UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

PHENOTYPE OPERATORS FOR IMPROVED PERFORMANCE OF HEURISTIC ENCODING WITHIN GENETIC ALGORITHMS

A DISSERTATION

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

Degree of

DOCTOR OF PHILOSOPHY

By

BENJAMIN PAUL CARLSON
Norman, Oklahoma
2016

PHENOTYPE OPERATORS FOR IMPROVED PERFORMANCE
OF HEURISTIC ENCODING WITHIN GENETIC ALGORITHMS

A DISSERTATION APPROVED FOR THE
SCHOOL OF COMPUTER SCIENCE

BY

Dedicated to my brother-in-law, Dennis Reust, without whose support this work would not have been possible

Acknowledgements

There are many people that I would like to thank for all of their help and support. First, my wonderful family who has endured much as part of my road to the PhD. Eleanor, my wife, has worked hard to make up for my excess time and energy working on the dissertation, and my kids, Isaac and Hannah, have had to put up with upheavals in their young lives which they handled very well.

My advisor, Professor Dean Hougen, and the entire School of Computer Science staff have helped me whenever asked, without question and always with a smile. Dean has maintained confidence in me even though I have tried his patience with leaving the program for a long break. Indeed, when I asked to return, Dean and the Director of Computer Science, Professor Sridhar Radhakrishnan, were entirely supportive and did all they could to get me back into the program where I had left off. The school of Computer Science has also supported me with graduate assistantships and scholarships every semester. Indeed, I have spent the last year working for Professor Chris Weaver as a research assistant on a very interesting NSF grant. Also, Professor Don Potter with the University of Georgia has provided much guidance and assistance through his background and expertise on the SITB problem.

The College of Engineering has helped me with moral support and with the Foster Fellowship, which has been a tremendous assistance to me and my family. Thus, I also want to thank the Foster family for their generous support to COE students. The COE library has been an excellent resource and the staff there has been very helpful in tracking down documents and books.

Finally, the University of Oklahoma and the Graduate College have helped in more ways than I can explain here. This is a wonderful university and I feel privileged to have been able to attend. The Graduate College gracefully restored my Whethington Fellowship after I had left for 9 years, and has also provided much advice and support over the years.

There are many other people that helped me and my family through this difficult but extremely valuable process both on campus and in the Norman community and I thank them all.

Table of Contents

List of Tables

List of Figures

Abstract

Many approaches to applying *Genetic Algorithms* (GAs) to *Nondeterministic Polynomial time Complete* (NPC) problems involve population members encoded directly from the problem solution space. While this technique enables trivial mapping of the population members to solutions, it can cause complex problems for GA operators as they attempt to direct the evolution of the population toward more promising areas of the solution space. These operators, using inspiration from genetics and evolution in the biological world, combine and manipulate the current population to produce a new population that, it is hoped, will eventually converge toward better solutions to the original problem. However, many problems, especially graph-space problems, cannot be so easily manipulated when GA members consist of direct encodings. In such cases, GA operators must perform awkward transformations to convert the progeny into viable solutions. Here is where *heuristic encoding* comes into play, in that any combination of genes will produce a viable solution. However, this additional level of abstraction does cause other problems and tends to weaken the guiding effects of traditional GA operators. Thus, I have designed custom GA operators that mitigate these problems by using the solutions produced by the heuristic encoded members to better guide the manipulation when producing the next generation. This dissertation shows that heuristic encoding is an effective technique for the representation of solutions to graph-space problems. It also shows that, when using heuristic encoding, GAs with traditional operators perform well compared to more direct encoding techniques. Finally it shows the combination of heuristic encoding and GA operators designed to work with them increases GA performance and can be

competitive with other techniques. I believe that these techniques will also work well for other types of problems for which GAs are commonly applied.

Chapter 1

Background and Introduction to the Heuristic Encoding Technique

Genetic Algorithms (GAs) are a type of stochastic search algorithm inspired by nature, and commonly applied to *Nondeterministic Polynomial Time-Complete* (NP-Complete, or NPC) problems. GAs create and manipulate a population of solutions in an attempt to discover good instances. *Graph-space problems* are defined on a set of nodes and a set of links. Many graph-space problems involve building a constrained path between these connected nodes. While GAs can be applied to these problems, the implementation can be difficult due to the mechanics of GAs and issues with population dynamics. This is caused primarily by two problems: representation issues and dimensionality. The obvious and most direct representation scheme involves simply listing either the nodes or the links that will represent a path. This technique, while simple, typically does not work well for GAs due to the nature of GA operators (see Chapter 3 for a description). Also, by *dimensionality*, we mean that as nodes are added to the search space, the hardness of the problem grows exponentially. While this is typical of NP-Complete problems, GAs seem to suffer more than other techniques, at least for path-building graph-space problems.

Due to these issues, it seems most successful techniques involve either deterministic algorithms (Palombo et al, 2015), or other *Evolutionary Computation* (EC) techniques (Zhang and Ma, 2014). Indeed, much work has been done using GAs on smaller instances of the *Traveling Salesman Problem* (TSP) but we have found no modern day papers using GAs on larger TSP instances (greater than 200 nodes).

This research attempts to address these issues with GAs as applied to path building in NPC problems. We show that an alternative encoding scheme can mitigate the representation issues that typically result from the GA operators being applied to more direct encoding schemes. Also, by viewing the GA as a *hyper-heuristic*, a heuristic to manipulate other heuristics, we have devised alternative GA operators designed to work with this new encoding scheme which greatly increase the effectiveness of the GA on these types of problems.

Our intent is not to find solutions to problems that are competitive with targeted techniques (techniques specifically designed for one and only one problem), but to show that our technique is generalizable and can produce good solutions to a variety of problems. Burke et al (2013, page 1710) states this well:

"One of the goals of hyper-heuristic research is raising the level of generality. In this context, it is often the case that a hyper-heuristic does not aim to outperform a custom-made solver for a given problem. In such an environment, applicability over a wide range of problem domains is more crucial."

An example of a targeted approach to a specific problem and one of the few contemporary examples of a GA applied to the *Snake-In-The-Box* (SITB) problem (other than this work) can be found in Ruiz (2014). Ruiz (2014) implements a custom GA, the *Mitosis Genetic Algorithm*, which was designed specifically to work on the SITB problem. Again, our focus is to show that heuristic encoding with phenotype operators is not a narrowly focused targeted solution, but can be applied to multiple problem domains.

1.1 Introduction to Heuristic Encoding

By *encoding*, we mean the technique used to represent solutions to a problem as a string of symbols such that a GA can manipulate them. Primarily, this encompasses the set of symbols that represent the individual genes and what these symbols represent.

Heuristic encoding, encodings of instructions for producing solutions, can mitigate many of the issues typically associated with the representation of graph-space problems for the application of a GA. A good example is the Traveling Salesman Problem where encoding the nodes of the solution space directly as genes in population members, while straightforward, causes GA operators many problems when attempting to manipulate the population to produce better solutions. This research uses heuristic encoding and *phenotype operators*, operators that use the solutions produced, for both the TSP and the SITB problems where the efficacy of the technique to both deal with encoding/recombination problems and in producing good results is demonstrated.

The design and selection of individual heuristics is vitally important for this technique to work, and knowledge of the problem space must be applied when assembling the heuristic set in order to fully exploit this novel technique. Indeed, if impotent heuristics are included with a good set, they cause the GA to waste resources searching areas that are of little value. Also, if needed heuristics are left out, the best areas to search may never be found. This work develops an effective set of low level heuristics for these two problems that can be used in other evolutionary computation work as well as with deterministic algorithms.

This encoding technique produces results that are quite competitive with traditional GA applications and with other techniques. We show this through experimentation and results, where the best known solution to the SITB in a dimension 8 hypercube was found using this encoding scheme combined with phenotype GA operators (Carlson and Hougen, 2010). Indeed, Ostergard and Pettersson (2015) use the result of this work to prove various bounds in the SITB problem.

1.2 Introduction to Phenotype Operators

Heuristic encoding does add an additional layer of abstraction between GA population members and the problem's solution space. This layer causes a weakening of the GA's ability to manipulate the population toward more promising areas of the search space. This issue is discussed in Burke et al (2013) where it is referred to as consisting of two different search spaces: the heuristic space, the set of all possible combinations of the heuristics, and the solution space, the set of all possible solutions to the problem that the heuristic space maps to. This problem should be dealt with in order for this scheme to reach its potential. This research addresses this problem as well as investigating the value of the heuristic encoding method. We have developed custom GA operators of mating/crossover and mutation, which tighten the coupling between the *phenotype* (the final product or the solution to the problem) and the *genotype* (the GA population members). These phenotype operators make use of the phenotype to guide the application of the GA operators when producing new members from the current population. These operators produce better results with this encoding scheme than the typical GA operators of mutation and crossover. The development of these operators is a primary contribution of this work as there is little evidence of such work currently in the literature. Indeed, the scope of this project is greater than any other work done involving heuristic encoding or phenotype operators for GAs. Finally, as outside justification of the value of this work, a quote from Falkenauer (1998, page 42) emphasizes these points nicely as one of the basic claims of his book: "a GA's encoding and operators must be adapted to the problem being solved."

1.3 Genetic Algorithms and the Problems Studied

GAs have been used to deal with "*The Curse of Dimensionality*" (Bellman, 1961): as the problem grows linearly, the solution space grows exponentially. GAs were developed by John Holland and his colleagues (Holland, 1992) at the University of Michigan. The first step in using a GA is devising an encoding scheme in which solutions to the problem can be easily manipulated by the GA. A very common scheme involves encoding the solutions directly as strings of symbols, quite often binary (Bäck, 2000, pages 132-135). Also, binary encoding is traditionally used for teaching the subject of GAs (Mitchell, 1996, pages10-12). However, often in graph-space problems or any problem where solutions are permutations of nodes or symbols, encoding paths or permutations directly can be used (Goldberg and Lingle, 1985). Due to the requirements and constraints involved, this encoding scheme can cause problems for the

GA. Still, direct encoding of solutions for GA population members is very common and for most types of problems, works well.

The Traveling Salesman Problem is a good example of a *Non-deterministic Polynomial time-Hard* (NP-Hard) graph-space problem, and many attempts have been made to apply GAs to it. Solutions to the TSP consist of a linear, non-duplicating, and complete list of the nodes. Directly encoding these paths can cause problems for the GA as it attempts to create new solutions from old through manipulation of population members using GA operators. This is the case for other graph-space problems, such as the Snake-in-the-Box problem where paths must not contain duplicate nodes. With this direct encoding scheme, population members can be initially encoded successfully, but the GA cannot directly manipulate, using its crossover and mutation operators, these solutions without creating duplicate nodes or edges. Therefore, either the GA operators must be designed to avoid this, or the new, invalid population members must be "fixed" to conform to valid points in the solution space (see description of Grefenstette in Goldberg, 1989, pages 204-205). Both of these options can be difficult and can reduce the effectiveness of the technique.

The heuristic encoding scheme can mitigate these problems for GA operators and still produce good solutions to graph-space problems. In addition, this encoding scheme can implicitly prune the solution space in that the set of heuristics may only allow a small portion of the space to be accessed. This is due to the fact that there may be far more selections possible at any point in the path building process than there are heuristics to choose from. Therefore, a good set of heuristics should still allow exploration of promising areas while restricting access to less promising ones. These constraints are necessary in order to deal with the combinatorial explosion of NP-Complete and NP-Hard problems. Indeed, Wynn (2012) also uses built-in constraints for this purpose.

So far we have made some broad and strong claims regarding the value of this work. How can we support these claims in a scientific manner? The next chapter formalizes these statements through four primary hypotheses with statements supporting the value and motivation of each. Next, we introduce the basic concepts of GAs (Chapter 3), heuristics and hyper-heuristics (Chapter 4), the problems studied (Chapter 5), the heuristic sets (Chapter 6), and the specifics of our GA operators (Chapter 7). Later we explain the experimental framework used (Chapter 8), the specific experiments and the results obtained (Chapter 9), the analysis and explanation of these results (Chapter 10), and the conclusions that can be drawn (Chapter 11). Finally, we discuss several areas of future work for the continuation of this research (Chapter 12).

Chapter 2

Hypotheses and Motivational Statements

Heuristic encoding does produce competitive results when compared to more traditional GA implementations and to other types of algorithms and techniques (Carlson and Hougen, 2010). However, the design and selection of the heuristic set can have a strong effect on the ability of the GA to discover promising areas of the solution space. Indeed, it is the heuristic set that provides the mapping mechanism from the problem to the solution space. In order for the GA to take advantage of this mapping ability, changes should be made to the canonical or typical GA operators to account for this novel form of encoding. This dissertation will show this through investigation of the following four hypotheses and supporting experiments and analysis.

2.1 Hypothesis H1, Heuristic Representation

H1: Heuristic encoding schemes can effectively represent solutions to graph-space problems.

If this encoding scheme is not at least as good as others, there is little reason to use it. As claimed, the original purpose of heuristic encoding was to simplify the GA on permutation type graph-space problems such as the TSP. However, the true usefulness lies in its ability to prune the search space. An example is the TSP: When going from node one to node two in a 100 node problem, there are 99 nodes to choose from. Of course, as the path grows the number of selections decreases. However, this is still a very large decision space with roughly 99 factorial ($99! = 9.3326*10^{155}$) possible paths.

Using a 25 heuristic set limits the number of choices at each point of the path to a maximum of 25 (roughly $25^{n-1} = 2.4892*10^{138}$ for a 100 node problem, still large, but much more reasonable), a drastic reduction. Using the node scheme where the solutions are represented directly as a sequence of nodes, the size of the search space overtakes the heuristic scheme at about 62 nodes. Therefore, for any TSP larger than 61 nodes, the heuristic encoding scheme will increasingly prune the space below the size of the node scheme.

In contrast, what if the heuristics limit the choice in such a way that a good or the best path cannot be created? This is where the value of H1 is demonstrated. If it can be shown that the best known path in various problems can be reconstructed with the given heuristic set, then we have shown empirically that this set is capable of representing this solution. Can it be proven that the heuristic set is capable of always being able to represent the best solution? No, as we are dealing with NP-Complete problems. Thus, in general, it is not even possible to show whether a given solution is the best, without exhaustive search and comparison of every possible solution. Therefore, we show through empirical experiments that the heuristic set is at least capable of representing a broad set of different, world record solutions. This is done for a set of four problems in the TSP and four from the SITB.

NOTE: Of course, there are proven bounds for some problems that can be used to show a given solution has reached the limit, but this is only useful in certain situations. See

Arora (1998) for work on a *Polynomial-Time Approximation Scheme* (PTAS) for the TSP and other geometric problems.

2.2 Hypothesis H2, Encoding Scheme Comparisons

H2: Heuristic encoding used with traditional GA operators and parameter settings performs no better, and may perform worse, with more traditional encoding schemes using the same GA operators and parameters.

When initial work was done using heuristic encoding on both the TSP and the SITB, traditional versions of crossover and mutation along with fairly traditional parameter values, such as 40% probability of crossover, selection with replacement, et cetera were used. However, results were mediocre at best. The form of encoding was likely causing greater disruption of the population members when changes occurred, such as from crossover and mutation, than anticipated. Based on these prior results, heuristic encoding may not perform any better with traditional operators and parameter values than more traditional encoding schemes.

2.3 Hypothesis H3, Phenotype GA Operators

H3: When using heuristic encoding, phenotype operators improve average GA performance over that obtained with traditional GA operators.

As stated in Section 2.2, initial findings showed that heuristic encoding by itself was not sufficient to achieve good results with a GA. This is likely caused by the disruptive nature of the heuristic encoding scheme. Indeed, Mitchell (1996) states: “Some types of encodings require specially defined crossover and mutation operators.” While she may be speaking to the necessity of maintaining valid population members under the effects of GA operators, this statement also applies to the need for the operators to evolve better members. For, indeed, if GA operators are ineffective at this, there is no point to the GA.

In order to address these issues, different techniques were researched to mitigate the disruptive nature of the heuristic encoding scheme while still maintaining the value of the GA operators in exploring the solution space. The result was new GA operators that cause less disruption to population members when using this encoding scheme. These *phenotype operators* use information from solutions being produced, the phenotype, and are explained more fully in Chapter 7.

In the following discussions, *canonical* refers to accepted, well researched methods for GA operators, and *traditional* refers to parameter settings for various aspects of GAs that are more typical and more commonly used.

2.3.1 Hypothesis H3-1, Phenotype Crossover

H3-1: When using heuristic encoding, the phenotype crossover operator increases average GA performance above that obtained with the canonical, linear multi-point crossover operator.

2.3.2 Hypothesis H3-2, Phenotype Mutation

H3-2: When using heuristic encoding, the phenotype mutation operator increases average GA performance above that obtained with the canonical mutation operator.

2.3.3 Hypothesis H3-3, Combination of Phenotype Operators

H3-3: When using heuristic encoding, the combination of phenotype crossover and phenotype mutation operators increases average GA performance above that obtained using any other combination of canonical GA operators for crossover and mutation.

2.4 The Heuristic Set

The design and selection of the heuristic set should incorporate knowledge of the problem space appropriate for mapping to the solution space but should not include useless heuristics. If key heuristics are missing, the GA is unable to find promising areas of the solution space. Also, if useless heuristics are included, they detract from the guiding effects of the GA.

These statements would seem to make sense based on simple ideas of filter effects on signals, and noise within signals. Using the signal analogy, if a signal is overly filtered, especially if it has no noise component, valuable signal content will be lost. Similarly with a set of heuristics, if valuable heuristics (heuristics that have been shown to be useful in discovery of good areas of the solution space) are removed, the GA will be unable to reach potentially high payoff areas of the solution space. Also, if noise is added, the original signal becomes harder to distinguish. Given a set of heuristics that seem to work well, if noise is added, i.e., useless or unneeded heuristics are added, one would expect that the GA guiding the set of heuristics toward a solution to a problem, might have a more difficult time.

We have identified a small set of heuristics for the TSP and SITB that appear to be very valuable in contributing to finding good solutions. These heuristics were identified by observing how often each heuristic could be used during the process of re-creating the best known solution to a problem. For the TSP, the four problems (Table 8.1) used for Hypothesis H1 were again used for this purpose, and for the SITB, the best known solutions for D7 to D10 (Table 8.2) were used. When attempting to determine if the heuristic set can be used to encode a known world record path, at each step, we see if a given heuristic will select the next node according to the known path. If said heuristic will select the correct node, a counter for this heuristic is incremented. This process is performed for each node in the path, and the check made for each heuristic, for all of the problems identified. When complete, the heuristics are ranked based on counts of how often they made the correct selection. The larger the count, the more useful the heuristic is likely to be, when used with the GA. After ranking the complete heuristic set, a small sub-set from the top of the list (having the larger counts) was identified for both the TSP and the SITB to be used for these experiments. However, as the TSP is initially encoded using only heuristic number 1, this heuristic was not considered for this purpose (see discussion of TSP encoding in Section 6.1).

By removing this subset from each, the average performance is expected to decrease as this will reduce the effective searching ability of the remaining heuristic set and the GA managing them. Also, as this will restrict the area of the solution space reachable by the heuristic set, we also expect to find that the best solution found using the entire set is likely not possible with the reduced set.

Using this same set of counts for the heuristics, a small sub-set at the lower end was identified as possibly not useful. For the TSP, the original set contained 33 heuristics, and 8 were removed to create a standard set of 25 by this process. For the SITB problem, the original set contained 26 heuristics, and 9 were removed to create a standard set of 17. The worst (lowest count) heuristic removed from each set will be used as the “noise” heuristic for the experiments in support of Hypothesis H4-2.

2.4.1 Hypothesis H4-1, Removing Key Heuristics

H4-1: When key heuristics are removed, both upper end, and average GA performance decreases.

2.4.2 Hypothesis H4-2, Adding Noise Heuristics

H4-2: When additional, unneeded heuristics are added, average GA performance decreases.

Now that the plan for supporting this work has been formalized through the list of hypotheses, we need a better background in the algorithmic framework used in this work. Therefore, the following chapter gives a brief history and introduction to the mechanics of the typical genetic algorithm.

Chapter 3

An Introduction to Genetic Algorithms

While the primary focus of this work is solution representation and manipulation, the framework for this is the Genetic Algorithm (GA). Therefore, in order to understand the value and application of this work, we must understand the basics of this algorithmic technique, inspired by nature, as it applies to NP-Complete problems.

GAs are a type of Evolutionary Computation (EC) scheme inspired by the idea of genetic encoding and population dynamics from nature (Holland, 1975). There are many other types of EC algorithm, such as *Estimation of Distribution Algorithms* (EDAs), but this work focuses on GAs. (See Hauschild and Pelikan, 2011, for an excellent introduction and survey of EDAs.)

GAs are used in many areas including classification and control systems. Two primary areas are:

1) In modeling and simulation of various biological problems/processes to include population dynamics. There are many ways that the GA has been applied to study various problems and areas of biology and sociology (Mitchell, 1996, pages 15-16). In this role, they have proven useful in the natural sciences (Holland, 1975).

2) To actually solve (or at least find good solutions to) difficult problems whose solutions have practical import in industry, engineering, and scientific fields. There are

many good examples of this in the books by Mitchell (1996) and Falkenauer (1998). Also see Louis and Xu (1996) for an application to the Open Shop Scheduling problem. This is the area on which this dissertation focuses.

Initially the GA was created to study various aspects of evolutionary processes and population dynamics (Holland, 1975). However, they eventually became useful for many areas of problem solving and optimization. They do have limits and are not applicable to all NP-Complete problems. Some problems prove to be quite difficult and even deceptive for GAs as discussed in Goldberg (2002, especially Chapter 7). Also, for a thorough discussion of the intricacies of GA design, parameter tuning, and optimization techniques see De Jung (1993).

3.1 The Canonical GA

The canonical GA was developed and introduced by John Holland and his associates at the University of Michigan in the 1960s and 70s (Holland, 1975, Mitchell, 1996), and was initially referred to as Genetic Plans. Holland was motivated by the desire to use computers to simulate natural systems and for the study of parameter changes on these systems. The basics of the canonical Holland GA will be explained along with some of the common variations based on application to the second primary application area above: finding good solutions to difficult problems.

In order to use a GA, an encoding scheme must first be devised to represent potential solutions as a string of symbols, which will be referred to as *population members*. A group of population members is the *population* the GA will work with.

Based on the defined encoding scheme, it may be useful to derive custom GA operators as quoted by Mitchell (1996, page 173) “Some types of encodings require specially defined crossover and mutation operators.” While this is not “required” for heuristic encoding, it is expected to improve overall results. Once accomplished, the initial population can be generated. Unlike many problem solving techniques, GAs do not work with a single potential solution, but instead, a population of potential solutions. Starting with this initial population, the GA uses functions known as GA *operators* to create a new population from the initial. One complete iteration of this process is known as a *generation*, with the idea that the previous population is used to create the next generation of potential solutions. As this process continues, the population should evolve better solutions. Thus, the number of generations allowed usually is a function of how much time is available, the quality of solution desired, or various dynamics of the population (whether it has converged, when the last improvement was seen, et cetera).

3.2 An Example of a Simple GA

The following steps must be accomplished as part of the design and implementation of a GA solution. The first three set up and initialize the GA, while the last four constitute the generational cycle.

1) Determine an appropriate *fitness function*, or a function that can be used to rank the quality of the population members. The fitness function will be used to evaluate

individual population members for use in creating the next population.

2) Determine an appropriate encoding scheme and create an initial population of candidate solutions. This usually takes the form of a string of characters or symbols. Binary strings are very common. The individual symbols are referred to as *alleles*, with the set of possible symbols called the *allelic values* or *allelic set*. Integers representing heuristic identifiers are used for the allelic values in this research. Next, a random value from a uniform distribution is selected for each of the individual genes in the initial population.

3) Determine the stopping criteria for the GA.
This is typically some measure of the fitness of the best individual, the convergence of the population, or is related to resources such as running time.

4) Evaluate the current population based on a fitness function determined by some measure of a good potential solution (what is the expected outcome?), derived in step 1.

5) Determine if the stopping criteria has been met. Stop and report results if it has. Otherwise, continue.

6) Using the evaluations from step 4, create the next population through GA operators of selection, crossover, and mutation.

7) Return to 4 and continue.

For the fitness function in step 1, “fitness” refers to *objective fitness* in that we are trying to determine how well a given population member solves the problem at hand (does it produce a better path than other solutions, does it reduce the time necessary to traverse a graph, et cetera). This function is sometimes referred to simply as “the objective function” in the literature. To implement a fitness function, we need to determine a measure that can be used to grade or rank how well each individual solves the problem, and this is referred to as the fitness of the individual. This value will be very important to the GA as it evolves the population searching for more fit individuals. This definition of fitness is quite different than seen in the biological sciences, where it typically refers to the ability of an individual to survive and reproduce. Of course, in biology, there may be problems in the environment that individuals need to solve (finding food, defending territory, mate selection, and defense), but these are simply part of the overall goal of producing offspring that will themselves survive and reproduce. Thus, in biology, the primary measure of fitness is how many offspring a given individual can produce, whereas our measure of fitness involves the problem objective, objective fitness, or simply, fitness.

3.3 GA Specifics as Applied to NP-Complete Problem Solving

The following discussion focuses on application of GAs to solving NP-Complete problems in engineering, not as used in the biological sciences or simulation. The key components of the GA are as follows.

3.3.1 Measure of Fitness (Fitness Function)

We must devise an appropriate measure of the performance of an individual population member. Based on the problem being solved, each member will map to a specific point in the solution space. We need to determine what metrics of this point will be used as the measure of fitness. This fitness function will be used in various ways by the GA operators to produce the next generation. It may also be used as part of the stopping criteria. Finally, it may be used as a measure of the quality of the best solution found by the GA.

3.3.2 Encoding Scheme

In order to have a population of potential solutions for the GA to operate, we must determine how to represent solutions to the problem. Typically some form of character or symbol string is utilized. In its simplest form, a binary string is used. However, any fixed set of characters will suffice. The individual character locations are referred to as *loci*, the item occupying a given loci is a *gene*, and the specific values each gene can obtain are known as alleles. Thus, in a binary encoding scheme, the only allelic values are 0 and 1. In the heuristic encoding scheme, the allelic values are integers representing the heuristics in the set. Once the general form has been determined, we must decide how to map these strings of symbols to actual solutions. As an example, if the problem is determining the correct numeric values for a set of variables to optimize an equation, we might use a floating point format encoded into binary with each variable being assigned a certain number of positions, or genes, within a population member. During evaluation, these values are decoded to their numeric counterparts and applied to the equation to determine the calculated value. This final value will be mapped to the fitness function to determine the overall fitness of the member. There are an infinite variety of encoding schemes for a given problem, and the scheme used can have drastic consequences on the outcome of the GA. Indeed, Falkenauer (1998, page 180) states "While it is certain that an NP-hard problem cannot be made trivial by any encoding, it is the case that an inadequate encoding can make a problem look harder to a GA than it actually is." Finally, we must decide on the *ordering scheme* used for the genes. Linear ordering is most common where the genes are evaluated, interpreted, or mapped linearly from left to right. Other ordering schemes are also used and will be discussed as needed. Linear ordering is used for the examples in this section.

3.3.3 Stopping Criteria

Before running the GA, the stopping criteria must be determined. As with exhaustive search on a large problem, the GA can run, for all practical purposes, forever. Indeed, we may find that many improvements are made for the first hour of operation, but the next improvement may take ten additional years, in an extreme case. Some common criteria used are:

1) Is the current best solution good enough? This is based on the problem being solved and what is considered an acceptable solution.

2) Has the population converged? Has a single member of the population grown to occupy a certain percentage (determined by the developer and the dynamics being

considered), such as 75%?

3) How long since the last improvement was seen? An example might be if the GA evolves regularly improved solutions for *x* generations, but no new improvements have been seen for 2*x* generations, we may decide to stop.

4) How much time or computational resources can we afford? We may decide, based on empirical studies, that 5,000 generations will produce good results. However, if one can wait for 50,000 in the hopes of minor improvements, then this may be advisable.

3.3.4 GA Operator - Selection

We must devise some scheme for selecting members from the current population to participate in constructing the next. There are typically three mechanisms that allow a current population member to contribute: Either selection for direct copy, selection with an additional member for mating/crossover, or selection for mutation. Often, these can be combined, such as new offspring created through crossover may also be mutated. There are many systems for selecting but a very common form known as *tournament selection* is common and is used in some of the following experiments. Tournament selection randomly selects a small subset, and performs a tournament between them by selecting the individual with the highest fitness. The number of elements in the subset is known as the *tournament number*. If selecting for crossover, then two tournament selection operations will be needed in order to select two members for crossover.

3.3.5 GA Operator - Mating or Crossover

Once two members of the current population have been selected, they will be combined in some meaningful way to produce two new members for the next population. This mating scheme typically involves random determination of one or more points within the encoded population string which will be used as focal points for swapping subsections between the two members. When we refer to the crossover point, we mean the index value of the first gene used for the swapping process. Thus, if the crossover point is 4, then gene 4 and all genes with a larger index value will be included. Following is an example using single point crossover in a 10 character string where the crossover point is 4, using the alphabet as the encoding characters or allelic values:

Old member 1:	ABC BETOOPU	New member 1:	ABC ABEDTPB
Old member 2:	RRU ABEDTPB	New member 2:	RRU BETOOPU

This can be generalized to any number of crossover points and is known as *linear multi-point crossover*. Linear refers to the fact that the sections between the crossover points are swapped in a linear fashion. The more points used, the more disruptive the operation. Therefore, we use multi-point crossover with a single point for illustration. There are other forms that use many more crossover points, such as *parameterized uniform* crossover (Spears and De Jung, 1995), which is becoming more popular.

Crossover attempts to combine the best parts of individuals in random ways to produce better, more fit, offspring. Of course, often the offspring are less fit, in which case they should eventually be removed from the population by not being selected. This operator is the strength of the GA and typically provides the greatest increase in performance. “Being the most potent force in the GA, the crossover is also the most used operator in the algorithm. Indeed, in a typical GA most of the new individuals produced (i.e., new points of the search space explored) are generated by this operator." (Falkenauer, 1998, page 38).

3.3.6 GA Operator - Mutation

Mutation involves the low probability change of a gene from one value to another. Thus, for a binary encoding, this would be a bit-flip. When a gene is selected for mutation, typically a new value is chosen using a uniform distribution of all possible values. At its simplest, mutation is a random walk through the solution space, and is no better than any other form of enumerative search. As such, its usefulness and power are much less than crossover. However, its primary benefit is to assist in maintaining diversity and as an insurance policy against losing useful genetic material. The crossover operator has great power to explore the solution space. However, if a given allele does not exist in the population, or has been removed through selection, crossover can never bring it back. The mutation operator can, however, in that, when a gene has been selected for mutation, any allele can be selected as the new value. This does introduce another level of randomness, but empirical evidence suggests its value (Falkenauer, 1998, pages 40-43).

3.4 GAs in Conclusion

The real power of the GA lies in the GA operators and their ability to balance exploration of the solution space with exploitation of good potential solutions. However, the encoding scheme is central to the success of the GA (Mitchell, 1996). The encoding scheme and the development of appropriate GA operators is the focus of this research. The study, comparison, and analysis of these techniques is the domain of this dissertation.

Finally, a problem known as *epistatic interaction* can cause problems for the GA, and indeed, is a known problem in this work. Epistatic interactions occur between the genes in such a way that the contributions of a specific gene are affected by others (in the literature, the number of interactions is a constant value denoted by *k*, see Kauffman, 1989 for a thorough discussion). Due to this, it is not possible to state that a specific gene contributes *x* to the overall solution while another gene contributes *y* as the contributions of these genes may be intertwined with each other as well as with others. This problem is even more acute with heuristic encoding where a path is built from the genes in a population member. In this technique, when one gene is changed, it may affect the contributions of all downstream genes (genes that have yet to contribute to this specific path). This problem is dealt with to some extent through the use of the *locus-based approach* for the ordering of the genes within a population member (Jung and Moon, 2002). Using the locus-based approach for gene ordering, each gene represents a specific node in the graph-space, instead of a specific node in a given solution (linear ordering). These ordering schemes are discussed more completely in Chapter 6.

Another technique for mitigation of the negative effects of epistatic interactions involves mutation. For the SITB, a maximum of one gene is allowed to be mutated per population member. Finally, the phenotype crossover operator helps greatly to mitigate this problem. See Altenberg (1994) for a thorough study of epistatic interactions in GAs.

Next, we must understand how the GA can be used to manage low level heuristics in order to construct paths in a graph-space. For this we introduce the idea of heuristics and explain more fully how they are applied in this work.

Chapter 4

An Introduction to Heuristics and Hyper-Heuristics

4.1 Heuristics and Hyper-Heuristics

Heuristics are simple "rules of thumb" used to make decisions. They can be used to determine the next step to take given a specific state, as part of the solution to a larger problem. We may have a group of these rules of thumb which will each take the current problem state to the next state based on the current conditions. Each heuristic decides what selection to make based on the current state of the problem. Thus, heuristics may be viewed as simple state transition rules. The question of which heuristic to apply in any given situation is a separate problem which the GA attempts to solve. Therefore, the GA serves as a type of *hyper-heuristic*, a heuristic to choose or select lower level heuristics. The idea of a hyper-heuristic for solving NP-Complete problems is well known in the literature and examples can be found in Terashima-Marin et al (2008) and Garcia-Villoria et al (2011).

The following quote highlights the primary purpose of using a heuristic encoding for a GA; the pruning of the search space. "We resort to heuristic programming whenever an algorithmic solution is prohibitively expensive or impossible to follow, or is unavailable. The role of heuristics is to cut down the time and memory requirements of search." (Ralston, 1976, page 606).

Indeed, when the solution space to be searched grows exponentially, a good algorithm must be able to prune areas of little value, without removing areas containing better solutions. So, from an evolutionary computation standpoint, heuristics are often combined with other techniques for this purpose. Burke et al (2013) presents an excellent survey of the state of the art in heuristic and hyper-heuristic techniques.

4.2 The Genetic Algorithm as a Hyper-Heuristic

When used as the encoding scheme for a GA, heuristics do add an additional layer of abstraction that may mitigate some of the GA operator's manipulation problems, but may also introduce other problems. This scheme involves using heuristics as the values of genes for the population members in a GA, instead of encoding actual solutions. Several good examples of heuristic encoding for GAs applied to packing problems can be found in Terashima-Marin et al (2008), and Lopez-Camacho et al (2010). Thus, a population member is evaluated for fitness based on the goodness of the solution that can be built from its heuristics. In this type of scheme, the GA acts as a hyper-heuristic in that it guides the application of low level heuristics in an attempt to create a better sequence of heuristics (better population members) that will create better solutions. The GA operators simply need to operate on population members that consist of heuristic identifiers, any combination of which will produce a valid solution. The goal of the GA is to manipulate the population members such that the heuristics are arranged so that the ordering of their application produces better solutions as the population evolves. While heuristic encodings are not new (Hart, 1998, Terashima-Marin et al, 2008), their use as a GA encoding scheme for graph-space problems is rare (Carlson, 2002, Carlson and Hougen, 2010).

The idea of a hyper-heuristic used in this manner is well known in the EC community and an excellent survey of the state of the art can be found in Burke et al (2013). In this paper they state: “When using hyper-heuristics, we are attempting to find the right method or sequence of heuristics in a given situation rather than trying to solve the problem directly.”

Another quote from Burke et al (2013) strengthens the argument for using a hyper-heuristic to select or manage a set of lower level heuristics as is done in this work: “In particular, searching over a space of heuristics may be more effective than directly searching the underlying problem space, as heuristics may provide an advantageous search space structure.”

This quote does highlight the fact that heuristic encoding adds an additional layer of abstraction that causes the GA additional problems that we deal with using specialized GA operators discussed in Chapter 7.

In this work, we have devised a custom set of heuristics for the two problems studied, which the GA, as the hyper-heuristic, will arrange in such a way as to find “good” solutions to the two different graph-space problems. A different heuristic set will need to be designed for each different type of problem for which this technique will be applied. However, before a heuristic set can be created, the problem space must be studied in order to decide what feature set should be used to create the heuristics. While this research does not focus on the problem of feature set determination, a good example of this can be found in Lopez-Camacho et al (2010).

Examples of features used for the TSP heuristic set are:

- Distance between nodes
- Distance of a node from the Euclidean center of mass
- Relationship between distance to current path nodes and the Euclidean center of mass

Examples of features used for the SITB heuristic set are:

- Position on the link table (see Table 5.1 for link table examples)
- Number of available neighbors
- Relative position on link table with the inbound link
- Tightness of space around node

Heuristics have been studied and used in-depth in the application to modern problem solving. Indeed, several fine books have been written that cover many different techniques including various deterministic schemes, stochastic methods, and EC approaches (Michalewicz and Fogel, 2004, Pearl, 1985). However, the idea of developing a set of low-level heuristics, while not unknown, is not as common and highlights one of the primary contributions of this work: The process of low-level heuristic set development for NP-Complete problems.

With a good understanding of heuristics and the GA as a hyper-heuristic to manage them, we will explore the two graph-space problems used to develop, test, and support this research. While this research should be applicable to other NP-Complete problems, we believe the use of graph-space problems, where a path must be built, will best illustrate the techniques and ideas developed here.

Chapter 5

The Problems Studied

Two example problems from the realm of graph theory are used to explore and show the usefulness of combining heuristics with GAs and the implementation of phenotype operators. In both problems, the GA builds a path through the space by evaluating the heuristics that a population member consists of, instead of directly manipulating paths as population members. The two problems, the Traveling Salesman Problem (TSP) and the Snake-in-the-Box problem (SITB), are explained below. We use these problems due to their reputations and the large amount of work the EC community has expended on them. However, we have found very few others using GAs on large instances of these problems. Indeed, Zhang and Ma (2014) use an EDA, a type of EC to solve smaller instances of the TSP up to 136 nodes, whereas this work has been applied to problems over 2,000 nodes in size. Also, for the SITB, we are aware of only a single additional instance of contemporary GA research (Ruiz, 2014) on larger instances. However, Diaz-Gomez and Hougen (2006) apply GAs to smaller instances of the SITB. The primary reason for this lack of GA research on the SITB seems to be that typical approaches using GAs do not scale well on larger problems. Indeed, our work suffers this fate to some extent, but still produces reasonable results even on larger problems.

5.1 The Traveling Salesman Problem (TSP)

The TSP is a simple, easy-to-understand problem that is quite hard to solve (it is NP-hard). It is a path minimization problem where all of the nodes must be connected in a closed loop. The nodes themselves can be viewed as not connected initially, but are only defined with *x* and *y* coordinates in a 2D plane. We seek to connect them in a complete, non-duplicating, closed path. As a simple analogy: there is a group of cities that a salesman must visit. He must start and end at the same city, and he can only visit each city once. As fuel and time are valuable, he must find the shortest path possible.

There are infinite variations on this basic problem including 2D and 3D, but this work involves the Euclidean, 2D symmetric (the distances are the same in both directions) TSP. In this version, all we are interested in is the shortest closed path. Thus, we are not concerned with defining a start/end or whether we go from city A to B or B to A. In this scheme, there are many permutations of the nodes that actually represent the same path. As an example, given nodes, A B C D E F G H I J, the following paths are equivalent:

A-B-C-D-E-F-G-H-I-J
C-D-E-F-G-H-I-J-A-B
H-G-F-E-D-C-B-A-J-I

The end node connects back to the first, and this connection distance is included in the final path distance, but may not be shown in the paths themselves. Figure 5.1 is an example of a TSP named berlin52 from Reinelt's TSPLIB95 showing the best known path. A problem of this size can be solved without much difficulty, even by a GA. However, GAs do not scale well using node-based encoding on problems much larger.

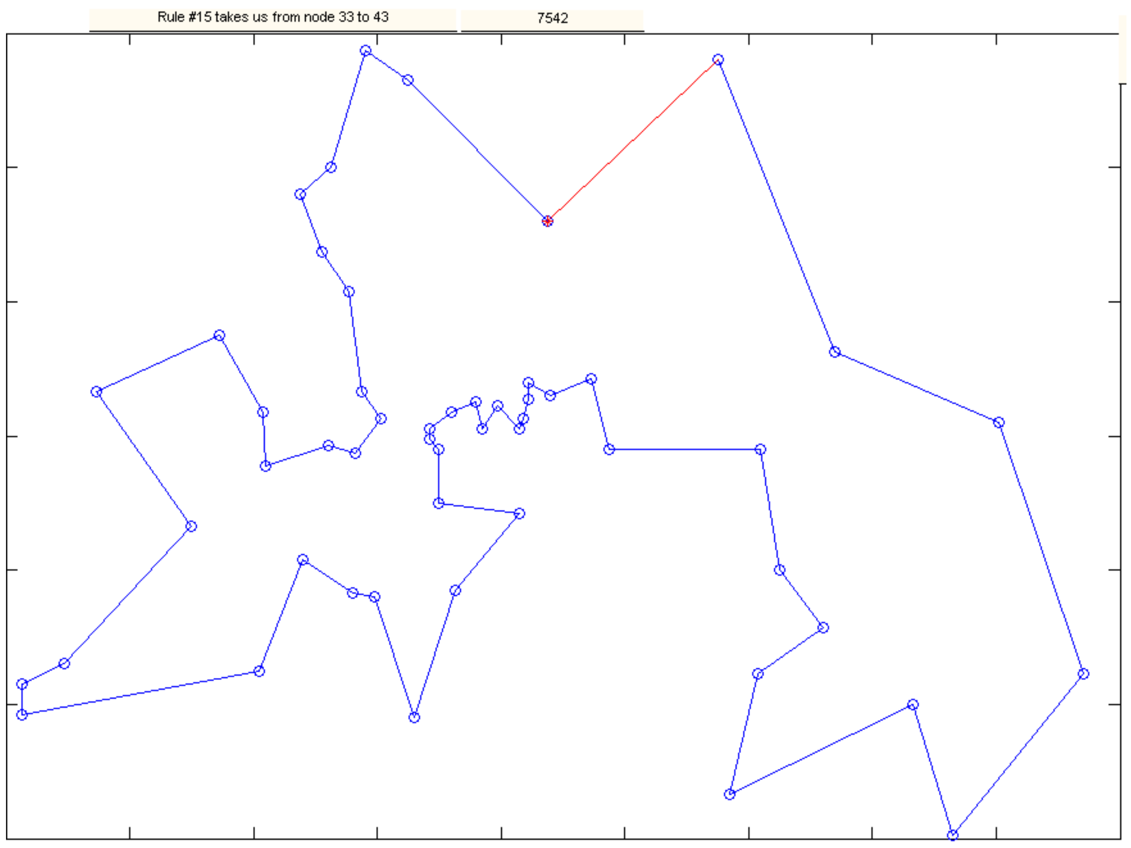

Figure 5.1: berlin52.tsp from TSPLIB95 with best path of 7542 shown. This solution was produced by the GA using heuristic encoding and the phenotype operators. The red link connects the last node in the path back to the root node, with the red asterisk.

5.2 The Snake-in-the-Box Problem (SITB)

The SITB is similar to the TSP in that it is a graph problem where an optimal path is sought. However, there are several key differences, first of which is that in the initial space, the nodes have a limited number of fixed links, instead of being a fully connected graph as the TSP is. The links are very structured and form a *hypercube*. These are the only links allowed. In the SITB we want to find the longest path that connects as many nodes within a given *n* dimensional hypercube as possible governed by certain constraints, explained in Section 5.2.2.

5.2.1 Hypercubes

The idea of finding snakes in hypercubes of different dimensions is referred to as "The Snake in the Box Problem" and was originally proposed by Kautz (1958). It involves a graph-space that extends the idea of a single point (a dimension 0 hypercube), to a line (1 Dimension), square (2D) and a cube (3D) further to dimensions not easy to represent in visual space, a hypercube. Figure 5.2 shows examples of a two dimensional representation of hypercubes of dimension 2, 3, and 4. This representation scheme is unique to this work (Carlson and Hougen, 2010). As the dimension is increased by one, the number of nodes is doubled. A hypercube of dimension *n* has 2^n nodes. Each node has *n* connections to neighbors creating a very structured space, a 2-colored graph. This is the *box*, the problem space.

5.2.2 Snakes

A *snake*, as put by Diaz-Gomez and Hougen (2006) is "a connected path in the hypercube d, where each node in the path has exactly two neighbors, except the head or source, and the tail, destination, that have only one neighbor."

The snake is a constrained open path in the "box" formed by the hypercube. Not only can it contain no duplicate nodes or edges but it also cannot have any *chords*, that is, a node in the path cannot have any path nodes next to it other than the one directly before it and the one directly after it in the path. This is a maximization problem as it is desired to find the longest snake possible in a given dimension. Thus, the set of all possible snakes (paths) of any length for a given hypercube dimension is the solution space for that problem space. Longest paths in hypercubes have use in coding theory (Kautz, 1958), hypercube computer communication schemes (Livingston and Stout, 1988), disjunctive normal form simplification (Potter et al, 1994) and other areas of science and engineering. Figure 5.3 has examples of the best snakes for dimensions 4 and 5 in two dimensional representation. In this figure, only the hypercube links forming the snakes are shown.

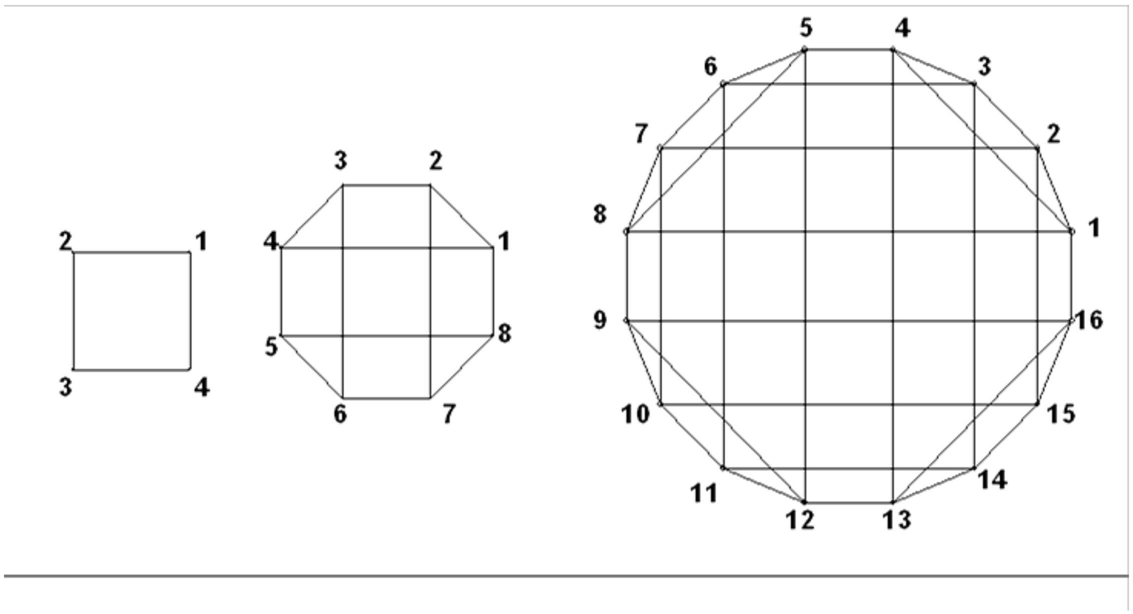

Figure 5.2: 2D circular graph for hypercubes of dimensions 2, 3, and 4. These graphs show the nodes as numbered in this work, which may be different from schemes seen elsewhere.

Also, in the literature, snakes are usually listed by the total number of links they contain. However, we use the node count. Therefore, most published snake lengths will be one less than the values contained in this document.

5.2.3 Snakes in Canonical Form

The search space for snakes in hypercubes can be pruned by taking advantage of some of the symmetries in the space. A simple constraint is used by Tuohy, et al (2007) but was originally proposed by Kochut (1996) and is quite simple: "Always begin at node 0." Tuohy starts node numbering at 0, whereas we start at node 1 in this work. Due to the symmetric structure of the space, all nodes are equivalent before a path has been started. This constraint greatly reduces the search space while not removing the best snake.

The second constraint is even more useful and is also used by Tuohy, et al (2007) and proposed by Kochut (1996): "Only consider snakes in *canonical form*." As stated by Tuohy, et al (2007), "Snakes in canonical form are those which only use higher-order dimensions after every lower-order dimension has been used at least once." This is actually a very simple idea. The hypercube structure has many symmetries and one is the fact that each dimension is created by connecting two copies of the previous dimension. Thus, D7 simply consists of two copies of D6 with matching nodes linked together. This additional set of links is always the largest number in the link table (see Table 5.1). Therefore, if we are building a snake in D5, we must include a node from the D4 portion of the hypercube before adding a node from the D5 addition. But first, a node from the D3 portion must be included before the first from D4. This continues until we are at the root dimension of D0 which only consists of node number 1. D1 consists of two nodes, the node from D0, already in the snake, and a second node with a single link between them, which must both be in the snake before a node from D2 (a square) is added. Table 5.1 shows the link table for D2, D3 and D4. In this table it can clearly be seen that D2 is embedded in D3 which is embedded in the D4 table, as D4 will be embedded in the D5 table, et cetera. Finally, Figure 5.3 shows the best snakes in D4 and D5 but not in canonical form, whereas Figure 5.4 has the logically equivalent snakes in canonical form.

Finally, with a solid understanding of GAs, the two problems used, and how heuristics can be combined with a GA to find solutions for these two problems, we will introduce the specific heuristic sets. The following chapter will introduce the general heuristic sets and give specific examples, while a complete list of the heuristics can be found in Appendix A. Also, the concept of gene ordering and the different implementations of this will be more completely explained.

Table 5.1: Link tables for hypercubes of dimension 2, 3, and 4. Each row shows the nodes that a given node has connections to, which are bi-directional. It can be seen here how the link table for d2 is embedded in the table for d3, and the table for d3 is embedded in the table for d4. Refer to Figure 5.2 to see the actual links as they appear between the nodes.

Node	Link	list
number	1	2
1	2	4
2	1	3
3	4	2
4	3	1

Node	Link	list	
number	1	2	3
1	2	4	8
2	1	3	7
3	4	2	6
4	3	1	5
5	6	8	4
6	5	7	3
7	8	6	2
8	7	5	1

Node	Link	list		
number	1	2	3	4
1	2	4	8	16
2	1	3	7	15
3	4	2	6	14
4	3	1	5	13
5	6	8	4	12
6	5	7	3	11
7	8	6	2	10
8	7	5	1	9
9	10	12	16	8
10	9	11	15	7
11	12	10	14	6
12	11	9	13	5
13	14	16	12	4
14	13	15	11	3
15	16	14	10	2
16	15	13	9	1

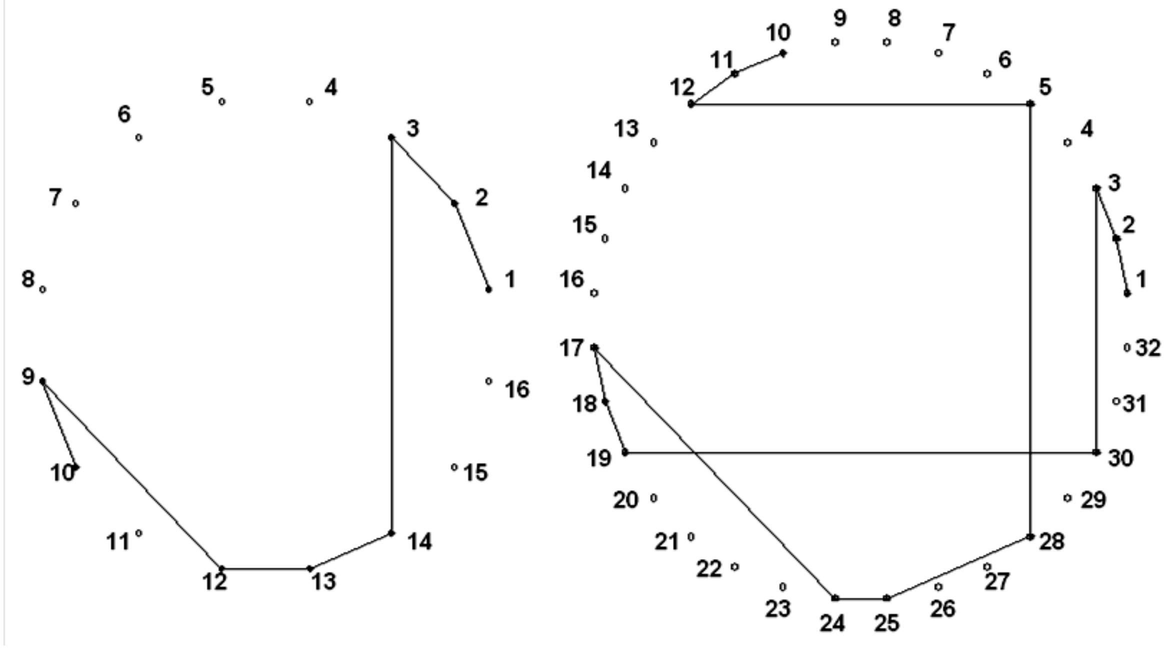

Figure 5.3: 2D circular graph of best snakes in dimensions 4 and 5. Lengths are 8 and 14 nodes. In order to focus on the snakes and their links, the unused links are not shown.

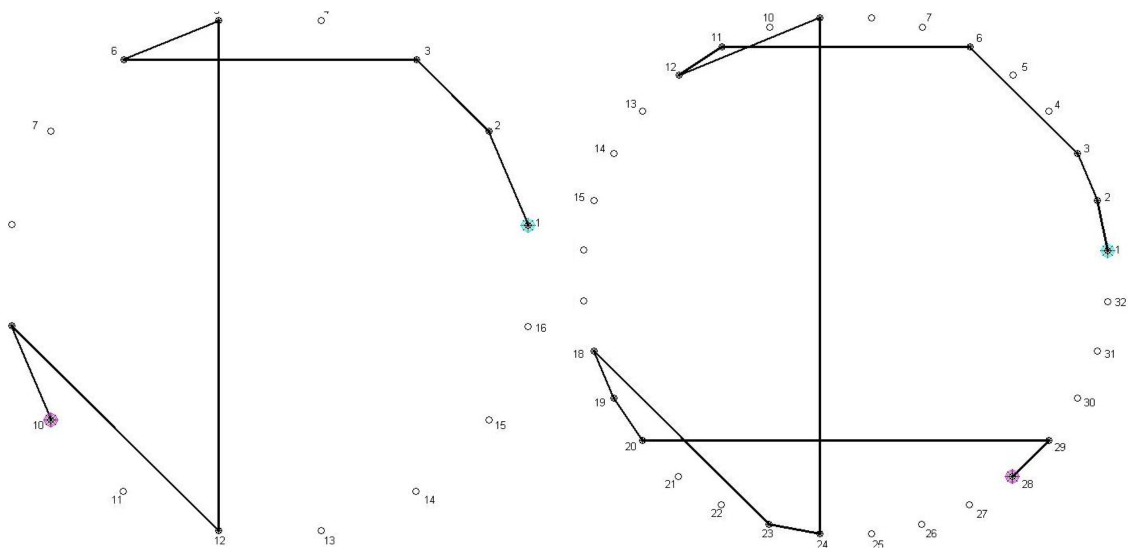

Figure 5.4: Same snakes from Figure 5.3 in canonical form. In order to focus on the snakes and their links, the unused links are not shown.

Chapter 6

The Heuristic Sets and Gene Ordering Schemes

As previously discussed, rather than directly encoding solutions as population members, instructions for producing solutions are encoded as the population members. Using knowledge of the problem space, a set of low level heuristics was devised which use the current status of the problem space to make a decision of which node to add to the path next.

There are several techniques for determining the order of gene processing or evaluation within a population member. In a *linear ordering scheme*, each population member's genes is evaluated linearly, from left to right. Thus, using heuristic encoding as an example, if we had a linear population member of 5 genes, these genes would be evaluated individually, in linear order, from left to right as such:

Index:	1	2	3	4	5
Genes:	8	4	1	6	3

Evaluate gene number 1, which has a value of 8 first. Then evaluate gene 2 with a value of 4, and continue until the final gene, gene 5 is reached, with a value of 3.

Rather than using linear gene ordering, Jung and Moon (2002) use a gene ordering scheme referred to as the *locus-based approach*. With this approach, each gene corresponds to a node and when that node is added to the path, its allele is used for the selection of the next node. This technique was found to work better than linear gene ordering for our work and it is used for both the TSP and the SITB.

Locus-based ordering evaluates based on the gene's position, or loci, as a path is built, and may not be applicable to all types of problems that a GA can be applied to. However, it seems natural for path building problems such as the TSP and the SITB. Using the example above, we would still start at the far left, evaluating gene 1. However, based on which node heuristic 8 leads to in building the path, the ordering of the gene evaluations may change. Thus, if heuristic 8 directs the GA to add node 3, the gene at index 3, which is 1, would be evaluated to determine what node to add next. In this way, the GA will skip around evaluating the genes in the order that the nodes are added to the path being built.

Through experiments, it was found that the locus-based approach does indeed produce better results than the typical linear alternative. This is likely due to the fact that individual heuristics are more easily tied to a specific node. However, this is more advantageous for the TSP than for the SITB. In the SITB, due to the symmetric nature of the space, it seems better to have the allelic values tied to positions in the path itself, rather than the space (the hypercube). Given this statement, however, we have still found that the locus-based gene ordering scheme works better even with the SITB, and it is used for both the TSP and the SITB throughout this work.

One additional detail is the determination of the starting node, or *root node*. For the TSP, we simply choose a starting node at random for each population member and encode this as an additional gene at the end of each population member. This will be the node that the population member's path will start from. Thus, for N nodes, the root node value is stored at the N+1 position in each population member. For the SITB, we always start at node 1 due to the symmetries in the space. Thus, a “root node” need not be encoded for the SITB population members.

Finally, one difference between evaluation of SITB population members and TSP members involves the fact that the TSP uses all nodes while the SITB does not. Since a constrained path is being built in the hypercube of a given dimension, over half of the total nodes will not be used. Therefore, when the path has been built as far as possible through adding nodes to the head, it might be possible to grow the snake (add additional nodes) to the tail. The evaluation algorithm for the SITB does this. This quite often leads to snakes a bit longer than would have been possible otherwise. This idea was suggested through a personal communication with Lee Altenberg in 2009.

Following is a brief description of the heuristic sets devised for the TSP and the SITB problems. A complete description of each heuristic including background definitions can be found in Appendix A.

6.1 TSP Heuristics

There are 25 heuristics in the TSP set. For all, if the requirements of the heuristic cannot be fulfilled, then the closest node to the current node that has not been visited will be selected. The most used heuristic is heuristic number 1, which selects the closest available node. Indeed, initially, the entire population consists of random root nodes but all gene values are 1. This initialization scheme is also used by Zhang and Ma (2014) to initialize their population for their hybrid EDA. However, the other heuristics must be available or improvements cannot be made. As part of the heuristic scheme for the TSP, the Euclidean center of the problem based on the complete set of nodes is determined. This "Center of Mass" is used in several of the heuristics, for example

- Heuristic number 3: Select the node that is the closest to the center.
- Heuristic number 4: Select the node that is furthest from the center.

Appendix A.1 contains a complete list and description of the TSP heuristic set.

6.2 SITB Heuristics

For finding maximal length snakes, it is useful to exploit the regularities of hypercubes. Our approach uses heuristics that choose a link from the link table (See Table 5.1 for an example of a link table) based on the current node (head node) and the *state of the search space*. The state of the space changes with every node added to the snake. When a node is added, several nodes may be eliminated from consideration later due to being neighbors of the previous head. Heuristics that evaluate the state of nodes near the current head or combine this information with knowledge of the global state could be used to add intelligence to the selection process (Carlson and Hougen, 2010).

We have created a set of 17 heuristics that will select the next node when building a path. Each heuristic was designed in such a way that if the link it would normally choose is not valid due to the node it points to already having a neighbor in the snake, the heuristic will look for the next most attractive link based on what the heuristic was designed to achieve. Some of the heuristics use position information in the link table and others use information about the state of a node's neighbors in making a decision. Some example heuristics are:

Heuristic number 1: Select the first node moving from left to right across the link table.

Each node has a row in the link table that shows the nodes it is connected to. For this heuristic, we find the row for the current head node and simply select the first available node from this row, starting at the left.

Heuristic number 6: Select the node with the largest invalid count value. Select the right node in a tie.

This heuristic evaluates each available node and determines how many nodes each connects to that cannot be added to the snake (thus, they are invalid). It selects the node that has the largest value of this count. However, the node that is selected must have at least one available node, or the snake could not grow past the selected node. This heuristic seeks to select nodes that keep the snake tightly coiled.

The idea of *tightness* means that, when a node is added, we want as few nodes to be disqualified, or removed from possible future inclusion, as possible. Thus, if we add a node that has 5 neighbors that are currently available to be added (they have no neighbors in the snake), then 4 of those nodes will be wasted upon adding said node, due to the fact that, once said node is added, we can only make use of one of the 5 available neighbors to continue the path. The other four neighbors will be wasted and can never be included. However, if we have a possible node that only has one available neighbor, then, if we add said node, we will waste no nodes as the only available node will be added next. Thus, a tight snake is one that makes good use of the nodes as they are added. The drawback is that we are more likely to get stuck in a dead end if we overuse the concept of tightness.

A complete list and full description of the heuristics can be found in Appendix A.2.

While the idea of a GA has been explained, we must flesh out the ideas of the phenotype operators developed specifically for heuristic encoding. The following chapter will explain, in detail, the specifics of the GA as used in this work.

Chapter 7

The GA Operators

The generation of the initial population for the TSP involves several idiosyncrasies that must be explained in order to understand and appreciate the descriptions of the GA operators. It was found through trials with the TSP training set that, while the entire heuristic set is needed to achieve good results, it is best to start off by having each node simply connect to the closest available node. Thus, initially heuristic 1, "select the closest available node," is used as the allelic value for all genes. This starts each population member off with a relatively short path that will slowly be improved using mutation to replace heuristic 1 with different values, and using crossover to intermix these improved population members.

Also, the root node is a separate value for each population member. During initial generation, the root node for each population member is randomly selected from all nodes using a uniform distribution. This root node will stay with this population member unless it is changed through mutation.

The initial population for the SITB is typical of GAs in that each gene is simply randomly set to one of the available heuristics with no bias (uniform distribution). The population is always evaluated starting from node 1 so there is no issue with selecting a root node as there is with the TSP.

7.1 The Fitness Function

The fitness function is used by the GA as part of the evolutionary process. It is also, for use in problem solving applications, the measure of success of the GA run. Since we are typically only interested in one solution, the fitness function will be the qualifying value for the selection of the best population member which will be used as the overall solution to the problem. Following is a brief description of the specific fitness functions used in this research.

7.1.1 Fitness Function for the TSP

With the TSP, we are interested in the shortest distance connecting all nodes in a closed path. We use the same scheme for measuring this distance as the TSPLIB95 web site (Reinelt) so that our values can be directly compared. For the node coordinates, floating point values must be used. However, the final calculation of distance between each pair of nodes is rounded to the nearest integer. The following gives an example:

$$x_d = x_i - x_j$$

$$y_d = y_i - y_j$$

$$d_{ij} = \lfloor \sqrt{x_d^2 + y_d^2} + 0.5 \rfloor$$

where all values are floating point except the final distance; d_{ij}. In this way, the distances between each pair of nodes in the path are added together as integers to produce an integral final value which is unit independent. During the GA evolutionary process, this path distance value is computed for the path constructed by each population member and these values are used in selection and crossover for the next population. Also, the best, shortest path found for the final population will be the end product of the GA.

7.1.2 Fitness Function for the SITB

The SITB fitness function is a bit more complicated than for the TSP. While we are still looking for a specific path distance, here the path is measured in number of nodes, and it is a maximization problem. However, the biggest difference is that a separate function is used for the GA evolution and for the final solution. The reason for this is the granularity of the solution space is not as fine as with the TSP. There may be many paths that are quite different but that contain the same number of nodes, whereas with the TSP, this does not occur very often. As an example, in D7 where the maximum/best path is 51 nodes, there may be 20 very different population members of length 49. How can they be ranked to determine which is more likely to eventually produce a 51 node path? This problem actually occurs often with GAs and a way must be determined to judge the goodness of a population member outside of the solution to the problem that it is capable of producing. For the SITB problem, this is done by determining whether two paths of equal length also have equal probability of being easily modified, through crossover or mutation, to a longer path by examining the state of the nodes that are not part of the current path. These nodes can be classified in two ways:

1) They are invalid and cannot be added to a path.

2) They are valid (they have no neighbors in the path) and could eventually be added to the path.

The more valid nodes, the more likely an existing path can be modified to include them. Thus, for the fitness values that the GA uses, we combine the current path length with a fraction based on the number of available nodes as follows:

F_i = (length in nodes) + (number of available nodes) / (total number of nodes)

where F_i is the final fitness value for population member *i*. This value still puts the emphasis on length and the fractional part will never be very large (less than one). Thus, a snake of length 49 will not be valued higher than a snake of length 50 no matter how many available nodes it has. This scheme allows the ranking of snakes of the same length and has proven empirically to be much better than using length alone. Indeed, while we discovered this scheme independently, it is used by others (Tuohy et al, 2007). This extra measure of fitness is referred to as tightness in that it measures how closely the snake has grown to itself (refer to Section 6.2 for a more thorough discussion of tightness). As the snake grows, it can either move into areas of the hypercube that are largely untouched, a *loose snake*, or it can try to make use of the nodes near the nodes that are currently in the snake, a *tight snake*. A tight snake has a higher probability of growing further with minor modifications. The final product of the GA only includes the length, as that is what is of interest overall. Thus, the results in this document all contain only length since the fractional values are only used within the GA.

This is not the only measure of fitness that could be used for the SITB problem. Indeed, Diaz-Gomez and Hougen (2006) use various other factors when determining the fitness of a population member, to include partial paths remaining within the hypercube. Their technique may make better use of determining the growth potential of a given population member, whereas our fitness measure looks primarily at the length of the valid snake produced, with only a secondary value reflecting growth potential.

7.2 The Selection Operator

As stated in Section 3.3.4, tournament selection, while not the first proposed selection operator, is very commonly used today. As explained below, tournament selection is used for the TSP, but not for the SITB. Various other selection schemes were attempted, including some custom algorithms, but did not perform as well.

Selection is used in two ways for both problems:

1) Select *x* percent *without replacement* from the old population for direct copy to the new. In selection without replacement, each member can only be selected one time. The value for *x* found to work best for both problems is 70%, which means only 30% of the new population will be created through crossover.

2) Selection for crossover. Here, a selection operator is used for selecting two members from the current population *with replacement* for crossover where they will only be used to create two new members for the next population, and will not themselves be copied to the new population. In selection with replacement, there is no limit on the number of times a given member may be selected. For this, tournament selection is used

for both problems. Other selection mechanisms were tried but found to perform worse. Also, a tournament number of two is used with tournament selection for crossover in both problems.

Finally, a form of selection elitism is used in that, regardless of which selection operator is used, the best population member from the current population is guaranteed to be selected at least once for direct copy. This ensures that the high point found so far is retained and available for future exploration of the solution space. This does have the drawback of increasing the chance of the population getting stuck and converging to a local maximum that is not global, as is shown by Gonzalez (2009) where selection elitism is shown to cause stagnation in certain situations. However, our experiments have shown that this is still a good technique.

7.2.1 Selection for the TSP

As stated above, several selection operators were tried but tournament selection was found to work best for the TSP (refer to Section 3.3.4 for a description of tournament selection). After trying several values for tournament number, two was found to work best. As the tournament number increases, the selection pressure increases as well, meaning that the better members of the population will be more likely to be selected and, thus, more quickly take over the population. Indeed, we found with larger values, the population often converges too early to a member that is not as good as can be achieved with a smaller tournament number. Here, we want to balance selection pressure with the need to explore the solution space. Too much exploration and the GA simply performs a random search. Too little and the GA converges too quickly to sub-optimal areas. The concept of exploration versus exploitation was originally discussed in Holland (1975), but is also seen in Mitchell (1996, page 118).

7.2.2 Selection for the SITB

It was found that selection percent works better and more consistently for the SITB. While tournament selection has a stochastic nature to it since the elements are chosen randomly, selection percent does not. With *selection percent*, the best *x* percent of the old population is selected for direct copy to the new. Tuohy et al (2007) claims that this technique helps to maintain diversity, and it seems to be true for the SITB based on our experiments. However, tournament selection clearly works better for the TSP in the experimental framework we are using. This may be caused by the population dynamics and the range of possible fitness values. With the TSP, there is a large variation of fitness values, and it is unlikely that two very different population members will have the exact same fitness value. Thus, when choosing two for a tournament, it is less likely that a tie will occur. However, this is not true with the SITB where a constant problem has been maintaining a diversity of fitness values, due to the fact that there are many symmetries in the solution space. In other words, there are many paths that are the same length, but are actually logically different from each other. Therefore, under these conditions, tournament selection is likely to result in many ties, which will weaken the value of this selection technique.

7.3 The Phenotype Crossover Operator

The basic GA crossover operator does not use any information from either the fitness function or the actual solutions produced by the population members being mated as part of the crossover function. It only uses the population members themselves, the genotype. The *phenotype* refers to the actual solution or creature (in biology) produced by the genotype. This is where the phenotype crossover operator differs dramatically. Since the heuristic encoding scheme adds an additional layer of abstraction to the GA, the phenotype is no longer identical, or even similar to the genotype (as is usually the case). Thus, there is a loosening of the guiding effect of the GA and the binding between the population members and the solutions they produce. When we first started work on the heuristic encoding scheme, it was found that performance was not what was hoped. This may have been due to the additional layer of abstraction and, at the time, we could not determine how to deal with it. Eventually, it was realized that, if we could use the solutions produced to guide the crossover operator, we may be able to mitigate this problem. Indeed, this has been done for both of the problems studied. Shortly after implementing this new crossover operator for the SITB, the world record length snake in an 8D hypercube was found using it (Carlson and Hougen, 2010). There is one serious difference in how this operator works for the two problems. Thus, each is explained separately in Sections 7.3.1 and 7.3.2.

The mechanism for selecting two population members for crossover is fairly typical in that tournament selection with a tournament number of two is used to select two from the old population for crossover. Many other selection techniques were tried but this seems to work best for both the TSP and SITB. Also, a form of elitism is used in that the best population member from the prior population is guaranteed to be selected for crossover at least once.

7.3.1 Phenotype Crossover for the TSP

The TSP crossover operator is designed to work with the locus-based gene ordering of the population members in order to keep alleles tied to specific nodes in the problem space. In this way the appropriate heuristics will eventually be assigned to the nodes that they will work best with through the evolutionary cycles of the GA. In this scheme, the node values are encoded based on the value at the node's position, or locus within the population member. Thus, if the value at index 3, which is node 3's value (it belongs to the third node in the problem space), is 6, this means that from node 3 we would visit node 6 next. From node 6, we see what value is at index/locus 6, and that is the next node visited (added to the path). Finally, we must determine what the starting, or root, node should be. This is determined by attaching an additional value at the end of the population member. This value is the root node. Thus, if the problem has 9 nodes, each population member will have 10 values. An example with 9 nodes using node based encoding follows where the first line shows index values and the second line contains the data.

Given the locus-based population member, with the root node as the last value:

Index: 1-2-3-4-5-6-7-8-9
Genes: 9-3-5-7-8-4-1-6-2-1

The associated path would be:

Index: 1-2-3-4-5-6-7-8-9
Nodes: 1-9-2-3-5-8-6-4-7-1

As can be seen, the last node in the open path is automatically connected back to the first node. Therefore, the last node's allelic value does not really matter and is not evaluated: the path will be connected back to the root no matter what node seven's value is. In this example, we see that the value of a node in the population member itself represents the node that should be visited next in the path. The idea is the same with the heuristic encoding scheme. The only difference is that, instead of node values for the genes, there will be integers representing heuristics. In this scheme, if the current node in the path is node 4, then gene (locus or index) 4's heuristic is used to determine the next node to visit.

With an understanding of the locus-based scheme, it will be much easier to explain the crossover operator. With the phenotype crossover operator, the actual path produced is used to determine how to intermix the two selected members. This will be illustrated using a simple 9 node problem and single point crossover at position 4. Also, the following will only use a made-up set of 5 heuristics, which need not be defined. Again, in the paths and population members listed below, the first line will simply be index values for reference. Also, for all examples, population members will be abbreviated M and paths as P.

Given the two paths (* indicates dividing point for crossover in the paths):

```
        Index:  1-2-3*4-5-6-7-8-9
P1)     Nodes: 1-2-3*4-5-6-7-8-9-1
P2)     Nodes: 6-1-9*2-3-4-5-7-8-6
```

And the associated population members, where the last value represents the root node to be used:

```
        Index:  1-2-3-4-5-6-7-8-9
M1)     Genes: 1-1-5-4-1-5-2-1-2-1
M2)     Genes: 1-2-5-1-2-1-4-1-5-6
```

We start by copying these two population members into the new population. After copying, we perform the crossover on the two, new copied members. In this example, the fourth position of P1 is 4 so we take the allele at gene 4 from M1 and put it into gene 4 of M2. Next, the fifth node in P1 is 5, so we copy the value from position 5 of M1 into position 5 of M2. We continue this process until we get to the last node in the P1 path, 9, before returning to the root node of 1, which causes the value 2 to be copied from gene 9 of M1 into gene 9 of M2. Next, the same process is performed on M1 using P2 and M2.

Following are the two new population members created using this process:

```
        Index:  1-2-3-4-5-6-7-8-9
M1)     Genes: 1-2-5-1-2-5-4-5-2-1
M2)     Genes: 1-2-5-4-1-5-2-1-2-6
```

While this example uses a single crossover point, all experiments use two. From this example, we can see the need for mutation in that, heuristic 3 is not represented. If 3 does not occur anywhere in the population, mutation is the only tool available to re-introduce it.

7.3.2 Phenotype Crossover for the SITB

Phenotype crossover for the SITB is very similar to the TSP crossover operation with one exception. Whereas for the TSP, we try to keep the allelic values with the nodes to which they are assigned, with the SITB, we want to keep the allelic values tied to a position in the snake. Of course, the GA operators will slowly change them as the generations proceed. Thus, if a current population member produces a snake of 50 nodes, we want the allelic values that selected each node to remain in the position where it will make the same selection when crossed over. In order to accomplish this, we need to know not only the snake that each population member creates, we also need to dynamically build two snakes as we cross over two members, to keep track of which heuristic selects which node at which snake location in each new population member as they are being constructed through crossover.

As with the TSP, initially a copy of both population members is put into the new population. Next, these copies are used to start building snakes up to the first crossover point. Here, the snake nodes from the opposite population member are used to determine which member node to use in continuing to build each snake. The gene used is actually cross copied to the other population member once it has been determined. This continues until the second crossover point is reached, at which point we change back to using the snake nodes from the original population members. Also, when one of the snakes being built dynamically to guide the gene selection and copying can no longer grow, the crossover operation is discontinued for both new members and they remain as is.

This is a complicated process and requires an example. Following, we illustrate the process using a 5D hypercube and a single cross-over point at index 6 (index 7 in the snake, meaning that the first 6 nodes in the path will be identical in the new members created). As can be seen from the snakes created from the original population members and those from the new, the snake nodes past node 6 have been swapped. This is accomplished by using the snake nodes to determine which gene values to swap. As an example, the sixth node in each snake is node 12 hypercube. However, in population member 1 (M1), the next node in its snake is hypercube node 13 while in M2 the next node in its snake is hypercube node 9. We want to swap the heuristics at these two loci (12 in both) so that the gene at locus 12 in M1 will select hypercube node 9 next and the gene at locus 12 in M2 will select hypercube node 13 next. We see this in the new population members where the heuristic in M1 at locus 12 has changed from 6 to 7, and in M2, from 7 to 6. Next, in the snake produced by the old M1, we see that hypercube node 20 follows hypercube node 13, and in M2's snake, hypercube node 10 follows hypercube node 9. Therefore, in M1 we want a heuristic at locus 9 that will select hypercube node 10 next, and a heuristic at locus 13 in M2 that will select hypercube node 20 next. Looking at the new M1, we see that the heuristic that had been at locus 9 (heuristic 3) has changed to have the value at locus 9 in M2 (heuristic 2). Also, in the new M2, the heuristic value that had been at locus 13 (heuristic 21) has changed to the heuristic value at locus 13 in M1 (heuristic 12). This same process will continue until no more nodes can be added to either snake, or the original snakes have reached their ends. In the new/recreated population members, in M1, the gene at locus 12 changed from heuristic 6 to heuristic 7 and from heuristic 7 to heuristic 6 in M2. The goal of this operator is to keep the heuristic that selected node number *x* in the snake (not node *x* in the hypercube, but in the numbered sequence of snake nodes) such that it still selects node *x* in the newly created population member. Finally, the first 4 nodes in the snakes are 1, 2, 3, and 6. These are the base nodes when using canonical form, where 1 is always the initial node. An additional example, without the explanation can be found in Appendix D.

* is the crossover point

Initial Population Members and Snake

Index values:
```
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
```

Selected members (M1 and M2):
```
26 20  3 13 16  6 13  1  3 19 16  6 12 20  1 21
 4 10 24 10  4  8  1 10  4 24  1  1  1  6  8 10

16 26 14 16  3 19 12 21  2 21  4  7 21 12 13 14
19 10 21 20 10 24  4 24  4  4  7 10 13  1 16 21
```

Selected snakes:
```
 1   2   3   6   5  12 * 13  20  17  18  23  26  25
 1   2   3   6   5  12 *  9  10  23  22  19  20  17
```

Final Population Members and Snake

Index values:
```
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
```

Reconstructed members (new M1 and M2):
```
26 20  3 13 16  6 13  1  2 21 16  7 12 20  1 21
19 10 21 20  4 24  4 10  4 24  1  1  1  6  8 10

16 26 14 16  3 19 12 21  2 21  4  6 12 12 13 14
 4 10 21 10 10 24  1 24  4 24  7 10 13  1 16 21
```

Corresponding snakes:
```
 1   2   3   6   5  12 *  9  10  23  22  19  20  17
 1   2   3   6   5  12 * 13  20  17  18  23  26  25
```

7.4 The Phenotype Mutation Operator

The phenotype mutation operator has a strong phenotypic side when applied to the SITB, since not all of the nodes are used to create a path. While the TSP mutation operator can more properly be described as probabilistic, it will still be referred to as a phenotype operator to simplify discussion. However, both use the idea of re-assigning probabilities for selection based on the current heuristic use statistics of the best 10% of the population, with the SITB including only those heuristics actively being used to create snakes. This redistribution of probabilities is nearly identical to the way that an Estimation of Distribution Algorithm reassigns probabilities as it evolves a set of solutions, as a quote from Hauschild and Pelikan (2011) illustrates: “The important step that differentiates EDAs from many other metaheuristics is the construction of the model that attempts to capture the probability distribution of the promising solutions.”

This technique has proven quite successful for EDAs and is generating much interest in the EC community. Therefore, its use in this research for reassignment of probabilities for allelic selection during gene mutation is quite justified.

The typical GA will use a uniform probability distribution when selecting a new allele for a mutating gene. We realized that there may be heuristics that work better than others, on average, and some that work better on specific problems. Thus, some way for the GA to adjust the selection probabilities seemed a good idea. In both problems, every 20 generations, the heuristic use statistics for the best 10% of the current population are used to create a new, weighted probabilistic selection distribution for mutation. In addition, the very best population member gets a double weight (if heuristic # 2 is used 5 times in the best population member, it will count as 2×5 or 10). Also, the minimum selection probability will never drop below 0.2%. This seems a small number, but it does allow all heuristics to maintain some presence in the population.

The actual equation for calculating the N individual probabilities is:

$$P(n) = 0.002 + div \times (2 \times CB(n) + C(n))$$

where n is the number of the heuristic being calculated from the total of N heuristics, N is the number of heuristics in use (17 for the SITB and 25 for the TSP), $CB(n)$ is the count of how often n is used by the best population member, $C(n)$ is the count of how often n is used by the remaining best 10% of the current population,

$$div = (1.0 - N \times 0.002) / (2 \times TCB + TC)$$

TCB is the total count of heuristic use by the best population member, and TC is the total count of heuristic use for the remaining best 10% of the population.

In a typical GA, mutation is applied to all population members in the new population: those directly copied, and those produced through some mating or crossover operator. It would appear that the new members produced through crossover should be immune from mutation since they have not been evaluated for performance yet. Therefore, we apply mutation only to those members directly copied from the previous generation. Finally, of the directly copied members, the best is immune to mutation as we wish to maintain the best solution found so far for future use by the GA in exploring the solution space through crossover. This form of elitism is not unknown in the field (Engelbrecht, 2007, page 139).

7.4.1 Phenotype Mutation for the TSP

With the TSP, all genes are used for each population member, with the exception of the final two nodes where the last connects back to the root, and the second to the last only has one node to choose from, so the heuristic need not be evaluated. However, all of the heuristics from the best 10% of the population members are used for calculating the new mutation selection distribution, with the heuristics for the best receiving double weight. This technique is only applied to the node values, not to the root node. The root node is generated from a uniform random distribution, and, if the root node is selected for mutation, a random value is selected, again, using a uniform distribution. Note, however, that the same value may be selected again.

For the TSP, since we generate the entire initial population to use heuristic 1, "select the closest available node," this will cause heuristic 1 to be overly weighted during initial generations. This is compensated for by not allowing the same heuristic to be selected as the replacement heuristic. When a gene is selected for mutation, if the replacement value is the same as the current, then the portion of the probability distribution for this heuristic is removed from the distribution by remapping the random number generated so as not to include the current heuristic. Thus, if heuristic 1 is the current value of a gene selected for mutation, the portion of the probability distribution assigned to this value is removed and redistributed to the other heuristics. In this way, a new value is guaranteed whenever a node is selected for mutation. This technique is not used with the mutation of the root node, however, as the root node starts off random.

7.4.2 Phenotype Mutation for the SITB

The SITB mutation operator is very similar to the TSP mutation operator with four exceptions:

1) Since not all of the genes are used to produce a snake, only those in use are included in the calculations for mutation probability selection.

2) Only the genes actively in use will be mutated. This is where the phenotype part of the name comes from in that the phenotype is used for gene selection during mutation.

3) Due to the high rate of epistatic interaction, only one gene per population member will be mutated at a time. While epistatic interaction is also a problem with the TSP, it is much worse for the SITB. Due to this fact, the mutation rate calculation is also a bit different. Typically the mutation rate, which is usually set quite low, is applied to each gene in a population member. However, since only one gene at most per population member will be mutated, we simply perform one check for mutation on the entire population member. For this, we must determine an equivalent mutation rate based on

the number of active genes (the number currently being used to produce the snake). This can be done by taking the mutation rate and multiplying it by the snake length. However, rather than performing this calculation for each population member, based on its length, we simply use the current best/longest snake and multiply the mutation rate by it. Thus, if the current rate is 95 parts per ten thousand (pp10k), the rate used for the 8D experiments, and the current longest snake is 90 nodes, then the chance that a single population member will be selected to have one random gene mutated will be $95 \times 90 = 8550$ pp10k.

4) While the current gene's value for the TSP will not be selected, this restriction has been removed from the SITB. Thus, when a new allele is selected for the gene, it may be the same as before.

Chapter 8

Experimental Approach and Philosophy

8.1 The General Approach

The background information for this research has now been completely explored. However, we have conducted no experiments in support of our claims. The following chapter gives a broad description of how we will conduct experiments in support of the four primary hypotheses. Next we give details of each specific experiment and the results obtained (Chapter 9). Then the results will be discussed (Chapter 10). Finally, we cover the conclusions that can be drawn from this work (Chapter 11).

Each hypothesis will be supported with a set of experiments using both the TSP and the SITB in order to show that the approach used is generalizable, as it was never the intent to focus on a specific problem, but on an improved technique for using GAs on a class of problems (graph problems).

The first hypothesis does not involve the GA but simply encoding schemes, so it will be different than the experiments conducted for H2 though H4 which all involve the GA. The GA experiments all involve performing 30 GA runs where, for each run, the best evaluation will be kept for comparison. Of these 30 runs (30 values for each experiment, one from each GA run), the best population evaluations will be used as such: the best and worst evaluations will be recorded along with the arithmetic mean and the standard deviation. Primarily, pairs of experiments will be conducted to investigate a hypothesis. Thus, focus will be on the arithmetic mean of the best values (rather than the best or worst) of the 30 runs for each of two experiments and statistical analysis will be performed on sets of runs for comparison. The question is whether these values will support each hypothesis. Statistical analysis will be used to show that the difference between the means of the two sets of 30 values are indeed statistically significant.

We performed Kolmogorov-Smirnov goodness-of-fit hypothesis tests using the kstest in MATLAB to compare against a normal distribution on each set of data. These tests showed that roughly half of the experiments were not normally distributed. Thus, the Student's *t*-test would be inappropriate as a statistical test for group comparisons. Therefore, the Wilcoxon Rank Sum test (the ranksum test in MATLAB, which is similar to the Mann-Whitney U test) was used to determine whether any apparent difference in the mean values from each pair of results is significant or not (are they from the same distribution or not).

8.2 The TSP Experimental Setup

For the TSP, problems from the TSPLIB95 (Reinelt) TSP library were used. This library contains subgroups of TSPs in many forms but only those that are 2D symmetric (referred to as EUC_2D in the documentation) will be used. By symmetric we mean that the distance from node A to B is the same as from node B to A. In this kind of TSP, all pairs of nodes have equivalent distance or cost values for either direction of travel. From this subgroup, a training set of 7 was selected for training and development, and a set of 7 for hypothesis testing experiments, each of which is similar in size to one in the training set (one is used for both: pr2392). Thus, other than the one exception, the test set is not used for any purpose except the hypothesis experiments. The training set is used for all code development, parameter tuning, and timing comparisons. Using the training set, once the parameters have been tuned, timing comparisons are performed in a controlled environment so that each experiment with results that are to be compared will have taken approximately the same amount of CPU time. We do not want to give any approach an advantage in more time or resources. Indeed, this work would be of little value if controlled timing comparisons were not performed, since it is not uncommon for a poorly designed GA to be able to eventually catch up to the performance of a better one given enough time.

For each of the TSPs, the number of nodes is listed as part of the problem name. Table 8.1 shows the training and development set on the left and the problems of similar size for hypotheses testing on the right. The best known path lengths (shortest known) have been taken as stated on the TSPLIB95 (Reinelt) web site. However, based on the documentation provided, it is unclear where these values actually originate. Also, some of the problems have the associated best paths available on the web site and some do not. Again, it is unclear where these paths came from. Problem pr2392, is listed as a training and testing problem, but is only used for H1 since it is a rather large problem with a listed best path solution, which is what is needed for H1. In addition to pr2392, only problems eil51, lin105, and a280 will be used for H1 as these are the only problems from the testing set that have best paths available from the TSPLIB95 (Reinelt) library.

8.3 The SITB Experimental Setup

For the SITB, there is only one problem for each hypercube dimension, so a separate training and testing set is not an option. Instead, the appropriate parameters for each set of experiments are determined through separate experiments, then, timing trials in a controlled environment are performed. For the actual experiments performed in support of the hypotheses, the time and resources allotted to each set is comparable. Also, dimension 10 is not used for H2 as the results from the D7, 8, and 9 hypercubes seem definitive. However, the average experimental values produced from the experiments for H3 and H4 are not as clear, and, thus, D10 has been added for additional support of the hypotheses statements. Table 8.2 shows the hypercube dimensions used with the total number of nodes and the current world record length snakes.

Table 8.1: TSPs used for training on left and similar sized problems for testing on right. The current world record shortest paths are listed under "Best known".

Training set	Best known	Testing set	Best known
berlin52	7542	eil51	426
eil101	629	lin105	14379
d198	15780	rat195	2323
lin318	42029	a280	2579
pcb442	50778	d493	35002
pr1002	259045	u1060	224094
pr2392	378032	pr2392	378032

Table 8.2: Hypercubes used for the SITB experiments, total number of nodes in each, and the current record length snakes

Hypercube Dimension	Number of Nodes	Longest Known Snake
7	128	51
8	256	99
9	512	191
10	1024	371

Chapter 9

Hypotheses Experiments and Results

In this chapter, the individual hypotheses are restated and the individual experiments designed to investigate the hypotheses are explained. Finally, the actual data from the experiments is presented along with the statistical test results. The discussion of the results and their meanings can be found in Chapter 10, Analysis.

9.1 H1 Experiments and Results

H1: Heuristic encoding schemes can effectively represent solutions to graph-space problems.

The process for these experiments involves using the best known path to guide the selection of heuristics to artificially construct a GA population member. This is done by going through the path and selecting a heuristic from the set that will select the next node in the path for which we are constructing a member. If none of the heuristics is capable of selecting the next node, then this path cannot be recreated with the given heuristic set. Once complete, the population member is evaluated by the evaluation function to produce a solution path, which is then compared to the best path on record. The two paths should be isomorphic, and for the problems analyzed (and others not listed), the process was successful.

9.1.1 TSP Results

The TSP set used for this hypothesis consists of four problems: Three from the hypotheses testing set and an additional larger problem of 2,392 nodes that will not be used for the other hypotheses. The paths for these problems are created using a set of 25 heuristics. The results are listed in order of size. Also, only the path and population member data for the smallest is listed here. The larger problems can be found in Appendix C.

9.1.1.1 eil51

The best known path for this problem is 426 units long. Following is the heuristic, locus-based population member that reproduces the best known path for this problem. The last value is the root node. Thus, there are 52 values in the population member. The first, third, and fifth lines represent index values for reference, and the data are below these:

```
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 7  7  9  1  1  1  1  1  1  1  1  1  1  7  1  1  1  1  1  5  5  9  1
28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 24 25 26 27 43 44 45 46
 4  1  1  8  1  1  1  1  1  1  1  1  1  1  1  1  1  1  4  1  7  1  1
47 48 49 50 51 root
 8  1  7  7  1  1
```

Below is the path created by the heuristic set using the locus-based evaluation function. This path is identical to the best known path for this problem:

```
1   22  8   26  31  28  3   36  35  20  2   29  21  16  50  34  30  9
49  10  39  33  45  15  44  42  40  19  41  13  25  14  24  43  7   23
48  6   27  51  46  12  47  18  4   17  37  5   38  11  32  1
```

Here, we see that the root node (the node the GA will always start from for this population member) is 1. From node 1, the second node is 22. Thus, the GA uses the heuristic at position 1 (heuristic 7 in this case) to make the decision to move to node 22. From node 22, the heuristic at this position, heuristic 9, will be used to select the next node, node 8. This process continues until the GA reaches node 11, the second to the last. From node 11, there is only one node left to select, node 32. Therefore, the heuristic at position 11, heuristic 1, is not used. Finally, the path is closed by connecting node 32 to node 1.

9.1.1.2 Additional Three TSP Instances

This technique is able to recreate the best known (as listed in Reinelt's TSPLIB95 web site) path for all three of the following problems: lin105, a280, and pr2392. The best path and the population member which created it for each, are in Appendix C.

9.1.2 SITB Results

The same approach is taken for this problem as for the TSP. The best known paths are created for dimension 7, 8, 9, and 10 hypercubes using a heuristic set of 17. In each case, the best path is converted to linked list format, then to canonical form. Finally, a list of heuristics is found that will reproduce it. This technique was successful for all four problems as shown below.

9.1.2.1 7D Hypercube

The best known snake in the 7D hypercube is 51 nodes in length (Kochut, 1996). Following is the locus-based heuristic list that will produce this snake. The active genes, the genes used to produce the path, are in bold and underlined. Also, if the nodes selected by a set of heuristics are contiguous in the member, then they are underlined together. Again, the odd lines show the index values for reference:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	**1**	**1**	14	**1**	**1**	20	8	**1**	**1**	2	**1**	20	4	10	14	20
18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
2	**1**	**1**	7	16	**2**	24	6	2	21	19	**1**	14	12	20	20	**1**
35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51
1	**1**	21	16	**3**	16	**4**	12	**26**	1	16	3	21	**1**	**1**	**1**	**1**
52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68
7	26	**2**	14	3	**1**	**1**	3	14	26	26	26	1	16	2	**1**	**1**
69	70	71	72	73	74	75	76	77	78	78	80	81	82	83	84	85
16	**1**	6	**3**	7	21	13	26	**1**	7	10	20	14	**1**	**1**	**1**	16
86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101	102
7	**1**	**1**	6	6	**1**	**1**	8	3	10	2	**1**	**1**	16	16	**1**	7
103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119
3	3	21	6	**1**	**1**	10	10	3	**2**	**1**	**1**	**1**	10	14	**1**	16
120	121	122	123	124	125	126	127	128								
14	**1**	**1**	7	10	7	1	26	4								

The world record snake of 51 nodes that this heuristic list produces follows:

```
1     2     3     6     5     12    9     10    23    18    19    20    29    36
35    34    39    58    57    72    121   122   103   98    97    112   113   114
115   118   107   108   101   92    91    70    67    68    77    84    83    82
87    88    41    48    49    50    51    54    43
```

With the SITB, not all nodes are used to produce a snake, since there are constraints on node inclusion. Therefore, the population members must contain a value for each node and will be longer than the path produced. In this example, hypercube node 1, which is always the root node for the SITB problems, has a value of 1. Therefore, heuristic 1 will be used to select the next snake node, which is hypercube node 2. Hypercube node 2's heuristic is 1 also, which will select hypercube node 3. Hypercube node 3's heuristic is

1 which will select hypercube node 6, and hypercube node 6’s heuristic is 1, which will select hypercube node 5. This continues until, at the end of the snake, hypercube node 54, the second to the last node in the path, has a heuristic value of 2, which will select the last node, hypercube node 43.

9.1.2.2 Additional Three Hypercube Results

The best known snake in an 8D hypercube is 99 nodes in length and was found using this technique (Carlson and Hougen, 2010). The world record length snake for the 9D hypercube is 191 nodes in length (Wynn, 2012). The world record snake in 10D is 371 nodes in length (Kinny, 2012). The population members that will produce these snakes and the snakes themselves can be found in Appendix C.

9.2 H2 Experiments and Results

H2: Heuristic encoding used with traditional GA operators and parameter settings performs no better, and may perform worse, with more traditional encoding schemes using the same GA operators and parameters.

Following are experiments that compare the results of 30 GA runs between the more traditional encoding scheme and the heuristic encoding, both using the same GA operators and parameter values (minor variations may be necessary and will be explained as needed). Also, a complete list of the parameters used can be found in Appendix B. For all H2 experiments, linear gene ordering is used, which is more common than locus based ordering (see Chapter 6 for a description).

9.2.1 TSP Results

For the TSP crossover operator, the Partially-Mapped Crossover (PMX) from Goldberg and Lingle (1985) is used as the traditional crossover operator for node encoding. For heuristic encoding, typical linear multi-point crossover, using 2 crossover points, is used (see Section 3.3.5 for a description).

NOTE: Several papers refer to this as the Partially-Matched Crossover (Goldberg, 1989, Jung and Moon, 2002). However, the original paper uses the term Mapped.

Node swapping, where two nodes are selected at random and their values are swapped, is used as the traditional mutation operator (Louis and Li, 1997) for node encoding. Typical uniform random mutation is used as the traditional mutation operator for heuristic encoding (see Section 3.3.6 for a description).

Table 9.1 shows the six test problems with the best (shortest) average and worst path length from 30 separate GA runs using both the node and heuristic encoding and standard GA operators. In this and all subsequent tables, “mean” refers to the arithmetic mean of the set of values. Also, S. D. is the standard deviation and is shown in the column next to the mean. Table 9.2 shows the Wilcoxon rank sum test probability values (p-values). These values indicate the probability of observing the two sample sets assuming the distributions the sets were drawn from are equivalent.

Table 9.1: TSP results comparing node and heuristic encoding for Hypothesis H2. The values under "Best" that are gray show the current world record path lengths. Each row labeled "Nodes" or "Heuristics" represents a set of 30 GA runs using the Canonical GA operators and typical parameter settings with the stated encoding scheme.

Hypothesis	Two	TSP	Results	
	Best	Mean	S. D.	Worst
eil51	426			
Nodes	437	488	25.1	535
Heuristics	428	430	1.7	435
lin105	14379			
Nodes	19301	24249	2268	27821
Heuristics	14514	14758	196.2	15184
rat195	2323			
Nodes	3871	4524	310.5	5253
Heuristics	2366	2412	26.8	2471
a280	2579			
Nodes	5583	6387	455.4	7630
Heuristics	2703	2751	26.1	2808
d493	35002			
Nodes	81749	90208	3829.5	96523
Heuristics	37753	38136	210.3	38591
u1060	224094			
Nodes	1130880	1227520	48151	1318680
Heuristics	251073	256512	2146	261208

Table 9.2: TSP Wilcoxon Rank Sum p-values when node encoding final values are compared to heuristic encoding using 30 GA runs for each. As can be seen from the p-values, each compared set has a p-value of less than 5%.

Hypothesis	Two	Wilcoxon	Rank	Sum	Results
eil51	lin105	rat195	a280	d493	u1060
2.5142E-11	3.0161E-11	3.0161E-11	3.0142E-11	3.0199E-11	3.0199E-11

9.2.2 SITP Results

For the SITB, heuristic encoding is compared with two alternative, more direct schemes. In all experiments, linear based gene ordering is used. Also, linear multi-point crossover with two crossover points is used. Finally, typical uniform mutation is used.

The first alternative encoding scheme is simple *link based encoding*, where each gene is assigned a value from 1 to the dimension size, which represents which link from the link table to take from the current head node when attempting to add a node (see Table 5.1 for an example of link tables). Using this technique, there is no search at all and if the node the given link leads to is not available, then this snake is done and the current length will be the value given the population member. This scheme is abbreviated L for Link.

The second encoding scheme involves the same technique but with minor search added. In this technique, if the node selected is not available, then the immediate neighbors of the given link in the link table are attempted. Each row of the link table is treated as a circular list. Thus, if the link taken is the far right entry, then the first entry of that row will be treated as one of its neighbors, and vice versa. If the primary link is not available but either of its neighbors is, then this node is added and we continue building the snake for this population member. This scheme is abbreviated LS or Link+, for Link Search.

For both L and LS, the initial population generation consists of randomly generating a number from 1 to the dimension size for each gene, using a uniform distribution. The results of 30 runs using each of these encoding schemes and the heuristic scheme, all using traditional GA operators and typical parameter values is shown in Table 9.3. For this comparison, D10 was left out as it was clear from D7 to D9 that there is a definite difference in performance between the encoding schemes. Table 9.4 contains the Wilcoxon rank sum p-values comparing both the simple link encoding to the heuristic encoding, and the link-search encoding to the heuristic encoding.

Table 9.3: SITB results comparing simple link encoding, links with basic search (Links+), and heuristic encoding for Hypothesis H2. World record best values are highlighted in gray. Each row represents values from a set of 30 GA runs using identical parameters for all 30. Also, the GA operators and parameter values are identical between all experiments.

Hypothesis	Two	SITB	Results	
	Best	Mean	S.D.	Worst
7D	51			
Links Only	41	37.13	2.1	32
Links+	49	47.80	0.7	47
Heuristics	51	49.67	1.2	48
8D	99			
Links Only	73	64.77	4	58
Links+	90	86.53	1.5	83
Heuristics	93	90.00	1.6	87
9D	191			
Links Only	127	112.53	5.8	100
Links+	158	151.77	3.1	145
Heuristics	171	161.43	3.2	156

Table 9.4: SITB Wilcoxon Rank Sum p-values when link encoding (L) and Links with basic search (LS) final values are compared to heuristic encoding (H) using 30 GA runs for each. As can be seen from the p-values, each compared set has a p-value of less than 5%.

Hypothesis	Two	Rank	Sum	Tests	
D7 L to H	D7 LS to H	D8 L to H	D8 LS to H	D9 L to H	D9 LS to H
2.0816E-11	1.7865E-08	2.5922E-11	2.3915E-09	2.8163E-11	5.2474E-11

9.3 H3 Experiments and Results

H3: When using heuristic encoding, phenotype operators improve average GA performance over that obtained with traditional GA operators.

In this set of experiments for the three sub-hypotheses, we compare the results of the GA using heuristic encoding and the traditional GA operators with the GA using the new operators of phenotype crossover and mutation developed to work with heuristic encoding. The results from the H2 tests of the heuristic encodings will be used for some of the comparisons. In all of the new experiments (those whose results were not copied from the H2 experiments), locus based gene ordering is used. Also, linear multi-point crossover with two crossover points is used as the traditional crossover operator. Finally, uniform mutation is used as the traditional mutation operator.

The first step in evaluating the performance of the GA with the new operators is tuning the various GA parameters to work best with the new operators. For parameter tuning, we use the training and development set for the TSP and the 4 dimensions being evaluated for the SITB. Many parameters are changed and a complete list of the parameters and their values can be found in Appendix B. After determining the best parameter settings, timing trials are performed using these new settings with the new GA operators (phenotype crossover and mutation) to determine the size of the population and the number of generations to run for equivalent timing.

Finally, a new set of 30 GA runs are performed for each of the TSP test problems and the four SITB dimensions using the new parameter settings but the typical/traditional GA operators. This is done to show that the improvements when the new operators are used are a result of the new operators, and not the change in parameter values. Due to this, the focus is on comparing between using the new GA operators and the new parameters with the traditional GA operators and the new parameters.

Tables 9.5 and 9.7 list the experimental results showing the best, mean, standard deviation and worst of the 30 GA runs for each experiment. In these tables, “Base” refers to the traditional GA operators using the new parameters, which is what the other three experiments for each problem will be compared against. “Crossover” is the experiments performed with the new parameters, the traditional mutation operator, and the phenotype crossover operator. “Mutation” uses the new parameters, the traditional crossover operator, and the phenotype mutation operator. “Both” uses the new parameters and both the phenotype crossover and mutation operators.

Tables 9.6 and 9.8 show the results of the Wilcoxon rank sum tests. Here, as stated above, the interest is in comparing the “Base” experiments against substituting the traditional operators with the new operators individually and together. Therefore, in these tables, the last three rows contain the p-values for these comparisons. Also, in these tables, the first four rows compare against the traditional operators and the typical parameters, which is abbreviated as “totp.”

Table 9.5: TSP results comparing typical GA operators of mutation and crossover with phenotype crossover, phenotype mutation, and both. Rows labeled "Base" use typical operators with new parameters. Rows labeled "Crossover" use the phenotype crossover operator and the typical mutation operator. Rows labeled "Mutation" used the phenotype mutation operator and the linear, 2-point crossover operator. Underlined values had means not as expected.

Hypothesis	Three	TSP	Results	
	Best	Mean	S. D.	Worst
eil51	426			
Base	427	430.70	3.70	439
Crossover	427	429.93	2.40	438
Mutation	427	<u>431.03</u>	3.82	439
Both	427	<u>430.60</u>	3.55	439
lin105	14379			
Base	14442	14645.10	136.10	15029
Crossover	14430	14607.20	121.50	14946
Mutation	14416	14571.00	119.60	14815
Both	14412	14522.30	97.30	14788
rat195	2323			
Base	2344	2379.17	21.50	2433
Crossover	2364	<u>2404.27</u>	31.00	2466
Mutation	2348	2375.17	18.10	2416
Both	2338	2370.70	18.90	2427
a280	2579			
Base	2630	2671.90	21.70	2715
Crossover	2629	<u>2677.07</u>	30.80	2729
Mutation	2626	2648.90	14.80	2673
Both	2623	2646.80	15.60	2679
d493	35002			
Base	36157	36842.40	308.00	37697
Crossover	36276	<u>36933.40</u>	327.00	37459
Mutation	35852	36413.00	298.60	37222
Both	35909	36409.00	196.80	36739
u1060	224094			
Base	237634	243013.00	2340.30	248726
Crossover	239023	241732.00	1727.60	246087
Mutation	235067	239444.00	2079.00	244878
Both	234984	238816.00	1939.20	242315

Table 9.6: TSP Wilcoxon Rank Sum p-values from comparing heuristic encoding with traditional operators and typical parameters (totp), traditional operators and new parameters (Base), and combinations of the new operators with the old operators. Only sets labeled totp use the typical parameter values. Bold values are not statistically significant at 5%. Underlined values had means not as expected.

Bold values not	Hypothesis Three Rank Sum Tests					
significant at 5%	eil51	lin105	rat195	a280	d493	u1060
totp vs Base	**7.57E-01**	**5.19E-02**	6.26E-06	6.01E-11	3.02E-11	3.02E-11
totp vs Crossover	**5.97E-01**	6.50E-03	**1.52E-01**	3.62E-10	3.02E-11	3.02E-11
totp vs Mutation	**7.17E-01**	1.17E-04	3.63E-07	3.00E-11	3.02E-11	3.02E-11
totp vs Both	**5.76E-01**	6.80E-07	6.22E-08	3.00E-11	3.01E-11	3.02E-11
Base vs Crossover	**9.15E-01**	**2.46E-01**	7.67E-04	**6.31E-01**	1.41E-01	1.70E-02
Base vs Mutation	**4.06E-01**	2.70E-02	**5.10E-01**	4.45E-05	1.93E-06	7.60E-07
Base vs Both	**8.04E-01**	1.05E-04	**1.17E-01**	1.20E-05	1.36E-07	2.83E-08
Both vs Crossover	**9.58E-01**	4.80E-03	5.61E-06	9.45E-05	1.87E-07	7.60E-07
Both vs Mutation	**3.06E-01**	**1.24E-01**	**3.07E-01**	**6.26E-01**	**7.62E-01**	**3.79E-01**

The first row in Table 9.6 and the first row in Table 9.8 each compares the typical parameter values with the new values, but both use the traditional GA operators.

In all of the tables, if an average value is not what was expected, then it is underlined. A discussion of these values and their implications can be found in Chapter 10, Analysis. In the tables with p-values, if two groups are not statistically significantly different at the 5% level, then they appear bold.

NOTE: As D10 was not used for H2, there are no typical parameters using traditional operators (totp) experimental values to compare against. Therefore, the totp rows for D10 in Table 9.8 contain NA.

9.3.1 H3-1 Experiments and Results

H3-1: When using heuristic encoding, the phenotype crossover operator increases average GA performance above that obtained with the canonical, linear multi-point crossover operator.

This is shown by comparing the Base case explained above with experiments using all the same parameters but with the traditional crossover replaced with phenotype crossover. The traditional crossover operator being used for both the TSP and the SITB is the simple multi-point linear crossover operator. In all cases, two point crossover is used. Tables 9.5 and 9.7 show the results of the experiments with the relevant comparisons between the Base and Crossover rows of data. Tables 9.6 and 9.8 contain the Wilcoxon Rank Sum p-values with relevant rows labeled "Base vs Crossover."

9.3.2 H3-2 Experiments and Results

H3-2: When using heuristic encoding, the phenotype mutation operator increases average GA performance above that obtained with the canonical mutation operator.

As with H3-1, here we compare the Base results with the results replacing the typical GA mutation operator for the TSP and SITB with the phenotype mutation operator. The relevant rows in Tables 9.5 and 9.7 are the Base and Mutation rows. The Wilcoxon Rank Sum p-values are in Tables 9.6 and 9.8 in the "Base vs Mutation" rows.

Table 9.7: SITB experiments of 30 GA runs each showing the performance using typical GA operators (Base), phenotype crossover with typical mutation (Crossover), linear crossover with phenotype mutation (Mutation), and phenotype crossover with phenotype mutation (Both). Underlined values had means not as expected.

Hypothesis	Three	SITB	Results	
	Best	Mean	S.D.	Worst
7D	51			
Base	51	50.13	1.10	48
Crossover	51	50.33	0.96	49
Mutation	51	50.37	1.00	48
Both	51	50.67	0.76	49
8D	99			
Base	95	92.27	1.26	89
Crossover	95	92.83	2.49	89
Mutation	95	<u>92.27</u>	1.20	90
Both	96	93.83	1.27	92
9D	191			
Base	172	165.70	3.31	158
Crossover	176	170.00	3.61	159
Mutation	174	167.20	3.37	161
Both	177	171.27	2.45	167
10D	371			
Base	315	297.23	7.47	282
Crossover	321	314.10	3.55	307
Mutation	312	298.13	4.75	290
Both	325	315.23	4.22	308

9.3.3 H3-3 Experiments and Results

H3-3: When using heuristic encoding, the combination of phenotype crossover and phenotype mutation operators increases average GA performance above that obtained using any other combination of canonical GA operators for crossover and mutation.

Here, as with H3-1 and H3-2, we are interested in comparing performance using the new parameters and traditional GA operators with the new parameters and both new GA operators. In addition, we show that using both of the new operators together performs better than using only one (i.e., using one traditional operator and one new). With this in mind, rows in Tables 9.6 and 9.8 with both of the new operators are labeled "Both," and we compare primarily the average value with the average from all three other rows for each experiment (Base, Crossover, and Mutation rows). If the hypothesis has merit, the average for "Both" should be better than any of the other experiments in these tables. Also, Tables 9.7 and 9.8 have the Wilcoxon Rank Sum p-values for statistical significance in the rows labeled "Base vs Both," "Both vs Crossover," and "Both vs Mutation."

Table 9.8: SITB Wilcoxon Rank Sum p-values from comparing heuristic encoding with traditional operators and typical parameters (totp), traditional operators and new parameters (Base), and combinations of the new operators with the old operators. Only sets labeled totp use the typical parameter values. Bold values are not statistically significant at 5%. Underlined values had means not as expected.

Bold values are not significant at 5%	Hypothesis Three Rank Sum Tests			
	D7	D8	D9	D10
totp vs Base	**1.06E-01**	7.66E-07	1.20E-05	NA
totp vs Crossover	1.82E-02	5.83E-05	4.13E-09	NA
totp vs Mutation	1.38E-02	8.25E-07	1.19E-07	NA
totp vs Both	2.87E-04	1.28E-10	1.29E-10	NA
Base vs Crossover	**4.88E-01**	**1.20E-01**	9.87E-06	3.58E-10
Base vs Mutation	**4.01E-01**	**<u>8.31E-01</u>**	**1.69E-01**	**5.69E-01**
Base vs Both	3.92E-02	4.34E-05	4.51E-08	1.94E-10
Both vs Crossover	**1.42E-01**	**2.73E-01**	**2.48E-01**	**4.44E-01**
Both vs Mutation	**2.12E-01**	3.37E-05	9.41E-06	4.66E-11

9.4 H4 Experiments and Results

The design and selection of the heuristic set should incorporate knowledge of the problem space appropriate for mapping to the solution space but should not include useless heuristics. If key heuristics are missing, the GA is unable to find promising areas of the solution space. Also, if useless heuristics are included, they detract from the guiding effects of the GA.

9.4.1 H4-1 Experiments and Results

H4-1: When key heuristics are removed, both upper end, and average GA performance decreases.

For the TSP, heuristics 4, 5, 6, 7, and 11 (see Appendix A.1 for a description of these) are removed from the standard set of 25 (see Section 2.4 for a discussion of how this subset was selected). With this subset of 20, 30 GA runs on each problem are performed and compared with the base set of heuristics using the results labeled "Both" from the H3 experiments. All other parameters are the same. The results are in Table 9.9 in rows labeled “Base Set” and “Minus.” The Wilcoxon Rank Sum probability test results are in Table 9.10.

For the SITB, the heuristics removed are 4, 6, 20, 24, and 26 (see Appendix A.2 for a description) from the base set of 17 for a subset of 12 (see Section 2.4 for a discussion of how this subset was selected). Again, 30 runs for each of the problems are performed and compared with the base set of heuristics using the results labeled “Both” from the

H3 experiments. The results are in Table 9.11 in rows labeled “Base set” and “Minus.” The Wilcoxon Rank Sum probability test results are in Table 9.12.

9.4.2 H4-2 Experiments and Results

H4-2: When additional, unneeded heuristics are added, average GA performance decreases.

This is tested in a similar way for both the TSP and the SITB. For both, copies of a heuristic that had previously been removed from the set as not very helpful, are added. Refer to Section 2.4 for a discussion of how this heuristic was selected. For the TSP, the added heuristic instructs the GA to add the furthest node from the current node. While this heuristic may occasionally be a good idea, it has been determined through experimentation that this heuristic is normally a bad idea and should generally not be used. Thus, for the TSP, 8 copies of this heuristic are added to the base set of 25 for a total of 33 heuristics. 30 GA runs are conducted for each TSP test problem using this new set of 33, and results compared to performance using the base set of 25 heuristics labeled “Both” from the H3 experiments. All other parameters and GA operators are identical. Results are in Table 9.9 in rows labeled “Base Set” and “Plus,” with the Wilcoxon Rank Sum statistical data in Table 9.10.

For the SITB, a similar scheme is followed. The noise heuristic will add the node with the largest number of non-dead end neighbors. (If one potential candidate has two neighbors that are not dead ends, and another has three, add the one with three.) Here *non-dead end* means that, after adding the stated node, there is at least one additional node that can be added). To the base set of 17 heuristics are added 8 copies of this noise heuristic. As with the TSP, 30 GA runs are performed and compared to the base set of 17 heuristics labeled “Both” from the H3 experiments. All other parameters and GA operators are identical. Results are in Table 9.11 in rows labeled “Base Set” and “Plus.” Table 9.12 contains the Wilcoxon Rank Sum statistical comparisons.

Table 9.9: TSP experiments of 30 GA runs each showing the performance using the base set of heuristics found to work best, with the removal of 5 useful heuristics (Minus), and the addition of 8 heuristics of little value (Plus). While all averages were ranked as expected, not all differences are statistically significant.

Hypothesis	Four	TSP	Results	
	Best	Mean	S. D.	Worst
eil51	426			
Base Set	427	430.60	3.55	439
Minus	429	433.20	3.96	439
Plus	427	431.63	4.21	439
lin105	14379			
Base Set	14412	14522.30	97.30	14788
Minus	14499	14639.90	113.72	14923
Plus	14416	14530.50	74.33	14737
rat195	2323			
Base Set	2338	2370.70	18.90	2427
Minus	2360	2377.97	11.25	2412
Plus	2339	2371.83	15.41	2412
a280	2579			
Base Set	2623	2646.80	15.60	2679
Minus	2624	2656.03	16.03	2691
Plus	2599	2648.47	21.05	2711
d493	35002			
Base Set	35909	36409.00	196.80	36739
Minus	36188	36650.60	202.29	37044
Plus	35713	36456.70	269.00	37018
u1060	224094			
Base Set	234984	238816.00	1939.20	242315
Minus	236054	240077.00	1885.39	243872
Plus	235100	240017.00	2427.33	245089

Table 9.10: TSP Wilcoxon Rank Sum p-values from comparing base set of heuristics with the removal of 5 useful heuristics (Minus), and the addition of 8 heuristics of little value (Plus). Bold values not statistically significant at 5%.

Bold values not significant at 5%	Hypothesis	Four	Rank	Sum	Tests	
	eil51	lin105	rat195	a280	d493	u1060
Base set vs Minus	7.9403E-04	3.2826E-05	2.5000E-02	3.3200E-02	5.2587E-05	1.5600E-02
Base set vs Plus	**4.1330E-01**	**3.5830E-01**	**5.8410E-01**	**7.9570E-01**	**3.3650E-01**	3.3900E-02

Table 9.11: SITB experiments of 30 GA runs each showing the performance using the base set of heuristics found to work best, with the removal of 5 useful heuristics (Minus), and the addition of 8 heuristics of little value (Plus). While all averages were ranked as expected, not all differences are statistically significant.

Hypothesis	Four	SITB	Results	
	Best	Mean	S.D.	Worst
7D	51			
Base Set	51	50.67	0.76	49
Minus	51	49.93	1.05	48
Plus	51	49.47	0.97	48
8D	99			
Base Set	96	93.83	1.27	92
Minus	94	92.63	0.67	91
Plus	95	92.67	0.84	90
9D	191			
Base Set	177	171.27	2.45	167
Minus	169	165.00	2.27	159
Plus	177	170.13	2.61	165
10D	371			
Base Set	325	315.23	4.22	308
Minus	308	298.47	4.97	289
Plus	324	315.00	4.17	308

Table 9.12: SITB Wilcoxon Rank Sum p-values from comparing base set of heuristics with the removal of 5 useful heuristics (Minus), and the addition of 8 heuristics of little value (Plus). Bold values not statistically significant at 5%.

Bold Values are not significant at 5%	Hypothesis Four Rank Sum Tests			
	D7	D8	D9	D10
Base set vs Minus	3.5000E-03	2.1322E-04	2.8826E-10	2.9784E-11
Base set vs Plus	1.0996E-05	5.6556E-04	**6.1500E-02**	**8.1790E-01**

Chapter 10

Analysis of Experimental Results

We have presented much experimental data which will now be analyzed and explored. In this chapter, we attempt to understand the results and their implications to the hypotheses and this work in general.

As part of the analysis for H2 through H4, the Wilcoxon Rank Sum test for comparison is performed using the MATLAB ranksum function between relevant sets of data. This function performs a two sided test on two sets to calculate the probability that they came from the same distribution. This function indicates at the 5% confidence level, if two sets of 30 experimental values are indeed drawn from populations with different distributions. For all experiments the p-values from these tests are included in the tables. For these tables, if the compared sets have a value greater than 5%, its p-value is bold, indicating a greater than 5% chance the sets are from the same distribution. Also, for experiments where the mean value of the 30 GA runs was not as expected, this mean value in the data table and its associated p-value are underlined.

10.1 H1 Experimental Analysis

H1: Heuristic encoding schemes can effectively represent solutions to graph-space problems.

For the four representative TSPs used, it has been shown that the heuristic set of only 25 can reproduce the world record path. Indeed, we have been able to successfully reproduce the best path for all of the training set problems for which the path is available as well. While this does not prove that this heuristic set is sufficient to reproduce the best path in any TSP, it shows the efficacy of this set in covering the solution space.

For the SITB, as with the TSP, we find that a series of heuristics from the set of 17 is capable of reproducing the world record snakes in D7, 8, 9, and 10. These findings adequately support the hypothesis that this is a good encoding technique for a GA.

While showing that a specific point in a large search space can be represented using a given scheme is good, it does not necessarily mean that the path in the search space to these points can be found by a GA. To show this, we must also show that, in general, and under the right conditions, a GA, using this encoding scheme can, on average, achieve reasonably good results. This is the focus of the remaining hypotheses: showing that, when combined with appropriate GA operators, good results are produced.

10.2 H2 Experimental Analysis

H2: Heuristic encoding used with traditional GA operators and parameter settings performs no better, and may perform worse, with more traditional encoding schemes using the same GA operators and parameters.

For both the TSP and the SITB, the experiments did not support the hypothesis. One common reason for this is that the heuristics encode knowledge of the problem space directly into the population members. This allows the GA to apply this knowledge during population manipulation instead of the more random nature of traditional encoding schemes where the GA must evolve the population to discover useful knowledge of the problem space. With these results, the addition of customized GA operators for the heuristic sets may not provide as much improvement as initially expected, although improvements should still be seen. Also, these findings imply there may be less degradation in performance when good heuristics are removed, or when noise heuristics are added.

10.2.1 TSP Analysis

The experimental data for this hypothesis and the TSP are in Table 9.1 with the statistical tests in Table 9.2. It was observed on the large problems (d493 and u1060 and the training problems) using node encoding that the GA continued to evolve better solutions no matter the number of generations it was allowed to run. The reason for this, and the poor performance overall, is likely the random nature of the initial population. The nodes are arranged in a completely random manner which leaves great room for improvement, and the GA will continue to improve for a much longer time than is typical when using GAs. However, the heuristic encoding scheme starts the population off at a much better state where less improvement can be made. Thus, the population is able to converge much more quickly to reasonably good solutions. Also, the fact that there are far fewer heuristics than there are nodes allows the heuristic encoding scheme to prune the search space. Thus, the GA has a much smaller area to explore when using heuristic encoding for this problem. Note that this statement does not apply to the SITB where the number of heuristics is larger than the number of links that can be taken.

For the TSP results, it seems obvious from the data in Table 9.1 that the values for the node and the heuristic encoding schemes are drawn from different distributions (their means are statistically different). Indeed, the Wilcoxon Rank Sum test for comparison also indicates that all of the TSP comparisons result from different distributions, at the 5% confidence level. This indicates that the differences seen in the averages of the 30 tests are indeed statistically significant.

10.2.2 SITB Analysis

The experimental data for this hypothesis and the SITB are in Table 9.3 with the statistical tests in Table 9.4. As with the TSP, results from Table 9.3 clearly show the difference in performance using the heuristic encoding scheme when compared to more traditional encoding schemes, even when traditional GA operators and parameters are used. Unlike the TSP, this cannot be explained with the argument of search space pruning in that we have more heuristics than links. As an example, in an 8 dimensional hypercube, each node links to 8 other nodes. When building a snake, this allows for only 7 choices at most (one of the links leads to the previous node in the path and, thus, cannot be used). However, with a heuristic set of 17, there are 17 different possible values or choices compared to only 7 using link encoding. Thus, we believe the reasons for the better performance using heuristics come primarily from the built-in knowledge of the problem space, and from the fact that the heuristics will only select a dead-end node as a last resort. Finally, the Wilcoxon Rank Sum tests support the apparent improvement of heuristic encoding over both of the link encoding variants. As with the TSP, the Wilcoxon Rank Sum tests show that each pair is drawn from a different distribution, at the 5% confidence level. As expected, the LS (Link Search) encoding does perform better than the simple link encoding. However LS is still markedly worse than heuristic encoding, again, not supporting the hypothesis.

10.2.3 Final Comments

Ultimately, the experiments showed this hypothesis incorrect! Indeed, the heuristic encoding in all cases outperformed the traditional encoding schemes even when traditional GA operators and parameters were used. The difference in performance becomes worse as the problem size grows.

10.3 H3 Experimental Analysis

H3: When using heuristic encoding, phenotype operators improve average GA performance over that obtained with traditional GA operators.

The first row in Tables 9.6 and 9.8 compares the typical parameter values with the new values, but both use the traditional GA operators. In all cases for the TSP and the SITB, except eil51, the new parameters perform better. The next three rows compare using the typical parameter values and the new values with the given new GA operators. These first four rows are mostly for completeness and it is the last five rows that are of most interest and are discussed below.

H3-1: When using heuristic encoding, the phenotype crossover operator increases average GA performance above that obtained with the canonical, linear multi-point crossover operator.

H3-2: When using heuristic encoding, the phenotype mutation operator increases average GA performance above that obtained with the canonical mutation operator.

H3-3: When using heuristic encoding, the combination of phenotype crossover and phenotype mutation operators increases average GA performance above that obtained using any other combination of canonical GA operators for crossover and mutation.

For all three sub-hypotheses, results were mostly as expected, with some interesting exceptions which are reviewed in the separate TSP and SITB analysis sections. As discussed in 10.2, the heuristic encoding, even with traditional GA operators, performed better than expected. This causes the addition of custom designed operators to not improve performance as much as expected, but still to some extent. In Tables 9.5 and 9.7, we start by performing a set of experiments using the traditional operators, but with the parameters tuned for the new operators. These experiments are labeled “Base” in the tables to show that they are the baseline for the other experiments to be compared against. Also, while we are not interested in comparing performance between the phenotype mutation operator and the phenotype crossover operator, it is important to show that the combination of these two operators works better than either alone. Thus, these mean value comparisons will be discussed and have the "Both" mean value underlined if this assumption is not true. As with H2, all relevant experiments are compared using the Wilcoxon rank sum tests in Tables 9.6 and 9.8. These tables also compare between both new operators and each individually.

10.3.1 TSP Analysis

While the intent was to find a good set of TSP test problems of various sizes starting under 100 nodes, this caused problems when using the smallest problem, due to ceiling effects. Results for the eil51 problem are so similar as to be statistically insignificant, and the Wilcoxon results bear this out as seen in the first column of Table 9.6.

Another interesting finding is it appears from the tests that phenotype crossover, on its own (i.e., in combination with traditional mutation) actually detracts from performance over linear crossover, at least for all but the largest problem of 1,060 nodes. Indeed, even for lin105, where phenotype crossover performed marginally better than Base on average, it fails the Wilcoxon test for statistical significance. We are not sure why the phenotype crossover only shows improvement when combined with the phenotype mutation operator, but not with traditional mutation. However, this may be caused by the initial gene values all being set to one. This initial uniform population may affect the performance of the more complicated phenotype crossover operator more than linear multi-point crossover. It is interesting to note, however, that on the largest, most difficult problem of 1,060 nodes, phenotype crossover does show statistically significant improvement over linear multi-point crossover.

The phenotype mutation operator, however, seems to perform statistically significantly better than the traditional mutation, with the exception of rat195, where its mean was still higher, but not statistically significantly so.

Finally, the most interesting result from these experiments is, with the exception of eil51 which can be discounted due to ceiling effects, the combination of the two new operators always outperforms all other combinations of old and new operators, even when individually, performance may drop with only one of them (primarily the phenotype crossover as discussed above). We believe this is due to the fact that the phenotype mutation operator redistributes the allelic selection probabilities such that the values contributing more to shorter paths quickly get incorporated into the population, which allows the phenotype crossover operator to be effective quicker than with a uniform random mutation operator. Therefore, when used together, the average performance increases over either phenotype operator alone.

Regardless of the actual differences in mean values, for all comparisons with the Wilcoxon Rank Sum test, the combination of both new operators and only the new mutation operator is not statistically significant. This lessens the value of the previous paragraph, even though the average for the combination tests is better for every problem except eil51. Also, the best/shortest path found is better for the combination in all cases except d493, again indicating the combination may be better overall. Since the tests for statistical significance do not have low enough p-values to show at a high enough confidence level that the combination does perform better than phenotype mutation alone, no definite conclusions can be drawn.

10.3.2 SITB Analysis

These experiments were conducted very similarly to the TSP, as we expect to find improvements with the new operators individually, and again, greater improvement when they are combined. Results of experiments in Table 9.7 show that this is largely the case. However, oddly enough we get similar results from the TSP except that the roll of the phenotype mutation and crossover operators has reversed. Table 9.7 shows that the phenotype mutation operator does not increase average path lengths by much, and Table 9.8 shows the p-values larger than 5% (the smallest is 17%) for all four hypercube dimensions, implying that the new mutation operator did not actually change the average results by a significant amount.

Even more interesting, however, is the fact that, when combined with the phenotype crossover operator, which generally does improve performance, it does seem to improve the mean, but not in a statistically significant manner, as seen from the "Both vs Crossover" rows in Table 9.8. As stated above, this is very similar to results from the TSP experiments except that there, phenotype crossover was the weaker by itself.

While phenotype crossover seems to improve mean performance in all cases, it fails to show this in D7 and D8 with the Wilcoxon Rank Sum tests as seen in Table 9.8. As the problem gets larger, the value of phenotype crossover should, and does increase as seen in the p-values for D9 and D10. However, in all cases, the combination of both phenotype operators improves performance over the base case in a statistically significant way, as seen from the row labeled “Base vs Both” in Table 9.8. As with the TSP, there seems to be some dynamic interaction involved in using both of the phenotype operators that is able to take advantage of the heuristic encoding, and which is not seen in either operator individually. There are many potential causes, but parameter tuning may be a partial cause.

The last two rows of Table 9.8, show that the mutation operator seems to weaken as the dimension grows while the crossover operator strengthens. This is shown by the p-values growing smaller for the “Both vs Mutation” row and larger for the “Both vs Crossover” row, implying the distributions are growing further apart for the former, but closer together for the latter. This may indicate that, as the problem space grows, phenotype crossover tends to dominate the evolution of the population toward better solutions, while the significance of mutation decreases. However, the best solution found (longest snake, or the snake with the largest number of nodes) for the D8 to D10 set of 30 runs was better for the combination of the new operators than for any other combination. We see, perhaps, a ceiling effect for the best paths in D7 as they are 51 for all combinations of operators.

The likely reasons for phenotype mutation being less effective for the SITB than for the TSP are four:

1) There are more heuristics than link choices for the SITB. This implies that multiple heuristics will choose the same node, thus weakening the value of the mutation operator's ability to re-introduce lost alleles.

2) The population starts off with random heuristic values for the SITB but all heuristic 1 for the TSP. Thus, phenotype mutation is not used for the SITB to randomize the population and bring in new values.

3) Phenotype mutation for the SITB is not limited to selecting a new value different from the current. Thus, when a gene is selected for mutation in the SITB, there is a chance the same value may be selected.

4) Only one gene per population member can be mutated for the SITB unlike the TSP, where multiple genes may be selected.

These arguments are not all specific to phenotype mutation, and may apply to traditional uniform mutation as well. However, they do show that, in general, mutation is likely to play less of a roll in our implementation of a GA for the SITB than it does for the TSP. Thus, even an improved mutation operator is unlikely to show as dramatic improvements for the SITB as for the TSP, and this is what is found in our results.

10.3.3 Final Comments

For both problems, while overall performance did not increase as much as expected, the phenotype GA operators did show a performance increase, especially when combined. There are cases where, individually, the new operators actually decreased performance, but this was mitigated when combined with the other new operator. Indeed, this should be studied further, and may lead to other improvements to the heuristic encoding technique and better GA operators for it.

10.4 H4 Experimental Analysis

The design and selection of the heuristic set should incorporate knowledge of the problem space appropriate for mapping to the solution space but should not include useless heuristics. If key heuristics are missing, the GA is unable to find promising areas of the solution space. Also, if useless heuristics are included, they detract from the guiding effects of the GA.

H4-1: When key heuristics are removed, both upper end, and average GA performance decreases.

H4-2: When additional, unneeded heuristics are added, average GA performance decreases.

As explained in Section 9.4, we study the effects of the removal of needed heuristics, and that of adding useless heuristics simulating noise in the heuristic set. It is surprising that we did not get the dramatic results expected, but still observe a general decrease in overall effectiveness of the GA with heuristic encoding. In Tables 9.9 to 9.12, “Base set” refers to results from the “Both” experiments from H3 where both of the new GA operators were used with the “Base” heuristic set found to work best. This base set is compared to the two experiments for this hypothesis, where a key set of heuristics is removed (“Minus” in the tables), and extra “noise” heuristics are added (“Plus” in the tables.)

10.4.1 TSP Analysis

Results for both sub-hypotheses as applied to the TSP can be found in Table 9.9 with the Wilcoxon Rank Sum statistical results in Table 9.10. When a small subset of useful heuristics is removed from the operational set, we see a definite and statistically significant degradation in performance as witnessed in the tables. Indeed, for even the smallest problem, eil51, the average and best performance decreased from the base set, and these differences are statistically significant as seen in Table 9.10. The reason for this is quite simple: The new heuristic set does not have the reach into the solution space that the larger set had. There are simply valuable areas of the solution space that can no longer be reached. Of course, we want the heuristic set to prune the search space and, indeed, this is one of the motivating factors behind heuristic encoding. However, we want to design and select heuristics in a manner that will prune unfruitful areas, whereas, here, the more valuable heuristics were removed.

Results of the Plus TSP tests are actually quite interesting. While adding the noise heuristics seems to cause a slight decrease in average performance, it is mostly insignificant. Indeed, the Wilcoxon Rank Sum tests fail to show statistical significance at the 5% level for all but the largest problem. This implies that adding noise does not affect performance as much as expected, and, in most cases, not even to a statistically significant level. The reason for this is likely the phenotype mutation operator. The primary point of this operator is to allow the GA to tailor the selection of heuristics during mutation based on their use within the best 10% of the current population. Thus, it would appear that, for population members with noise heuristics, their presence degrades population members so that they are less likely to be within the best 10%. With this being the case, if the GA is allowed to run long enough, the noise heuristics will slowly be removed from the active population, thus reducing their effect on the population's performance. Also, since no valuable heuristics were actually removed, the same areas of the solution space reachable without the noise heuristics are also reachable with them.

10.4.2 SITB Analysis

Results for both sub-hypothesis as applied to the SITB can be found in Table 9.11 with the Wilcoxon Rank Sum statistical results in Table 9.12. In all cases, the removal of a small subset of useful heuristics noticeably degrades the performance both on average and for the best found, with the exception of D7. Indeed, this difference is also statistically significant as seen in Table 9.12. Also, this degradation worsens as the problem size increases. This may be due to the fact that, as the search space increases, it is more necessary to have a larger number of options for selection of the next node. As the options are restricted and the search space increases, the performance is bound to decrease. As with the TSP results, this is not surprising, since much of the exploration ability of the heuristic set has been removed. In such a case, even with good GA operators and the best parameter settings, the GA may not be able to reach promising areas of the search space that were accessible with the more complete set (the Base Set).

Results when noise heuristics are added are very similar to what was observed with the TSP. In all cases there is a minor reduction in mean performance. However, this reduction is only statistically significant for the two smaller dimensions, D7 and D8. The Wilcoxon Rank Sum test results in Table 9.12 show that the difference in D9 is minor (6.15%) and that there is virtually no difference between the results produced without and with the noise heuristics in D10. Less of a difference is observed in these larger problems due to the fact that they run longer (the larger the problem size/population members, the more generations are required by a GA to reach good solutions), and this allows the phenotype mutation operator more time to remove the noise heuristics from the population (see discussion for the TSP in Section 10.4.1).

10.4.3 Final Comments

Results for the Minus tests were as expected and support the hypothesis. However, adding 8 noise heuristics had less effect than expected. While the phenotype mutation operator was designed to be able to identify and remove unneeded heuristics, it was not expected to be so effective. Indeed, the longer the GA is allowed to run, the less degradation is seen from the noise heuristics. Thus, while the mean with the noise heuristics is generally worse than without, this difference is quite minor. However, removal of valuable heuristics does indeed affect performance strongly in nearly all cases. In the smaller problems the effect is less, due to the reduced size of the space to be explored. In these smaller problems, the remaining heuristics are more able to make up for the loss.

Chapter 11

Conclusions

This work contributes much to the area of GAs as applied to NP-Complete problems and to the use of heuristics in general. Some of the specific contributions are:

1) The specific study of heuristic encoding for GAs and graph-space problems. While there are a few others who have applied heuristic encoding for GAs (Hart and Ross, 1998; Lopez-Camacho et al, 2010), the application to graph-space problems where the low level heuristics are used to dynamically construct a path seems rare (Carlson, 2002; Carlson and Hougen, 2010). Indeed, the study of this technique on two rather diverse problems can inform and guide others in applying heuristics to GAs. Burke et al (2013) mentions that an area of heuristic research needing further work is their application to multiple problem domains, and that this should be shown to produce good results on average across these domains, rather than targeting specific problems.

2) The development of low level heuristic sets for the TSP and the SITB. These heuristic sets are shown to be quite useful in finding good solutions to these problems, and can be used by others both for GAs and with other techniques. The process itself of studying the effectiveness and application of these sets is also a strong contribution.

3) Scope and application. The scope of this project is larger than any known regarding the study and application of heuristics to GA encoding schemes. While others have focused on individual projects, this research attempts to show that this is a generalizable

technique with broad application.

4) The development of phenotype GA operators. These new operators developed for heuristic encoding are quite different than most in the evolutionary computation community, and seem to work well with heuristic encoding. While most GA operators work directly with the GA's population members (the genotype) the phenotype operators use the final product and apply information from solutions back to the manipulation of the GA's population. This work and the path taken to develop such operators should prove useful to others in the evolutionary computation community. Burke et al (2013) lists this as one of the areas for further research in heuristic techniques.

5) Generally good results were obtained and a world record was broken for the 8D hypercube. The previous record of 98 (Rajan and Shende, 1999) had stood for 11 years before being broken by using heuristic encoding combined with phenotype GA operators (Carlson and Hougen, 2010).

6) Already this work is proving useful to others: Ostergard and Pettersson (2015) have used the 99 node snake found in this work as a starting point for their exhaustive search of the 8D hypercube space. They show that our 99 node snake is the longest possible within the 8D hypercube.

Based on the good performance in these two representative graph-space NP-Hard and NP-Complete problems, it appears the heuristic encoding technique performs well on at least graph-space problems, if not on any class of NP-Complete problem to which a GA can be applied. Certainly the performance, when compared to other, more typical, encoding schemes and sets of GA operators, justifies this approach. Indeed, even before the GA starts, the average evaluations for the initial populations using heuristic encodings- are quite good. This fact, of course, caused issues with H2 where the claim was that performance under these conditions would not be good. The good results before implementing custom GA operators and parameter tuning for the heuristics was unexpected.

The disruptive nature of this encoding scheme is mitigated well by using locus-based gene ordering, phenotype crossover and phenotype mutation operators. It seems clear these new operators do contribute to the success as applied to the TSP and SITB problem. It is quite interesting that the phenotype mutation operator was so good at removing the noise heuristics from the operating set, as seen from the results of the H4-2 experiments.

11.1 Overall TSP Findings

The TSP has been studied in depth for many years and has had many different types of algorithms applied to it. It seems from the literature that the best are non-EC techniques, and are often deterministic such as polyhedral, or branch and bound (Carlier and Villon, 1990). Indeed, the LK heuristic algorithm works quite well, even though the initial paper was only applied to smaller problems (Lin and Kernighan, 1973). However, the purpose of this work was never to compete with other's work in finding good solutions to the TSP. Instead, the TSP was used as one of two problems for developing and testing the idea of heuristic encodings for GAs. In this, it has served well and, although not specifically competing for good solutions, the heuristic encoded GA does reasonably well, even in larger problems over a thousand nodes.

Table 11.1 lists some of the better solutions found to various TSPs in this project and compared to the known best (bold matches the best known). For most of the problems under 200 nodes, the best known path was found. For those larger, the heuristic encoded GA still finds reasonably good solutions within a short time (perhaps an hour for 1,000 nodes). However, once these good solutions are found, the GA must run for much longer before meaningful improvements are made. An example is pr2392: In about one hour the GA can typically find solutions around 410,000. However, it takes perhaps another day to get to 400,000 and a week to get to 390,000 (the best known is 378,032).

Table 11.1: TSP results comparing best known to best found paths using heuristic encoding. Bold values found by the heuristic GA are equal to known record. Best known values taken from TSPLIB95 web site.

TSP	Best Known	Heuristic Best
berlin52	7,542	**7,542**
eil101	629	**629**
ch150	6,528	**6,528**
d198	15,780	15,825
lin318	42,029	42,584
pcb442	50,778	51,551
pr1002	259,045	267,199
pr2392	378,032	390,557

11.2 Overall SITB Findings

As with the TSP, the Snake-In-The-Box problem is used primarily as a test and development platform for heuristic encoded GAs. While also a graph-space problem, unlike the TSP, it has a very structured space. Therefore, it makes a good match with the TSP for vetting the heuristic encoded GA with phenotype operators. When a heuristic encoded GA for this problem was initially studied, the performance was not very good. After developing the idea of the phenotype crossover and mutation operators and implementing them, average performance increased dramatically and the world record length snake of 99 nodes in an 8 dimensional hypercube was quickly found. Figure 11.1 shows this longest snake in canonical form and in the 2-D circular format. In this figure, the nodes are numbered from 1 to 256 starting on the right side and going counterclockwise. The tail is light blue, on the right, and the head is dark pink, on the left. This snake has since been proven to be optimal in D8 by Ostergard and Pettersson (2015).

In addition, this technique does quite well in other dimensions. However, performance does noticeably drop off after D9. The best snake of 27 nodes in D6, which an exhaustive search algorithm can find in about two minutes, is found by the GA immediately (within seconds), and sometimes appears as an initial population member before the GA has even started. Also, our exhaustive search implemented using MATLAB has never found better than 49 in D7 (it has been allowed to run for several weeks), where the best is 51. However the heuristic encoded GA finds the best of 51 nearly every time within several minutes. Table 11.2 has additional comparisons of the current record snakes and the best found in this work.

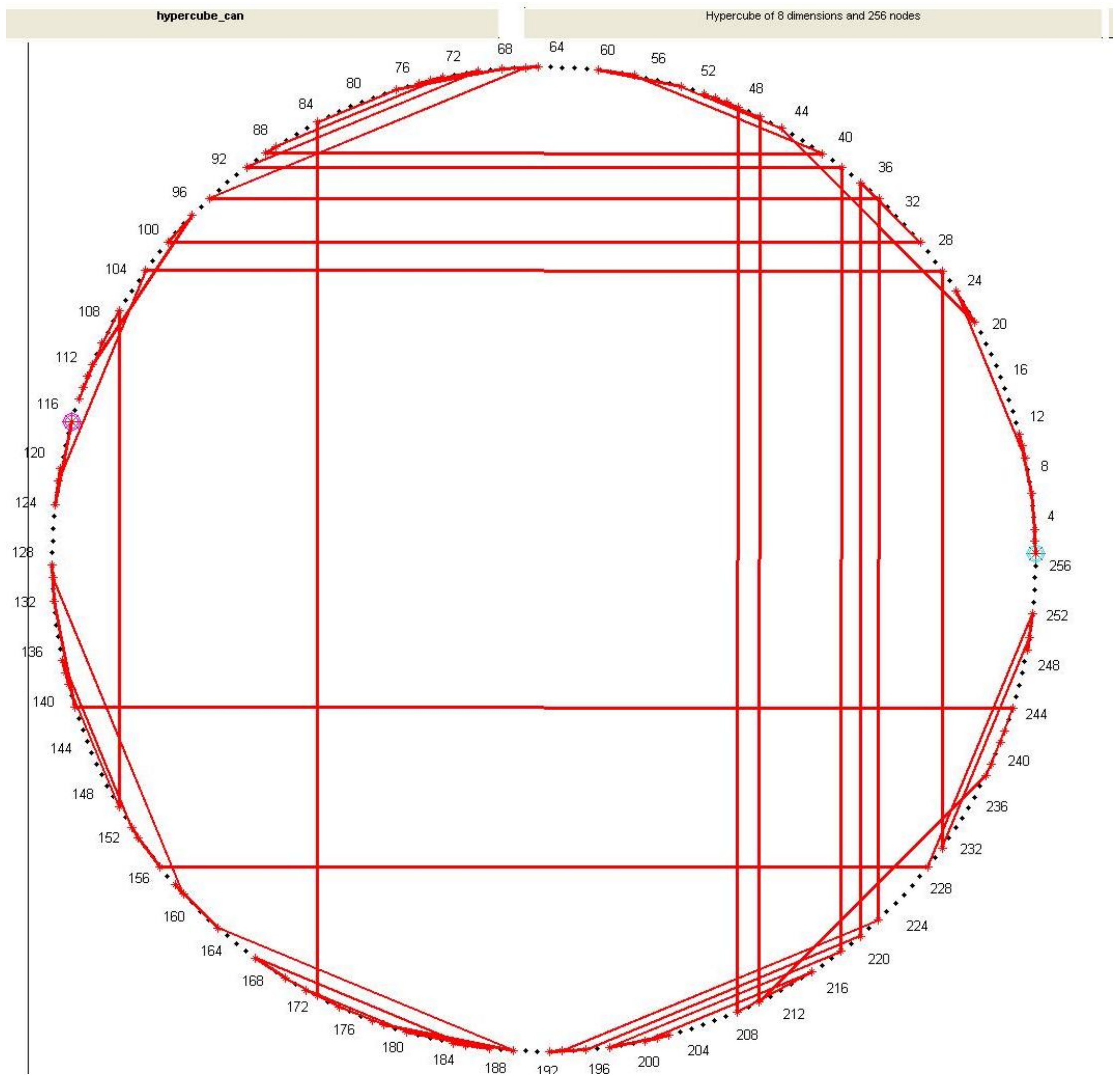

Figure 11.1: The world record D8 snake of 99 nodes in canonical form found with the Heuristic encoded GA using phenotype operators. Light blue node on right is the start/root node or tail of the snake. Pink node on left is the final or head node.

11.3 Final Concluding Remarks

Many GA approaches to graph-space problems (or problems that, while not graphs on the surface, can easily be turned into graphs) use GAs combined with local search optimization. With these techniques, as part of the GA evaluation scheme, some form of deterministic local search is performed on the solutions created by the population members in an attempt to squeeze just a bit more performance out of them. However, our technique is considered a pure Evolutionary Algorithm or EA (Jung and Moon, 2002), in that no additional searching is performed outside of the normal GA evolutionary process. One might argue that the heuristics themselves perform a local search, but it can be argued that this is simply a side effect of the encoding scheme, not an actual optimization technique. Indeed, often the given heuristic selects a node that is not a good choice based on local optimization, but ends up being a good choice from a global perspective. This is one reason for using this technique: to avoid choices that appear good short term (locally, known as a greedy algorithm), but that might detract from the overall success of the population member producing a quality solution!

Finally, while conducting this research, I have found few research projects that use GAs for larger problems, but instead use some form of deterministic algorithm (Wynn, 2012, also Carlier and Villon, 1990), or other form of stochastic algorithms that have been highly modified for the problem at hand (Kinny, 2012, also Allison and Paulusma, 2016). Also, some use other techniques to find seed solutions for a GA (Louis and Xu, 1996). Finally, some use a GA to evolve pruning rules for some other form of search technique (Tuohy, et al, 2007). However, I found no other projects involving a pure EA as applied to very large NP-Complete graph-space problems, as this work does. See Jung and Moon (2002) for a more complete evaluation and discussion of historic evolutionary computation techniques for the TSP.

Table 11.2: SITB results comparing best known to best found with the heuristic encoded GA using phenotype operators. Bold values found using the heuristic GA are equal to known best values. Best known values found by: [1] Davies, 1965, [2] Potter et al, 1994, [3] Carlson and Hougen, 2010, [4] Wynn, 2012, [5] Kinny, 2012, [6] Allison and Paulusma, 2016

SITB Dimension	Best Known	Heuristic Best
6	27 [1]	**27**
7	51 [2]	**51**
8	99 [3]	**99**
9	191 [4]	180
10	371 [5]	332
11	709 [6]	603
12	1,358 [6]	1,087

Chapter 12

Future Work

There are primarily two areas of future work:

1) While phenotype operators were developed for crossover and mutation, a phenotype selection operator should be developed. Currently, in selection for direct copy and selection for crossover, only the evaluation score of the individual population members are used. Some form of selection that also examines the solutions produced by the population members as part of the determination for selection should be developed. This operator should help the population maintain diversity which will allow for better exploration of the solution space.

2) Application to other problems, including non-graph-space problems. This technique should be demonstrated on other problems to determine its efficacy in more broad terms. Indeed, this is the primary focus of this work: to develop the heuristic encoding technique to be a general encoding technique for NP-Complete problems with GAs. If it can be shown that this technique works well on a broader class of problems, this will validate its usefulness and illustrate how it can be adapted to many different problems.

Finally, when these are accomplished, it would make sense to bring this material together in a single volume, in order to more completely communicate the technique of heuristic encoding of problems for GA application, and the idea of phenotype operators.

This would allow others to more quickly develop solutions to their own problems using heuristic encoding and phenotype operators. It would also bring together my work and the work of others in this area.

References

NOTE: Information retrieved regarding world record snakes from Prof. Don Potters SITB web site at:
http://ai1.ai.uga.edu/sib/sibwiki/doku.php/records

Allison, D., Paulusma, D. 2016. New Bounds for the Snake-in-the-Box Problem. arXiv:1603.05119v1, 10 pages, not numbered.

Altenberg, L. 1994. Evolving Better Representations through Selective Genome Growth. *IEEE World Congress on Computational Intelligence*, pages182-187

Arora, S. 1998. Polynomial Time Approximation Schemes for Euclidean Traveling Salesman and Other Geometric Problems. *Journal of the ACM*, volume 45, number 5, pages 753-782.

Bäck, T., Fogel, D. B., Michalewicz, T. 2000. *Evolutionary Computation 1: Basic Algorithms and Operators*. Taylor & Francis Group, LLC.

Bellman, R. 1961. *Adaptive Control Processes: A Guided Tour*. Princeton, NJ: Princeton University Press.

Burke, E. K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., Qu, R. 2013. Hyper-Heuristics: A Survey of the State of the Art. *Journal of the Operational Research Society*, 64, pages 1695-1724.

Carlier, J., Villon, P. 1990. A New Heuristic for the Traveling Salesman Problem. *RAIRO - Operations Research - Recherché Opérationnelle* 24.3, pages 245-253.

Carlson, B. 2002. Rule Coding for Genetic Algorithms: an Alternative Solution to the Traveling Salesman Problem. *International Conference on Artificial Intelligence*, Las Vegas, NV, pages 878-883.

Carlson, B., Hougen, D. 2010. Phenotype Feedback Genetic Algorithm Operators for Heuristic Encoding of Snakes within Hypercubes. *Genetic and Evolutionary Computation Conference*, July 7-11, Portland, OR, 2010, pages 791-798.

Davies, D.W. 1965. Longest -Separated- Paths and Loops in an N Cube, *IEEE Transactions on Electronic Computers*, Vol. 14, page 261.

De Jong, K. A. 1993. Genetic Algorithms are NOT Function Optimizers. In L. D. Whitley, ed., *Foundations of Genetic Algorithms 2*. Morgan Kaufmann, pages 5-18.

Diaz-Gomez, P., Hougen, D. 2006. Analysis of the Snake in the Box Problem: Mathematical Conjecture and Genetic Algorithm Approach. *Genetic and Evolutionary*

Computation Conference, pages 1409-1410.

Engelbrecht, A. P. 2007. *Computational Intelligence: An Introduction, Second Edition*. John Wiley & Sons Ltd.

Falkenauer, E. 1998. *Genetic Algorithms and Grouping Problems*. John Wiley & Sons Ltd.

Garcia-Villoria, A., Salhi, S., Corominas, A., Pastor, R. 2011. Hyper-Heuristic Approaches for the Response Time Variability Problem. *European Journal of Operational Research*, pages 160-169.

Goldberg, D. E. 2002. *The Design of Innovation: Lessons from and for Competent Genetic Algorithms*. Springer Science+Business Media Dordrecht.

Goldberg, D. E. 1989. *Genetic Algorithms in Search, Optimization & Machine Learning*. Addison Wesley Longman, Inc.

Goldberg, D., Lingle, R. 1985. Alleles, Loci, and the Traveling Salesman Problem. *International Conference on Genetic Algorithms and Their Applications*, pages 154-159.

Gonzalez, G. 2009. Elitism, Fitness, and Growth. University of Oklahoma Master's Thesis.

Hart, E., Ross, P. 1998. A Heuristic Combination Method for Solving Job-Shop Scheduling Problems. *Proceedings Parallel Problem Solving from Nature* 5th International Conference, Lecture Notes in Computer Science Volume 1498, pages 845-854.

Hauschild, M., Pelikan, M. 2011. An Introduction and Survey of Estimation of Distribution Algorithms. *Swarm and Evolutionary Computation 1*, pages 111-128.

Holland, J. H. 1975. *Adaptation in Natural and Artificial Systems*. University of Michigan Press. (Second edition: MIT Press, 1992)

Jung, S., Moon, B. 2002. Toward Minimal Restriction of Genetic Encoding and Crossovers for the Two-Dimensional Euclidean TSP. *IEEE Transactions on Evolutionary Computation, 6,* pages 557-565.

Kauffman, S. A. 1989. Adaptation on Rugged Fitness Landscapes. Stein, D., editor, *Lectures in the Sciences of Complexity*, pages 527-618, Redwood City: Addison-Wesley. SFI Studies in the Sciences of Complexity, Lecture Volume I.

Kautz, W. 1958. Unit-Distance Error-Checking codes. *IRE Transactions on Electronic Computers 1958*, 7, pages 179-180.

Kinny, D. 2012. A New Approach to the Snake-In-The-Box Problem. *European Conference on Artificial Intelligence*, doi:10.3233/978-1-61499-098-7, pages 462-467.

Kochut, K. 1996. Snake-in-the-Box Codes for Dimension 7. *Journal of Combinatorial Mathematics and Combinatorial Computations* 20, pages 175-185.

Lin, S., Kernighan, B. 1973. An Effective Heuristic Algorithm for the Traveling Salesman Problem. *Operations Research*, *21*, pages 498-516.

Livingston, M., Stout, Q. 1988. Distributed resources in hypercube computers. *3rd Conference on Hypercube Concurrent Computers and Applications*, pages 222-231.

Lopez-Camacho, E., Terashima-Marin, H., Ross, P. 2010. Defining a Problem-State Representation with Data Mining within a Hyper-heuristic Model which Solves 2D Irregular Bin Packing Problems. *12th Ibero-American Conference on Advances in Artificial Intelligence*, pages 204-213.

Louis, S., Xu, Z. 1996. Genetic Algorithms for Open Shop Scheduling and Re-Scheduling. *11th ISCA International Conference on Computers and Their Applications*, pages 99-102

Louis, S, Li, G. 1997. Augmenting Genetic Algorithms with Memory to Solve Traveling Salesman Problems. Wang, P. P., editor, *Third Joint Conference on Information Sciences*, pages 108-111.

Michalewicz, Z, Fogel, D. B. 2004. How to Solve It: Modern Heuristics, Second Edition. Springer-Verlag.

Mitchell, M. 1996. An Introduction to Genetic Algorithms. MIT Press

Ostergard, P.R.J., Pettersson, V.H. 2015. Exhaustive Search for Snake-in-the-Box Codes. *Graphs and Combinatorics*, 31, pages 1019-1028.

Palombo, A., Stern, R., Puzis, R., Felner, A., Kiesel, S., Ruml, W. 2015. Solving the Snake in the Box Problem with Heuristic Search: First Results. *Eighth International Symposium on Combinatorial Search*, pages 96-104.

Pearl, J. 1985. Heuristics: Intelligent Search Strategies for Computer Problem Solving. Addison-Wesley Publishing Company, Inc.

Potter, W., Robinson, R., Miller, J., Kochut, K., Redys, D. 1994. Using the Genetic Algorithm to find Snake-in-the-Box Codes. *7th International Conference on Industrial & Engineering Applications of Artificial Intelligence and Expert Systems*, pages 307-314.

Rajan, D., Shende, A. 1999. Maximal and Reversible Snakes in Hypercubes. *24th Annual Australasian Conference on Combinatorial Mathematics and Combinatorial Computation*.

Ralston, A. (Editor) 1976. *Encyclopedia of Computer Science, First Edition*. Litton Educational Publishing, Inc.

Ruiz, KH. 2014. Search for Maximal Snake-in-the-Box Using New Genetic Algorithm. *Genetic and Evolutionary Computation Conference*, pages 831-838.

Spears, W. M., De Jung, K. A. 1995. On the Virtues of Parameterized Uniform Crossover. Naval Research Lab Washington DC, 7 pages.

Terashima-Marin, H., Ross, P., Farias-Zarate, C. J., Lopez-Camacho, E., Valenzuela-Rendon, M. 2008. Generalized Hyper-Heuristics for solving 2D Regular and Irregular Packing Problems. *Annals of Operations Research*, pages 369-392.

Reinelt, R. TSPLIB95, University of Heidelberg, Gerhard, e-mail: Gerhard.Reinelt@informatik.uni-heidelberg.de
http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/index.html

Tuohy, D., Potter, W., Casella, D. 2007. Searching for Snake-in-the-Box Codes with Evolved Pruning Models. *International Conference on Genetic and Evolutionary Methods*, pages 3-9.

Wynn, E. 2012. Constructing Circuit Codes by Permuting Initial Sequences. arXiv:1201.1647v1, 9 pages, not numbered.

Zhang, X., Ma, Y. 2014. Solving TSP Problems with Hybrid Estimation of Distribution Algorithms. International Conference on Intelligent Computing, Lecture Notes in Computer Science 8588, pages 73-81.

Appendices

A. List of Heuristics and their Descriptions

This appendix contains the complete list and description of the two heuristic sets for the TSP and the SITB. As needed, it also contains additional explanations and supporting figures for various concepts involved.

A.1 TSP Heuristics

The initial heuristic set contained 33 heuristics. However, it was found that 8 acted as noise and did not contribute to finding shorter paths. Due to this, the heuristic numbers may have gaps. In all of the descriptions, it is implied that only available nodes will be considered, and that the current node is implied (in "Select the closest node" it is implied that this refers to the closest node to the current node.) Also, *center* refers to the Euclidean center of mass of the entire problem, which is calculated at the beginning and kept static throughout the path building process. The current set below contains 25 numbered from 1 to 26:

1) Select the closest node.

3) Select the closest node to the center.

4) Select furthest node from the center.

5) Select the node whose sum of distance from the current node and distance from the center is the smallest.

6) Select the node that is the closest and also is closer to the center than the current node.

7) Select the node that is the closest and is also further from the center than the current

node.

8) Select the second closest node from all available nodes.

9) Select the third closest node from all available nodes.

10) Select the fourth closest node from all available nodes.

Heuristics 11 to 18 use the idea of dividing the space into Euclidian quadrants. Figure A.1 shows the four quadrants used by numbers 11 to 14 on the left, and the four quadrants used by numbers 15 to 18 on the right. For all of these heuristics, the current node is at the origin. I refer to quads one through four on the right as alternate quads.

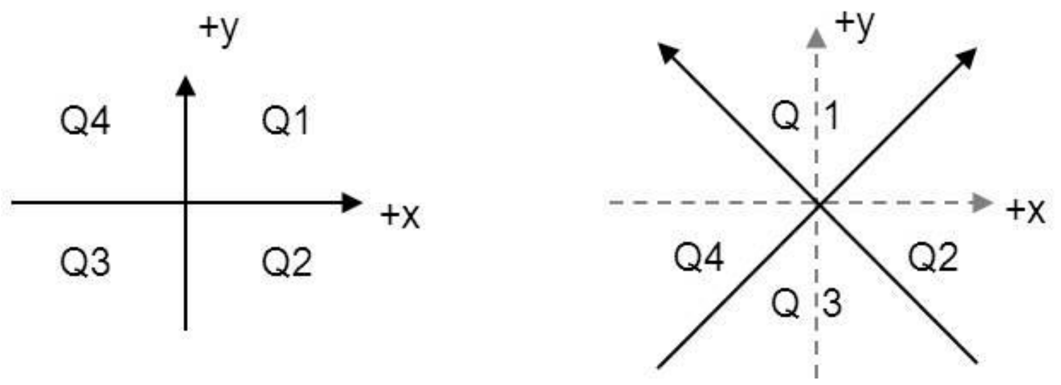

Figure A.1: The eight quadrants used for TSP heuristics 11 to 18. The current node is at the origin.

11) Select the nearest node in quad 1.

12) Select the nearest node in quad 2.

13) Select the nearest node in quad 3.

14) Select the nearest node in quad 4.

15) Select the nearest node in alternative quad 1.

16) Select the nearest node in alternative quad 2.

17) Select the nearest node in alternative quad 3.

18) Select the nearest node in alternative quad 4.

19) Select the node that is the closest in the *x* coordinate only.

20) Select the node that is the closest in the *x* coordinate that is also closer to the center.

21) Select the node that is the closest in the *x* coordinate that is also further from the center.

22) Select the node that is the closest in the *y* coordinate only.

23) Select the node that is the closest in the *y* coordinate that is also closer to the center.

24) Select the node that is the closest in the *y* coordinate that is also further from the center.

Heuristics 25 and 26 use the distance between the furthest two nodes in the problem space, referred to as *total distance*, which implies that it is the total distance the problem space occupies.

25) Select the closest node that is at least 10% of the total distance from the current node.

26) Select the closest node that is at least 20% of the total distance from the current node.

A.2 SITB Heuristics

The initial SITB heuristic set contained 26 heuristics. Through experimentation and analysis it was found that 9 of these contributed little or nothing to the success of the GA, and were removed. The remaining 17 are explained below. In the descriptions, an example of a link table is in Figure 5.1. This is a table showing the connections of all nodes. Each node has a row listing the nodes it connects to.

As with the TSP set, some from the original set have been removed. Thus, the remaining 17 are not contiguous in their numbering scheme. Also, a node that is a dead end, in that no other nodes after it can be selected, will only be selected as a last resort.

Some additional concepts need explanation. *Invalid* means that a node cannot be added to the snake, either because it is already in the snake, or because it already has a neighbor in the snake and that neighbor is not the head node of the current path.

Non-Dead End (NDE) node is an available node (it can be added to the snake) that has at least one neighbor that can be added to the snake. Thus, if this node is added, the snake can still grow. However, a *Dead End* (DE) node, if selected, will terminate the growth of the snake.

A node can have multiple neighbors in the snake, and this value is kept track of as part of the node's current status. When a NDE node is being considered, some of the heuristics will examine the number of invalid neighbors it has. This examination can be performed in two ways:

1) Count the number of invalid neighbors, referred to as the *invalid count*

2) Sum up the number of times each neighbor that is invalid has been invalidated due to a neighbor in the snake, referred to as the *invalid sum*.

Figure A.2 illustrates these two concepts. This figure is not meant to properly represent a hypercube, and does not accurately show all of the nodes and links.

1) Select the first node moving from left to right across the link table.

2) Select least invalid sum greater than one. This heuristic will select the neighbor that has the least value for invalid sum (explained above), but that is greater than one.

3) Select closest link with respect to the inbound link for the current head. Treat the row from the link table for the current head node as a circular list.

4) Select the node that has the largest invalid sum value.

6) Select the node with the largest invalid count value. Select the right node in a tie.

7) Select the furthest link with respect to the inbound link (opposite of #3).

8) Select the furthest link from the inbound link, but add one to the link number.

10) Select the node that has the largest invalid sum value. Select right in a tie.

12) Select the closest link from the inbound link. Select right in a tie.

13) Select the furthest link from the inbound link, but subtract one from the link number.

14) Select the closest link from the inbound link, but add one to the link number.

16) Select least invalid sum greater than one. Select right in a tie.

19) Select the furthest link from the inbound link. Select right in a tie.

20) Select the node with the largest invalid count.

21) Select closest link from the inbound link. However, start searching at the inbound link minus 2. Select right in a tie.

24) Select the neighbor that has the least number of NDE neighbors, but at least one.

26) Select the neighbor that has the least number of NDE neighbors whose sum of available neighbors is smallest, but at least one. This is a rather difficult heuristic to understand. It involves looking two levels deep past the node under consideration. Figure A.3 illustrates this heuristic.

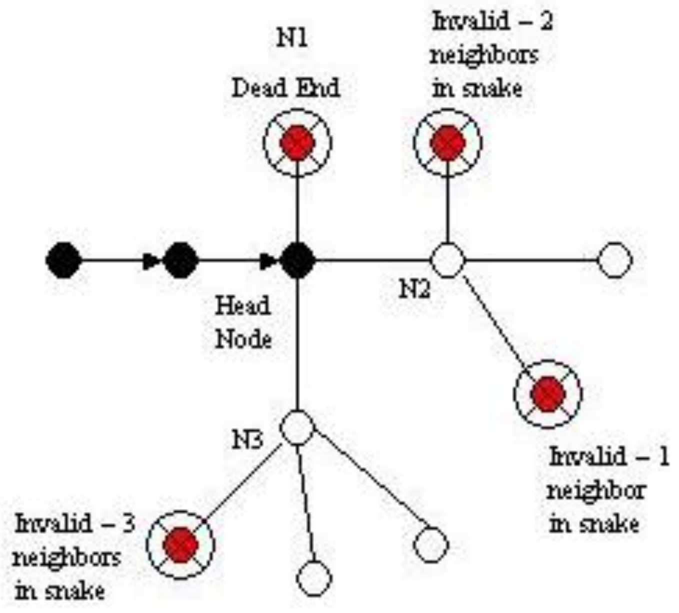
N1
Dead End
Invalid – 2 neighbors in snake
Head Node
N2
N3
Invalid – 1 neighbor in snake
Invalid – 3 neighbors in snake

Figure A.2: SITB example of invalid sum and invalid count determination. Black nodes are in the snake, white nodes are available to be added, and red nodes are invalid and cannot be added. From the current head node (far right black node), there are three options: N1, N2, and N3. N1 is invalid due to an immediate neighbor, other than the current head node, being in the snake and so is not considered. Both N2 and N3 are valid, NDE nodes. They are valid because they have no immediate neighbor in the snake, and they are NDE since both have an additional neighbor that can be added next. Now, to determine the sum and count of each using 1 and 2 above we first look at only the total number of invalid neighbors that each has: the invalid count. This value is 2 for N2 and 1 for N3. Therefore, if we were using heuristic 6 we would select N2, which has the largest invalid count value. The invalid sum value for N2 is 2 + 1 or 3, while the invalid sum for N3 is 3 also. Therefore, if we were to use heuristic 2, the link table would be consulted to break the tie and the node that appears first, starting from the left, would be selected.

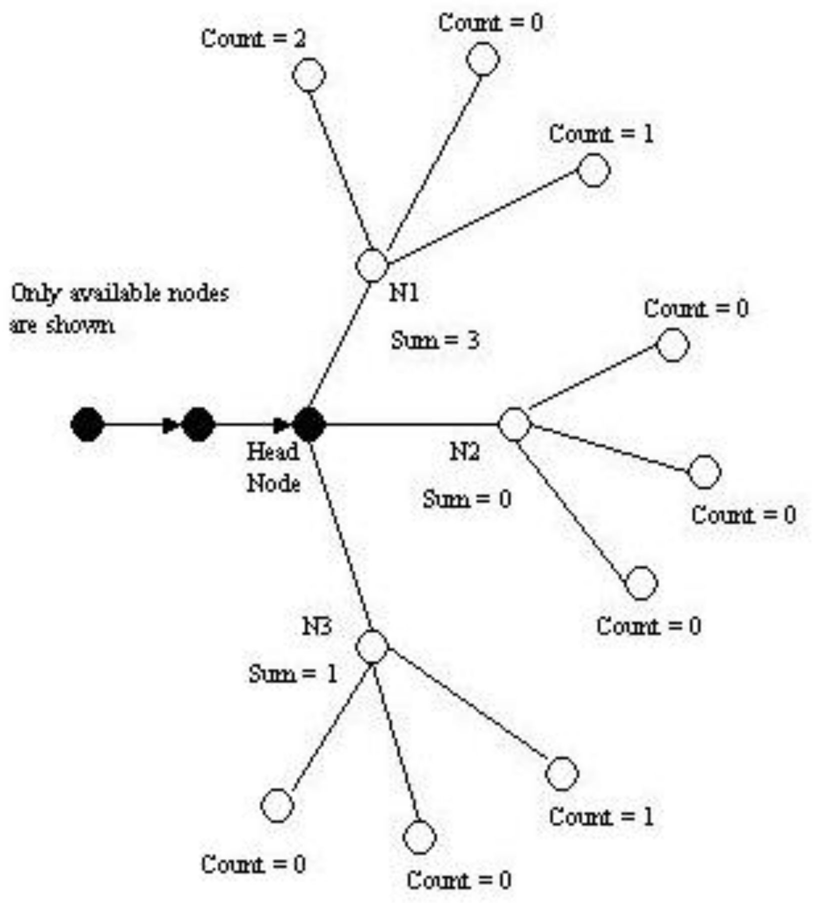

Figure A.3: SITB example of NDE sum for heuristic 26.
From the head node, we see three possible nodes under consideration. Heuristic 26 looks at how many available nodes each NDE neighbor of the node under consideration has. Thus, in the figure, we see the neighbors of N1 have a count of 2, 0, and 1. This indicates two available neighbors, zero available neighbors (a dead end neighbor), and one available respectively. Therefore, the score given N1 is 2+0+1 or 3. This heuristic is looking for the smallest score, not the largest. However, the score must be at least one or the node has no NDE neighbors, meaning only two additional nodes could be added. Due to this, N2 is not considered. The competition is between N1 with a score of 3 and N3 with a score of 1, and, thus, N3 will be selected.

B. Parameter Settings for Hypotheses Experiments

This appendix contains the specifics of the parameters used for the various experiments. Each section explains the settings for the experiments exploring a single hypothesis.

B.1 H2 Parameter Settings

The GA parameter settings for H2 have been selected to conform to the most common values used for a typical GA. These settings are mostly the same for the TSP and the SITB for all encoding schemes used. The list below gives a brief description and the values used.

Selection elitism is enabled for all experiments. This implies that the best population member from the current population will always be selected for being copied, without change, to the next population. Also, the best member is immune from mutation. Finally, the best member is used at least once as one of the two members used for crossover to produce two new members.

For all experiments, a certain percentage of the old population is directly copied to the new. This percentage for H2 is 30%. The mechanism for selecting these 30% is tournament selection without replacement. This means that no member will be copied twice.

For all experiments, the remaining 70% of the new population will be created through crossover (in the literature this is referred to as 70% probability for crossover). For all experiments, the selection mechanism for crossover is tournament selection with replacement. These are selected from the complete old population, including those already selected for direct copy.

Tournament number: In all experiments, both selection for direct copy and selection for crossover, the tournament number used is 2. This is the number of members selected to take part in the tournament, where the member with the best evaluation wins and is selected. If this number is increased, selection pressure (the likelihood that a better member will be selected) will go up. If it is set to unity, then we have simply random selection with no pressure to select based on quality of solution.

Crossover points: When performing crossover, 2 crossover points are used for all experiments.

Also, with the exception of the best member being immune from mutation, mutation is applied to every gene in every population member of the new population, once it has been created through selection and crossover.

Finally, linear gene ordering is used for all H2 experiments. An individual population member is evaluated by starting at index 1 (the gene at the first locus), determining the node that should be added based on this gene, then going to the gene at locus 2 for the next, then the gene at locus 3, et cetera, until the path cannot continue or it has been completed. Each population member has the same number of genes as the problem has nodes. This implies that over half of the genes for the SITB population members will not be used as the nodes that can be added are constrained and this is a maximization problem. As an example, there are 128 nodes in a 7 dimensional hypercube, but the longest known path is only 51 nodes. Also, for the TSP, an additional gene is added at the end which indicates the root node (the node to start building the path from). As mentioned previously, due to the symmetries in the hypercube, the start node is irrelevant. Thus, we always start at node 1.

The additional parameters that are typically tuned for each problem are:

1) Number of Generations.

2) Population size.

3) Mutation rate. Values are listed as parts per 10,000 (pp10k).

The values for the problems used in H2 are listed in Table B.1 for the TSP and Table B.2 for the SITB. In Table B.1, the first three rows are the values used for the traditional encoding schemes (nodes) while the last three are for the heuristic encoding scheme. In Table B.2, the first three rows are for the simple link encoding scheme, the middle three for the link-search scheme, and the last for the heuristic encoding scheme. The reason these values are different for the different encoding schemes are that we attempted to get the experiments to take approximately the same amount of time for a more equal, proper comparison. The traditional encoding schemes typically run faster and, therefore, are given a larger population size or more generations, or both.

Table B.1: Hypothesis H2 parameter values for TSP experiments. First group indicates settings for the Node encoding experiments, while the second shows the settings for the heuristic encoding experiments. Mutation rates are in parts per 10,000.

	eil51	lin105	rat195	a280	d493	u1060
Node Encoding						
Generations	2,000	3,000	5,000	8,000	15,000	30,000
Population Size	1,000	1,000	1,000	500	200	200
Mutation Rate	120	50	30	12	10	5
Heuristic Encoding						
Generations	2,000	2,000	4,000	5,000	6,000	7,000
Population Size	300	300	500	500	400	500
Mutation Rate	100	60	35	11	13	5

Table B.2: Hypothesis H2 parameter values for SITB experiments. First group indicates settings for the link only encoding, second for the link with simple search, and last for the heuristic encoding experiments. Mutation rates are all in parts per 10,000.

	7D	8D	9D
Link Encoding			
Generations	500	800	1500
Population Size	1200	9000	10000
Mutation Rate	100	80	60
Link+Search Encoding			
Generations	500	800	1500
Population Size	800	6000	7000
Mutation Rate	140	100	80
Heuristic Encoding			
Generations	500	800	1500
Population Size	600	4000	4000
Mutation Rate	120	100	80

B.2 H3 Parameter Settings

The GA parameter settings for H3 have been tuned to work best with the new GA operators, and these parameters are used throughout, even when the traditional GA operators are used.

As with H2, elitism is used throughout for selection, mutation, and crossover selection (see discussion in B.1 above.)

Selection, however, is reversed in that now all experiments will select for direct copy 70% with the remaining 30% being created through crossover. These numbers have been found to work best for heuristic encoding with phenotype operators. Also, for the SITB we now use selection percent (see Section 7.2.2) instead of tournament selection for direct copy. However, we still use tournament selection from the old population when selecting for crossover.

As with H2, the tournament number used is 2 for all experiments in both selection for direct copy and for crossover. Also, the number of crossover points is 2 for both problems and all experiments.

Mutation is only applied to directly copied members of the new population with the exception of the best member, which is immune to mutation. Finally, for the TSP, mutation is applied to every gene, whereas for the SITB, mutation is only applied to a single active gene currently being used to produce the snake.

Locus-based (also referred to in the literature as vertex-based) gene ordering is used for all experiments and for both problems. This scheme seems to work best for heuristic encoding, especially with graph-space problems.

The additional parameters that are typically tuned for each problem are:

1) Number of Generations.

2) Population size.

3) Mutation rate. Values are listed as parts per 10,000 (pp10k).

These values for the problems used in H3 are listed in Table B.3 for the TSP and B.4 for the SITB. These values are used for all experiments. As elsewhere, 30 GA runs are performed for each.

Table B.3: Hypotheses H3 and H4 parameter values for TSP experiments. The same parameters were used for both the subtraction of and the addition of heuristics experiments. Mutation rates are all in parts per 10,000.

	eil51	lin105	rat195	a280	d493	u1060
Generations	2,000	2,000	4,000	5,000	6,000	7,000
Population Size	300	300	500	500	600	800
Mutation Rate	130	100	65	35	20	9

Table B.4: Hypotheses H3 and H4 parameter values for SITB experiments. The same parameters were used for both the subtraction of and the addition of heuristics experiments. Mutation rates are all in parts per 10,000.

	7D	8D	9D	10D
Generations	500	800	1,300	2,000
Population Size	600	4,000	4,000	2,000
Mutation Rate	190	95	32	25

B.3 H4 Parameter Settings

All parameters for all experiments in support of H4 are the same as for H3. See Tables B.3 and B.4 for the settings for the generations, population size, and mutation rates.

The TSP heuristics removed for the H4-1 experiments were: 4, 5, 6, 7, and 11.

The SITB heuristics removed for the H4-1 experiments were: 4, 6, 20, 24, and 26.

The TSP noise heuristic duplicated 8 times for the H4-2 experiments was:

"Add the node that is furthest from the current node."

The SITB noise heuristic duplicated 8 times for the H4-2 experiments was:

"Add the node with the largest number of NDE neighbors."

C. H1 Population Members and the Paths they Create

This appendix contains the heuristic encoded population members that were artificially constructed to create the best known paths for three of the larger TSP instances and three of the larger SITB instances.

C.1 TSP Population Members and their Paths

The population members show the genomes of the solutions. These consist of the heuristics used to construct the paths, followed by an additional value representing the root node. Also, the corresponding paths created from the genome list the TSP node numbers, starting and ending with the root node. The paths produced are logically equivalent to the best path listed on the TSPLIB95 web-site.

lin105 locus-based population member that will produce the best known path of length 14,379:

```
1   1   1   1   1   1   1   1   1   1   1   1   4   1   1   7   1   1
1   7   1   1   1   1   1   1   10  1   1   1   1   6   1   1   1   1
1   1   1   8   1   8   1   9   1   1   1   1   8   1   1   1   1   1
1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   5   1   3
1   1   1   1   1   1   1   10  5   1   1   1   1   1   1   1   1   8
1   1   5   1   1   1   1   1   4   1   1   1   1   1   1   1
```

lin105 path produced by this population member:

```
1    2    6    7    10   11   15   103  21   22   29   30   31   32
33   28   23   20   12   19   24   27   16   17   18   25   26   36
37   42   41   43   46   52   53   58   57   54   51   47   44   104
40   49   45   48   50   55   56   59   105  62   63   70   69   74
75   81   73   76   80   86   79   77   72   64   67   68   71   78
82   83   84   85   91   92   96   97   101  102  93   89   90   98
99   100  95   94   88   87   66   65   61   60   39   38   35   34
14   13   4    5    9    8    3    1
```

a280 locus-based population member that will produce the best known path of length 2,579:

```
4   4   1   1   1   1   1   1   1   8   1   1   1   6   1   1   1   1   1   6   1   1   1   1
1   1   1   1   1   1   1   1   8   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
1   1   1   7   7   7   7   7   7   1   1   1   1   1   1   7   1   1   1   1   1   7   1   1
1   1   1   1   5   1   9   1   1   1   1   1   7   1   5   6   6   1   1   1   1   1   1   1
7   7   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   5   1   8   1   1   1
1   1   1   1   5   1   1   1   5   5   1   1   1   7   1   7   1   1   3   1   1   1   1   1
8   8   5   3   1   3   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
5   8   1   1   5   1   1   7   1   8   3   1   1   1   1   1   1   1   8   1   1   1   1   1
1   6   1   1   8   1   1   1   6   8   1   1   1   6   8   1   1   1   1   7   1   8   1   7
1   8   1   7   1   8   1   1   1   1   1   1   1   1   1   1   1   1   1   4   7   7   1   1
1   1   6   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
```

a280 path produced by this population member:

```
1     2     242   243   244   241   240   239   238   237   236   235   234   233
232   231   246   245   247   250   251   230   229   228   227   226   225   224
223   222   221   220   219   218   217   216   215   214   213   212   211   210
207   206   205   204   203   202   201   198   197   196   195   194   193   192
191   190   189   188   187   186   185   184   183   182   181   176   180   179
150   178   177   151   152   156   153   155   154   129   130   131   20    21
128   127   126   125   124   123   122   121   120   119   157   158   159   160
175   161   162   163   164   165   166   167   168   169   170   172   171   173
174   107   106   105   104   103   102   101   100   99    98    97    96    95
94    93    92    91    90    89    109   108   110   111   112   88    87    113
114   115   117   116   86    85    84    83    82    81    80    79    78    77
76    75    74    73    72    71    70    69    68    67    66    65    64    58
57    56    55    54    53    52    51    50    49    48    47    46    45    44
59    63    62    118   61    60    43    42    41    40    39    38    37    36
35    34    33    32    31    30    29    28    27    26    22    25    23    24
14    15    13    12    11    10    9     8     7     6     5     4     277   276
275   274   273   272   271   16    17    18    19    132   133   134   270   269
135   136   268   267   137   138   139   149   148   147   146   145   199   200
144   143   142   141   140   266   265   264   263   262   261   260   259   258
257   254   253   208   209   252   255   256   249   248   278   279   3     280
1
```

pr2392 locus-based population member that will produce the best known path length of 378,032:

```
1       4       1       1       1       1       1       1       1       1       1       1
1       1       1       1       1       1       1       1       1       1       1       1
1       1       1       1       1       1       1       1       1       1       1       1
1       1       1       1       1       1       1       1       1       1       1       1
1       1       1       1       1       1       1       1       1       1       1       1
1       1       1       1       1       1       1       1       1       1       1       1
1       1       1       1       1       1       1       1       1       1       1       1
1       1       1       1       1       1       1       1       1       1       1       1
1       1       1       1       1       1       1       1       1       1       1       1
1       1       1       1       1       7       1       1       1       1       1       1
1       1       1       1       1       8       1       1       8       1       1       1
1       1       1       1       1       1       1       1       1       1       9       1
1       1       1       1       1       1       1       1       8       1       1       1
5       1       1       1       6       1       8       1       1       5       1       1
1       8       1       1       1       1       1       1       1       1       1       1
1       1       1       1       8       1       7       1       1       1       1       1
1       8       1       1       1       1       1       7       1       1       1       1
1       1       1       1       1       1       1       1       9       1       1       7
11      1       8       8       1       1       1       1       1       1       1       1
1       1       1       1       1       1       1       1       1       1       1       1
1       1       1       1       6       1       1       1       1       1       7       8
8       1       1       8       1       1       1       1       1       1       1       9
1       1       1       1       1       1       1       1       1       1       5       1
1       5       1       1       1       9       1       1       1       1       1       1
1       1       1       1       5       1       1       1       1       1       5       1
8       1       1       1       5       1       5       1       5       1       1       1
1       1       5       1       1       7       1       1       1       1       1       1
1       1       5       5       1       1       1       8       1       1       1       9
1       1       1       1       1       1       1       1       5       1       1       1
1       1       1       1       1       10      1       1       1       8       1       1
1       1       1       1       1       1       8       1       5       1       1       1
1       1       1       1       1       1       1       1       1       8       1       1
7       9       1       7       1       1       1       1       1       1       1       8
6       6       1       1       1       1       1       1       1       1       1       1
1       1       1       1       1       1       1       1       1       1       1       1
1       1       1       1       1       1       1       1       1       1       1       5
1       1       8       7       1       7       7       1       1       1       1       1
1       1       1       1       1       1       1       1       1       1       1       1
1       1       1       1       1       1       1       10      1       1       1       1
1       1       6       9       8       1       5       1       1       1       1       1
1       1       9       1       1       1       1       1       1       1       1       1
1       6       1       1       8       1       1       1       9       1       1       1
1       1       9       1       1       1       1       1       5       1       1       8
1       1       1       1       7       1       6       8       1       1       1       7
1       8       1       1       1       1       1       1       1       8       1       1
1       1       22      1       1       1       1       1       6       1       7       1
1       1       1       1       1       1       1       1       1       5       9       8
1       1       1       8       8       1       1       1       5       1       1       1
5       1       1       1       1       1       1       1       1       5       1       1
1       1       1       1       1       1       5       1       1       1       7       1
1       1       1       1       1       1       1       8       1       8       1       1
1       1       1       1       1       6       1       1       7       1       8       1
1       1       1       1       1       1       1       1       1       1       1       1
5       5       1       1       1       1       1       1       1       1       1       1
```

1	1	1	1	1	1	1	1	1	8	1	1
1	1	1	1	1	1	8	1	1	1	8	1
9	1	1	1	9	1	8	1	1	1	7	1
1	8	1	1	8	1	1	1	1	1	1	1
1	7	1	6	1	1	1	1	1	5	1	1
6	1	1	1	1	8	1	10	8	1	1	6
6	8	1	8	1	1	1	1	1	1	1	6
1	1	1	1	7	1	1	1	1	1	1	5
1	1	1	1	1	1	1	1	1	1	1	1
1	1	8	1	1	1	1	1	1	1	1	5
1	1	1	1	1	8	9	8	1	1	1	1
1	1	1	1	1	1	1	1	8	7	1	1
1	1	1	1	1	1	5	1	1	1	5	1
1	1	1	7	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	8	8	1
1	1	8	1	1	10	1	1	6	5	1	1
7	1	5	1	1	1	1	1	1	1	1	1
8	8	1	1	1	8	1	1	1	5	1	1
1	5	8	8	1	1	5	1	1	1	1	1
1	1	1	3	1	1	1	1	1	1	1	1
1	1	13	1	1	1	1	1	6	1	1	1
1	1	1	1	1	1	1	5	1	1	1	1
1	1	1	1	1	8	1	1	1	8	1	1
8	1	1	8	1	1	1	7	1	1	1	1
1	1	1	1	1	1	1	1	1	1	7	1
8	1	1	1	1	1	1	6	1	1	1	1
1	1	1	5	6	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	5	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
5	1	1	1	1	1	1	1	1	7	1	1
1	8	1	9	1	1	1	9	1	6	1	1
1	8	1	1	8	1	1	6	1	1	1	1
1	1	1	1	1	1	1	1	1	1	6	1
1	1	1	1	1	1	1	1	7	1	1	1
1	1	1	1	1	1	9	1	7	1	1	1
1	1	1	1	1	1	9	8	1	1	1	1
1	1	1	1	1	1	1	1	6	1	9	1
1	1	1	8	1	9	1	1	1	1	1	1
9	1	1	1	1	1	1	1	1	1	1	1
1	1	1	8	1	1	1	1	1	1		
1	1	8	1	1	1	1	1	8	9	6	1
1	1	1	1	1	1	1	1	1	1	1	8
8	1	1	1	1	1	1	9	1	1	1	1
1	1	1	1	7	1	1	1	1	1	1	1
8	1	1	10	1	1	1	1	1	1	1	1
6	1	1	10	1	1	8	8	1	1	9	1
8	1	1	1	1	1	1	1	1	1	6	8
1	1	1	8	1	1	1	1	7	8	8	1
1	8	1	1	1	1	1	1	1	1	6	1
1	1	1	1	1	1	1	1	1	8	1	1
6	1	1	1	1	1	1	1	1	1	1	1
1	1	6	1	1	1	1	1	1	1	1	1
7	1	1	9	6	1	1	1	9	1	5	1
1	5	1	1	1	5	1	1	1	1	1	1

```
1    1    1    1    1    1    1    1    5    1    8    1
1    1    1    1    1    7    1    1    1    1    1    7
1    1    1    1    1    1    1    1    1    1    1    1
9    1    1    8    11   1    8    8    1    1    1    1
1    1    1    1    1    1    1    1    1    1    1    1
1    1    1    1    1    1    1    1    5    1    1    1
1    1    1    8    9    5    1    8    1    1    1    1
1    1    1    9    1    1    1    1    1    1    7    1
1    1    8    1    1    8    1    1    1    6    1    1
1    1    1    1    1    1    1    1    8    1    1    1
1    1    8    1    5    1    1    1    6    1    6    1
9    1    1    1    1    1    8    8    8    1    1    1
1    1    1    1    1    1    8    8    1    1    1    8
1    1    1    9    1    1    1    1    1    1    1    1
8    1    1    1    1    1    1    1    1    19   1    1
1    6    1    1    1    1    1    1    1    1    8    1
8    1    1    1    1    1    1    1    1    1    1    1
1    5    1    1    5    9    1    8    1    1    1    1
1    1    1    8    5    5    1    1    1    10   1    1
1    1    5    1    1    1    1    1    1    1    1    1
1    1    1    1    1    1    1    1    1    1    1    1
1    1    1    1    1    1    1    1    1    1    1    1
1    1    1    1    1    1    1    1    1    1    1    1
1    9    7    1    1    10   1    1    1    1    1    1
1    1    1    1    1    1    1    1    1    1    1    1
1    1    3    1    1    1    7    1    1    4    1    1
4    1    1    1    8    1    1    1    1    1    1    1
1    1    1    1    1    1    1    8    1    5    1    1
1    1    1    1    8    1    1    1    1    1    1    3
1    1    1    1    1    5    1    1    1    1    1    1
8    1    1    9    1    1    1    1    1    1    1    1
1    1    1    1    1    1    1    1    1    1    1    1
1    1    10   1    1    1    1    1    1    8    9    8
1    8    1    1    1    1    1    1    1    5    1    1
1    1    1    1    1    1    1    1    8    1    1    4
1    1    1    4    1    1    1    1    1    7    1    1
1    1    1    4    1    1    4    1    1    1    1    7
1    8    6    1    1    1    8    1    8    1    1    1
1    1    1    1    8    1    1    1    1    4    1    1
1    1    1    8    1    8    1    1    1    1    1    1
1    1    1    1    9    9    8    1    1    1    8    8
1    1    1    8    1    1    1    9    1    1    1    1
1    1    1    1    8    1    1    1    1    1    1    1
1    10   1    1    1    8    1    1    1    1    1    1
1    1    8    1    8    1    1    1    1    1    1    1
10   1    1    4    1    4    1    1    1    1    1    1
1    1    1    1    1    1    1    9    4    1    1    1
1    1    1    1    1    1    1    1    1    1    1    1
1    1    1    1    8    1    1    1    1    1    1    1
1    6    1    1    1    8    1    6    1    1    1    9
1    7    1    1    1    8    1    1    8    1    1    7
1    1    1    1    1    1    1    1    8    1    8    1
1    1    1    1    8    1    1    9    1    1    1    1
8    1    6    6    1    1    6    8    8    1    6    1
1    1    1    1    1    1    8    1    1    1    1    6
1    1    1    1    1    1    6    1    1    1    1    1
1    1    1    1    1    1    1    1    1    8    1    1
```

```
1       1       1       1       1       1       8       1       1       1       1       1
6       9       8       1       1       1       1       1       1       1       1       1
1       1       1       8       8       1       1       1       1       1       1       1
1       8       1       1       1       7       1       1       1       1       8       1
1       1       1       1       1       1       1       1       1       1       1       1
8       1       1       1       1       1       1       7       1       1       10      1
1       8       8       1       1       9       1       8       1       1       1       1
1       1       1       1       1       7       6       1       1       1       8       1
1       1       1       7       7       6       1       1       8       1       1       1
1       1       1       1       1       1       1       5       1       1       1       1
1       1       1       1       1       1       1       1       1       1       1       1
1       1       1       1       8       1       1       1       1       1       1       1
1       8       1       1       1       5       1       5       1       1       1       5
1       8       1       1       1       5       1       1       7       1       1       8
1       1       1       1       1       1       1       1       1       1       1       1
1       1       8       1       1       1       1       1       1       1       1       1
5       1       1       1       1       1       1       1       1       1       9       1
5       1       1       1       1       1       1       1       1       1       7       7
1       1       1       1       1       1       1       1       1       1       1       1
8       1       7       1       1       1       1       8       1       9       1       1
1       1       1       1       9       1       1       1       1       1       1       1
1       1       1       1       1       1       1       8       1       1       1       1
1       1       1       1       7       1       1       1       1       1       6       9
8       1       1       1       1       1       1       1       1       1       1       1
1       8       5       1       1       1       1       1       1       7       1       1
1       1       1       1       1       1       8       1       1       1       1       1
1       1       1       1       1       10      1       1       1       1       1       1
1       1       8       1       1       6       1       1       7       8       1       1
9       1       8       1       1       1       1       1       1       1       1       1
7       8       1       1       1       6       1       1       1       1       7       8
6       1       1       7       1       1       1       1       1       1       1       1
1       1       1       1       1       1       1622
```

pr2392 path produced by this population member:

```
1622    1621    1620    1619    1618    1617    1616    1615    1614    1613    1612    1611
1610    1609    1608    1607    1606    1605    1604    1603    1602    1601    1600    1599
1598    1597    1596    1595    1594    1593    1592    1591    1590    1589    1588    1587
1586    1585    1584    1583    1582    1581    1580    1579    1578    1577    1576    1575
1574    1573    1572    1571    1570    1569    1568    1567    1566    1565    1564    1563
1562    1561    1560    1559    1558    1557    1556    1555    1554    1553    1552    1551
1550    1549    1548    1547    1546    1545    1544    1543    1542    1541    1540    1539
1538    1537    1536    1535    1534    1533    1532    1531    1530    1529    1528    1527
1526    1525    1524    1523    1522    1521    1520    1519    1518    1517    1516    1515
1514    1513    1512    1511    1510    1509    1508    1507    1506    1505    1504    1503
1502    1501    1500    1499    1498    1497    1496    1495    1494    1493    1492    1491
1490    1489    1488    1487    1486    1485    1484    1483    1482    1481    1480    1479
1478    1477    1476    1475    1474    1473    1472    1471    1470    1469    1468    1467
1466    1465    1464    1463    1462    1461    1460    1459    1458    1457    1456    1455
1454    1453    1452    1451    1450    1449    1448    1447    1446    1445    1444    1443
1442    1441    1440    1439    1438    1437    1436    1435    1434    1433    1432    1431
1430    1429    1428    1427    1426    1425    1424    1423    1422    1421    1420    1419
1418    1417    1416    1415    1414    1413    1412    1411    1410    1409    1408    1407
1406    1405    1404    1403    1402    1401    1400    1399    1398    1397    1396    1395
1394    1393    1392    1391    1390    1389    1388    1387    1386    1385    1384    1383
1382    1381    1380    1379    1378    1377    1376    1375    1374    1373    1372    1371
```

1370	1369	1368	1367	1366	1365	1364	1363	1362	1361	1360	1359
1358	1357	1356	1355	1354	1353	1352	1351	1350	1349	1348	1347
1346	1345	1344	1343	1342	1341	1340	1339	1338	1337	1336	1335
1334	1333	1332	1331	1330	1329	1328	1327	1326	1325	1324	1323
1322	1321	1320	1319	1318	1317	1316	1315	1314	1313	1312	1311
1310	1309	1308	1307	1306	1305	1304	1303	1302	1301	1300	1299
1298	1297	1296	1295	1294	1293	1292	1291	1290	1289	1288	1287
1286	1285	1284	1283	1282	1281	1280	1279	1278	1277	1276	1275
1274	1273	1272	1271	1270	1269	1268	1267	1266	1265	1264	1263
1262	1261	1260	1259	1258	1257	1256	1255	1254	1253	1252	1251
1250	1249	1248	1247	1246	1245	1244	1243	1242	1241	1240	1239
1238	1237	1236	1235	1234	1233	1232	1231	1230	1229	1228	1227
1226	1225	1224	1223	1222	1221	1220	1219	1218	1217	1216	1215
1214	1213	1212	1211	1210	1209	1208	1207	1206	1205	1204	1203
1202	1201	1200	1199	1198	1197	1196	1195	1194	1193	1192	1191
1190	1189	1188	1187	1186	1185	1184	1183	1182	1181	1180	1179
1178	1177	1176	1175	1174	1173	1172	1171	1170	1169	1168	1167
1166	1165	1164	1163	1162	1161	1160	1159	1158	1157	1156	1155
1154	1153	1152	1151	1150	1149	1148	1147	1146	1145	1144	1143
1142	1141	1140	1139	1138	1137	1136	1135	1134	1133	1132	1131
1130	1129	1128	1127	1126	1125	1124	1123	1122	1121	1120	1119
1118	1117	1116	1115	1114	1113	1112	1111	1110	1109	1108	1107
1106	1105	1104	1103	1102	1101	1100	1099	1098	1097	1096	1095
1094	1093	1092	1091	1090	1089	1088	1087	1086	1085	1084	1083
1082	1081	1080	1079	1078	1077	1076	1075	1074	1073	1072	1071
1070	1069	1068	1067	1066	1065	1064	1063	1062	1061	1060	1059
1058	1057	1056	1055	1054	1053	1052	1051	1050	1049	1048	1047
1046	1045	1044	1043	1042	1041	1040	1039	1038	1037	1036	1035
1034	1033	1032	1031	1030	1029	1028	1027	1026	1025	1024	1023
1022	1021	1020	1019	1018	1017	1016	1015	1014	1013	1012	1011
1010	1009	1008	1007	1006	1005	1004	1003	1002	1001	1000	999
998	997	996	995	994	993	992	991	990	989	988	987
986	985	984	983	982	981	980	979	978	977	976	975
974	973	972	971	970	969	968	967	966	965	964	963
962	961	960	959	958	957	956	955	954	953	952	951
950	949	948	947	946	945	944	943	942	941	940	939
938	937	936	935	934	933	932	931	930	929	928	927
926	925	924	923	922	921	920	919	918	917	916	915
914	913	912	911	910	909	908	907	906	905	904	903
902	901	900	899	898	897	896	895	894	893	892	891
890	889	888	887	886	885	884	883	882	881	880	879
878	877	876	875	874	873	872	871	870	869	868	867
866	865	864	863	862	861	860	859	858	857	856	855
854	853	852	851	850	849	848	847	846	845	844	843
842	841	840	839	838	837	836	835	834	833	832	831
830	829	828	827	826	825	824	823	822	821	820	819
818	817	816	815	814	813	812	811	810	809	808	807
806	805	804	803	802	801	800	799	798	797	796	795
794	793	792	791	790	789	788	787	786	785	784	783
782	781	780	779	778	777	776	775	774	773	772	771
770	769	768	767	766	765	764	763	762	761	760	759
758	757	756	755	754	753	752	751	750	749	748	747
746	745	744	743	742	741	740	739	738	737	736	735
734	733	732	731	730	729	728	727	726	725	724	723
722	721	720	719	718	717	716	715	714	713	712	711
710	709	708	707	706	705	704	703	702	701	700	699
698	697	696	695	694	693	692	691	690	689	688	687

686	685	684	683	682	681	680	679	678	677	676	675
674	673	672	671	670	669	668	667	666	665	664	663
662	661	660	659	658	657	656	655	654	653	652	651
650	649	648	647	646	645	644	643	642	641	640	639
638	637	636	635	634	633	632	631	630	629	628	627
626	625	624	623	622	621	620	619	618	617	616	615
614	613	612	611	610	609	608	607	606	605	604	603
602	601	600	599	598	597	596	595	594	593	592	591
590	589	588	587	586	585	584	583	582	581	580	579
578	577	576	575	574	573	572	571	570	569	568	567
566	565	564	563	562	561	560	559	558	557	556	555
554	553	552	551	550	549	548	547	546	545	544	543
542	541	540	539	538	537	536	535	534	533	532	531
530	529	528	527	526	525	524	523	522	521	520	519
518	517	516	515	514	513	512	511	510	509	508	507
506	505	504	503	502	501	500	499	498	497	496	495
494	493	492	491	490	489	488	487	486	485	484	483
482	481	480	479	478	477	476	475	474	473	472	471
470	469	468	467	466	465	464	463	462	461	460	459
458	457	456	455	454	453	452	451	450	449	448	447
446	445	444	443	442	441	440	439	438	437	436	435
434	433	432	431	430	429	428	427	426	425	424	423
422	421	420	419	418	417	416	415	414	413	412	411
410	409	408	407	406	405	404	403	402	401	400	399
398	397	396	395	394	393	392	391	390	389	388	387
386	385	384	383	382	381	380	379	378	377	376	375
374	373	372	371	370	369	368	367	366	365	364	363
362	361	360	359	358	357	356	355	354	353	352	351
350	349	348	347	346	345	344	343	342	341	340	339
338	337	336	335	334	333	332	331	330	329	328	327
326	325	324	323	322	321	320	319	318	317	316	315
314	313	312	311	310	309	308	307	306	305	304	303
302	301	300	299	298	297	296	295	294	293	292	291
290	289	288	287	286	285	284	283	282	281	280	279
278	277	276	275	274	273	272	271	270	269	268	267
266	265	264	263	262	261	260	259	258	257	256	255
254	253	252	251	250	249	248	247	246	245	244	243
242	241	240	239	238	237	236	235	234	233	232	231
230	229	228	227	226	225	224	223	222	221	220	219
218	217	216	215	214	213	212	211	210	209	208	207
206	205	204	203	202	201	200	199	198	197	196	195
194	193	192	191	190	189	188	187	186	185	184	183
182	181	180	179	178	177	176	175	174	173	172	171
170	169	168	167	166	165	164	163	162	161	160	159
158	157	156	155	154	153	152	151	150	149	148	147
146	145	144	143	142	141	140	139	138	137	136	135
134	133	132	131	130	129	128	127	126	125	124	123
122	121	120	119	118	117	116	115	114	113	112	111
110	109	108	107	106	105	104	103	102	101	100	99
98	97	96	95	94	93	92	91	90	89	88	87
86	85	84	83	82	81	80	79	78	77	76	75
74	73	72	71	70	69	68	67	66	65	64	63
62	61	60	59	58	57	56	55	54	53	52	51
50	49	48	47	46	45	44	43	42	41	40	39
38	37	36	35	34	33	32	31	30	29	28	27
26	25	24	23	22	21	20	19	18	17	16	15
14	13	12	11	10	9	8	7	6	5	4	3

2	1	2392	2391	2390	2389	2388	2387	2386	2385	2384	2383
2382	2381	2380	2379	2378	2377	2376	2375	2374	2373	2372	2371
2370	2369	2368	2367	2366	2365	2364	2363	2362	2361	2360	2359
2358	2357	2356	2355	2354	2353	2352	2351	2350	2349	2348	2347
2346	2345	2344	2343	2342	2341	2340	2339	2338	2337	2336	2335
2334	2333	2332	2331	2330	2329	2328	2327	2326	2325	2324	2323
2322	2321	2320	2319	2318	2317	2316	2315	2314	2313	2312	2311
2310	2309	2308	2307	2306	2305	2304	2303	2302	2301	2300	2299
2298	2297	2296	2295	2294	2293	2292	2291	2290	2289	2288	2287
2286	2285	2284	2283	2282	2281	2280	2279	2278	2277	2276	2275
2274	2273	2272	2271	2270	2269	2268	2267	2266	2265	2264	2263
2262	2261	2260	2259	2258	2257	2256	2255	2254	2253	2252	2251
2250	2249	2248	2247	2246	2245	2244	2243	2242	2241	2240	2239
2238	2237	2236	2235	2234	2233	2232	2231	2230	2229	2228	2227
2226	2225	2224	2223	2222	2221	2220	2219	2218	2217	2216	2215
2214	2213	2212	2211	2210	2209	2208	2207	2206	2205	2204	2203
2202	2201	2200	2199	2198	2197	2196	2195	2194	2193	2192	2191
2190	2189	2188	2187	2186	2185	2184	2183	2182	2181	2180	2179
2178	2177	2176	2175	2174	2173	2172	2171	2170	2169	2168	2167
2166	2165	2164	2163	2162	2161	2160	2159	2158	2157	2156	2155
2154	2153	2152	2151	2150	2149	2148	2147	2146	2145	2144	2143
2142	2141	2140	2139	2138	2137	2136	2135	2134	2133	2132	2131
2130	2129	2128	2127	2126	2125	2124	2123	2122	2121	2120	2119
2118	2117	2116	2115	2114	2113	2112	2111	2110	2109	2108	2107
2106	2105	2104	2103	2102	2101	2100	2099	2098	2097	2096	2095
2094	2093	2092	2091	2090	2089	2088	2087	2086	2085	2084	2083
2082	2081	2080	2079	2078	2077	2076	2075	2074	2073	2072	2071
2070	2069	2068	2067	2066	2065	2064	2063	2062	2061	2060	2059
2058	2057	2056	2055	2054	2053	2052	2051	2050	2049	2048	2047
2046	2045	2044	2043	2042	2041	2040	2039	2038	2037	2036	2035
2034	2033	2032	2031	2030	2029	2028	2027	2026	2025	2024	2023
2022	2021	2020	2019	2018	2017	2016	2015	2014	2013	2012	2011
2010	2009	2008	2007	2006	2005	2004	2003	2002	2001	2000	1999
1998	1997	1996	1995	1994	1993	1992	1991	1990	1989	1988	1987
1986	1985	1984	1983	1982	1981	1980	1979	1978	1977	1976	1975
1974	1973	1972	1971	1970	1969	1968	1967	1966	1965	1964	1963
1962	1961	1960	1959	1958	1957	1956	1955	1954	1953	1952	1951
1950	1949	1948	1947	1946	1945	1944	1943	1942	1941	1940	1939
1938	1937	1936	1935	1934	1933	1932	1931	1930	1929	1928	1927
1926	1925	1924	1923	1922	1921	1920	1919	1918	1917	1916	1915
1914	1913	1912	1911	1910	1909	1908	1907	1906	1905	1904	1903
1902	1901	1900	1899	1898	1897	1896	1895	1894	1893	1892	1891
1890	1889	1888	1887	1886	1885	1884	1883	1882	1881	1880	1879
1878	1877	1876	1875	1874	1873	1872	1871	1870	1869	1868	1867
1866	1865	1864	1863	1862	1861	1860	1859	1858	1857	1856	1855
1854	1853	1852	1851	1850	1849	1848	1847	1846	1845	1844	1843
1842	1841	1840	1839	1838	1837	1836	1835	1834	1833	1832	1831
1830	1829	1828	1827	1826	1825	1824	1823	1822	1821	1820	1819
1818	1817	1816	1815	1814	1813	1812	1811	1810	1809	1808	1807
1806	1805	1804	1803	1802	1801	1800	1799	1798	1797	1796	1795
1794	1793	1792	1791	1790	1789	1788	1787	1786	1785	1784	1783
1782	1781	1780	1779	1778	1777	1776	1775	1774	1773	1772	1771
1770	1769	1768	1767	1766	1765	1764	1763	1762	1761	1760	1759
1758	1757	1756	1755	1754	1753	1752	1751	1750	1749	1748	1747
1746	1745	1744	1743	1742	1741	1740	1739	1738	1737	1736	1735
1734	1733	1732	1731	1730	1729	1728	1727	1726	1725	1724	1723
1722	1721	1720	1719	1718	1717	1716	1715	1714	1713	1712	1711

```
1710  1709  1708  1707  1706  1705  1704  1703  1702  1701  1700  1699
1698  1697  1696  1695  1694  1693  1692  1691  1690  1689  1688  1687
1686  1685  1684  1683  1682  1681  1680  1679  1678  1677  1676  1675
1674  1673  1672  1671  1670  1669  1668  1667  1666  1665  1664  1663
1662  1661  1660  1659  1658  1657  1656  1655  1654  1653  1652  1651
1650  1649  1648  1647  1646  1645  1644  1643  1642  1641  1640  1639
1638  1637  1636  1635  1634  1633  1632  1631  1630  1629  1628  1627
1626  1625  1624  1623  1622
```

C.2 SITB Population Members and their Paths

This section contains the genomes that will create the best known snakes in hypercubes of dimension 8, 9, and 10. For each dimension, the population genome is listed first, followed by the snake it produces. The produced snakes are logically equivalent to the best known.

D8 locus-based population member that will produce the longest snake of 99 nodes:

```
1   1   1   12  19  3   16  8   1   1   2   8   14  26  12  13  24  4
14  3   8   13  4   1   8   1   7   4   1   12  16  1   12  1   26  1
19  3   21  3   20  24  1   4   4   6   2   2   1   1   1   2   2   14
3   7   2   16  21  1   4   3   14  4   1   1   13  1   19  3   20  6
1   1   2   6   1   19  10  20  3   8   24  4   7   24  6   2   6   13
4   1   8   3   1   24  1   2   10  1   2   19  1   24  20  7   2   4
12  1   12  3   1   1   1   16  26  24  6   8   1   1   10  14  3   1
26  19  1   2   2   4   21  3   21  6   1   1   1   21  8   26  6   26
7   2   3   4   8   1   1   1   1   24  12  1   26  4   1   26  13  10
3   20  2   7   1   14  14  3   16  1   1   12  1   6   3   1   1   24
1   6   12  2   1   1   12  1   19  1   21  8   1   1   7   1   26  3
2   7   2   1   1   6   10  4   10  12  4   3   1   6   20  14  7   1
19  1   6   4   4   6   1   4   4   7   6   2   1   16  1   7   7   12
4   13  26  2   7   20  1   1   12  8   24  7   19  3   1   1   21  1
10  10  19  6
```

D8 99 node path produced by this population member:

```
1    2    3    6    11   10   9    24   21   44   53   60   57   40
89   88   73   74   75   70   91   38   219  198  203  202  201  216
209  48   49   50   51   46   211  238  239  242  241  244  141  132
129  130  159  158  163  190  179  178  175  170  167  186  185  188
181  172  173  84   77   68   65   66   95   34   223  194  193  196
221  36   29   100  97   112  113  114  115  110  107  150  139  138
137  152  153  156  229  252  249  250  231  26   103  122  121  124
117
```

D9 locus-based population member that will produce the longest snake of 191 nodes:

```
1   1   1   1   1   1   20  19  7   3   8   1   19  3   8   12  1   1
1   3   16  21  2   1   26  13  7   3   19  16  7   20  1   1   1   1
6   3   2   7   6   19  1   20  2   13  1   20  1   1   14  16  21  1
1   4   12  4   1   12  20  1   8   3   1   2   3   3   13  1   3   12
1   12  2   1   12  1   13  16  1   1   1   26  1   20  6   24  21  3
14  4   12  16  2   26  20  3   1   1   3   12  1   14  4   3   4   1
26  4   19  1   1   12  6   1   19  1   6   12  24  1   21  13  1   14
12  4   1   1   1   4   1   2   20  3   1   4   10  1   24  1   21  2
6   1   1   12  13  3   1   1   13  2   1   4   1   4   26  1   10  1
1   1   16  10  1   21  3   7   24  12  1   8   24  14  1   1   12  4
6   1   1   2   13  6   4   21  16  20  7   26  3   4   1   7   20  13
3   1   1   6   20  1   19  1   24  6   24  3   1   4   2   13  3   3
19  1   20  3   16  19  8   20  3   10  12  12  8   2   7   1   21  12
1   1   12  6   7   1   1   1   1   1   3   16  2   14  4   1   16  8
1   10  1   1   2   21  1   6   1   1   20  13  1   12  6   1   3   1
8   21  20  3   1   1   13  3   24  1   21  7   6   26  12  4   4   1
10  19  1   1   3   3   12  1   1   24  1   21  1   16  4   2   1   1
19  20  7   4   1   10  8   1   16  13  14  14  13  1   4   1   1   8
1   3   7   2   8   24  3   1   12  1   8   19  1   1   4   20  1   8
3   7   3   12  26  12  7   2   3   13  1   1   1   4   6   6   24  1
12  14  1   1   21  10  7   1   4   1   8   1   13  1   1   13  1   1
10  20  1   13  10  6   1   1   1   4   4   14  1   2   1   24  3   1
13  1   21  2   26  1   2   26  7   16  1   1   6   21  4   16  6   8
13  1   2   20  12  2   19  1   1   26  26  8   1   24  1   10  4   7
1   1   14  1   10  1   1   3   12  26  6   4   6   8   8   4   24  13
1   1   26  2   16  1   1   1   26  1   10  1   10  6   1   1   1   20
1   16  21  2   1   3   7   1   2   20  20  14  1   20  19  2   8   7
1   4   6   2   1   1   1   7   2   1   2   1   1   4   13  1   1   2
1   14  21  19  1   16  19  24
```

D9 191 node path produced by this population member:

```
1    2    3    6    5    12   9    24   23   18   19   20   45   52
49   50   55   54   43   38   35   34   33   40   89   72   73   76
85   108  107  118  119  122  103  98   99   100  125  116  113  112
81   82   83   78   67   70   187  182  183  178  177  180  173  164
163  162  167  218  215  210  211  206  195  196  253  244  241  242
247  250  249  200  201  204  213  236  235  230  155  156  133  140
137  152  151  146  147  142  131  130  129  160  225  288  287  354
355  356  381  372  369  368  337  338  339  334  323  322  321  320
305  306  311  310  299  294  291  292  301  276  275  270  259  262
261  268  265  280  297  296  345  360  377  378  375  374  363  364
341  332  325  444  389  396  393  408  407  402  403  398  387  386
385  416  417  420  429  436  433  434  439  438  427  422  423  474
487  488  505  456  457  460  469  492  491  502  503  498  497  496
465  466  467  462  451  452  509  484  483
```

D10 locus-based population member that will produce the longest snake of 371 nodes:

```
1   1   1   6   20  3   19  20  1   1   2   10  8   20  26  19  16  1
1   13  2   10  20  1   10  13  10  1   1   20  16  6   6   4   19  7
21  14  2   7   1   1   1   24  14  1   12  2   2   7   13  3   1   7
3   10  1   1   2   1   10  12  13  24  1   1   2   1   4   6   1   16
1   1   1   1   21  1   14  6   21  8   19  13  2   24  16  1   4   19
1   1   19  1   1   14  1   1   7   13  6   19  24  3   2   4   1   19
21  7   21  21  1   8   4   3   20  10  8   26  2   2   16  1   1   8
1   26  13  1   3   2   6   4   1   20  1   1   21  6   3   8   3   7
2   20  24  3   13  1   1   12  24  3   7   16  20  8   1   1   13  7
14  26  2   4   1   1   13  3   10  2   2   26  10  2   1   1   13  1
1   6   14  1   1   14  1   1   24  1   26  8   1   1   14  1   19  2
19  7   1   1   1   20  16  1   7   26  6   10  19  2   1   12  10  1
6   12  1   1   16  1   1   1   14  12  19  8   24  1   3   13  7   1
1   3   1   16  26  1   1   1   1   3   1   24  8   14  26  1   6   3
1   7   14  8   14  2   1   3   12  19  21  21  1   1   4   19  1   6
16  19  3   26  10  12  1   20  20  1   2   13  20  3   8   12  1   19
21  13  4   10  16  24  1   1   7   1   1   3   13  1   6   4   2   20
19  1   4   3   6   21  1   19  1   1   2   1   4   12  1   1   20  14
20  21  1   21  1   1   6   1   10  13  14  3   2   21  2   24  3   14
3   24  12  2   1   1   13  1   1   3   1   16  26  3   24  12  1   1
14  1   10  2   8   21  4   19  1   1   3   1   10  4   14  24  4   4
1   1   7   26  4   20  3   3   4   21  1   24  3   8   1   1   20  1
20  10  20  12  19  12  10  1   14  4   8   4   24  21  26  20  2   13
26  21  20  21  1   10  10  14  1   1   3   1   21  1   2   14  4   7
1   1   6   1   20  16  6   4   1   26  1   1   19  1   6   20  1   1
12  1   12  20  10  13  1   1   1   21  20  1   19  21  19  10  26  13
6   21  26  1   21  14  1   1   20  1   2   8   1   20  2   7   21  16
1   1   20  1   1   4   26  19  7   3   1   1   1   8   1   19  21  13
1   16  19  1   1   3   14  21  6   8   1   1   20  10  24  7   1   1
1   20  1   14  26  19  13  1   1   24  1   21  19  1   8   13  1   2
3   26  19  8   16  6   20  4   19  14  1   1   10  1   1   8   1   1
19  4   2   13  13  1   7   20  2   16  1   1   1   1   3   1   14  2
1   1   2   24  2   12  1   19  1   1   10  1   2   12  12  3   2   20
26  13  3   13  8   26  1   12  1   1   7   2   4   3   4   19  1   24
24  20  4   1   20  1   10  12  6   6   4   19  1   1   26  1   13  4
13  1   4   21  1   1   16  21  21  13  1   1   10  1   7   10  1   6
1   1   13  8   16  14  2   6   1   2   6   26  14  3   14  1   20  7
2   2   12  1   8   21  7   1   2   1   16  2   1   1   4   20  2   26
1   4   1   7   1   1   26  1   8   3   8   4   1   7   1   1   14  16
2   8   1   1   2   4   21  21  1   2   1   1   2   1   8   2   3   13
24  13  10  24  4   20  1   10  19  8   16  7   2   14  26  1   4   12
14  26  13  13  16  24  16  8   4   26  19  1   14  3   1   1   1   24
1   24  3   20  19  19  19  2   1   7   4   1   16  4   4   26  4   8
6   24  1   1   1   1   1   7   13  13  8   7   12  20  1   7   3   1
12  6   26  2   2   19  4   7   13  24  8   7   20  20  1   1   6   1
1   16  19  7   26  1   2   19  6   1   26  3   8   10  1   20  1   1
7   1   10  2   6   1   24  20  10  13  1   16  1   1   4   6   12  1
4   6   4   2   1   10  3   3   14  24  12  13  1   1   13  3   21  1
16  12  7   1   19  16  1   1   21  3   16  10  4   2   1   19  1   1
1   4   6   1   7   14  1   24  1   1   20  21  26  6   10  2   1   8
8   4   1   13  1   1   2   1   8   24  21  13  12  4   4   12  3   3
1   14  12  4   7   3   4   2   16  2   10  24  1   4   6   21  1   1
2   2   16  2   1   14  1   7   1   1   14  1   13  4   6   13  1   21
1   1   12  1   13  13  1   1   2   1   6   14  1   1   1   1   1   10
10  1   26  16  6   1   1   1   1   26  6   1   10  3   1   13  13  1
```

```
1   3   1   21  3   1   13  2   20  3   4   4   3   12  26  2   21  1
1   1   1   26  4   16  26  7   1   1   12  1   1   6   26  2
```

D10 371 node path produced by this population member:

```
1     2     3     6     11    10    9     24    21    28    29    36
33    48    49    52    53    60    57    58    39    42    43    46
19    18    111   114   115   78    75    74    73    88    85    92
91    94    95    66    65    68    125   124   121   104   97    160
159   158   131   190   187   188   185   168   167   170   175   178
177   180   173   148   237   240   241   242   243   206   203   202
201   216   213   220   219   222   223   194   193   196   253   252
245   140   137   138   135   250   231   234   235   150   107   406
491   490   487   488   481   496   497   498   499   462   459   458
457   472   469   476   475   478   479   450   449   452   509   508
501   396   393   394   391   386   387   446   443   444   441   424
423   426   431   434   433   436   429   404   413   356   285   284
277   280   265   266   267   374   379   380   377   360   359   362
367   370   369   372   269   308   305   304   337   352   321   322
327   330   329   332   341   348   347   350   339   302   299   298
295   296   313   316   315   318   259   258   287   738   799   770
771   830   827   828   825   808   807   810   811   814   851   862
859   860   853   844   841   842   839   834   833   864   849   816
817   820   781   884   881   882   879   874   871   872   889   892
891   886   779   778   777   792   789   796   797   868   925   924
917   908   905   906   903   898   899   958   955   956   953   936
935   938   943   946   945   948   941   980   981   984   969   970
971   974   1011  1010  1009  1008  993   1000  1017  1020  1021  964
961   962   991   990   987   998   539   540   533   536   521   522
523   630   635   636   633   616   615   618   623   626   625   628
525   564   561   560   593   608   577   578   583   586   585   588
597   604   603   606   611   670   667   662   683   694   699   700
697   680   679   674   687   690   689   692   685   676   733   736
705   706   711   714   713   716   757   764   765   644   641   642
647   650   649   664   657   752   753   754   755   750   531   558
555   554   551   552   569   572   571   574   515   514   513
```

D. Additional Examples

Following is another example of single point phenotype crossover in a D5 hypercube. This is similar to the example in Section 7.3.2 and should help in understanding of this operator as it is applied to the SITB.

Initial Population Members and Snake

Index values:
```
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
```

Selected population members:
```
10 13 12  7  8  1 21 10  7 10  8 24 21  2 12  8
13  6 19 10 20  8 14  2  1 24  1 20  1 12  7  6

 7 20 14 14  7  2 19  8  1 24 26 20 13 26 10  7
10 14 20  1 26 10 24  1  1  2  8 14  3  2 20 14
```

Selected snakes:
```
1   2   3   6   5  12 * 13  20  17  24  25  26
1   2   3   6   5  12 * 13  20  19  18  23  24  25
```

Final Population Members and Snake

Index values:
```
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
```

Reconstructed members:
```
10 13 12  7  8  1 21 10  7 10  8 24 13  2 12  8
13 14 20  1 20  8 24  1  1 24  1 20  1 12  7  6

 7 20 14 14  7  2 19  8  1 24 26 20 21 26 10  7
13 14 20 10 26 10 24  2  1 24  8 14  3  2 20 14
```

Reconstructed snakes:
```
 1   2   3   6   5  12 * 13  20  19  18  23  24  25
 1   2   3   6   5  12 * 13  20  17  24  25  26
```

An * indicates crossover point.