	25	10
FP9	90	17569	13706	21	10
HC4	6	106	14	10	10
HC5	10	239	72	10	10
HC6	15	803	512	12	10
HC7	21	3559	3281	18	10
HC8	28	8680	9355	25	10
HC9	36	34417	29668	61	10
HC10	45	174706	182521	200	10
HC11	55	640478	477833	595	10

Table 12: GA Performance (AVE^p, $p = 2$)

Appendix 1

Problem	Vars	Mean	St. Dev	Iterations	Trials
TP1	10	175	74	10	10
TP2	20	687	285	12	10
TP3	30	1198	717	11	10
TP4	40	2123	1073	12	10
TP5	50	2828	1816	12	10
TP6	60	3575	2367	12	10
TP7	70	6567	4094	16	10
TP8	80	9838	5671	19	10
TP9	90	14524	9490	23	10
FP1	10	238	153	11	10
FP2	20	987	509	15	10
FP3	30	2432	1614	19	10
FP4	40	3229	2293	17	10
FP5	50	6554	4724	22	10
FP6	60	10537	10873	28	10
FP7	70	7814	3438	19	10
FP8	80	22412	13671	39	10
FP9	90	16644	15119	26	10
HC4	6	106	13	10	10
HC5	10	222	70	10	10
HC6	15	594	755	11	10
HC7	21	1897	1096	18	10
HC8	28	12247	8444	52	10
HC9	36	46558	51804	148	10
HC10	45	155141	138554	377	10

Table 13: GA Performance (AVE^p, $p = 3$)

Appendix 1

Problem	Vars	Mean	St. Dev	Iterations	Trials
TP1	10	174	62	10	10
TP2	20	497	96	11	10
TP3	30	1062	478	12	10
TP4	40	1501	311	10	10
TP5	50	2930	1030	15	10
TP6	60	4566	3618	17	10
TP7	70	5649	2176	16	10
TP8	80	10601	7545	25	10
TP9	90	17426	6991	32	10
FP1	10	314	265	12	10
FP2	20	780	450	14	10
FP3	30	1771	1021	18	10
FP4	40	3015	2786	19	10
FP5	50	5477	4165	26	10
FP6	60	11224	9881	36	10
FP7	70	9479	5374	26	10
FP8	80	13340	7674	30	10
FP9	90	31299	23323	57	10
HC4	6	106	13	10	10
HC5	10	234	82	10	10
HC6	15	697	728	12	10
HC7	21	1993	2460	23	10
HC8	28	10506	8753	73	10
HC9	36	69114	77931	342	10

Table 14: GA Performance (AVE^p, $p = 4$)

Appendix 1

Problem	Vars	Mean	St. Dev	Iterations	Trials
TP1	10	165	40	10	10
TP2	20	433	84	10	10
TP3	30	961	266	12	10
TP4	40	1532	875	13	10
TP5	50	2402	1060	14	10
TP6	60	6167	4738	25	10
TP7	70	8637	4914	28	10
TP8	80	11774	9073	30	10
TP9	90	18892	11420	41	10
FP1	10	254	92	12	10
FP2	20	559	241	12	10
FP3	30	2030	1584	20	10
FP4	40	3988	2751	27	10
FP5	50	5240	4680	27	10
FP6	60	10850	5653	43	10
FP7	70	10663	6493	33	10
FP8	80	23963	23687	58	10
FP9	90	48010	34600	97	10
HC4	6	106	13	10	10
HC5	10	221	80	10	10
HC6	15	771	798	16	10
HC7	21	1784	1543	26	10
HC8	28	10605	9259	101	10
HC9	36	97837	90413	687	10

Table 15: GA Performance (AVE^p, $p = 5$)

Appendix 1

Problem	Vars	Mean	St. Dev	Iterations	Trials
TP1	10	6	4	50	50
TP2	20	12	6	50	50
TP3	30	24	11	50	50
TP4	40	35	14	50	50
TP5	50	51	14	50	50
TP6	60	64	15	50	50
TP7	70	78	15	50	50
TP8	80	97	18	50	50
TP9	90	113	19	50	50
TP10	100	132	23	50	50
FP1	10	49	106	57	50
FP2	20	18	10	50	50
FP3	30	31	16	50	50
FP4	40	44	18	50	50
FP5	50	76	127	51	50
FP6	60	77	25	50	50
FP7	70	90	26	50	50
FP8	80	108	27	50	50
FP9	90	126	30	50	50
FP10	100	171	256	51	50
HC4	6	15	10	10	10
HC5	10	51	34	10	10
HC6	15	169	84	10	10
HC7	21	426	265	10	10
HC8	28	1120	1058	12	10
HC9	36	6698	5927	26	10
HC10	45	99431	100052	99	10
HC11	55	1417388	1341593	384	10

Table 16: NN Performance

Vita

William M. Spears was born on April 3, 1962, in Providence, Rhode Island, and is an American citizen. He graduated from Saratoga Springs High School, Saratoga Springs, New York, in 1980. He received his Bachelor of Arts in Mathematics from Johns Hopkins University, Baltimore, Maryland, in 1984. He has been employed at the Naval Research Laboratory, Washington D.C, since 1985. His publications include:

De Jong, K.A. and W. Spears, "Using Genetic Algorithms to Solve NP-Complete Problems", International Conference on Genetic Algorithms, George Mason University, Fairfax, Virginia, 1989.

Pipitone, F., K.A. De Jong, and W. Spears, "An Artificial Intelligence Approach to Analog Systems Diagnosis", NRL Report 9219, Naval Research Laboratory, Washington D.C., 1989.

Pipitone, F., K.A. De Jong, W. Spears, and M. Marrone, "The FIS Electronic Troubleshooting Project", Expert System Applications to Telecommunications, J. Liebowitz, John Wiley, New York, 1988.

Spears, W. and K.A. De Jong, "Using Genetic Algorithms as a Heuristic for NP-Complete Decision Problems", Operations Research Society of America / The Institute of Management Sciences, New York City, New York, 1989.

Spears, W., "Using Neural Networks and Genetic Algorithms as Heuristics for NP-Complete Problems", International Joint Conference on Neural Networks, Washington D.C, 1990.