Problem Structure and Evolutionary Algorithm Difficulty

Susan Khor Lay Choo

A Thesis

in

The Department

of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy (Ph.D.) at
Concordia University
Montreal, Quebec, Canada

Summer, 2008

Your file Votre référence
ISBN: 978-0-494-45680-4
Our file Notre référence
ISBN: 978-0-494-45680-4

ABSTRACT

Problem Structure and Evolutionary Algorithm Difficulty

Susan Khor Lay Choo, Ph. D.
Concordia University, 2008

This study reexamines the Hierarchical-If-And-Only-If (HIFF) problem (of which there are two versions – discrete and continuous) introduced by Watson et al. (1998) to distinguish the evolutionary capability of hill climbers from genetic algorithms. It finds that it is possible to relax some of the conditions stipulated for the structure of the HIFF problem without destroying the ability of the modified problem to distinguish the evolutionary capability of hill climbers from genetic algorithms. In particular, full inter-dependency between problem variables is not necessary, i.e. there need not be an *iff* constraint between every distinct variable pair.

In addition, by reducing variable-to-variable inter-dependencies, the modified HIFF problem becomes more solvable by upGA, a genetic algorithm which does not explicitly maintain population genetic diversity. This is because the modified HIFF problem (called HIFF-M) permits mutation to be effective in a genetic algorithm. The maintenance of sufficient population genetic diversity through means other than mutation was a key component of genetic algorithm success in Watson's work on the HIFF problem.

The modular structure of the HIFF problem, along with the inter-dependencies between modules, is credited for the ability of the HIFF problem to distinguish the evolutionary capability of hill climbers from genetic algorithms. However, when the genetic algorithm in question is upGA, this study finds that degree distribution type is a better indicator than modularity of when a problem will be easier for a genetic algorithm than a hill climber to solve. In general, problems with real-world type degree distributions were more easily solved by upGA than by the random mutation hill climber or the macro-mutation hill climber. Suggestions are made for further research in this direction.

This study also examines the relationship between hierarchical levels in different HIFF problems. It finds differences in the speed at which levels can optimize relative to one another, the degree of inter-level inter-dependency, and the degree of top-down inter-level conflict. The consequences of these differences for evolutionary algorithm difficulty are discussed.

Acknowledgements

Many thanks to everyone who contributed to the completion of this thesis, especially to my thesis supervisor for funding a large part of this work.

Table of Contents

List of Figures

List of Tables

1. Introduction

Evolutionary algorithms are a class of algorithms that use the generate-and-test method for solving problems. New candidate solutions are generated by making random modifications to existing candidate solutions, and the quality of new candidate solutions are evaluated by a *fitness function.* A problem is solved when a solution is found with the desired level of fitness.

By generating and testing new candidate solutions, an evolutionary algorithm in effect explores the space of possible solutions in search of satisfactory solutions. Thus evolutionary algorithms are often described as *search* algorithms, and the problems evolutionary algorithms help solve, as search problems (section 2.1). Optimization, combinatorial and decision problems fit into this search paradigm nicely.

$f: \{0, 1\}^2 \rightarrow \mathbf{Z}$ is an example of a fitness function that takes two binary variables x and y, and returns an integer representing the fitness of the binary string (solution) whose elements are x and y.

x	y	$f(x, y)$
0	0	1
0	1	0
1	0	0
1	1	3

A simple fitness function

If the problem is to maximize f, then the string '11' is the best or *optimal solution.* If the problem is to find a combination of x and y such that $f(x, y) > 0$, then '00' and '11' are both optimal solutions. If the problem is to decide whether there exists a combination of x and y such that $f(x, y) = 1$, then the answer is yes if '00' is found. The *variables* in all these problems are x and y.

The work performed by an evolutionary algorithm to solve a problem is typically measured in terms of number of fitness function evaluations. A problem is more difficult for evolutionary algorithm *X* than evolutionary algorithm *Y* to solve, if *X* does more work than *Y* to solve the problem, or after having done the same amount of work, *X* is less successful than *Y* at solving the problem. In this case, evolutionary algorithm *X* has more difficulty solving the problem than *Y*.

The main goal of this dissertation is to understand what makes a problem more difficult for one evolutionary algorithm to solve as opposed another. Several types of evolutionary algorithms exists (section 2.2.2). The evolutionary algorithms of interest in this dissertation are *hill climbers* and *genetic algorithms*.

Hill climbers and genetic algorithms occupy opposite ends of the evolutionary algorithm complexity spectrum, in the sense that hill climbers (e.g. Figure 2.3) use only one variation operator (mutation), and store only one solution in memory at all times (single-individual population); while genetic algorithms (e.g. Figure 2.4) typically use more than one variation operator (mutation and crossover), and store more than one solution in memory at all times (multi-individual population).

Given these superficial syntactic differences, when does a genetic algorithm outperform a hill climber in terms of problem solving? This was the question Mitchell, et al. attempted to answer in 1992 with the *Royal-Roads* problem (section 2.3.3). The Royal-Roads was designed to epitomize the *Building-block Hypothesis* (section 2.2.2) at work in a genetic algorithm. A building-block is a highly fit combination of problem variables. The Building-block Hypothesis supposes that a genetic algorithm solves a problem incrementally, by constructing higher-order building-blocks from lower-order building-blocks in a step-like or hierarchical manner (section 3.3.3). But the Royal-Road problem was subsequently found to be unsatisfactory (section 2.4.3).

For the sake of brevity, the problem of identifying when a genetic algorithm outperforms a hill climber is referred to in this dissertation as *the HC < GA problem* (section 2.4). A solution to the HC < GA problem takes the form of an abstract problem specially designed to highlight aspects of the problem which makes it easier for genetic algorithms than hill climbers to solve.

Later in 1998, a solution to the HC < GA problem was proposed by Watson, et al. in the form of the *Hierarchical-If-And-Only-If* (HIFF) problem (chapter 3). The HIFF problem comes in two versions – discrete and continuous. The right understanding of *modularity*, according to Watson (2006, section 4.2), is the key to understanding what makes the HIFF problem more difficult for hill climbers than genetic algorithms to solve. Modularity is to be understood in terms of subsets of problem variables that are not only inter-dependent on other problem variables within their own respective subsets, but also inter-dependent on problem variables outside their own respective subsets (section 3.3.1).

A problem variable x is inter-dependent on another problem variable y if the effect x has on fitness of a solution depends also on the value of y, and vice versa. Such inter-dependencies between problem variables are also referred to as *epistasis* (section 2.5.1). Consider the simple fitness function, $f\colon \{0, 1\}^2 \rightarrow \mathbf{Z}$. The fitness values are calculated by awarding zero fitness point to the value '0', one fitness point to the value '1', one fitness point if x and y have the same value, and minus one fitness point if x and y do not have the same value. The last two reward conditions create inter-dependencies between the x and y variables.

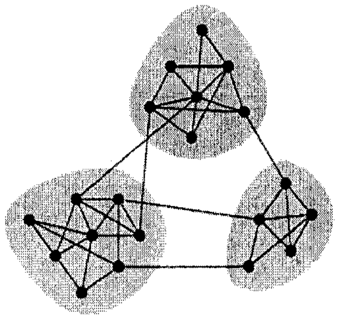
x	y	Calculation for the simple fitness function	$f(x, y)$
0	0	0 + 0 + 1	1
0	1	0 + 1 - 1	0
1	0	1 + 0 - 1	0
1	1	1 + 1 + 1	3

Inter-dependency between variables from different subsets was the missing ingredient in the Royal-Road problem. Inter-dependency between variables from different subsets introduces *non-separability* into a problem, and creates *frustration* for both hill climbers and genetic algorithms (section 3.3.2). Frustration is a common source of problem difficulty for evolutionary algorithms (section 2.5.2).

However, problem difficulty of HIFF is more severe for hill climbers than genetic algorithms because unlike hill climbers, genetic algorithms are able to swap subsets of variables between solutions due to their use of the crossover or recombination operator. Thus, the HC < GA problem also addresses indirectly the utility of crossover over mutation, or in Watson's terminology, compositional evolution over gradual evolution.

But for there to be identifiable subsets of variables, i.e. *modules*, and inter-dependencies between subsets; the inter-dependencies between variables from the same subset need to be much stronger relative to the inter-dependencies between variables belonging to different subsets; or alternatively there needs to be many more inter-dependencies between variables from the same subset than between variables belonging to different subsets.

In the following figure (excerpted from Newman, 2006), let the points represent problem variables, and the lines represent inter-dependencies between variables. The three identifiable subsets of variables or modules in the figure are shaded.



To exemplify Watson's modularity argument, the HIFF problem with inter-dependencies as those represented by the following matrix was created. According to the inter-dependency matrix (Part I) below, there are inter-dependencies between every distinct variable pair, but some inter-dependencies are stronger (larger valued entries) than others, and the "the large values are diagonalizable" (Watson, 2006 p.14).

	1	2	3	4	5	6	7	8
1	-	16	4	4	1	1	1	1
2	16	-	4	4	1	1	1	1
3	4	4	-	16	1	1	1	1
4	4	4	16	-	1	1	1	1
5	1	1	1	1	-	16	4	4
6	1	1	1	1	16	-	4	4
7	1	1	1	1	4	4	-	16
8	1	1	1	1	4	4	16	-

Inter-dependency matrix

Motivation

Watson (2006, p. 14) also says, referring to the matrix above: "But significantly, the values of the matrix that are off the diagonal, representing the strength of dependencies of variables across modules, are nonzero." This sentence served as catalyst for the research in this dissertation because it was not accompanied with further explanation with regards to its necessity.

Research objective

Thus, the primary research objective of this dissertation is to determine whether it is necessary for the entries in a HIFF inter-dependency matrix which are off the main diagonal to be nonzero. It is not necessary if the resulting HIFF problem is still a solution to the HC < GA problem.

Part I

To meet the above research objective, Part I introduces two new HIFF problems, HIFF-II and HIFF-M, which have off diagonal zero entries in their inter-dependency matrices. In addition, all non-zero entries in HIFF-M's inter-dependency matrix have the same value.

Part II

Part II investigates how HIFF-M fares on the HC < GA problem, and compares its performance with the HIFF-D problem, which is the discrete version of the HIFF problem introduced by Watson.

The results from Part II confirm via the HIFF-M problem that the inter-dependency matrix of a HIFF problem can have off diagonal zero entries and uniform valued non-zero entries, and still be a solution to the HC < GA problem. Furthermore, HIFF-M is a different kind of solution to the HC < GA problem than HIFF-D (or HIFF-C and HIFF-II) because HIFF-M is more easily solved than HIFF-D by upGA, a genetic algorithm that does not explicitly manage population diversity and relies on mutation to recover loss alleles.

Part III

All four HIFF problems defined in Part I (HIFF-C, HIFF-D, HIFF-II and HIFF-M) have the same amount of modularity for problems of the same size, but produce different upGA success rates in Part II, and thus distinguish the problem solving ability of hill climbers from genetic algorithms to varying degrees. This leads Part III to question whether modularity is still the determining factor to what makes HIFF problems more difficult for hill climbers than genetic algorithms to solve, and to find an alternative factor if the question is answered in the negative. This is the second research objective.

The results in Part III confirm that modularity correlates strongly with evolutionary difficulty for hill climbers, but when the genetic algorithm relies on mutation for population diversification as does upGA, degree distribution type emerged as a better indicator than modularity, epistasis or network transitivity of when a problem is more difficult for hill climbers than genetic algorithms to solve. Problems with strong modular structure are more likely to be difficult for hill climbers to solve, but may also be difficult for genetic algorithms to solve due to premature loss of population diversity.

Although the structure of the HIFF-M problem has the same amount of modularity as the other HIFF problems introduced in Part I (HIFF-C, HIFF-D and HIFF-II), its degree distribution is markedly distinct from that of HIFF-C, HIFF-D and HIFF-II. HIFF-M's degree distribution is more similar than that of HIFF-C, HIFF-D and HIFF-II to degree distributions in the real-world.

With this find, a new connection is made between problem structure and evolutionary algorithm difficulty. Problems with real-world type degree distributions are found to be more easily solved by genetic algorithms than hill climbers in general.

2. Background: Evolutionary Algorithms

This chapter covers the necessary background material on Evolutionary Algorithms, a class of search algorithms. It begins by describing the problem of search from a direct point-to-point perspective and search algorithms using solution-neutral terminology. Special attention is paid to Hill Climbers (HC) and Genetic Algorithms (GA), two types of evolutionary algorithms featured in this dissertation. Finally, the HC < GA problem investigated in this thesis is explained, followed by an overview of related work.

2.1 Basic search terminology

Informally, a search problem involves finding one or more elements of an *object* or *solution space* O, which meet some criteria. A search algorithm seeks to solve a search problem (which may be optimization or combinatorial in nature) by first representing elements of the given object space in a form it can process, and then organizing the elements of the resulting *representation space* R, into elements of the *search space* S in which the search algorithm will search. The elements of S called *search point*s are multi-sets of the elements of R (Jones, 1995). For example, we will see in the next section that the relationship between R and S in a hill climbing algorithm is one-to-one whereas it is many-to-one in a genetic algorithm.

The relationship between O and R is one aspect of the representation problem. At minimum, the elements of R must be able to represent all the elements of O. If R is a larger set than O, some elements in R may be infeasible (they do not represent any element in O), or the mapping between O and R is one-to-many. The other part of the representation problem is choosing the function to do the transformation between the elements of *O* and *R*. Using the wrong encoding can make a problem more difficult for an EA to solve (Rothlauf, 2002). These two aspects of the representation problem influence the distances a search algorithm can expect to traverse between elements of *O* in the search space. Figure 2.1 illustrates the relationships amongst the *O*, *R* and *S* sets, and helps with the subsequent discussion on fitness landscapes.

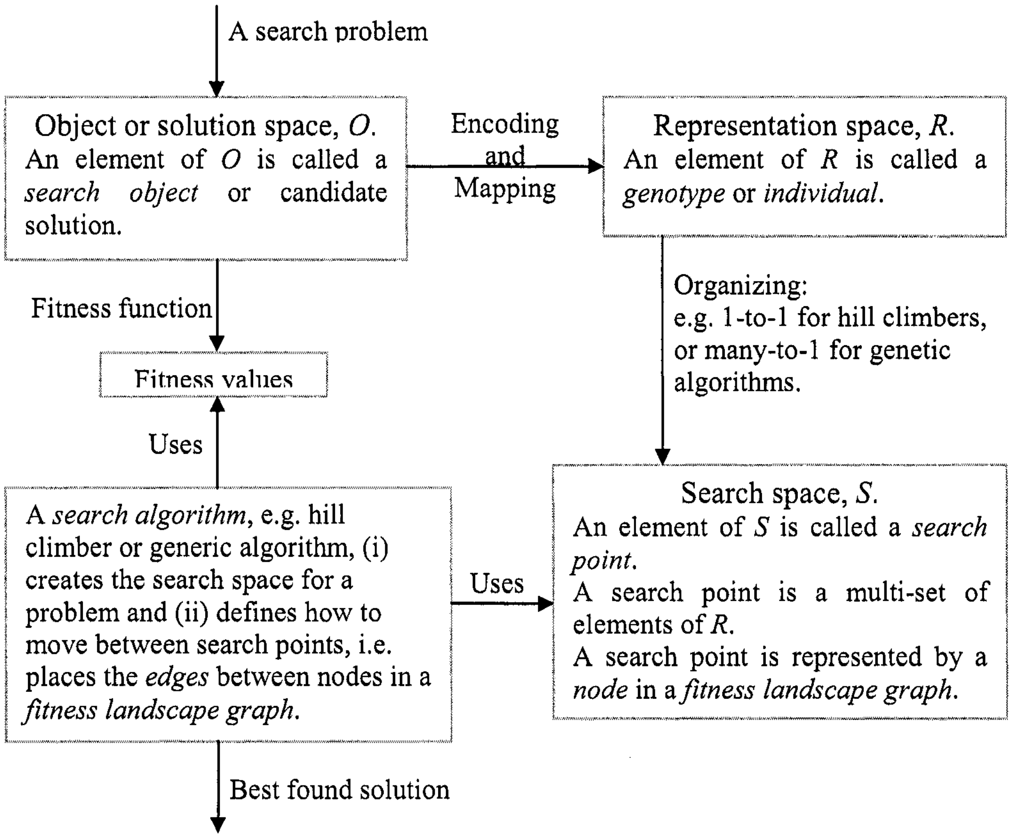

A search problem
Object or solution space, *O*.
An element of *O* is called a *search object* or candidate solution.
Encoding and Mapping
Representation space, *R*.
An element of *R* is called a *genotype* or *individual.*
Fitness function
Fitness values
Organizing:
e.g. 1-to-1 for hill climbers, or many-to-1 for genetic algorithms.
Uses
A *search algorithm*, e.g. hill climber or generic algorithm, (i) creates the search space for a problem and (ii) defines how to move between search points, i.e. places the *edges* between nodes in a *fitness landscape graph.*
Uses
Search space, *S*.
An element of *S* is called a *search point.*
A search point is a multi-set of elements of *R*.
A search point is represented by a *node* in a *fitness landscape graph.*
Best found solution

Figure 2.1 Relationships amongst the O, R and S sets

Each search object (element of *O*) needs to have a unique fitness value. A fitness value is an assessment of quality of how well a search object meets some criteria at the point in time when the measurement is taken. The fitness value of a search object may vary with some criteria external to the search object. It is common for fitness values to come from the set of real numbers **R** and to use the natural total order of real numbers to rank search objects by their fitness values.

Fitness functions are used to quantify the quality of a search object with respect to some criteria. A *fitness function* F maps O to a set of fitness values. If F is static, the fitness values are said to be *objective.* Otherwise F is dynamic, and the fitness of search objects depends also on factors external to the search objects. For example, a search object may increase in fitness simply because it appears, through its representation, to a search algorithm less frequently during the course of a search. Or, in the case of more subjective qualities, the quality of search objects may vary with the sample of observers used to make the fitness evaluation.

In this dissertation, the search objects are binary ($\{0, 1\}$) strings, $O = R$, and F is static and maps O to **R**.

2.1.1 Fitness landscape

A search algorithm is often described as moving through a fitness (adaptive) landscape (Wright, 1967). A *fitness landscape* comprises a search space S and a fitness function F for the search objects making up the search space, and a search algorithm defining how the search space can be traversed. A natural way to accommodate the multi-dimensionality of a search space is to represent a fitness landscape as a directed graph (V, E) (Jones, 1995). Each node (vertex) in V represents a search point and may be labeled with the corresponding fitness value. A directed edge is placed between two nodes in V if there is some non-zero transition probability that the search algorithm moves from the node at the source of the edge to the node at the sink of the edge. Each edge in E is labeled with its transition probability and all out-edges of a node must sum to 1.0. From this description, one can see how different search algorithms can have different fitness landscapes for the same fitness function and even for the same search space.

Different search algorithms can produce different *E*s (edge sets) for the same fitness function and even for the same search space because (i) their search strategies are different, and/or (ii) they move differently within the search space (the topology of the search space is defined differently). For an edge (u, v) to exist in the fitness landscape graph for a search algorithm, the search algorithm must first be able to move between the search points represented by u and v. The ability of a search algorithm to move from one search point to another search point in its search space depends on the specifics (operators) of the search algorithm. Each search operator has a *radius* or *maximum reach* r which determines the size of the largest transformation it can make on any search point. For example, in a search space where the elements are binary strings, a bit-flip operator with a radius of two can produce or reach all strings that are at most two Hamming distance from any search point. The Hamming Distance (HD) between two strings of equal length is the number of locations where the strings are dissimilar from each other. For example, '0111' and '1101' are two HD from each other. The minimum reach of a search operator is similarly defined.

The ability of a search algorithm to move from one node to another node in its fitness landscape depends on the specifics (operators) of the search algorithm and its search strategy which takes fitness of search objects into account. There may be more than one operator in a search algorithm. Hence, it is possible to construct more than one fitness landscape for a search algorithm. This is the 'one operator one (fitness) landscape' notion suggested by Jones (1995).

The set of nodes *directly* reachable from node v in a fitness landscape is the *neighbourhood* of v denoted $N(v)$. If the neighbours of v are all as fit as v, v and its neighbours, $v \cup N(v)$, form a *plateau*. If $N(v)$ is empty, or v is at least as fit as all nodes in $N(v)$, v is a *local optimum*. If v is a local optimum and $F(v)$ is optimal, that is v has the minimum (maximum) fitness for a minimization (maximization) problem, v is a *global optimum*. Each local optimum has its own *basin of attraction*. The basin of attraction of a local optimum is the set of nodes reachable from v while traversing the edges of the fitness landscape graph in the reverse direction. A node may belong to more than one basin of attraction. A *saddle node* is a node on a path between two local optima u and v (a saddle node resides in both of their basins of attraction) that enables a search algorithm to move from u to v with minimal decrease in fitness. In a minimization (maximization) problem, a decrease in fitness is an increase (decrease) in fitness value. The magnitude of the minimal fitness decrease is the *fitness barrier* enclosing u.

To circumvent a non-zero fitness barrier, one could imagine an edge between local optimum u and a node w that is fitter than or as fit as u. Since it is an imaginary edge, we need some operator-neutral way of measuring its length. (All non-imaginary edges in a fitness landscape have a length of one in the unit of the relevant search operator.) In this dissertation (and following Watson), Hamming distance is used to measure the length of imaginary edges. The shortest imaginary edge spans a *fitness saddle*, and its length is the width of the fitness saddle.

2.2 Search algorithms

2.2.1 Random search algorithm

A random search algorithm does a random walk between points in a search space. All points are equally fit in the eyes of a random searcher. A (non-random) search algorithm is more discerning about how it moves. Fitter search points are commonly favoured, but this depends on the search strategy and the assumptions of a search algorithm. For example, if the assumption is that fit search points lead to the discovery of even fitter search points, the search strategy may be a greedy one, i.e. always and only move to a fittest point found so far.

2.2.2 Evolutionary algorithms

Evolutionary algorithms are a class of search algorithms that mimic the trial and error process of biological evolution by natural selection and random variation to solve problems. Figure 2.2 depicts an idealized view of biological evolution. Biological individuals compete with each other for limited resources. Those that are successful in this competition (this is natural selection) are more likely to produce offspring. But reproduction is error-prone. Hence, offspring are likely to be variants of their parents. These random variations may or may not be beneficial to the survival of the offspring.

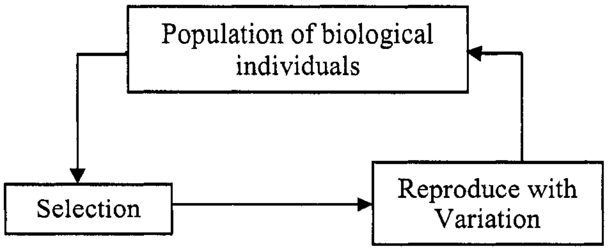


Figure 2.2 A simplified view of biological evolution

The three classic types of evolutionary algorithms are evolutionary programming (EP) (Fogel et al., 1966), evolution strategies (ES) (Rechenberg, 1973) and genetic algorithms (GA) (Holland, 1975). Nevertheless, as each approach developed, there has been healthy cross-pollination of ideas from one camp to another. Genetic Programming (GP) (Koza, 1992), is another well-known type of evolutionary algorithm (EA). A brief history of the development of evolutionary algorithms as a field of study can be found in de Jong (2006). Spears (1998) gives a brief comparison of the three classic evolutionary algorithms.

Evolutionary algorithms are stochastic but they are not random search algorithms (section 2.2.1). The heuristic an evolutionary algorithm (EA) uses to navigate its search space is intimately tied with the fitness function *F*. In contrast, a random searcher picks search points uniformly at random from *S* until an optimal search object is found.

What makes evolutionary algorithms evolutionary in the Darwinian sense is their employment of an inheritance and variation mechanism that is modeled on the biological transfer of genetic material from one generation to the next, and a selection mechanism that is modeled on natural selection. The inheritance and variation mechanism are the operators that enable an EA to move in its search space in a less random manner. The variation mechanism is commonly implemented as mutation and/or crossover operators. The inheritance mechanism is simple copy or reproduction and it prevents an EA from doing random moves. The selection mechanism is the search strategy of an EA. It directs the movement of an EA.

An EA is run until one or more optimal solutions are found or some criterion is met. One possible criterion is the exhaustion of the allocated number of iterations, generations or function evaluations. Alternatively, an EA could continue until no significant improvement is detected.

The term *genotype* is commonly used by evolutionary algorithms to refer to an element of the representation space *R*. An EA *genotype* comprises distinct locations or sites called *genes*. Each gene can hold one of a number of values called *alleles*. The set of possible values for a gene is the gene's *alphabet*. Each gene may have its own alphabet or there may be a common alphabet for all genes.

Problem vs. Solution Terminology

Two sets of terms are used in this dissertation (where possible and natural to do so) to distinguish between problem definition and solution description. When defining the problem, the terms 'string' and 'binary variable' or 'variable' are used; and when speaking from the solution (evolutionary algorithm) perspective, the terms 'genotype' and 'gene' or 'site' are used.

Each element in a string is a binary variable or *variable*. x is a binary variable if the value that can be held by or assigned to x come from the set $\{0, 1\}$ only. All variables are binary in this dissertation, so binary variables will be referred to simply as variables. In evolutionary algorithms, binary strings are represented as (linear) genotypes, and elements of a string or variables correspond to genes.

Hill Climbers

Two evolutionary algorithms relevant to this dissertation are genetic algorithms and hill climbers. Hill climbers are a most basic kind of EA. They are single-individual (the cardinality of the multi-set at each search point is exactly one) evolutionary algorithms and use only one variation operator – mutation.

A variety of hill climbers exists and there are several ways to classify them including how mutation is implemented, the number of mutants produced in each step, how the next search point is chosen from the pool of genotypes made up by a parent and its offspring, whether side stepping is allowed and whether there is a chance to restart a climb from another search point. Hill climbers in this dissertation use random- (RMHC in section 5.1.1), flip- (FMHC in section 3.3.3) or macro-mutation (MMHC in section 5.1.2), produce only one mutant in each step, do not restart, and are >= which means they move to fitter points and side step to equally fit points.

The outline for a generic hill climber is given in Figure 2.3. In evolution strategies (ES) parlance, this HC is a (1+1) EA or EA (1+1) since an individual is selected from the pool made up of the parent and its offspring.

Step	**A Generic Hill Climber**
1.	Create an initial (possibly random) genotype p.
	Calculate $F(p)$, the fitness of p.
2.	While stopping criteria not met
3.	Create a mutant genotype m by first reproducing p and then mutating the copy.
	Calculate $F(m)$.
4.	Replace p with m if $F(m)$ >= $F(p)$.
	Update variables for stopping criteria.
	End while

Figure 2.3 A Generic Hill Climber

Genetic Algorithms

Genetic algorithms are multi-individual (the cardinality of the multi-set at each search point is more than one) evolutionary algorithms. The distinguishing algorithmic feature of genetic algorithms from other evolutionary algorithms is their reliance on the crossover or the recombination operator to evolve or find good solutions.

```
Step   The Simple Genetic Algorithm (SGA)
1.     Create a population P of initial (possibly random) individuals. P is a
       multi-set. PS is the size of P.
2.     While stopping criteria not met
           For each individual p in P, calculate the fitness of p, F(p).
3.         Create a population C of offspring individuals as follows:
           While size of C < PS
4.             Select with replacement, from P, two individuals p1 and p2.
               The probability of selecting an individual is proportionate to its
               fitness.
5.             Create two offspring c1 and c2 by first reproducing p1 and p2.
6.             With probability $P_c$, crossover c1 with c2.
7.             With probability $P_m$, mutate c1 and c2.
8.             Add c1 and c2 to C.
           End while.
9.         Replace P with C.
           Update variables for stopping criteria.
       End while.
```

Figure 2.4 The Simple Genetic Algorithm

Figure 2.4 outlines the steps for the simple genetic algorithm (SGA). The SGA is also known as the canonical GA. Versions of the SGA can be found in evolutionary computation textbooks and tutorials, e.g. Goldberg (1989), Whitley (1993), Mitchell (1998), Eiben and Smith (2003) and de Jong (2006). Nevertheless, the common characteristics of all SGA versions are binary string representation, multi-individual search points, fitness-proportionate (parent) selection or viability selection, multiple variation operators (commonly one-point crossover and bit-flip mutation), a heavier reliance on crossover than other variation operators, and generational replacement.

In some SGA versions, in place of steps 4 to 8 in Figure 2.4, an intermediate or mating population of PS individuals is first created (Whitley, 1993). This intermediate population comprises true copies of selected parent individuals. Then individuals from the intermediate population are either paired up in sequence of their creation or at random without replacement, to produce offspring via crossover and mutation. In the formal model of the SGA developed by Vose and Liepins (1991), one of the recombinants chosen at random, is discarded.

Reasons for the use of binary string representation are discussed in Goldberg (1989) and Whitley (1993). The idea behind fitness-proportionate (parent) selection is for genotypes with above average fitness in the current generation to produce on average more offspring than genotypes with below average fitness in the current generation. Fitness-proportionate (parent) selection can be implemented using the 'biased roulette-wheel sampling' technique (Goldberg, 1989) or using a more direct approach, the 'remainder stochastic sampling' technique (Whitley, 1993) combined with 'Stochastic Universal Sampling' (Baker, 1987) to reduce sampling variance. In evolution strategies (ES) terminology and using the labels in Figure 2.4, the SGA is a (P, C) EA or EA (P, C) since the offspring generation replaces the parent generation entirely.

Compared with 'non-simple' or optimizing genetic algorithms, the SGA adheres more closely to Holland's blueprint for 'adaptive plans'. A (canonical) GA is envisioned to work as follows (Holland 1992, p.96):

> "We will see that $\mathcal{B}(t)$ [population at time *t*] is used basically as a pool of schemata. ... Past history is recorded in terms of the ranking (number of instances) of each schema in $\mathcal{B}(t)$ ... From this point of view *crossing-over* acts to generate new instances of schemata already in the pool while simultaneously generating (instances of) new schemata. *Inversion* affects the pool of schemata by changing the linkage (association) of alleles (attributes) defining various schemata. In combination with reproduction, the net effect is to increase the linkage of schemata of high rank (coadapted sets of alleles), making such schemata less subject to decomposition. *Mutation* generally has a background role, supplying new alleles or new instances of lost alleles."

Schemata and the Building-block Hypothesis

A *schema* is a 'similarity template' (Goldberg, 1989). It describes the set of strings that have the same values as it at its *defined locations*. The character '*' (or '#') is used to mark undefined locations in schemata. A schema of *order K* has *K* defined locations, and describes $|A|^{N-K}$ distinct strings of length N with common alphabet A. There are $(|A| + 1)^N$ schemata in total partitioning a representation space of size $|A|^N$. A string of length N and common alphabet A is a member of 2^N schemata. The all '*' schema refers to the entire representation space and is sometimes not considered a schema (Whitley, 1993). The *defining length* of a schema is the maximum number of *positions* between all pairs of defined locations.

The following example assumes linear, binary {0, 1} strings. '*10* ***0' is a schema of order 3, for strings of length 8. The defining length of this schema is 6. There are 2^{8-3} distinct strings belonging to this schema. They include '0100 0000', '1100 0000' and '0101 1100'. There are three *string instances* of this schema in the population comprising strings '0100 0000', '1100 0000', '1100 0000' and '1000 1110'. Other schemata that describe string '1100 0000' include '**00 0000' and '1**0 000*'.

The order and defining length properties of schemata matter because they affect the survivability of schemata after reproduction-with-variation events. A schema is disrupted when a reproduction-with-variation event involves one or more instances of the schema but none of the offspring produced belong to the schema. A schema is constructed when the parent genotype(s) of a reproduction-with-variation event are not instances of the schema, but one or more of their offspring are. In general, higher order schemata and schemata with longer defining lengths are easier to disrupt and harder to construct. But this also depends on the make-up of the population (distribution of genotypes), the fitness function and selection mechanism, and variation operators used. Schemata disruption and construction is investigated in Spears (1998).

The *observed* fitness of a schema is the average fitness of all its genotype instances in the current population. By having a population of diverse genotypes, a schema could potentially be evaluated in more than one context, thus increasing the accuracy of its evaluation relative to other existing schemata. Schemata with above average fitness are meant to represent promising partial solutions or combinations of alleles that go together. With a fitness-proportionate (parent) selection strategy (and the right mix of variation operators), such promising partial solutions are expected to increase in frequency in a SGA population. By randomly mixing and matching promising partial solutions, the SGA is also expected to happen upon new schemata with even higher observed fitness and possibly of higher order.

The process of discovering increasingly fitter schemata of increasingly higher order is believed to be the way that the canonical or simple genetic algorithm (SGA) behaves, and has been called the *Building-block Hypothesis* (Goldberg, 1989). The *building-blocks* of the Building-block Hypothesis (BBH) are special schemata which are highly fit and highly resilient to disruption. For this reason, of two schemata with equal above average fitness, the one with a shorter defining length is a better (more useful) building-block. The *inversion* operator was introduced to make schemata more compact (i.e. have shorter defining lengths) but its application requires a separation between a gene and its location in a genotype (Goldberg, 1989 elaborates).

By evaluating and selectively reproducing with variation a population of genotypes, the SGA is also selectively storing and processing, in parallel, a pool of schemata. Holland calls this *implicit* or *intrinsic parallelism* (1992 p. 71). The number of schemata in a population depends on its make-up. It can range between 2^N (maximum homogeneity) and PS × 2^N (maximum diversity) (Goldberg, 1989 p. 20). The population size (PS) is usually smaller than the required value to make $PS \times 2^N > (|A| + 1)^N$. Goldberg (1989) estimates that as many as PS^3 schemata are processed in each SGA generation.

Crossover and Mutation

The primary method of mixing and matching in genetic algorithms is crossover. A crossing-over or recombination involves an exchange of alleles between at least two genotypes. In an *n*-point crossover, *n* positions are chosen at random as cut-points. If *n* is odd, the position between the first and last genotype locations which is 0, is a cut-point by default. Alternate segments of alleles between adjacent cut-points are exchanged. Besides being able to exchange multiple alleles at one time, the crossover operator is also able to create new combinations of alleles while preserving some of the similarities between the parent genotypes. Therefore if both parents are instances of a schema, so are their recombinants. And if the parent genotypes are each instances of non-overlapping fit schema, one of their offspring may inherit both schemata and be more fit then both of its parents.

Crossover does not change the pool of alleles presented to it by parent genotypes. This can be a limitation in insufficiently diverse populations and if the missing allele is of importance. The *mutation* operator was introduced to prevent the irreversible loss of alleles. Mutation changes the value of a gene to one of its alleles chosen at random.

Exploration versus exploitation

Given limited resources (e.g. finite lifespan) and an unfamiliar and changing environment, a trade-off encountered by adapting systems, e.g. evolutionary algorithms, is between exploitation and exploration. On the one hand, too much exploitation (and too little exploration) may lead systems to be over-adapted to their current environment, and miss out on 'greener pastures'. On the other hand, exploration is risky because systems may not have anything positive to show for the cost incurred (fitness loss or increased time to optimality) at the end. There is no one solution to this dilemma. The ideal strategy is to properly balance exploitation and exploration throughout the evolutionary process. In this respect, evolutionary algorithms with multi-individual search points (e.g. genetic algorithms) have an advantage over single-individual evolutionary algorithms (e.g. hill climbers) because they do not need to 'put all their eggs in one basket'. But this advantage is contingent upon sufficient population diversity which amongst other things depends on the selection mechanism.

A highly selective parent-selection mechanism would give a disproportionate number of chances to a small group of genotypes to reproduce-with-variation and thus limit an EA's ability to explore its search space. In addition, if survival-selection is random or generational as in the SGA, the building-blocks of a small group of genotypes, through inter-breeding, will quickly takeover the finite size population thus further limiting exploration and intensifying exploitation. Restricting the ability of an evolutionary algorithm to explore once it has stumbled on a promising patch in the fitness landscape may be a good search strategy for some optimization problems, e.g. Two-Max (van Hoyweghen et al., 2002), but not for others.

There are opposing schools of thought in the field of evolutionary computation whether crossover is a mechanism for exploitation or exploration. On the one hand, crossover is more likely to cause larger perturbations to individual genotypes than random mutation and on this basis a genotype undergoing crossover could be viewed as doing exploration. In this view, mutation becomes a mechanism for exploitation. On the other hand, crossover enables highly fit individuals to broadcast about their respective promising areas in the fitness landscape and to enlist the help of other individuals in the population to search these promising areas more intensely by drawing less fit individuals nearer to them and their promising areas. By increasing the homogeneity of a population through the reproduction and propagation of building-blocks, crossover helps an EA focus its search. Hence from this perspective, crossover is a mechanism for exploitation and mutation becomes a tool for exploration. This latter point of view is the one adopted by this dissertation for reasons that will become clear in Part II.

The tension between exploitation and exploration in an evolutionary algorithm (EA) can be observed through changes in the distribution of alleles per gene in its population over evolutionary time. Exploitative mechanisms tend to increase population homogeneity while explorative mechanisms tend to diversify a population.

Genetic drift

In this dissertation, *genetic drift* refers to changes in the distribution of alleles per gene in an EA population over evolutionary time due to stochastic exploitative and explorative events. The phrase 'genetic drift' comes from population genetics where it describes the effects of random processes (random parent-selection and survival-selection or replacement) on gene frequencies in finite populations. In contrast, changes in gene frequencies attributed to natural selection are associated with adaptive and reproductive success. This dissertation does not split this hair and defines genetic drift as *the accidental and purposeful (in retrospect) redistribution of alleles for each gene in a population.*

Genetic algorithms as function optimizers: a metaphor mismatch?

The discussion about the SGA (simple or canonical GA) and genetic algorithms in general has so far focused on schema processing, and the mixing and matching of building-blocks. But genotypes are the constructs that represent complete solutions to optimization problems, not a population of building-blocks. The combination of fitness-proportionate (parent) selection and generational replacement in finite populations has subtle effects which are interesting from an adaptation point of view, but they do nothing to preserve highly fit genotypes explicitly, or to ensure that the next generation is fitter on average than the current one (this may or may not be desirable). Further, schema processing is based on observed fitness, which depending on what has been observed hitherto, can be very different from actual or static fitness if one exists. Thus, the BBH may be working as described, but its success in optimizing a problem with a static fitness function also depends on the sufficient supply of appropriate building-blocks at the right time in evolution. The fitness-proportionate (parent) selection, generational replacement and SGA in general, are often painted in a negative light from an optimization viewpoint (e.g. Altenberg, 1995).

However, de Jong (1992) argues that this sort of criticism of the canonical GA stems from a basic lack of understanding about the nature of the SGA. As a model for studying adaptation in natural systems, the SGA is more about how a population as a whole adapts for continued survival in an unknown and changing environment at minimum cost to itself, and less about producing one super fit individual efficiently and at the expense of other individuals in the population, as is often required when seeking a globally optimal solution to an optimization problem. In other words, the canonical GA is designed to maximize cumulative pay-off (on-line performance) and not the bottom-line (off-line performance). In some cases, these two ways of measuring GA performance coincide, and the SGA behaves as a function optimizer and produces good results. In other cases, the SGA is a very poor optimizer and requires a variety of modifications to improve its behavior as a function optimizer. Nevertheless, genetic algorithms have empirically proven themselves useful as optimizers. But a theoretical explanation for their effectiveness has been tenuous at best. Holland's theory on schema processing is designed to capture the behavior of the canonical GA or SGA, and need not predict the effectives of genetic algorithms as optimizers (de Jong, 1992).

2.3 Canonical EA fitness functions

2.3.1 One-Max

The One-Max fitness function has a single globally optimal string and awards a fitness point for *every* occurrence of a value match between a string and the globally optimal string. It is commonly implemented as a unitation function (sum of ones) on a binary alphabet: $F(S) = \sum_{i=0}^{N-1} S_i$, with an all-ones string (11...11) as the optimal solution. N is the length of string *S*. S_i is the value of *S*'s ith element.

2.3.2 Two-Max

The Two-Max problem is similar to the One-Max problem in all aspects, except as its name implies, there are two global optima. Fitness is awarded by taking the maximum of the number of matches: $F(S) = \left|\frac{N}{2} - \sum_{i=0}^{N-1} S_i\right| + \frac{N}{2}$. For the binary {0, 1} alphabet, the two global optima are commonly the all-zeroes (00...00) and the all-ones (11...11) strings.

2.3.3 Royal-Road

One of the earliest attempts at demonstrating the building-block behavior of a genetic algorithm is the Royal-Road problem (Mitchell et al., 1992). In the basic Royal-Road problem (RR), a string is divided into a set of non-overlapping segments, and fitness points are awarded to a segment only when it is optimal. Each RR segment has at least two elements. A RR segment is optimal only when it is identical to the corresponding segment in the optimal solution. A non-optimal segment has zero fitness. Fitness of a string *F*(*S*) is the sum over all segment fitness. A RR segment is meant to correspond to a building-block (fit schemata). Additional fitness points (bonus) may be awarded to consecutive optimal segments to acknowledge the formation of higher order building-blocks. The Royal-Road fitness function is: $F(S) = \sum_i f_i s_i(S)$ where $s_i(S) = 1$ if *S* is an instance of schema s_i and 0 otherwise. f_i is the fitness value associated with s_i. Example (all segments are of the same size, 4) follows:

Optimal string: 1011 1111 0000 1111
Optimal or maximum fitness: 16 (without bonus), 32 (with bonus)
Schema (s) & fitness (f):

s_1	1011 **** **** ****	$f_1 = 4$
s_2	**** 1111 **** ****	$f_2 = 4$
s_3	**** **** 0000 ****	$f_3 = 4$
s_4	**** **** **** 1111	$f_4 = 4$
s_5	1011 1111 **** ****	$f_5 = 8$
s_6	**** **** 0000 1111	$f_6 = 8$

String: 1011 0000 0000 1111
Basic Royal Road fitness: $4 + 0 + 4 + 4 = 12$
Royal Road with Bonus: $12 + 8 = 20$

2.4 The HC < GA problem

When are genetic algorithms effective as optimizers? Since the Royal-Road experiments (Mitchell et al., 1992) researchers have addressed this problem (rightly or wrongly) by comparing GA performance with the performance of hill climbers on static optimization problems. By comparing Figures 2.3 and 2.4, we see clearly that the SGA involves significantly more computation than the HC. Therefore it is natural to ask whether these additional steps contribute positively and significantly to problem solving (optimization), and if so when and how.

The HC < GA problem is to find a static fitness function that a GA optimizes more successfully and/or efficiently than a HC. The comparison between the HC and the GA is made on the basis of *optimizing performance*, which is the number of successful runs (SUCC) and Mean First Passage Time (MFPT). MFPT is the average number of function evaluations to discovery of a (the first) globally optimal solution. Convergence quality of an EA is discussed in more detail in Part II.

The problem of defining which HC and GA pair to compare is discussed further below. But the convention in this area of research at the time of this thesis is to state clearly which hill climbing algorithm and genetic algorithm a proposed solution to the HC < GA problem holds, and to define the hill climbing algorithm and the genetic algorithm concerned with their parameters.

The HC < GA problem is not as straightforward as it might first appear. First, there are a variety of hill climbers and genetic algorithms to choose from and compare. Researchers investigating this problem have not always been consistent in their choice (and given the apparent variety, it would be limiting if they did). Part of the challenge of this problem lies with the evolution of genetic algorithms from its canonical form to encompass a family of algorithms. Even within a particular genetic algorithm there are parameters to tweak. In addition, hill climbers are single-individual evolutionary algorithms, whereas genetic algorithms are multi-individual evolutionary algorithms. So besides the difference in variation operators used, there is difference in population size to consider when comparing hill climbers with genetic algorithms. The nature of crossover itself has needed clarification in the past to distinguish it from macro-mutation, e.g. (Jones, 1995).

Second, the boundary between hill climbing behavior and expected evolutionary behavior of genetic algorithms is not always entirely clear. Is a genetic algorithm just a more efficient hill climber, or a fundamentally different kind of EA? How would this be detected? Studies that focus solely on optimizing performance might overlook this more fundamental dimension of the problem. Naudts (1998) devotes some attention to this issue for NNI (Nearest-Neighbour-Interaction) problems and suggests the use of 'site fixation chronology'. A genetic algorithm (GA) is expected to exhibit more parallelism in its site fixation pattern than a hill climber (HC).

Third, to complicate matters, the HC < GA problem has become entangled, rightly or wrongly, with the 'mutation versus crossover problem' which in one interpretation seeks to discover 'what crossover can do that mutation cannot', e.g. (Watson, 2002); and in another interpretation, to understand and identify any distinction between the roles that crossover and mutation play in genetic algorithms, e.g. (Spears, 1998).

2.4.1 Rules of the game

Like Watson (2002) and others, this dissertation will follow the common practice and make conclusions about problem difficulty with respect to specific evolutionary algorithms namely RMHC and upGA (introduced in Part II). The conclusions are made primarily on the basis of optimizing performance, but some attention is also given to behavior. Unlike Watson (2002), this work does not aspire to demonstrate the imperativeness of crossover or any other compositional mechanisms e.g. symbiogenesis, to artificial or biological evolution. Imperativeness of a variation operator is judged by its ability to evolve something that other variation operators cannot within a reasonable amount of time. In this dissertation, both mutation and crossover are regarded as essential components of a GA for optimization. Site fixation analysis and formalization of the roles that mutation and crossover play in a genetic algorithm within the context of the new HIFF problems are left for future work.

2.4.2 No Free Lunch or why a solution should exist

Assuming HC and GA are fundamentally different algorithms, previous researchers have appealed to the No Free Lunch Theorem (NFLT) (Wolpert and Macready, 1997) to justify the existence of a solution to the HC < GA problem.

The main point of this theorem is: Averaged over all possible search problems, the expected performance of all search algorithms is exactly the same. The corollary of this is: For evolutionary algorithms to work effectively, the heuristics they use to traverse the search space must match the biases implicit in the problem.

There are several proofs of the basic NFLT referred to above. Loosely, since a search algorithm $a \in A$ can visit a set of search points in a certain order (permutation) only (no revisiting of search points) and in the corresponding set of all fitness functions F there exists every order of fitness values, a cannot perform the best over all fitness functions in F (visit the point with the optimal value earlier than all other search algorithms in A). There must be some fitness function in F that orders the fitness values such that the order in which a visits the search points is the worst (visits the point with the optimal value later than all other search algorithms in A). One challenge to this NFLT is whether it is realistic to assume that all functions in F are equally likely. See www.no-free-lunch.org for other NFLT related theorems.

2.4.3 History and Related work

In spite of having well-identified building-blocks to pave a royal road for genetic algorithms, the Royal-Road problem was eventually shown to fail as a solution to the HC < GA problem. Forrest and Mitchell (1993) used the *Random Mutation Hill Climber* or RMHC (section 5.1.1) to demonstrate that the Royal-Road problem is more easily solved by a HC than a GA. Subsequently, an Idealized-GA (IGA) was designed to demonstrate that a GA can in principle outperform a RMHC on the Royal-Road problem (Mitchell et al., 1994).

The Royal-Road experiments and the IGA revealed several areas where the picture painted by schema processing and the Building-block Hypothesis for a canonical GA, does not align with the desired behavior from a GA for static function optimization (Mitchell, 1998 pp.131-138). Essentially, the non-IGA (the GA used in Forrest and Mitchell, 1993) performed worse than RMHC on the Royal-Road problem because the non-IGA did not discover and preserve all the building-blocks necessary to construct a globally optimal genotype. There was insufficient exploration and population diversity.

The 'failure' of the Royal-Road experiments inspired a slew of Royal-Road variant problems, as well as other fitness functions, to address the HC < GA problem. While Mitchell and colleagues focused on how a GA might be modified to be more efficient than RMHC on the Royal-Road problem, other researchers modified the Royal-Road problem to increase its difficulty for hill climbers , e.g. Holland's extension (Jones, 1995 section 3.8.6), van Nimwegen and Crutchfield (2000), Jansen and Wegener (2005), and Watson and Jansen (2007). The HIFF problems discussed in this dissertation can be viewed as descendents of the Royal-Road problem as well.

Other abstract or formal fitness functions used to investigate the HC < GA problem include (in no particular order of importance) those by David E. Goldberg and his research team, e.g. Hierarchical Trap, Bipolar and Tobacco Road functions; Engel's function (described in Rosé et al., 1996); Shapiro et al. (1994); Prügel-Bennett (2004) and those based on the Ising model, e.g. Naudts (1998), Fischer and Wegener (2005) and Sudholt (2005).

In general, the GAs used to demonstrate the 'success' of a fitness function as a solution to the HC < GA problem have hitherto required some form of explicit population diversification mechanism, e.g. fitness-sharing, deterministic-crowding and multi-population. More so, as problem size increases. The original HIFF problems are no exception. This dissertation demonstrates in Part II with a new HIFF problem that this need not be the case.

2.5 Discussion

The Royal-Road problem also highlights an important but subtle distinction between epistasis and frustration.

2.5.1 Epistasis and the site-wise optimization (SWO) measure

Epistasis is a term appropriated by evolutionary computation from biology to refer to dependencies or interactions between genes. Biological systems are replete with polygenic genes (genes that depend on other genes) and genes with pleiotropic effects (genes that influence many other genes). This has led some researchers to wonder how biological systems manage to evolve at all if small changes to genotypes produce large often detrimental effects to the overall phenotype. This apparent problem has been an impetus for the study of modularity in adapting systems (Wagner and Altenberg, 1996; Altenberg, 2005).

In evolutionary computation, epistasis refers to dependencies between problem variables, i.e. the fitness contribution a value at a variable can make depends on the values of other variables. Naudts (1998, p.21): "Epistasis expresses the extent to which a fitness function can be decomposed into lower-dimensional functions. In fitness landscapes with a high epistasis, there is a higher interaction between the sites..." Spears (1998, p.14): "A system has low (high) epistasis if the optimal allele for any locus depends on a small (large) number of alleles at other loci. Systems with independent loci (the optimal allele for each locus can be decided independently of the alleles at the other loci) have no epistasis." Watson (2006, p.296) defines epistasis as "Functional interactions between alleles at different loci, meaning that the effect of a mutation at one locus depends on the allelic state at one or more other loci. See also frustration of variables." Goldberg (1989, p.22) equates epistasis with nonlinearity.

Epistasis has been linked with problem difficulty for evolutionary algorithms: Schaffer and Eshelman (1991), Kauffman (1993), Naudts et al. (1997) and Spears (1998), but with no definitive results. What is clear though is that the kind of epistasis is more relevant to problem difficulty for evolutionary algorithms than the amount of epistasis. Naudts (1998, p.3): "It is not the *amount* of interaction, but the kind of interaction that counts." The kind of epistasis that creates problem difficulty for an evolutionary algorithm does so through *frustration* (section 2.5.2). Since frustration is a concept intimately tied to the fitness landscape of an evolutionary algorithm, a problem may be frustrating and hence difficult for one EA and not for another. Naudts (1998, p.62): "it is not the amount of interaction which influences the convergence. Rather, one has to search for the presence of symmetry or frustration within the interaction....Dependencies between sites can also have a positive effect on the convergence behavior....We conclude that quantitative information about higher order interactions is not a priori related to the convergence behavior of the GA."

Several ways to quantify the amount of epistasis in a problem have been proposed: epistasis variance by Davidor (1991), the site-wise optimization measure (SWO) by

Naudts and Kallel (2000) and an information-theoretic approach by Dong-II and Byung-Ro (2007).

Site-wise optimization measure (SWO)

Site-wise optimization measure or SWO (Naudts and Kallel, 2000) measures the amount of epistasis in a problem. *SWO* values are measured for a fitness function *F* based on a sample of (random) individuals, *PS*. The size of *PS* is 128 for the *SWO* values in Table 2.1.

The SWO measure is the ratio of population diversity (width) measured after each individual in the population has independently performed a site-wise hill climb (SWHC), to the diversity of the population measured prior to the hill climbs. Population diversity is the average Hamming distance between its members.

$$\text{SWO}_{PS}(F) = \text{width (SWHC, } (F, PS)) / \text{width}(PS)$$

A site-wise hill climb by a genotype *g* constructs for every gene in *g*, a set *B* of values that strictly increases *g*'s fitness using the genetic background of *g*. The output of the site-wise hill climber is a genotype *g′* whose value of gene *i* is a random choice from B_i if B_i is not empty, or the value of *i* in *g*.

The SWO measure defined for problems with a single global optimum is bounded within the closed interval [0.0, 1.0]. Problems with *SWO* values closer to 0 are less epistatic. One-Max's *SWO* is 0.0 since a population of initially random genotypes will converge to the optimal genotype after doing the site-wise hill climb and therefore would have a population width (diversity) of 0.0. For problems with two complementary global optima, e.g. Two-Max, the approach in this dissertation is to complement a genotype after their SWHC where necessary, to bring the genotype closer in Hamming distance, to one of the global optima before calculating population width.

2.5.2 Frustration and symmetry

Frustration is a term appropriated from the branch of physics which studies systems of particles interacting locally to produce collective effects. Magnetism is one such global effect and the Ising model is commonly used to model such systems. An Ising model consists of a set of sites connected to each other following some topology. Each site σ_i is occupied by a spin variable which can take either a -1 or +1 value and may have a constant value h_i (the external magnetic field strength of the site). The site-site connections represent interactions between spin variables and may have constant values J_{ij} associated with them (the interaction coefficients). The Ising problem is to find a configuration of spins that minimizes the energy function E given by $-\frac{1}{2}\sum_{i,j} J_{ij}\sigma_i\sigma_j - \sum_i h_i\sigma_i$ for each site, and globally. Such a configuration is called a ground state (global optimum). Ideally one would like to find all ground states for an Ising problem. However, the geometry of an Ising problem can make this very difficult, if not impossible. If J > (<) 0.0, aligned (mis-aligned) spins are favoured because they minimize E.

For an evolutionary algorithm, frustration sets in when there is no clear fitness gradient to follow and not all fitness gradients lead to a global optimum; or there are conflicting conditions in the problem. The former situation is a result of *symmetry* in the fitness landscape.

RR2

The frustrating effects of symmetry can be introduced into the RR problem by setting up a complementary global optimum and by using bonus fitness points to create inter-dependencies between segments. Let this new problem be called RR2. With more than one optimal configuration for each segment and no external pressure to favour either global optimum, individual RR2 segments are free to align themselves to different global optima. But only when all segments cooperate and favour the same global optimum is a RR2 problem solved at the global level. The HIFF problems use this same trick to create frustration for evolutionary algorithms (section 3.3).

The RR2 problem is an example where increasing the number of global optima actually increases problem difficulty for EA, while at the same time increases the probability (albeit by an inconsequential amount if the search space is large) of finding a solution through pure random search. This phenomenon is observed in many classical NP-hard optimization problems (Prügel-Bennett, 2004).

2.5.3 An experiment

Table 2.1 compares the *SWO* values and the RMHC results for One-Max, Two-Max and RR* problems. These results demonstrate that problems which are easily solved by RMHC do not necessarily have low amounts of epistatic, i.e. small *SWO* values.

RR(*n*) is a basic Royal-Road problem with uniform segments of size *n*. The RR2 function (section 2.5.2) is a RR(*n*) problem with two global optima. Epistasis increases as *SWO* values approach 1.0 from 0.0 (section 2.5.1). The RMHC (Figure 5.1) is very similar to the generic hill climber in Figure 2.3. P_m is the mutation rate which is used to determine the maximum number of changes (mutations) that can be made to a genotype, i.e. the maximum reach of the RMHC. P_m, RMHC and other parameters used in the runs reported in Table 2.1 are explained further in Part II.

Table 2.1 SWO values and RMHC (Pm = 0.0625) optimizing performance

Problem size N=128	*SWO* values averaged over 5 random runs ± 99% confidence interval	SUCC	MFPT
One-Max	0.0000 (0.0000)	30	1,843 (540)
Two-Max	0.1210 (0.0496) ± 0.0571	30	1,790 (563)
RR (8)	0.9999 (0.0002) ± 0.0002	30	43,976 (23,515)
RR (16)	1.0000 (0.0000)	0	-
RR2 (8)	0.9939 (0.0019) ± 0.0022	0	-

Standard deviation is given in parentheses. MFPT = mean first passage time.

Except RR(16) and RR2(8), all RMHC runs in Table 2.1 succeeded in finding a global optimum within 250,000 function evaluations. Problems with similar amounts of epistasis, RR(8), and RR(16), have vastly different RMHC SUCC values. RR2(8) is less epistatic than RR(8) (their SWO 99% confidence intervals do not overlap), but RR2(8) is not solved by RMHC.

Adjusting the mutation rate did little to change the result for RR(16). The reason why RMHC is unsuccessful on RR(16) is related to its segment size being larger than $\sqrt{128}$. Watson and Jansen (2007) make this (segment size > $\sqrt{N}$) one of the conditions in their version of the Royal-Road problem. According to Naudts (1998 Chapter 3), the epistasis level indicative of problem difficulty for the Royal-Road problem depends only on segment length, and not the number of segments which is proportionate to problem size.

The results in Table 2.1 show that problems with high amounts of epistasis need not be difficult for RMHC to solve, e.g. RR(8). Hence the amount of epistasis is not a sufficient indicator of EA difficulty. Other factors such as noise and deception may also contribute to problem difficulty.

2.5.4 Size and type of interaction neighbourhoods

The NK model (Kauffman, 1989) is archetypal of a problem which can be frustrating for evolutionary algorithms. It is similar to a variant of the Ising model called the spin-glass model (Kauffman 1993 p. 43). In the spin glass model, J_{ij} are stochastic variables, i.e. they are assigned random values.

In the NK model, N is the number of sites (or string length), and K is the non-zero number of other sites (i.e. the neighbours) a site is dependent on. The number of neighbours may be different or uniform for each site, and the neighbours may be nearest or chosen at random. The fitness contribution of a site is determined by a lookup table which comprises randomly generated fitness values indexed by the combination made up of the values at the site and its neighbours. For example if site 2 has sites 1 and 4 as its neighbours, and the value of sites 1, 2 and 4 are 0, 1 and 0 respectively, then '010' is used to lookup the fitness contribution for site 2. Fitness of an NK genotype is the average over all site fitness contributions. From this description, it is clear how the best combination of values for each individual site can come into conflict with each other.

Unlike the Royal-Road problem and its variants, the interactions in a NK problem are not confined to within some local neighbourhood, and the interaction strengths are randomly assigned. Hence NK problems, especially when K approaches N are said to have random (fitness) landscapes (Sherrington, 1997).

Kauffman (1993 p. 114) argues that in general, recombination is not effective on random landscapes because for recombination to be useful, parts of a genome need to be functionally independent from each other, in other words have an 'additivity' property. For this to happen, epistatic interactions need to be bounded to within some local neighbourhood. The Royal-Road problem and the problem proposed by Watson and

Jansen (2007) which is similar[1] to RR(16) in Table 2.1, have this additivity property. However, according to Kauffman, this local-epistasis model does not conform to the realities of biological systems.

Fogel (1995) also argues that recombination will not be effective for problems with high epistasis because there would not be good building-blocks for recombination to exploit. Thus, Altenberg (1994) proposed the use of a 'constructional selection' method where genes (a site and its interactions) are incrementally added to an NK genotype to reduce overall site-site interactions in the genotype[2]. But epistasis promotes communication and coordination between genes, and can be beneficial to an EA in certain circumstances[3]. Spears (1998, p.15) point to the existence of some evidence that recombination is effective for solving problems with medium but not high epistasis.

Nevertheless, Kauffman also argues that recombination can be useful when K is small relative to N without the local neighbourhood restriction if the fitness landscape (for mutation with maximum radius of one Hamming Distance) is such that (i) high peaks (local optima) are located near one another and have large basins of attraction, and (ii) high peaks contain mutually beneficial information so that recombination of genotypes located at high peaks lead to the discovery of higher peaks which lie somewhere in between the two peaks. Note that Kauffman does not experiment with genetic algorithms

[1] In the Watson and Jensen (2007) problem, a segment need not wait till it is optimal before being awarded any fitness points, but the problem for each segment can be defined to frustrate an EA.

[2] Usefulness of the constructional selection (CS) method to GAs was investigated in Khor (2006). The results were not promising because CS does not create relatively isolated subsets of variables, i.e. modules, within a genotype. Instead, CS may be increasing inter-site dependencies when trying to reduce epistasis for a genotype as a whole and in turn increase the number of linkages crossover has to maneuver.

[3] Usefulness of epistasis was investigated in Khor (2007b) with a positive outcome under certain circumstances – it appears that some flexibility in the value required at each site is required. The proposed method – RMHC with allele transposition – does not work for the discrete HXOR problem. Like HIFF-D, RMHC with allele transposition also solves the HIFF-M problem introduced in chapter 3.

directly. His experiment in this area alternates a pure hill climbing phase with a pure recombination phase.

The HIFF problems proposed by Watson (2002) are additive to a certain extent, but the interactions between their sites are not random and not local (K=N-1). This combination of apparently contradictory properties is possible through the notion of modularity (section 3.3). High peaks in the HIFF mutation landscape also contain mutually beneficial information that when recombined in the right manner can lead to the discovery of higher peaks (Figure 2.5). There are no local optima (peaks) in the HIFF crossover landscape. The imaginary edges spanning fitness saddles in the HIFF mutation landscape is made real by recombination.

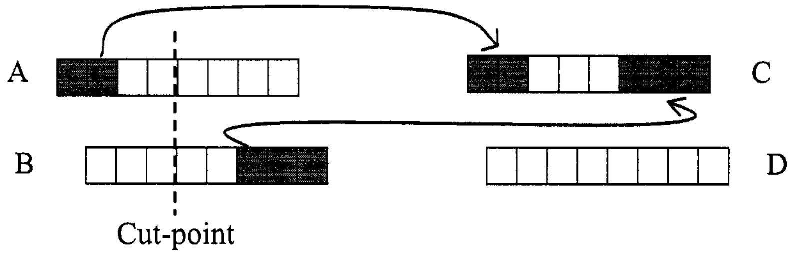


Figure 2.5 Recombination of two fit genotypes (A and B) at cut-point produces a fitter genotype (C)

However, as Watson and others (e.g. Ebner et al., 2001; van Hoyweghen et al., 2002) have confirmed, a GA needs some extra help via population diversification mechanisms or the genotype-phenotype map (mapping between search objects or phenotypes and their representation or genotypes), to arrive at different high peaks in a HIFF mutation landscape. In Part II of this dissertation, a new HIFF problem called HIFF-M (section 3.2.4) demonstrates that this need not be the case. HIFF-M also has non-random non-local interactions, and high peaks in the mutation landscape holding mutually beneficial information that when recombined can lead to the discovery of even higher peaks. However, less effort in terms of maintaining population diversity is required to entice HIFF-M genotypes in a GA population to reach different high peaks.

Part I

Part I delves into the *Hierarchical-If-And-Only-If* (HIFF) problems. The (original) HIFF problem was introduced as a formal or abstract problem that is more difficult for a gradual evolutionary process, i.e. hill climbing, than for a compositional evolutionary process, i.e. evolution with a genetic algorithm, to solve (Watson et al., 1998; Watson, 2002). In other words, the HIFF problem is proposed as a solution to the HC < GA problem defined in section 2.4.

There are several versions of the HIFF problem in the evolutionary computation literature (Watson and Pollack, 1999). There is a complementary version of the HIFF problem called Hierarchical-Exclusive-Or (HXOR), and also a Hierarchical Problem Generator to produce randomized hierarchical problems (de Jong et al., 2005).

HIFF problems in general

In keeping with the more common version of the HIFF problem, a HIFF problem in this dissertation is one of transforming a $\{0, 1\}^N$ string to a string of either all-zeroes $\{0\}^N$ or all-ones $\{1\}^N$. The size of a HIFF problem refers to the number of elements in its string, i.e. N. The all-zeroes and all-ones strings are referred to as maximally similar strings and are the globally optimal solutions for a HIFF problem as their fitness values are optimal. When a globally optimal solution is found for a HIFF problem, the HIFF problem is fully optimized or solved. All HIFF fitness values are greater than or equal to zero, and better solutions (genotypes in evolutionary algorithm parlance) have higher fitness values. In other words, HIFF problems are maximizing problems.

Parts I and II of this dissertation use the HIFF problems defined in chapter 3. Another set of HIFF problems are introduced in chapter 9 of Part III. The definitions and explanations given here in Part I apply to all HIFF problems in this dissertation, unless otherwise stated.

The basic idea behind a HIFF problem and its fitness function

A HIFF problem is defined by its *fitness function* $F(S)$. The fitness function $F(S)$ of a HIFF problem consists of a set of *iff* constraints (c_i) specified between variable pairs of the problem. Each *iff* constraint is associated with a real number representing its weight or fitness contribution (w_i). In general, the fitness function of a HIFF problem awards to a string fitness points according to the number of constraints satisfied by the string, i.e. $F(S) = \sum_i w_i c_i(S)$ where $c_i(S) = 1$ if S satisfies c_i and 0 otherwise. An *iff* constraint between two variables i and j is satisfied if and only if i and j hold the same values, i.e. $i = j$. When there is an *iff* constraint between two variables i and j in a HIFF problem, the variables i and j are said to be inter-dependent on each other, or equivalently to interact with each other. Heavier weights or larger fitness contributions correspond to stronger inter-dependencies or interactions.

In general[4], a HIFF fitness function can be represented by an *inter-dependency matrix* **M**. M_{ij} denotes the fitness contribution of the *iff* constraint between variables i and j or alternatively, the strength of the interaction between the variable pair (i, j). All M_{ij} are ≥ 0.0. Because self-interaction is not rewarded, and all variable-to-variable interactions are bi-directional and equally weighted in both directions, **M** is an N × N symmetric

[4] An inter-dependency matrix as it is defined here need not capture all aspects of a HIFF fitness function, e.g. the inter-dependency matrix for HIFF-D (section 3.2.2) does not represent the all-or-nothing approach of HIFF-D's fitness function. However, the HIFF-D fitness function is the only exception in this dissertation. All other HIFF fitness functions are completely represented by their respective inter-dependency matrix.

matrix with zero-valued entries on the main diagonal. The inter-dependency matrices also capture the problem structure of HIFF problems.

An example

Figure I.A depicts in graph form a HIFF problem with four variables and three *iff* constraints. Variables are represented by nodes and *iff* constraints by links. There are *iff* constraints, denoted c_1 to c_3, between the following variable (unordered) pairs: (0, 2), (0, 1) and (3, 2). The fitness function for this HIFF problem is $F(S)=\sum_i w_i c_i(S)$ where $c_i(S)=1$ if S satisfies c_i constraint and 0 otherwise, and w_1 = 0.5 and w_2 = w_3 = 1.0. According to the fitness function for this HIFF problem, the weight of the c_1 constraint is 0.5 fitness points, while satisfaction of the c_2 or c_3 constraint contributes 1.0 fitness points each. Figure I.B represents the fitness function for this HIFF problem in an inter-dependency matrix **M**. The attentive reader would notice the similarity of this fitness function with the Royal-Road fitness function (section 2.3.3). The set of *iff* constraints corresponds to the set of fit schemata in the Royal-Road problem. In this example, the fit schemata are: 1*1* and 0*0* representing c_1, 11** and 00** representing c_2 and **11 and **00 representing c_3.

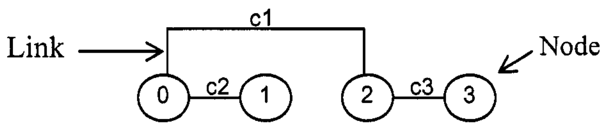


Figure I.A A HIFF problem, N = 4

	0	1	2	3
0	0.0	0.5	0.25	0.0
1	0.5	0.0	0.0	0.0
2	0.25	0.0	0.0	0.5
3	0.0	0.0	0.5	0.0

Figure I.B Inter-dependency matrix (M) for the HIFF problem in Figure I.A

Suppose the string given to the HIFF problem in Figure I.A is '1011', then variable 0 is assigned value '1', variable 1 is assigned value '0', variable 2 is assigned value '1' and variable 3 is assigned value '1'. The string '1011' satisfies constraints c_1 and c_3 only and therefore has a fitness value of 1.5. The maximum fitness value for this HIFF problem is 2.5 which is attained when the value of all variables is either '0' only or '1' only. Hence the strings '0000' and '1111' are both globally optimal solutions for this HIFF problem.

Why a HIFF problem is a solution to the HC < GA problem

The string '1011' is represented by genotype '1011' in an evolution algorithm (hill climber or genetic algorithm). It would appear that all a hill climber (e.g. Figure 2.3) has to do in the example above to transform '1011' to '1111' is to assign '1' to gene 1, the second gene from the left. However, mutation (re-assignment of a value to a gene) is performed uniformly at random, that is any one of the four genes in genotype '1011' could be chosen for mutation. Therefore, it is equally likely that gene 0, the first gene from the left, is chosen for mutation and re-assigned to value '0', thus transforming '1011' to '0011'. This mutation succeeds (the mutant genotype replaces the parent genotype) because '0011' is fitter than '1011'. The genotype '0011' satisfies constraints c_2 and c_3 which are worth 1.0 fitness points each. Thus '0011' has a fitness value of 2.0 and is fitter than '1011' whose fitness value is 1.5.

If the hill climber can only mutate a single gene at a time, the hill climber is now stuck at a local optimum. No matter which gene it mutates, '0011' will be fitter and yet '0011' is not a globally optimal solution. In contrast, a genetic algorithm (Figure 2.4) might be able to escape from this local optimum because the crossover operator exchanges segments of genotypes at one time, and thus if a crossover between '0011' and

‘1110’ genotypes happens, the ‘1111’ genotype might be produced as a result. This in a nutshell is why the HIFF problem is claimed to be more difficult for a gradual evolutionary process, i.e. hill climbing, than for a compositional evolutionary process, i.e. evolution with a genetic algorithm, to solve (Watson et al., 1998; Watson, 2002).

Importance of appropriate *iff* constraint weights

According to Watson, the relative weights or fitness contributions of the *iff* constraints play a significant role in making a HIFF problem more difficult for a hill climber than a genetic algorithm to solve. Watson (2002 p.139): “However, it should be noted that the choice of strength values is not in general arbitrary. It is important that the strength of inter-module dependencies is low enough that intra-module dependencies create local optima.” A *module* is a subset of problem variables that are more strongly inter-dependent on variables within the same subset than on variables outside the subset. A system is modular if “there are subsets of variables in which the variables are more strongly dependent on other variables within their own subset than on variables in other subsets” (Watson, 2006 p.13). Modularity and its role in HIFF problems are discussed further in section 3.3.

Figure I.C gives the inter-dependency matrix on page 14 of Watson (2006). A ‘-’ entry denotes no inter-dependency. In this inter-dependency matrix “the large values are diagonalizable” (Watson, 2006 p.14).

	1	2	3	4	5	6	7	8
1	-	16	4	4	1	1	1	1
2	16	-	4	4	1	1	1	1
3	4	4	-	16	1	1	1	1
4	4	4	16	-	1	1	1	1
5	1	1	1	1	-	16	4	4
6	1	1	1	1	16	-	4	4
7	1	1	1	1	4	4	-	16
8	1	1	1	1	4	4	16	-

Figure I.C An inter-dependency matrix from Watson (2006, p.14)

To illustrate this point on the importance of appropriate *iff* constraint weights, suppose the weights for the HIFF problem in Figure 1.A were inverted, i.e. w_1 = 1.0 and w_2 = w_3 = 0.5. The inter-dependency matrix now looks like that in Figure I.D. The larger values are no longer diagonalizable. *F*('1011') is still 1.5. But now, *F*('0011') is 1.0 which is less than 1.5 so the mutation of gene 0 would not succeed, i.e. the hill climber would not transform '1011' to '0011'. In this example, the closest (in Hamming distance) genotype fitter than '1011' is '1111', which is a global optimum and reachable by a hill climber that is restricted to mutating one gene at a time. Hence, this "HIFF problem" would not be more difficult for a hill climber than a genetic algorithm to solve.

	0	1	2	3
0	0.0	0.25	0.5	0.0
1	0.25	0.0	0.0	0.0
2	0.5	0.0	0.0	0.25
3	0.0	0.0	0.25	0.0

Figure I.D Inter-dependency matrix with inverted weights

Some Terminology

To assist with subsequent discussion, the concept of *blocks* and *levels* are introduced next.

A *block* (*b*) is a subset of variables contiguously located on a HIFF string. A variable may belong to more than one block but the blocks which have the same variable as a member must be of different sizes.

To decompose a HIFF string into its set of blocks B, first add all the variables of the HIFF string (assume this action does not change the string) as a block to B, then recursively divide the string into half. On each division add the variables of the two substrings as two new blocks into B. Stop dividing the string when its length is 2.

This method of obtaining the set of blocks for a HIFF string is possible when N = 2^i and i ∈ $\mathbf{Z}^+$ as it is for the HIFF problems introduced in chapter 3. As a result, the size of the set of blocks |B| for a HIFF string is N-1, and blocks of size 2^λ where $\lambda = 1\ldots i$ and $\lambda \in \mathbf{Z}^+$ are elements of B.

For example, Figure I.E shows a HIFF string of length eight decomposed into seven blocks denoted b_1 to b_7. The elements of b_1 are variables 0 to 7, block b_2 encompasses variables 0 to 3 and so on.

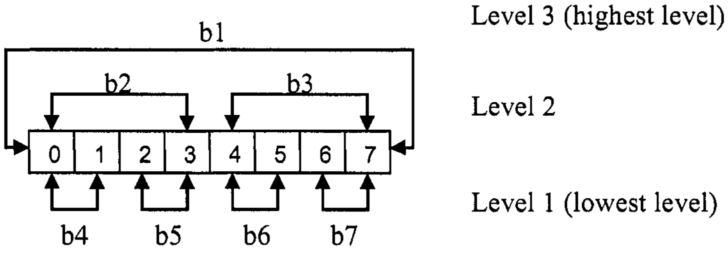


Figure I.E Decomposition of a HIFF string into blocks and levels

The inter-dependencies or interactions of a block refer to inter-dependencies or interactions between variables of the block but do *not* include the inter-dependencies of nested blocks. For example, in Figure I.A the block which includes all variables has only one interaction, c_1.

A block is optimal when the values of its variables satisfy all the block's inter-dependencies. The optimal fitness of a block for all HIFF problems introduced in chapter 3 is 1.0. This restriction is not necessary, but is imposed to maintain uniformity amongst the HIFF problems introduced in chapter 3. As there are N-1 blocks, the maximum fitness value for all HIFF problems introduced in chapter 3 is N-1.

Blocks are distinct from modules. Recall that a module is a subset of problem variables that are more strongly inter-dependent on variables within the same subset than on variables outside the subset. While the inter-dependencies of a block refer to inter-dependencies between variables of the block but do *not* include the inter-dependencies of nested blocks, the inter-dependencies of a module refer to all inter-dependencies between variables of the module. In Figure I.C for example, variables 1, 2, 3 and 4 form a module. The interactions of this module are the six inter-dependencies amongst variables 1, 2, 3 and 4. But the interactions of the block encompassing variables 1, 2, 3 and 4 are only the four 4 point weighted inter-dependencies.

A module, like a block, is optimal when all its inter-dependencies are satisfied. An optimal module implies that all blocks within the module are optimal. A block is within a module if all variables of the block belong to the module. But an optimal block need not imply optimality of its nested blocks (section 3.4.1).

A *level* (λ) is a set of same-sized blocks. Level λ contains all blocks of size 2^{λ} and there are $N/2^{\lambda}$ blocks at level λ. For example, in Figure I.E, all blocks of size two belong to level 1 and all blocks of size four belong to level 2. Larger sized blocks belong to higher levels than smaller sized blocks. A HIFF string of length $N = 2^i$ where $i \in \mathbf{Z}^+$ has $\log_2 N$ levels.

The inter-dependencies or interactions of a level refer to inter-dependencies or interactions of all blocks belonging to the level. So with reference to the inter-dependency matrix in Figure I.C, it can be said that an appropriate weight configuration for a HIFF problem (if it is to be a solution to the HC < GA problem) is one where lower-level interactions or inter-dependencies (represented by entries closer to the main diagonal) are stronger than higher-level interactions or inter-dependencies (represented by entries farther from the main diagonal).

A level is optimal when all its blocks are optimal. The term *phenotype* is used to refer to a sequence of level fitness values, arranged in level descending order with the fitness value for the highest level at the leftmost position. For example, the phenotype for '1011' using the fitness function in Figure I.B is $\langle 0.5, 1.0 \rangle$. The globally optimal phenotype has the following form: $\langle 2^0, 2^1, \ldots, 2^{i-2}, 2^{i-1} \rangle$ for all HIFF problems introduced in chapter 3. 'Phenotype' has a different meaning here than in chapter 2 where it means search object. This is not confusing since phenotype in sense of chapter 2 is not used in the rest of this dissertation.

A HIFF string is globally optimal when all its levels are optimal. Because levels are superimposed, a HIFF string is globally optimal only when all its variables have the same value. If a {0, 1} alphabet is used (as is the case in this dissertation), there are two globally optimal HIFF solutions: the all-zeroes (00...00) and the all-ones (11...11) strings. Hence, problem difficulty for pure random search which is $2 / 2^N$ is the same for all HIFF problems of the same N.

The HIFF problems at a glance

Different HIFF problems can be generated by varying the set of *iff* constraints and their fitness contributions. This is the basic approach taken in chapter 3 of this dissertation. In chapter 9, the size of blocks and the number of blocks that can be directly nested within a block are also varied.

Four HIFF problems are the subject of attention in Parts I and II. They are HIFF-D, HIFF-C, HIFF-II and HIFF-M. HIFF-D[5] and HIFF-C are part of the original set of HIFF problems introduced by Watson (2002). The new HIFF problems introduced in chapter 3 of this dissertation are HIFF-II and HIFF-M.

The HIFF-D (section 3.2.2) problem is distinct from the other three HIFF problems because fitness of a string is not calculated by aggregating the number of satisfied *iff* constraints but by aggregating the number of optimal blocks. Nonetheless, HIFF-D has the same inter-dependency matrix as HIFF-C (section 3.2.1).

What differentiates the other three HIFF problems from each other is the number of *iff* constraints. HIFF-C has the most *iff* constraints while HIFF-M (section 3.2.4) has the fewest. The weights of the *iff* constraints in HIFF-II (section 3.2.3) and HIFF-M are adjusted accordingly so that the optimal phenotype for all HIFF problems of the same size in chapter 3 is identical.

The purpose of introducing the new HIFF problems in chapter 3 is to understand whether it is necessary, if the new HIFF problems are also to be solutions to the HC < GA problem, for the entries in the inter-dependency matrix which are off the main

[5] In the evolutionary computation literature, HIFF commonly refers to the discrete version of the HIFF problem, although Watson also describes a continuous version. To avoid confusion, in this dissertation *HIFF-D* and *HIFF-C* are used respectively to refer to the discrete and continuous versions of the HIFF problem.

diagonal to be nonzero. Watson (2006, p. 14) referring to the matrix in Figure I.C: "But significantly, the values of the matrix that are off the diagonal, representing the strength of dependencies of variables across modules, are nonzero." This investigation is taken up in Part II with the HIFF-M problem. A feature present in the HIFF-II problem but not in the HIFF-C or HIFF-M problem, i.e. top-down inter-level conflict (section 3.4.2), was explored in Khor (2007d).

3. Hierarchical-If-And-Only-If (HIFF) Problems

This chapter defines the following HIFF problems: HIFF-D, HIFF-C, HIFF-II and HIFF-M, and compares the design of the new HIFF problems (HIFF-II and HIFF-M) to that of the original HIFF problems (HIFF-D and HIFF-C). It also examines the HIFF problems from a level perspective. The relationship between hierarchical levels is one dimension in which the new HIFF problems differ significantly from the original HIFF problems.

Recall from the preamble in Part I that for all HIFF problems defined in this chapter:

(i) problem size N is 2^i where $i \in \mathbf{Z}+$ and total number of levels is $\log_2 N$;

(ii) total number of blocks is N-1 and the size of blocks at level λ is 2^λ;

(iii) fitness of an optimal block is 1.0 and maximum fitness of a string is N-1 (which is also the sum of all entries in an inter-dependency matrix); and

(iv) the globally optimal phenotype is: $\langle 2^0, 2^1, \ldots, 2^{i-2}, 2^{i-1} \rangle$.

The above list of properties need not hold for problems defined in chapter 9.

3.1 The basic fitness function

For all HIFF problems defined in this chapter, fitness of a string *F*(*S*) is the sum over all block fitness values $f(b)$ as follows: $F(S) = \begin{cases} 0 & |S| = 1; \\ f(b) + F(S_L) + F(S_R) & |S| > 1. \end{cases}$

S_L and S_R are the left and right halves of *S* respectively, and block *b* comprises all variables in *S* in each recursion. The difference between the fitness functions in section 3.2 lies in the calculation of block fitness.

Table 3.1 enumerates the fitness values for all binary strings of length four. F*i* is string fitness at level *i*. F is (total) string fitness. The maximum value for F2, F1 and F are respectively, 1, 2 and 3.

Table 3.1 Fitness values for all strings N=4

String	HIFF-D			HIFF-C			HIFF-II			HIFF-M		
	F2	F1	F	F2	F1	F	F2	F1	F	F2	F1	F
0000, 1111	1	2	3	1.0	2.0	3.0	1.0	2.0	3.0	1	2	3
0001, 1110	0	1	1	0.5	1.0	1.5	0.5	1.0	1.5	1	1	2
0010, 1101	0	1	1	0.5	1.0	1.5	0.5	1.0	1.5	0	1	1
0011, 1100	0	2	2	0.0	2.0	2.0	0.0	2.0	2.0	0	2	2
0100, 1011	0	1	1	0.5	1.0	1.5	0.5	1.0	1.5	1	1	2
0101, 1010	0	0	0	0.5	0.0	0.5	1.0	0.0	1.0	1	0	1
0110, 1001	0	0	0	0.5	0.0	0.5	0.0	0.0	0.0	0	0	0
0111, 1000	0	1	1	0.5	1.0	1.5	0.5	1.0	1.5	0	1	1

3.2 Fitness Functions

3.2.1 HIFF-C

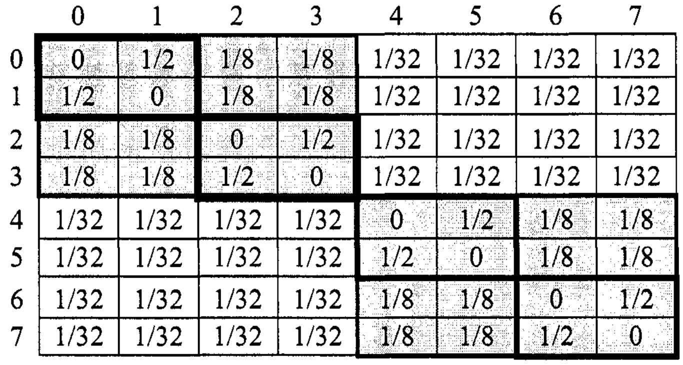
$f(b) = (p \times q) + (1 - p) \times (1 - q)$ where p and q are the proportion of ones in the left and right halves of b respectively (Watson, 2002 p.121).

Table 3.2 demonstrates with string '0110 0001'. This string of length 8 is decomposed into seven blocks and three levels. At level 1, the (values of the) blocks are '01', '10', '00' and '01'. f('01') = f('10') = f('01') = 0.0 and f('00') = 1.0. Hence fitness for level 1 is 1.0. At level 2, the (values of the) blocks are '0110' and '0001'. The proportion of ones in the left and right halves (the p and q values) of '0001' is 0/2 and 1/2 respectively. Therefore, f('0001') = (0.0 × 0.5) + (1.0 × 0.5) = 0.5. At level 3, the (values of the) block is the string itself. Block fitness f('0110 0001') = 0.5 but string fitness F ('0110 0001') = 2.5 because as mentioned previously in the preamble for Part I, the inter-dependencies or interactions of a block refer to inter-dependencies or interactions between variables of the block but do *not* include the inter-dependencies of nested blocks.

Table 3.2 Example of a HIFF-C fitness calculation

Level	String or Genotype								Level Fitness
	0	1	1	0	0	0	0	1	
1	0.0		0.0		1.0		0.0		1.0
2	(0.5 × 0.5) + (0.5 × 0.5) = 0.5				(0.0 × 0.5) + (1.0 × 0.5) = 0.5				1.0
3	(0.5 × 0.25) + (0.5 × 0.75) = 0.5								0.5
								Phenotype	⟨0.5, 1.0, 1.0⟩
								Total Fitness	2.5

Figure 3.1 is the inter-dependency matrix for HIFF-C, N=8. *F* ('0110 0001') can also be obtained by accumulating the fitness contributions (entries in the inter-dependency matrix) of the *iff* constraints satisfied by '0110 0001'. The matrix in Figure 3.1 is the matrix in Figure I.C divided by 32. Hence the relative weights of the inter-dependencies are maintained.



	0	1	2	3	4	5	6	7
0	0	1/2	1/8	1/8	1/32	1/32	1/32	1/32
1	1/2	0	1/8	1/8	1/32	1/32	1/32	1/32
2	1/8	1/8	0	1/2	1/32	1/32	1/32	1/32
3	1/8	1/8	1/2	0	1/32	1/32	1/32	1/32
4	1/32	1/32	1/32	1/32	0	1/2	1/8	1/8
5	1/32	1/32	1/32	1/32	1/2	0	1/8	1/8
6	1/32	1/32	1/32	1/32	1/8	1/8	0	1/2
7	1/32	1/32	1/32	1/32	1/8	1/8	1/2	0

Figure 3.1 HIFF-C's inter-dependency matrix, N=8. Sets of inter-dependencies belonging to the same level are indicated by borders of equal thickness in the matrix. The shaded areas represent two modules.

3.2.2 HIFF-D

HIFF-D uses the same inter-dependency matrix as HIFF-C (Figure 3.1) but it has the additional condition that a block has 0.0 fitness value until it is optimal (a block is optimal when all its variables have the same value). Thus, the possible fitness values for a block are either 0.0 or 1.0 only. For this reason, HIFF-D is said to use an *all-or-nothing* approach and is called a discrete (-D) problem.

$$f(b)=\begin{cases}1 & \forall i\, b_i=0 \vee \forall i\, b_i=1 \\ 0 & otherwise\,.\end{cases}$$ b_i is the i^{th} variable of block b.

Table 3.3 demonstrates for string '1011 0000'. There is only one block at level three and its value is '1011 0000'. This is not an optimal block, thus *f* ('1011 0000') = 0.0. The (value of the) two blocks at level two are '1011' and '0000'. *f* ('1011') = 0.0 because '1011' is not an optimal block but *f* ('0000') = 1.0 because '0000' is an optimal block.

Table 3.3 Example of a HIFF-D fitness calculation

Level	String or Genotype								Level Fitness
	1	0	1	1	0	0	0	0	
1	0.0		1.0		1.0		1.0		3.0
2	0.0				1.0				1.0
3	0.0								0.0
Phenotype									⟨0.0, 1.0, 3.0⟩
Total Fitness									4.0

If HIFF-C's block fitness function was used, *f* ('1011') = (0.5 × 1.0) + (0.5 × 0.0) = 0.5. '1011' is almost an optimal block. It is only one Hamming distance away from the optimal block '1111'. Unlike the HIFF-C fitness function which gives a search algorithm some indication of this, the HIFF-D fitness function which uses an all-or-nothing approach does not. Both '1100' and '1011' have zero level 2 fitness by HIFF-D's fitness function.

HIFF-D is the only HIFF problem in this dissertation that employs the all-or-nothing approach. This approach is an extreme way of ensuring that lower-level interactions are satisfied before higher-level interactions are to create wide fitness saddles for hill climbers, i.e. a hill climber would need to change a large segment of a genotype to the same value at one time.

The all-or-nothing approach makes the differences in fitness contribution between levels unnecessary for ensuring that a HIFF problem is more difficult for hill climbers than genetic algorithms to solve. Watson (2006 p.127): "This all-or-nothing aspect ... means that the resultant intermodule dependencies cannot be described in terms of the sum of pairwise interactions... This has the consequence that we can now change the relative importance of intra- and inter- module interdependencies without the risk of intramodule dependencies being over-powered as discussed earlier."

3.2.3 HIFF-II

$f(b) = \frac{2}{|b|} \times \left(\sum_{i} iff\left(b_i, b_{|b|/2+i}\right) \right)$ where $0 \leq i < |b| / 2$ and $|b|$ is the size of (number of variables in) block b. The previous equation says that the fitness of a block $f\ (b)$ for HIFF-II is the number of variable pairs that have the same value, divided by half the block size. The variables are paired position-wise between the left and right halves of a block.

Figure 3.2 depicts the interaction pattern for a HIFF-II problem with 4 variables. Unlike HIFF-C, the interactions in HIFF-II do not occur between every distinct pair of variables but follow a specific pattern.

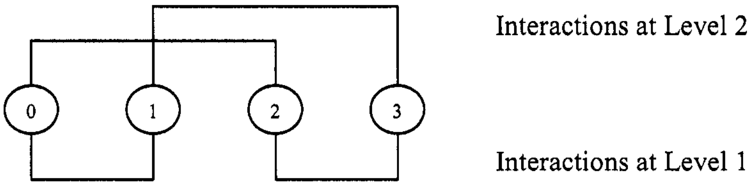


Figure 3.2 Interaction pattern for the HIFF-II problem with 4 variables

Table 3.4 demonstrates by calculating the HIFF-II fitness for string '1000 1100'. As with the other HIFF problems in this chapter, this string of length 8 is decomposed into seven blocks and three levels. At level 2, the (values of the) blocks are '1000' and '1100'. f('1000') = 0.5 because variables 1 and 3 both have the value '0' and since this is the only block inter-dependency which is satisfied, the number of variables pairs that have the same value is one and one divided by half the block size, i.e. two, gives 0.5.

Table 3.4 Example of a HIFF-II fitness calculation

Level	String or Genotype								Level Fitness
	1	0	0	0	1	1	0	0	
1	0.0		1.0		1.0		1.0		3.0
2	0.5				0.0				0.5
3	0.75								0.75
Phenotype									⟨0.75, 0.5, 3.0⟩
Total Fitness									4.25

With fewer variable to variable interactions than HIFF-C, the inter-dependency matrix for HIFF-II is not surprisingly more sparse (has more '0' entries) than that for HIFF-C (Figure 3.1). Figure 3.3 gives the inter-dependency matrix for the HIFF-II problem with 8 variables. The weights of the *iff* constraints for HIFF-II are adjusted so that for problems of the same size, HIFF-II has the same globally optimal phenotype as HIFF-C.

3.2.4 HIFF-M

$f(b) = iff(b_0, b_{|b|/2})$. The previous equation says that fitness of a block $f(b)$ for HIFF-M is 1.0 if the first variable and the variable at $|b|/2$ of block b have the same values, and 0.0 otherwise.

	0	1	2	3	4	5	6	7
0	0	1/2	1/4	0	1/8	0	0	0
1	1/2	0	0	1/4	0	1/8	0	0
2	1/4	0	0	1/2	0	0	1/8	0
3	0	1/4	1/2	0	0	0	0	1/8
4	1/8	0	0	0	0	1/2	1/4	0
5	0	1/8	0	0	1/2	0	0	1/4
6	0	0	1/8	0	1/4	0	0	1/2
7	0	0	0	1/8	0	1/4	1/2	0

Figure 3.3 HIFF-II's inter-dependency matrix, N = 8. Sets of inter-dependencies belonging to the same level are indicated by borders of equal thickness in the matrix. The shaded areas represent two modules.

Like HIFF-C and HIFF-II, the all-or-nothing approach to fitness calculation also does not apply to HIFF-M. But unlike the fitness values for HIFF-C and HIFF-II blocks which are real numbers in [0.0, 1.0], the fitness values for HIFF-M blocks like HIFF-D, are either 0.0 or 1.0 only. Fitness of a HIFF-M block depends on two variables only. Since a pair of variables must either have the same values or have different values, the possible fitness values for a HIFF-M block are either 0.0 or 1.0 only.

A HIFF-M problem has even fewer interactions than HIFF-II. HIFF-M interactions occur between pairs of variables occupying the first location in the left and in the right halves of blocks. Figure 3.4 depicts the interaction pattern for a HIFF-M problem with 4 variables.

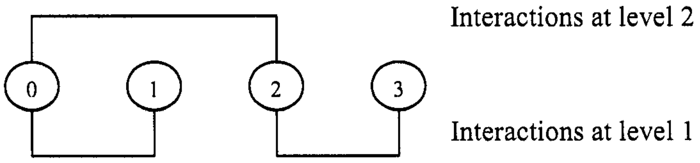


Figure 3.4 Interaction pattern for a HIFF-M problem with 4 variables

Table 3.5 demonstrates the calculation of HIFF-II fitness for string '1000 1100'. As with the other HIFF problems in this chapter, this string of length 8 is decomposed into seven blocks and three levels. At level 2, the (values of the) blocks are '1000' and '1110'. *f* ('1000') = 0.0 because variables 0 and 2 of '1000' do not have the same value. *f* ('1110') = 1.0 because variables 0 and 2 of '1110' have the same value.

Table 3.5 Example of a HIFF-M fitness calculation

Level	Genotype								Level Fitness
	1	0	0	0	1	1	1	0	
1	0.0		1.0		1.0		0.0		2.0
2	0.0				1.0				1.0
3	1.0								1.0
Phenotype									⟨1.0, 1.0, 2.0⟩
Total Fitness									4.0

In Figure 3.5 is the inter-dependency matrix for the HIFF-M problem with 8 variables. It is sparser (has more '0' entries) than the matrix in Figure 3.3 since HIFF-M has fewer variable to variable inter-dependencies than HIFF-II. The weights of the *iff* constraints for HIFF-M are adjusted so that for problems of the same size, HIFF-M has the same globally optimal phenotype as HIFF-C and HIFF-II.

What is interesting about the matrix in Figure 3.5 is now all the non-zero entries have the same value, 1/2, and this applies to all HIFF-M problems of any size in this dissertation. This is a departure from the structure of the other HIFF problems. Since HIFF-M does not use the all-or-nothing approach and accordingly this should make the differentiation of weights between levels important if HIFF-M is to be a solution to the HC < GA problem, the effect of HIFF-M's problem structure on optimizing performance of evolutionary algorithms is examined in Part II.

	0	1	2	3	4	5	6	7
0	0	1/2	1/2	0	1/2	0	0	0
1	1/2	0	0	0	0	0	0	0
2	1/2	0	0	1/2	0	0	0	0
3	0	0	1/2	0	0	0	0	0
4	1/2	0	0	0	0	1/2	1/2	0
5	0	0	0	0	1/2	0	0	0
6	0	0	0	0	1/2	0	0	1/2
7	0	0	0	0	0	0	1/2	0

Figure 3.5 HIFF-M's inter-dependency matrix, N = 8. Sets of inter-dependencies belonging to the same level are indicated by borders of equal thickness in the matrix. The shaded areas represent two modules.

3.3 Design features

This section discusses the design features of the original HIFF problems that Watson (2002) considered important for producing the HC < GA result, and highlights how the designs of the new HIFF problems comply with or depart from the original design features.

3.3.1 Modularity

The cornerstone of the original HIFF problems (HIFF-D and HIFF-C) is modularity. A module is a subset of problem variables which have stronger inter-dependencies with one another than they do with variables not in the same subset. Watson (2006, p.13): "there are subsets of variables in which the variables are more strongly dependent on other variables within their own subset than on variables in other subsets."

The presence of modularity helps ensure that subsets of variables evolve to have the same values. Since HIFF problems have two global optima with equally large basins of attraction, in the absence of global coordination, different subsets of variables will adapt to different global optima. As a result, wide fitness saddles emerge for evolutionary algorithms like hill climbers (an example was given in the preamble of Part I).

For the HIFF problems, stronger inter-dependencies between a pair of variables translate to larger fitness contributions. The modular structure of the original HIFF problems is evident from their inter-dependency matrix (HIFF-D and HIFF-C have the same inter-dependency matrix). The shaded areas in Figure 3.1 mark two modules. Entries within the shaded areas (the intra-module interactions) are larger in value than entries outside the shaded areas (inter-module interactions). This means that the intra-module interactions are stronger than the inter-module interactions. Nonetheless, it is still important to satisfy the inter-module *iff* constraints to solve HIFF problems.

The modularity condition is satisfied by the new HIFF problems. HIFF-II's inter-dependency matrix (Figure 3.3) clearly shows that intra-module interactions are stronger than inter-module interactions. For HIFF-M, although all interactions are equally strong, there are more interactions within the shaded areas of the inter-dependency matrix in Figure 3.5, than outside the shaded areas.

In chapter 9, the Q metric, a measure of modularity proposed by Newman (2006) is used to show that HIFF-M, HIFF-II and HIFF-C have the same level of modularity. Since HIFF-D uses the same inter-dependency matrix as HIFF-C, and Q values are calculated on inter-dependency matrices, HIFF-D has the same Q value as HIFF-C. However, since HIFF-D uses an all-or-nothing approach to fitness calculation and therefore difference in weights of inter-dependencies is unnecessary (section 3.2.2), the exercise of calculating Q values for HIFF-D is really unnecessary to establish that HIFF-D has modular structure. The all-or-nothing approach to fitness calculation ensures HIFF-D's modular structure.

3.3.2 Non-separability

A problem is non-separable if it is not possible to solve the whole problem by solving parts of it independently and then putting the sub-solutions together. Watson et al. (1998): "a *separable* problem is a problem which can be divided into sub-problems each of which has a fixed optimal solution that is independent of how other sub-problems are solved".

To test for non-separability, Whitley et al. (1995) suggest the 'line search algorithm'. Processing one variable at a time, this algorithm enumerates over all possible values for a variable, looking for a value that improves overall fitness the most before moving on to the next variable. If a problem is solved using this algorithm, then it is separable. To solve any of the HIFF problems defined in this chapter using the line search algorithm, one would need to treat the largest module, i.e. the whole HIFF string itself, as one variable. This essentially becomes pure random search and hence the HIFF problems are non-separable.

Non-separability is satisfied in the original HIFF problems by having more than one equally fit optimal solution for each module, but restricting globally optimal solutions to a subset of the possible combinations. That is, a random combination of optimal modules need not produce optimal modules of higher order. For example, '00' and '11' are optimal solutions for size two modules, but optimal solutions for size four modules are '0000' and '1111' only; '0011' and '1100' combinations are not optimal. Watson calls this kind of non-separability *modular interdependency* (2006, Chapter 4).

Because the structure of the new HIFF problems is modular (section 3.3.1) and this modularity does not preclude the importance of inter-module inter-dependencies (globally optimal solutions require satisfaction of all *iff* constraints which includes those between modules), the new HIFF problems are non-separable also, and their non-separability is the result of modular inter-dependency. This makes it possible for the new HIFF problems to be candidate solutions to the HC < GA problem.

Discussion

Modular inter-dependency alone is insufficient to produce a HC < GA result from the HIFF problems. The modularity condition (section 3.3.1, which is satisfied by all HIFF problems defined in this chapter) also needs to be present so that modular inter-dependency behavior can manifest itself during evolution, i.e. modules need to appear before they can be swapped in a genetic algorithm (GA). Hence modular inter-dependency is a dynamic feature and depends on the fitness function to induce the formation of modules. To use the terminology from statistical physics (Naudts and Naudts, 1998): modularity helps to *pin domain walls* at specific positions (i.e. module boundaries) in HIFF strings so that modules can form. For HIFF strings, a *domain* is a consecutive sequence of variables set to the same value, and a domain wall is formed by a pair of adjacent variables set to different values. If domain walls were free to move (i.e. do random walks), a HIFF problem becomes a Two-Max problem and easily solved by hill climbing (Table 2.1).

Consider the HIFF-C genotype ‘0100’. It has two domain walls denoted W1 and W2, located as follows: $0^{W1}1^{W2}00$. If a search algorithm happens to change the value of gene 1 from ‘1’ to ‘0’, W1 and W2 collide and disappear, and a globally optimal genotype ‘0000’ results. However, it is also equally likely for a search algorithm to change the value of gene 0 from ‘0’ to ‘1’. Because the (intra-module) interaction between genes 0 and 1 is stronger (1) than the combined (inter-module) interactions between gene 0 and genes 2 and 3 (1/2), there is a net increase (1/2) in genotype fitness as a result of moving from ‘01 00’ to ‘11 00’. The inter-dependency matrix for HIFF-C, N=4 is given below.

	0	1	2	3
0	0	1/2	1/8	1/8
1	1/2	0	1/8	1/8
2	1/8	1/8	0	1/2
3	1/8	1/8	1/2	0

Genotype ‘11^{W3}00’ has one domain wall W3 and this domain wall is pinned with the help of HIFF-C interaction weights and now we can see clearly two domains (optimal modules ‘11’ and ‘00’). To dissolve W3, it needs to move either to the far left or the far right of the genotype, and this can only happen (without changing the HIFF-C function or using transposition of alleles (Khor, 2007b)), if genes 0 and 1 or alternatively genes 2 and 3 are flipped simultaneously.

Modularity alone is also insufficient to produce a HC < GA result from the HIFF problems. This can be tested by adding a bias to a HIFF problem so that *iff* constraints adapting to a particular global optimum say the all-zeroes string, consistently make larger fitness contributions than *iff* constraints adapting to other global optima. In the limit, this bias destroys modular inter-dependency and the problem becomes One-Max, which is separable and easily solved by hill climbing (Table 2.1). This experiment was done in (Watson and Pollack, 1999).

The behavior described by modular inter-dependency is similar to the notion of *frustration* found in Ising type models (section 2.4.3). Frustration happens when more than one equally attractive move is available to a search algorithm but only a subset of the moves can lead to a globally optimal solution (Naudts, 1998). The choice between equally attractive moves could involve satisfying one of a set of equally weighted but conflicting constraints, or deciding between two or more alternative solutions. The search algorithm then needs to make a random choice. The choice is random since there is no additional information to guide the search algorithm or to break the *symmetry* of the situation. Symmetry increases problem difficulty because there is no unambiguous fitness gradient to follow. Many classical NP-hard optimization problems suffer from this condition (Prügel-Bennett, 2004).

The HIFF problems studied in this dissertation have *bit-flip symmetry* (van Hoyweghen et al., 2002a). A symptom of this is 0110 and 1001 are equally fit (more examples in Table 3.1). Bit-flip symmetry is a special case of symmetry due to fitness invariance on permutations of the alphabet (van Hoyweghen et al. 2002a). To see how bit-flip symmetry can create frustration for HIFF problems, consider replacing '0111' with '1000' in the following HIFF-D string: '1111 0111'. This replacement does not change the string's fitness but the search is taken further away in Hamming space from the all-ones global optimum. A similar situation results if '1100' is replaced with '0000' in the HIFF-M string '1100 1111'. In this second example, the *non-inferior building-blocks* (van Hoyweghen et al., 2001) are '11** ****' and '00** ****'. Non-inferior building-blocks are schemata which are not less fit than any other schemata of the same order and the same defined locations. For example, '00** ****' and '11** ****' are both fitter than (superior to) '01** ****' and '10** ****', but not less fit than (inferior to) each other.

Indiscriminate replacements of non-inferior building-blocks, coupled with uneven sampling of the population during parent selection and/or survival selection, can lead to the disappearance of some non-inferior building-blocks from the population. If this disappearance is later discovered by a search algorithm to be premature and the search algorithm is unable to reconstruct the lost non-inferior building-blocks, the population has a *synchronization problem* (van Hoyweghen et al., 2002a).

The synchronization problem is premature convergence of a population due to irrecoverable loss of non-inferior building-blocks. Genetic algorithms without explicit diversification measures have difficulty solving HIFF-D due to the synchronization problem. Increasing synchronization constraints e.g. by increasing the number or strength of constraints between modules (and thus reducing modularity), and introducing a backbone variable (e.g. a fitness bias towards one globally optimal solution if this can be known a priori) to reduce symmetry, decreases the severity of the synchronization problem induced by a search problem. While these changes might make a problem less difficult for a genetic algorithm to solve because of population diversity issues, they also make the problem easier for hill climbers.

3.3.3 Hierarchy

A hierarchic system "is composed of subsystems that, in turn, have their own subsystems, and so on" (Simon, 1969 p. 86). Watson (2006, p.14): "Clearly, the structure of dependencies in this example system [referring to the matrix in Figure I.C] are not just modular but also hierarchically modular – that is, there are clusters and subclusters of more strongly interdependent variables. This potentially allows an adaptive mechanism that is capable of exploiting modularity to decompose the system recursively or equivalently, to compose together subsolutions repeatedly. This kind of hierarchical decomposability is closely related to the notion of nearly decomposable systems discussed by Simon (1969)".

In the original HIFF problems, optimal modules form the building-blocks (schemata with above average observed fitness) of the Building-block Hypothesis (Goldberg, 1989). The idea is a genetic algorithm would be able to exchange optimal modules between genotypes via the crossover operator and eventually construct a globally optimal genotype for a HIFF problem. The hierarchical structure of the original HIFF problems assures that building-blocks of intermediate sizes exists to act as stepping stones (i.e. stable intermediate structures) towards the construction of optimal solutions of order N from lower-order solutions. This design is generally adhered to by the new HIFF problems as well.

For all HIFF problems in this chapter, optimal modules cannot form before their constituent modules become optimal. The reason for this stems from the definition of interactions for a module which includes all interactions amongst variables of a module (Part I). Since modules are nested, the interactions of a module necessarily include the interactions of its constituent modules. But the simple aggregation of optimal modules does not necessarily make an optimal higher-order (larger) module due to modular-interdependency.

However, the hierarchical structure of the original HIFF problems increases problem difficulty for hill climbers with limited reach, by creating exponential number of local optima and fitness saddles of widths up to N / 2 between these local optima (Watson, 2006 p. 151). Table 3.6 enumerates strings which are locally optimal for the original HIFF problems of size eight. Distances are in Hamming units. A hill climber with a maximum reach of 2^{λ} - 1 faces a fitness saddle of at least 2^{λ} wide and $2^{N/2^{\lambda}}$ local optima only two of which are globally optimal. For a hill climber with a maximum reach of one, the size of the widest fitness saddle is N / 2 and the total number of local optima is $2^{N/2}$.

Table 3.6 Local optima for HIFF-D and HIFF-C problems, N=8

Level λ	Fitness saddle width 2^{λ}	Number of local optima $2^{N/2}\lambda$	Locally optimal strings for a hill climber with a maximum reach of 2^{λ} - 1
3	8	2	0000 0000, 1111 1111
2	4	4	0000 0000, 0000 1111, 1111 0000, 1111 1111
1	2	16	0000 0000, 0000 0011, 0000 1100, 0000 1111, 0011 0000, 0011 0011, 0011 1100, 0011 1111, 1100 0000, 1100 0011, 1100 1100, 1100 1111, 1111 0000, 1111 0011, 1111 1100, 1111 1111

A similar role is played by the hierarchical structure of the new HIFF problems. The above calculation of local optima and fitness saddles apply also to the HIFF-II problem. But the calculation is significantly different for HIFF-M. For example, by inspecting the strings and their fitness values in Table 3.1, the local optima found for HIFF-II is the same as those for HIFF-D and HIFF-C, but no local optimum is found for HIFF-M.

The correlation between HIFF-M and HIFF-C fitness values (M-C) is weaker than the correlation between HIFF-II and HIFF-C fitness values (II-C). The correlation between HIFF-D and HIFF-C fitness values (D-C) is the strongest of the three. For N=8, the correlation coefficients for M-C, II-C and D-C are 0.8073, 0.9322 and 0.9828, respectively. The position of strings relative to each other in Hamming space is not affected by the fitness function, but whether a string becomes a local optimum is. Thus, a locally optimal string for HIFF-C is likely to be a local optimum for another HIFF function if HIFF-C and the other HIFF function assign fitness values which are positively and highly correlated with each other.

An experiment – FMHC

The objective of this experiment is to test whether there are differences in the mutation landscape of the HIFF problems as the previous discussion suggests. The test is conducted using the Flip Mutation Hill Climber (FMHC).

FMHC mutates a sequence of genes located between two positions x and y. The genotype is treated as a ring. Position x is chosen uniformly at random from [0, N-1]. Position y is determined by the mutation rate (P_m) as follows: $y = (x + k) \bmod N$ where k is chosen uniformly at random from [1, $P_m \times N$]. FMHC mutates by assigning to a gene a new value based on the gene's existing value (i.e. bit-flip mutation when the {0, 1} alphabet is used). FMHC keeps the mutant (new) genotype if it is fitter than or as fit as the parent (current) genotype. FMHC is similar to the recombinative hill climber described in (Watson, 2001). The recombinative hill climber was used to approximate how a GA might use crossover to avoid local optima. Thus it is no surprise that FMHC succeeds on HIFF problems[6].

The results for FMHC are reported in Table 3.7. The maximum number of function evaluations is 250,000. It is possible to solve HIFF-M problems with a lower mutation rate (P_m) than the other three HIFF problems. A mutation rate of 0.25 is sufficient for FMHC to be 100% successful on HIFF-M problems. But all three HIFF-D, HIFF-C and HIFF-II problems need a mutation rate of 0.5.

[6] FMHC is a specialized search algorithm or a 'cheat' because it no longer treats a HIFF problem as a black-box. Instead, FMHC is using knowledge about HIFF's problem structure (modularity) and the relationship between its global optima (they are complements of each other). For more realistic problems, a search algorithm would not be privy to this kind of 'inside' information at the start. At best, the bias of a search algorithm would coincide with or be coaxed through learning to approximate the bias needed to solve a problem. What the HC < GA result for HIFF problems intend to prove is that the *natural* bias of a GA is more suitable for solving HIFF problems than the *natural* bias of a HC. And consequently, genetic algorithms are distinct from hill climbers.

Table 3.7 Number of successful FMHC runs out of 30

		HIFF-D			HIFF-C	HIFF-II	HIFF-M		
P_m	N	64	128	256	128	128	64	128	256
0.5		**30**	**30**	**30**	**30**	**30**	30	30	30
0.25		14	14	18	22	25	**30**	**30**	**30**
0.125		-	-	-	-	-	27	28	26
0.0625		-	-	-	-	-	19	21	20

The inference from this result is that for a hill climber with a maximum reach of one, the size of the widest fitness saddle in a HIFF-M problem is N / 4, which is half the size of that for the other three HIFF problems. Thus, there are differences in the mutation landscape of the HIFF problems as the previous discussion suggests.

3.4 Relationship between hierarchical levels

In this section, the relationship between hierarchical levels in the four HIFF problems is studied. The relationship between hierarchical levels is one dimension in which the new HIFF problems differ significantly from the original HIFF problems. This aspect helps explain why HIFF-M has a narrower widest fitness saddle (section 3.3.3).

3.4.1 Relative speed of optimization between levels

In the original HIFF problems, optimal blocks cannot form before their constituent blocks become optimal because the inter-dependencies of a block imply the inter-dependencies of its constituent blocks. For example, in Figure 3.1, the inter-dependencies for block *b* with variables 0, 1, 2 and 3 are between variable pairs (0, 2), (0, 3), (1, 2) and (1, 3). For *b* to be optimal, all four of its *iff* constraints must be satisfied, but due to transitivity, this also means that the *iff* constraints between variable pairs (0, 1) and (2, 3) are satisfied. The *iff* constraints between variable pairs (0, 1) and (2, 3) belong to the blocks nested within *b*. Therefore, the inter-dependencies of a block (*b* in the example) imply the inter-dependencies of its constituent blocks.

Since the inter-dependencies of blocks at lower levels make larger fitness contributions in the original HIFF problems, satisfaction of lower level *iff* constraints would normally trump that of higher level *iff* constraints. As a result, blocks at lower levels would optimize faster than blocks at higher levels. This implies, since a level is optimal when all its blocks are optimal, that *lower levels would optimize faster than higher levels in the original HIFF problems*.

Having lower levels optimize faster than higher levels increases problem difficulty for search algorithms because it compels optimal modules to form independent of their context. Information about context is important for solving HIFF problems because of modular inter-dependency. Contextual information, if decipherable from fitness values, can guide a search algorithm towards a particular global optimum and thus reduce frustration. For hill climbers, the lack of contextual information leads to the manifestation of wide fitness saddles. For genetic algorithms, the lack of contextual information can create the synchronization problem (section 3.3.2).

The difference in optimization speed between levels in the original HIFF problems is made more pronounced in HIFF-D because in effect, the all-or-nothing approach to fitness calculation assigns zero fitness contribution to *iff* constraints of a block until the block is optimal. So while the block is not optimal, there is little choice but to increase string fitness by satisfying the *iff* constraints of lower-level (constituent) blocks. One consequence of this, besides creating wide fitness saddles for hill climbers and setting up genetic algorithms for the synchronization problem, is it is not possible to solve a HIFF-

D problem even when the satisfaction of higher level inter-dependencies is given priority by a search algorithm. This does not hold for HIFF-C however, as the experiment in section 3.4.2 demonstrates.

In contrast, in both the HIFF-II and the HIFF-M problem, it is possible for optimal blocks to form before their constituent blocks become optimal because the inter-dependencies of a block do not imply the inter-dependencies of its constituent blocks. Since a level is optimal when all its blocks are optimal, it is also possible in both the HIFF-II and the HIFF-M problem, for a level to achieve optimality before levels below it do. To illustrate, the blocks of '1100 1100' at levels 1 and 3 are optimal by both the HIFF-II and the HIFF-M fitness function, but the blocks at level 2 are not. More examples of strings where higher levels are optimal before lower levels can be found in Table 3.1. Hence, *it is not necessarily the case that lower levels would optimize faster than higher levels in the new HIFF problems.*

This has a direct consequence for evolutionary algorithm difficulty because inter-dependencies at higher levels provide longer range contextual information. The sustained satisfaction of long range inter-dependencies (the inter-module interactions), especially at an early stage in evolution when large optimal modules have not yet formed, can decrease search difficulty for evolutionary algorithms. In particular, the early and continued satisfaction of inter-module inter-dependencies can help hill climbers to avoid wide fitness saddles.

However, it would be more difficult for higher levels to optimize faster than lower levels in a HIFF-II problem than in a HIFF-M problem because HIFF-II's *iff* constraints at lower levels make larger fitness contributions than the *iff* constraints at higher levels.

In contrast, all inter-dependencies are equally weighted in HIFF-M (section 3.4.3 discusses the consequence of this further). Moreover, HIFF-M has fewer *iff* constraints per level at higher levels than HIFF-II. Therefore, of all the HIFF problems discussed in this chapter, it would be easiest for higher levels to optimize faster than lower levels in the HIFF-M problem.

3.4.2 Inter-level inter-dependency and top-down inter-level conflict

Since in the original HIFF problems (HIFF-D and HIFF-C), the inter-dependencies of a block imply the inter-dependencies of its constituent blocks (subsection 3.4.1), lower-level inter-dependencies are superfluous for constructing a globally optimal string if the satisfaction of higher level inter-dependencies are given priority and the all-or-nothing approach is not used. The experiment below in this subsection confirms this.

However, this is not true for the new HIFF problems (HIFF-II and HIFF-M) where the inter-dependencies of a block do not imply the inter-dependencies of its constituent blocks. In contrast to the original HIFF problems, no one level in the new HIFF problems possesses a set of *iff* constraints sufficient for constructing a globally optimal string. This creates a dependency between levels. This *inter-level dependency is bi-directional* in the sense that not only do the blocks at lower levels depend on the *iff* constraints at higher levels for contextual information (as is the case in the original HIFF problems), now blocks at higher levels also depend on the *iff* constraints at lower levels to fill in the gaps in information about the complete set of *iff* constraints for constructing a globally optimal string.

In Table 3.8, sets of level optimal strings are compared. Level optimal strings are strings that satisfy all the *iff* constraints of a level, i.e. a level optimal string for level 3 satisfies all level 3 inter-dependencies, and may or may not satisfy inter-dependencies of other levels. The set of level optimal strings for HIFF-D and HIFF-C at the highest level (level 2 in Table 3.8) comprise only globally optimal strings, while for HIFF-II and HIFF-M, the set includes non-globally optimal strings also. In the case of HIFF-II and HIFF-M, the inter-dependencies of levels other than the highest level are required also to whittle down the set of level optimal strings at the highest level to globally optimal strings only. Hence, the levels of HIFF-II and HIFF-M are said to be inter-dependent.

Table 3.8 Level optimal strings for HIFF problems, N = 4

Level λ	HIFF-D and HIFF-C		HIFF-II		HIFF-M	
	Size $2^{N/2^{\lambda}}$	Strings	Size $2^{N/2}$	Strings	Size $2^{N(1-1/2^{\lambda})}$	Strings
2	2	0000, 1111	4	0000, 0101, 1010, 1111	8	0*0*, 1*1*
1	4	0000, 0011, 1100, 1111	4	0000, 0011, 1100, 1111	4	0000, 0011, 1100, 1111

Section 3.4.1 stated that the early and sustained satisfaction of higher level inter-dependencies can decrease search difficulty for evolutionary algorithms especially hill climbers. This is because higher-level inter-dependencies provide long range information which can help guide adaptation at the local level in individual modules optimize towards the same global optimum. An exception to this is when the satisfaction of higher level inter-dependencies obstructs the satisfaction of lower level inter-dependencies, and as a result prevents a string from becoming globally optimal. The HIFF-II problem is an example of such an exception. In this dissertation, problems like HIFF-II are said to have *top-down inter-level conflict*.

Problems with top-down inter-level conflict cannot be solved by prioritizing the satisfaction of higher level *iff* constraints over lower level *iff* constraints, as the experiment below confirms. Giving optimization priority to higher levels over lower levels is a form of conflict mediation (Michod, 1999).

HIFF-M does not have top-down inter-level conflict. Neither do HIFF-D and HIFF-C, but for a different reason – their levels as discussed previously are not inter-dependent.

Experiment – RMHC2

The objectives of this experiment are to confirm that (i) the HIFF-C problem is solvable using only inter-dependencies of the highest level, and (ii) HIFF-II has top-down inter-level conflict but HIFF-M does not. To achieve these objectives, the RMHC2 hill climber is used.

RMHC2 mutates *k* genes of a genotype chosen uniformly at random each time and replaces the parent (current) genotype with the mutant (new) genotype if the multi-level selection scheme evaluates the mutant genotype to be fitter than or as fit as the parent genotype. The integer *k* is chosen uniformly at random from $[1, P_m \times N]$ and P_m is the mutation rate. Gene mutation is by reassigning the gene with a value chosen uniformly at random from the alphabet. Fitness of genotypes in RMHC2 is expressed as phenotypes (defined in the preamble for Part I). Phenotypes are compared by RMHC2 using the multi-level selection scheme which compares fitness level by level in the order specified by the sieve component. The comparison terminates as soon as un-equality is detected and the genotype with the larger fitness value is considered the fitter.

For example, suppose the HIFF problem is of size eight, the sieve is $\langle 2, 1, 3 \rangle$, the phenotype of the parent genotype is $p_1 = \langle x, y, z \rangle$ and the phenotype of the mutant genotype is $p_2 = \langle a, b, c \rangle$. Then RMHC2 first compares b with y, then c with z, and finally a with x. If $(b > y)$ or $(b = y$ and $c > z)$ or $(b = y$ and $c = z$ and $a > x)$ or $(b = y$ and $c = z$

and *a* = *x*), then the mutant genotype replaces the parent genotype. Otherwise, the mutant genotype is discarded.

Table 3.9 demonstrates with two pairs of HIFF-C genotypes. F*i* is fitness of genotype at level *i*, and F is (total) genotype fitness.

Table 3.9 RMHC2 multi-level selection with sieve ⟨3, 2, 1⟩

Genotype	HIFF-C Phenotype			F	Fitter genotype
	F3	F2	F1		
0000 0010	**0.75**	1.5	3.0	5.25	0000 0010
0000 0011	**0.50**	1.0	4.0	5.50	
1011 0011	0.5	**0.5**	3.0	4.0	1011 0110
1011 0110	0.5	**1.0**	1.0	2.5	

For the first hypothesis, that the HIFF-C problem is solvable using only inter-dependencies of the highest level, RMHC2's multi-level selection scheme compares phenotypes by their leftmost (highest level) fitness value only. This method (with P_m = 0.0625) solved HIFF-C (N = 128) in all 30 runs. On average, 1,922 function evaluations with a standard deviation of 676 were used to evolve an optimal HIFF-C genotype. Thus, the hypothesis is confirmed. HIFF-C problem is solvable using only inter-dependencies of the highest level. And therefore, the levels of HIFF-C are not inter-dependent on each other for producing globally optimal solutions.

For the second hypothesis, that HIFF-II has top-down inter-level conflict but HIFF-M does not, RMHC2's multi-level selection scheme compares phenotypes level by level from the highest level down, i.e. the sieve is ⟨*i*, *i*-1..., 2, 1⟩ where $N = 2^i$ and $i \in \mathbf{Z}^+$. Thus this RMHC2 gives optimizing priority to higher levels over lower levels. Since HIFF-II has top-down inter-level conflict, this RMHC2 is not expected to be successful for HIFF-

II. But HIFF-M has no top-down inter-level conflict and so should be solved by this RMHC2.

This RMHC2 is not successful in any of the 30 HIFF-II runs but is successful in all of the 30 HIFF-M runs. On average, 2,522 function evaluations with a standard deviation of 878 were used to evolve an optimal HIFF-M string. The maximum number of function evaluations allowed is 1 million. Thus, the hypothesis is confirmed. HIFF-II has top-down inter-level conflict while HIFF-M does not. Giving priority to higher levels to optimize is a successful strategy for the HIFF-M problem, but not for the HIFF-II problem.

The HIFF-II problem is solvable using a multi-level selection strategy if no level is permitted to increase its fitness at the expense of another level. This algorithm called RMHC3 is described in Khor (2007a). The RMHC2 algorithm, the HIFF-II problem and the differences between the HIFF problems from a level perspective is discussed in more detail in Khor (2007), Khor (2007a), Khor (2007b) and Khor (2007c).

3.4.3 HIFF-M: consequence of equally weighted interactions

In addition to having fewer *iff* constraints per level than HIFF-II, the HIFF-M problem has another feature which facilitates the satisfaction of long range constraints – all its interaction weights are the same. Thus, it is possible to substitute the satisfaction of a lower level *iff* constraint with a higher level one without adversely affecting total fitness for a string. But the reverse is also true. The satisfaction of a higher level *iff* constraint may also be replaced with a lower level one. This produces frustration and prevents HIFF-M from becoming a Two-Max problem.

Figure 3.6 illustrates the *iff* constraints HIFF-M places between four variables. The c_1 *iff* constraint directly encloses c_2 and c_3, and c_2 and c_3 are directly enclosed by c_1. String '11 00' satisfies *iff* constraints c_2 and c_3 but not c_1, and has a fitness value of 2.0. String '01 00' also has a fitness value of 2.0, but it satisfies c_1 and c_3 but not c_2. In the transformation of '11 00' to '01 00', satisfaction of c_2 is substituted with satisfaction of c_1. The reverse transformation from '01 00' to '11 00', is possible also. A one for one substitution of satisfied *iff* constraints between levels without sacrificing fitness is possible for HIFF-M strings because interactions at all levels weigh the same. In the other HIFF problems, more than one higher level *iff* constraint need to be satisfied to compensate for the fitness loss from un-satisfying a lower level *iff* constraint because lower level interactions make larger fitness contributions.

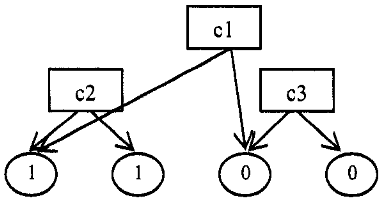


Figure 3.6 Interactions in a HIFF-M problem with four variables

The ease with which satisfied lower level *iff* constraints may be substituted with satisfied higher level *iff* constraints helps to explain why HIFF-M has a narrower widest fitness saddle than the other HIFF problems (section 3.3.3). The following explanation uses HIFF-D but applies also to HIFF-C and HIFF-II, and is with reference to a hill climber with a maximum reach of one Hamming unit.

For HIFF-D, the widest fitness saddles are located at strings of the form $0^{N/2}1^{N/2}$ and $1^{N/2}0^{N/2}$. Let T denote the set of such strings. Strings in T have a fitness value of N-2;

they satisfy all *iff* constraints except those at the highest level. For HIFF-M, although a hill climber may visit a genotype in *T*, the hill climber *may* be able to escape having to cross the N/2 wide fitness saddles directly via one of the $\binom{N-1}{N-2} \times 2 - 2$ equally fit genotypes not in *T* but less than N/2 HD away. *P* is used to denote the set of strings that are N – 2 fit and not in *T*. *P* is empty for HIFF-D, HIFF-C and HIFF-II.

T
1111 0000
0000 1111

P	
0000 0001	1111 1110
0000 0011	1111 1100
0000 0100	1111 1011
0001 0000	1110 1111
0011 0000	1100 1111
0100 0000	1011 1111

Figure 3.7 P and T sets for HIFF-M, N=8

A hill climber with a maximum reach *r* of < N / 2 will be stuck at a genotype in *T* only if it cannot at least make the minimal number of changes necessary *w* to move to a point in a non-empty *P*. The question is: what is *w* for HIFF-M? To move from a genotype in *T* and continue to satisfy N – 2 *iff* constraints, a genotype must satisfy c_1 and in exchange un-satisfy one other *iff* constraint. Satisfying c_1 affects *iff* constraints that involve either gene 0 or gene N/2. To keep the number of necessary changes to a genotype to a minimum, it is clear that we need to un-satisfy one of the *iff* constraints directly enclosed by c_1, that is either c_2 or c_3, and continue to satisfy all other *iff* constraints. The *iff* constraints c_2 and c_3 involve the variable pairs (0, N/4) and (N/2, 3N/4) respectively. Therefore, *w* is N/4 for HIFF-M.

Figure 3.8 illustrates this process. In this example, satisfaction of an *iff* constraint at level 2 (c_2), is exchanged for a satisfied *iff* constraint at level 1 (c_1) by flipping the first two bits.

String or Genotype	HIFF-M fitness	*iff* constraints satisfied
1111 0000	6	c2, c3, c4, c5, c6, c7
0111 0000	5	c1, c3, c5, c6, c7
0011 0000	6	c1, c3, c4, c5, c6, c7

Figure 3.8 Constraint substitution: c1 replaces c2 with no net change to HIFF-M fitness

3.5 Chapter Summary

Section 3.1 defined the basic HIFF fitness function. Section 3.2 defined the HIFF-C, HIFF-D, HIFF-II and HIFF-M problems. Section 3.3 compared the design of the new HIFF problems (HIFF-II and HIFF-M) with that of the original HIFF problems (HIFF-D and HIFF-C). It found that in general, the design of the new HIFF problems are similar to that of the original HIFF problems. Both the new and original HIFF problems (i) have modular structure (section 3.3.1), (ii) are non-separable due to modular-interdependency (section 3.3.2), and (iii) have many local optima and wide fitness saddles for hill climbers (section 3.3.3). However, the discussion in section 3.3.3 also found that HIFF-M has a narrower widest fitness saddle than the other three HIFF problems. An experiment with the Flip Mutation Hill Climber (FMHC) was carried out to confirm that there are differences in the mutation fitness landscapes of the HIFF problems (section 3.3.3).

Section 3.4 examined the HIFF problems from a level point of view and found fundamental differences between the original HIFF problems and the new HIFF problems. The first difference is related to the relative speed at which hierarchical levels could optimize (section 3.4.1). While it is not possible for higher levels to optimize faster than lower levels in the original HIFF problems, faster optimization of higher levels are not out of the question in the new HIFF problems, in particular HIFF-M. The implication of this for search (evolutionary) algorithm difficulty is discussed.

The second difference is related to inter-level inter-dependency and top-down inter-level conflict (section 3.4.2). The levels in the new HIFF problems are inter-dependent on each other since no one level has the complete set of *iff* constraints to produce a globally optimal solution. This is not so in the original HIFF problems. While prioritizing the optimization of higher levels can help reduce search (evolutionary) algorithm difficulty, this is provided satisfaction of higher level inter-dependencies does not hinder satisfaction of lower level inter-dependencies and prevent the evolution of a globally optimal string. HIFF-II is a problem where this obstruction is possible and it is an example of a problem with top-down inter-level conflict. HIFF-M does not have top-down inter-level conflict and helping higher levels to optimize before lower levels is a successful strategy for solving HIFF-M. Experiments with RMHC2 (section 3.4.2) were conducted to confirm that (i) the HIFF-C problem is solvable using only inter-dependencies of the highest level, and (ii) HIFF-II has top-down inter-level conflict but HIFF-M does not. RMHC2 is a random mutation hill climber that uses a multi-level selection scheme to decide if a mutant genotype replaces its parent genotype.

Finally, section 3.4.3 discusses the consequence of HIFF-M's equally weighted interactions which helps explain why HIFF-M has a narrower widest fitness saddle for hill climbers with a radius of one Hamming distance than the other HIFF problems in this chapter. This discovery prompts further investigation in the next chapter and in Part II of HIFF-M's ability to distinguish the evolutionary capabilities of genetic algorithms from hill climbers.

4. Preliminary Analysis of HIFF problems

This chapter does a preliminary investigation of the HIFF problems (HIFF-D, HIFF-C, HIFF-II and HIFF-M) using analytical tools to better understand the differences between the HIFF problems. The findings of this preliminary investigation lead us to expect HIFF-M to exhibit different evolutionary dynamics from the other three HIFF problems, and help us to formulate specific hypotheses for the experiments in Part II.

4.1 Site-wise optimization (SWO)

This section computes and compares the *SWO* values (section 2.5.1) of the HIFF problems (HIFF-D, HIFF-C, HIFF-II and HIFF-M) to assess if and how much differences in their problems structures (i.e. differences in inter-dependency matrices) affect the amount of epistasis. Population size (PS) is N for each *SWO* value in Table 4.1, that is N random genotypes are used in each run. Two sets of runs were made, one with a sample size of 5 and another 30. The *SWO* values are not dependent on sample size.

Table 4.1 *SWO* values

N	Averaged over 5 runs				Averaged over 30 runs			
	HIFF-D	HIFF-C	HIFF-II	HIFF-M	HIFF-D	HIFF-C	HIFF-II	HIFF-M
128	0.9949 (0.0021)	0.9949 (0.0021)	0.9949 (0.0021)	0.9845 (0.0019)	0.9946 (0.0019)	0.9946 (0.0019)	0.9946 (0.0019)	0.9845 (0.0024)
256	0.9973 (0.0005)	0.9973 (0.0005)	0.9973 (0.0005)	0.9923 (0.0007)	0.9974 (0.0005)	0.9974 (0.0005)	0.9974 (0.0005)	0.9924 (0.0009)

Standard deviations are in parentheses.

No significant differences in *SWO* values are detected between the HIFF-D, HIFF-C and HIFF-II runs for the same problem size, N. The significance probability that there is no difference between HIFF-M and any one of HIFF-D, HIFF-C or HIFF-II SWO values averaged over 30 runs for the same N is 0.000000% using the one-tailed t-test. Since this significance probability is less than 1%, HIFF-M's *SWO* value is smaller than any one of HIFF-D, HIFF-C or HIFF-II *SWO* values averaged over 30 runs for the same N. As such, HIFF-M is less epistatic than the other HIFF problems.

The *SWO* value for HIFF-II is unexpected because HIFF-II has fewer direct inter-dependencies between its variables than HIFF-C but HIFF-II has the same *SWO* value as HIFF-C. The result for HIFF-M is less unexpected, and indicates that the HIFF-M problem may produce different EA optimizing performance than the other three HIFF problems.

4.2 Fitness Distance Correlation (FDC)

Correlation is a measure of linear dependence. The sample correlation coefficient *r* between two random variables y1 and y2 is the sample covariance divided by the product of the sample standard deviations for y1 and y2 to remove scaling effects.

$$r=\frac{\sum(y1-\bar{y}1)(y2-\bar{y}2)/(n-1)}{s1s2}$$

If y1 and y2 were independent, their covariance and therefore correlation coefficient would be zero. The converse is not true.

Fitness Distance Correlation (FDC) as its name implies measures the correlation as defined above between fitness and distance from a nearest global optimum. FDC values are bounded within [-1.0, 1.0]. FDC was introduced by Jones and Forrest (1995) as a measure of problem difficulty for GAs. Intuitively, FDC values indicate whether fitness increases as distance to a nearest global optimum decreases. If it does, there is a strong negative correlation between fitness and distance, and the FDC value is -1.0. For example, both One-Max and Two-Max have FDC of -1.0. If represented properly, all fitness gradients lead to a global optimum in One-Max and Two-Max. A search problem with a positive FDC value is categorized as misleading. When there is no clear relationship between fitness and distance, the FDC value is uninformative and the actual scatter plot can provide additional information in some instances.

Base on data from several test problems, Jones and Forrest (1995) surmise that the FDC measure (all calculated using Hamming distance) is a "reliable, although not infallible, indicator of GA performance on a wide range of problems." They presume that the accuracy of FDC values would improve if distances were measured "*according to the operator in use by the algorithm*". The counter-example test problem constructed by Altenberg (1997) exploits just this. The test problem is GA-easy[7] when distance is measured in terms of the crossover operator, but GA-difficult according to the FDC value which is uninformative when Hamming distance is used. A more detailed critique of FDC and epistasis measures can be found in (Naudts, 1998). In general, finding one universal measure, which is computationally less intensive than running the EA itself, to predict EA-difficulty, is very challenging if not arguably intractable (Guo and Hsu, 2003). In the name of using 'a simple tool that does the job', the SWO and FDC measures are used in this dissertation.

Figure 4.1 plots the FDC and FDC-X values for HIFF problems of sizes 8 and 16. FDC-X is a measure introduced in this dissertation to gauge crossover-difficulty for HIFF problems. It is identical to FDC in all respects except *crossover distance* is used in place of Hamming distance.

[7] In this context, 'GA-easy' means that the proportion of search space sampled by a 'simple genetic algorithm' to find a global optimum is orders of magnitude smaller than what would be required by a random search. The maximum string length in Altenberg's experiments is 50 bits. Out of 200 runs, the SGA found the global optimum 41% of the time. The reason given for run failures is loss of population diversity.

Altenberg (1997) defined crossover distance (XD) as the minimum number of crossover operations needed to transform a pair of complementary genotypes to the global optimum and its complement. When this definition is applied to Altenberg's counter-example test problem and 1-point crossover is the recombination operator, crossover distance essentially becomes the number of discontinuities between gene values. For example there are one and two discontinuities in '0111' and '0010' respectively. XD assumes that the parent pair comprises complementary genotypes. Real crossover distances will depend on the genotype makeup of the population.

Like the HIFF problems, the building-blocks of Altenberg's counter-example test problem are segments of consecutively located genes whose values agree with each other and the global optimum is a genotype with maximally similar values and thus can have zero discontinuities and zero XD. This similarity makes it easy to apply Altenberg's discontinuity method to the HIFF problems. However, the application is modified accordingly to account for the 1-2 point crossover operator used by GAs in this dissertation. 1-2 point crossover operator permits both one and two point crossovers (Part II). For example, genotypes '0001', '0010', '0011', '0110' and '0111' all have one crossover distance (XD); and the XD for '0101' is two.

In Figure 4.1, the FDC-X values of a HIFF problem are much smaller (closer to -1.0) than the FDC values. This indicates that the HIFF problems are crossover-easy and mutation-hard. A problem is *crossover-easy* if it is easily solved by an EA with crossover as the primary search operator. A problem is *mutation-hard* if it is difficult to solve by an EA with mutation as the primary search. Since the objective of this chapter is to find HC

< GA problems, the crossover-easy and mutation-hard nature of the new HIFF problems fits the bill.

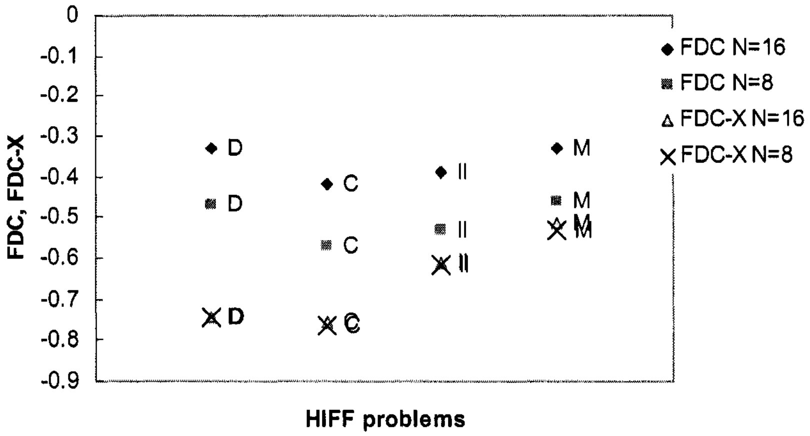


Figure 4.1 FDC and FDC-X values for HIFF problems, N = 8 and N = 16

Within each HIFF problem, the FDC-X values are much less influenced by changes in problem size than the FDC values. As problem size increases, so does the FDC value for each problem (they get closer to 0.0 from -1.0). Hence, the effectiveness of mutation as a search operator is expected to be less scalable than crossover.

The FDC-X values for the new HIFF problems (HIFF-II and HIFF-M) are larger (closer to 0.0) than the values for the original HIFF problems (HIFF-D and HIFF-C). This means that the new HIFF problems are less crossover-easy than the original HIFF problems.

The gap between FDC values and FDC-X values for problems of the same size are wider for the original HIFF problems (HIFF-D and HIFF-C) than for the new HIFF problems (HIFF-II and HIFF-M). The gap is widest for HIFF-D and much narrower for HIFF-M which implies that HIFF-D makes a greater distinction between the optimizing performances of mutation and crossover than HIFF-M. In other words, *HIFF-M is less crossover-easy and less mutation-hard than HIFF-D.*

The differences between the HIFF problems in terms of crossover and mutation difficulty become clearer when fitness is aggregated and correlated with averaged distance. Let $F_{[a, b)}$ denote all fitness values in [a, b) and $G_{[a, b)}$ denote the set of all genotypes *g* such that $a \leq F(g) < b$. The averaged distance corresponding to fitness level $F_{[a, b)}$ is the normalized distance (to a nearest global optimum) of all genotypes in $G_{[a, b)}$. The FDC-$\mathcal{M}$ and FDC-$\mathcal{X}$ measures are calculated with averaged distances. For FDC-$\mathcal{M}$, distance is measured in Hamming units. For FDC-$\mathcal{X}$, crossover distance as defined previously is used.

Figure 4.2 plots the FDC-$\mathcal{M}$ and FDC-$\mathcal{X}$ values for HIFF problems of sizes 8 and 16. The FDC-$\mathcal{M}$ values for HIFF-M is noticeably smaller (and close to -1) than HIFF-D's. Also, of all the HIFF problems, HIFF-M shows the least distinction between its FDC-$\mathcal{M}$ and FDC-$\mathcal{X}$ values. This further supports the statement that HIFF-M is less crossover-easy and less mutation-hard than HIFF-D.

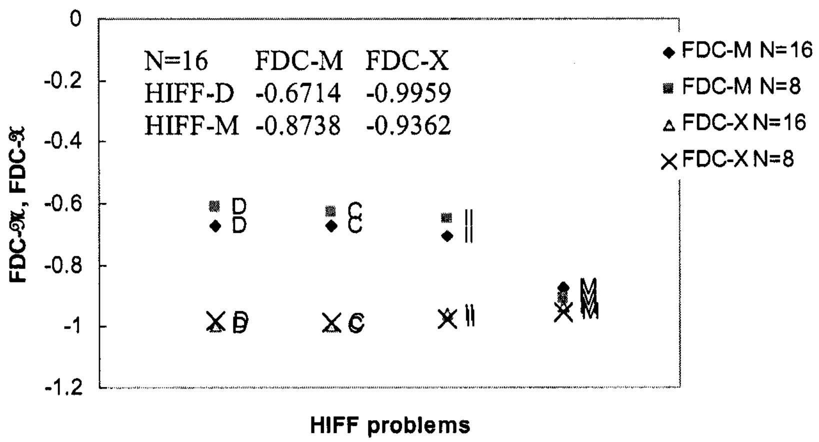


Figure 4.2 FDC-M and FDC-X values for HIFF problems, N = 8 and N = 16

The scatter plots underlying the FDC-$\mathcal{M}$ and FDC-$\mathcal{X}$ fitness distance correlation values for N=16 are given in Figure 4.3. These plots confirm that normality can be assumed and hence the correlation values are representative of the data.

The plots for HIFF-C and HIFF-II (Figure 4.3) are most similar to each other, while the plots for HIFF-D and HIFF-M are most distinct. Although the correlation between fitness and averaged Hamming distance (FDC-$\mathcal{M}$) for HIFF-M is negative and strong, the standard deviations (± one standard deviation is shown for HIFF-M when fitness > 8) increase in magnitude as fitness increases. Therefore a mutation only approach may not always be successful on HIFF-M.

Note in Figure 4.3 (bottom-left) that at fitness level 14 (N-2), the point furthest to the right for HIFF-D is 8 which is N/2 while for HIFF-M is 4 which is N/4 (right error bar). This agrees with the statement made in section 3.3.3 that the widest fitness saddle for HIFF-M is half the size of the widest fitness saddle for HIFF-D.

4.3 Fitness distribution

According to Naudts (1998) "... the faster the number of strings decreases with the fitness approaching the optimum, the harder the problem is for stochastic optimization methods like hill-climbers, simulated annealing and genetic algorithms. This is due to the fact that the number of possible paths to the optimum for such algorithms is related to the combinatorics of near-optimal strings." This assertion may seem counter-intuitive at first because surely it is easier to find an object in a smaller object (*O*) or representation (*R*) space. Nevertheless, this assertion is supported by the results from the HIFF experiments in Part II.

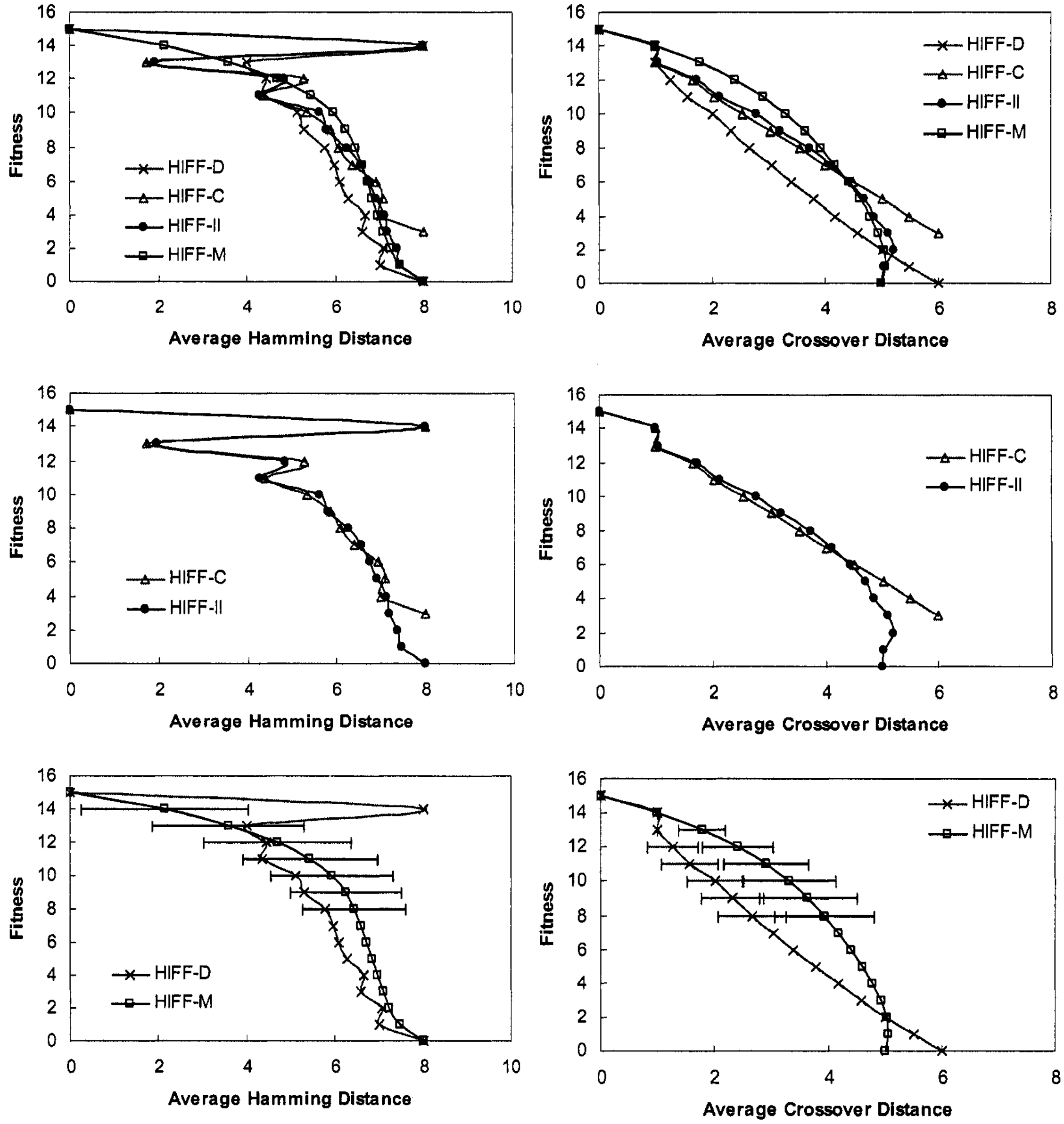


Figure 4.3 Fitness averaged distance scatter plots for HIFF problems, N = 16

The *fitness distribution* of a problem reveals how the size of its object or representation space changes at different fitness levels. The fitness distribution graphs in Figure 4.4 plots $|G_{[a,\ b)}|$ against $F_{[a,\ b)}$ for HIFF problems of size 16. The fitness distributions for HIFF-C and HIFF-II are most similar to each other, while the fitness distributions for HIFF-D and HIFF-M are most distinct. Of all the HIFF problems, the fitness distribution for HIFF-M is most symmetrical, and shows the slowest reduction in object (representation) space as fitness levels approach the optimum.

HIFF-D and HIFF-M have the same set of objective fitness values: {0, 1,... , N-2 , N-1}, but the fitness values as Figure 4.4 shows, are distributed differently. HIFF-M's fitness distribution is more symmetrical or less right-skewed than HIFF-D's. HIFF-M has $\binom{N-1}{F} \times 2$ objects with fitness value F. As a consequence, as fitter genotypes takeover (occupy a larger portion of) a population, it is possible for a HIFF-M population to be more genetically diverse in terms of genotype configurations, than a HIFF-D population.

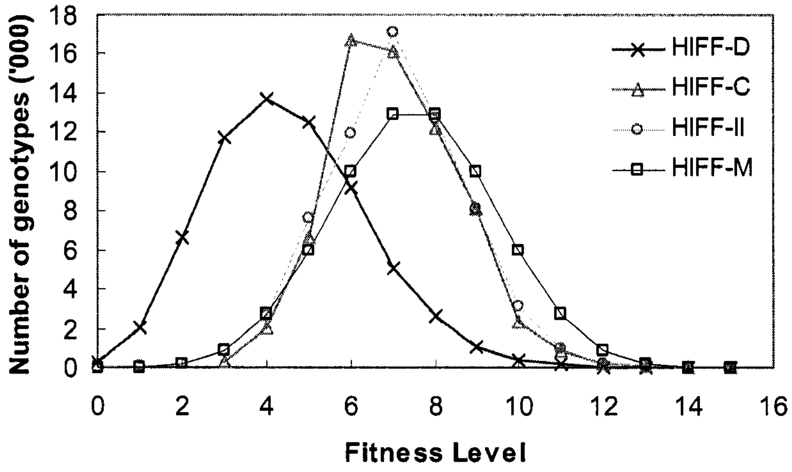


Figure 4.4 Fitness distributions for HIFF problems, N = 16

In section 3.3.4, we considered the sets P and T of HIFF-M genotypes with N-1 fitness (reproduced below). Since genotypes from both P and T sets are equally fit, each of them are (in the case of preferring fitter individuals over less fit ones) equally likely to survive in a population. We can see by inspection several alternative ways of recombining the genotypes in P and T or even mutating them to create a global optimum.

T
1111 0000
0000 1111

P	
0000 0001	1111 1110
0000 0011	1111 1100
0000 0100	1111 1011
0001 0000	1110 1111
0011 0000	1100 1111
0100 0000	1011 1111

In the case of HIFF-D, *P* is empty. So there are only two genotype configurations at fitness level N-2 and there is only one way to recombine the two genotypes in *T* to produce a global optimum. The problem is as Watson (2002) and van Hoyweghen et al. (2002) have confirmed, extra measures are needed to impel a GA to find the two complementary genotypes in *T*. Without these extra measures, what happens is a GA finds one of the genotypes in *T*. Because this genotype is fitter than all other configurations except its complement and the global optima, it begins to takeover the population via crossovers, thus making it increasingly difficult for a GA to find its complement quick enough before premature convergence sets in and removes all hope of further progress. Finding the complement genotype is crucial in the case of HIFF-D and can only be done with crossover because random mutation is helpless at higher levels of fitness (width of fitness saddles increase as fitness increases, section 3.3.3).

By having more genotype configurations per fitness level at higher levels of fitness, HIFF-M allows an EA more alternate routes to a global optimum, and increases the possibility of more *neutral networks* in the fitness landscape thus enabling an EA to cross fitness barriers by escaping via neutral paths (van Nimwegen and Crutchfield, 2000). A neutral network is a set of connected neutral nodes in a fitness landscape. Neutral nodes are neutral with respect to their fitness values.

With respect to a mutation landscape where each node represents a unique HIFF-M genotype, there are not only more equally fit nodes (neutral nodes) at higher levels of fitness, it is also easier for a mutation-only EA to move between neutral nodes HIFF-M is less mutation-hard (section 4.2).

Figure 4.5 shows two comparable snapshots of the HIFF-D and HIFF-M mutation landscapes. A coordinate (x, y) in the contour chart represents the HIFF genotype whose left and right halves are respectively the binary equivalents of the integers x and y. The coordinates at the farthest bottom-left (0, 0) and the farthest top-right (15, 15), represent respectively the all zeros global optimum '0000 0000' and the all ones global optimum '1111 1111'. The points at the farthest top-left (0, 15) and the farthest bottom-right (15, 0) are respectively the fittest non-global optima points '0000 1111' and '1111 0000'. Adjacent points on either axis of the contour charts are one Hamming distance from each other. The numbers in the ovals denote fitness values. Inspecting the charts in Figure 4.5, we find the following:

(i) HIFF-M has several neutral networks (two neutral networks are circled). As such, HIFF-M has several paths of non-decreasing fitness to a global optimum. There are no neutral networks for HIFF-D.

(ii) HIFF-M has fewer local optima than HIFF-D.

Neutrality is a form of symmetry, and symmetry as explained in section 2.5.2 can increase problem difficulty by creating frustration if paths of non-decreasing fitness in the fitness landscape do not all lead to a global optimum (some lead to a non-global local optimum). Hence what is actually preferable for efficient and successful optimization is for evolutionary algorithms to escape via the 'right' path through a neutral network, in other words for the neutrality to be the non-frustrating kind.

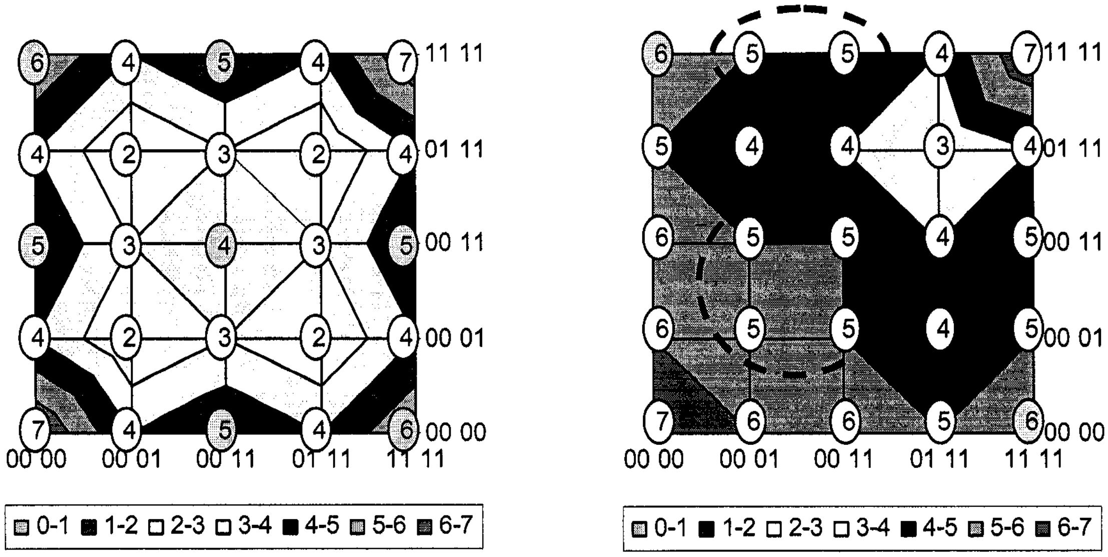


Figure 4.5 Comparable snapshots of HIFF-D (left) and HIFF-M (right) fitness landscapes, N=8, from the perspective of a one-bit mutation EA

In addition to the fitness function, the representation function and the search operators of an EA influence the degree of neutrality (the number and vastness of neutral networks) in a fitness landscape. Thus it would be challenging to characterize problem difficulty in terms of degree of neutrality. According to Ebner et al. (2001), it is possible for a hill climber to solve HIFF-D (N=16) if extensive neutral networks are created by the representation function (i.e. genotype-phenotype map).

4.4 Chapter Summary

When compared with the other HIFF problems (HIFF-D, HIFF-C and HIFF-II), in particular HIFF-D:

(i) HIFF-M is less epistatic (section 4.1). HIFF-M has a significantly smaller SWO value (Table 4.1).

(ii) HIFF-M is less mutation-hard and less crossover-easy (section 4.2). Fitness distance correlations computed on Hamming distance for the mutation operator (FDC-$\mathcal{M}$) and on crossover distance for the crossover operator (FDC-$\mathcal{X}$) are used to support this statement. The closer a fitness distance correlation value computed for an operator is to -1, the easier is it for an EA which uses the operator as its main search operator to reach a global optimum. FDC-$\mathcal{M}$ for HIFF-M (-0.8738) is smaller than FDC-$\mathcal{M}$ for HIFF-D (-0.6714). FDC-$\mathcal{X}$ for HIFF-M (-0.9362) is larger than FDC-$\mathcal{X}$ for HIFF-D (-0.9959). The FDC-$\mathcal{M}$ and FDC-$\mathcal{X}$ values for HIFF-M are also closer to each other (Figure 4.2). The widest fitness saddle in the HIFF-M mutation fitness landscape is only half the size of the same for HIFF-D (Table 3.7 and Figure 4.3).

(iii) HIFF-M has greater potential for maintaining genotype diversity and creating neutral networks even at higher levels of fitness (section 4.3). This is because HIFF-M's fitness distribution is more symmetrical (Figure 4.4). As such, there are more distinct genotypes at higher levels of fitness.

The above findings lead to the overall expectation that an evolutionary algorithm would exhibit different optimizing performance on HIFF-M than on the other HIFF problems. Since HIFF-M is less mutation-hard than HIFF-D, mutation-only evolutionary algorithms should perform better on HIFF-M than on HIFF-D. Since mutation is a diversification mechanism in genetic algorithms and crossover a homogenizing one, and

HIFF-M is less-mutation hard and less crossover-easy than HIFF-D, HIFF-M GA populations should remain diverse for longer. Points (ii) and (iii) combined also imply slower GA population convergence for HIFF-M.

Slower GA population convergence is not a goal in itself. A GA population that converges slowly may not be doing enough exploitation and increasing the average number of function evaluations to find a global optimum (MFPT) needlessly. However since the main difficulty faced by genetic algorithms on the original HIFF problems and other test problems designed to address the HC < GA problem is premature convergence (section 2.4.3), having a test problem like HIFF-M which shows signs of being inherently more resistant to premature convergence would be educational.

Part II

The objective of Part II is to ascertain whether HIFF-M is a solution to the HC < GA problem and if so, how HIFF-M is distinct from HIFF-D as a solution to the HC < GA problem. HIFF-D was proposed as a solution to the HC < GA problem by Watson et al. (1998). Part I introduced and defined the HIFF-M problem (section 3.2.4). HIFF-M has a different inter-dependency matrix from HIFF-D and its fitness function does not use the 'all-or-nothing' approach that HIFF-D's fitness function uses (section 3.2.2). The central question addressed in Part II is how the differences in inter-dependency matrix and fitness calculation affect the ability of a problem to be a solution to the HC < GA problem.

The HC < GA problem was described at length in section 2.4. Essentially, it is to find a static fitness function that a genetic algorithm (GA) optimizes or solves more easily (i.e. successfully and /or efficiently) than a hill climber (HC), within a given number of function evaluations.

Evaluation of evolutionary algorithms

The ability of an evolutionary algorithm (EA) to solve a problem is based on optimizing performance which comprises the number of successful runs (SUCC) and the average number of function evaluations to discover a globally optimal solution (MFPT) (section 2.4). A successful run is one that finds a global optimum within the maximum number of function evaluations specified. When comparing two evolutionary algorithms (EAs), the number of successful runs takes priority. When both EAs have similar SUCC values, their MFPT values are compared to decide EA superiority. Signal-to-noise ratios are also considered in certain cases to emphasize differences in MFPT variances. *Best-so-far fitness graphs* are plotted to observe evolutionary behavior. Best-so-far fitness graphs give the best fitness value found so far by an EA during a run (the genotype with the best fitness exists in the current population). Best-so-far fitness graphs for different EAs are compared to observe how much success rates depend on when the comparison is made (Spears, 1998).

Defining a rough and intuitive list of criteria for EA evaluation is not unique to this dissertation. For example, Naudts and Kallel (2000) define an informal hierarchy of five quality convergence classes for a given evolutionary algorithm and fitness function that take into account whether an EA finds the optimum of the fitness function independently of the choice of the randomly sampled initial population, and the size of the variance of the number of iterations needed to visit the optimum for the first time but note that "whether a given fitness function belongs to one convergence quality class or not can depend heavily on the choice of parameters of the algorithm."

The hill climbing algorithm and genetic algorithm

The HC and GA used in the HIFF-D experiments were respectively the Random Mutation Hill Climber (*RMHC*) (Forrest and Mitchell, 1993) and the Genetic Algorithm with Deterministic Crowding (*GADC*) (Mahfoud, 1995). RMHC was mentioned earlier in section 2.4.3 in connection with the 'failed' Royal-Road experiment. GADC (Figure 6.1) is a panmictic (single-population) GA but it imposes explicit restrictions on interactions between individuals to maintain genetic diversity in the population. Individuals in a population interact with each other through crossover and by competing with each other for reproduction and survival opportunities.

The HC used in HIFF-M experiments is RMHC also (section 5.1.1). However, the GA is *upGA* (Figure II.A) which like GADC is a panmictic GA, but upGA does not explicitly preserve population genetic diversity and is less restrictive on interactions between individuals than GADC. The upGA algorithm is explored in section 7.4. Experiments on HIFF-M were also carried out with GADC, but those runs performed as well as upGA (section 7.1).

Preview of findings

The experiments in Part II confirm that HIFF-M is a solution to the HC < GA problem and a distinct one at that from HIFF-D. As such, the differences in inter-dependency matrix and fitness calculation between the HIFF-D and HIFF-M problems have not adversely affected the ability of a problem to be a solution to the HC < GA problem. The specifications for the original HIFF problems described in chapter 3 can be relaxed without violating the HC < GA condition. In particular, it is not necessary: (i) for all variables to be directly inter-dependent on each other, (ii) for fitness contributions to be scaled so that inter-dependencies at lower levels far outweigh interactions at higher levels, or (iii) to use the 'all-or-nothing' approach.

Nevertheless, when direct inter-dependencies are reduced and all inter-dependencies make the same fitness contribution, e.g. in HIFF-M, the problem becomes less difficult for hill climbers to solve, but still not easily solved by RMHC or MMHC (section 5.1).

In addition, being a less mutation-hard problem (section 4.2) makes it easier for upGA to be successful because mutation is able to perform its role of reintroducing lost alleles back into the population even when individuals in the population are highly fit.

Thus, premature loss of genetic diversity is less severe when upGA works on HIFF-M than on HIFF-D (section 7.2).

Mutation is more effective in the HIFF-M problem than HIFF-D because HIFF-M's fitness distribution is less right-skewed than HIFF-D's (section 4.3). The connection between fitness distribution and GA-hardness has been suggested before (Borenstein and Poli, 2004), although a counter-example is found in chapter 9. Having a less right-skewed fitness distribution means that HIFF-M has more unique genotypes than HIFF-D at higher fitness levels. Consequently, it is possible for HIFF-M to have more fitness-neutral nodes in its fitness landscape with which to build neutral networks and create neutral paths along which an EA may escape from local optima (section 4.3).

Of all the evolutionary algorithms tested in Part II, upGA is one of the best performing EA for HIFF-M (section 7.1). The combination of upGA with HIFF-M creates a fitness landscape where both crossover and mutation have important roles to play in the search for an optimal solution (section 7.3). This leads to the conclusion that the exploitative and explorative forces are suitably balanced in the HIFF-M and upGA combination. Work reported in Part II is published in part in Khor (2008).

II.A Roadmap to the experiments

The objective of Part II is to ascertain whether HIFF-M is a solution to the HC < GA problem and if so, how HIFF-M is distinct from HIFF-D as a solution to the HC < GA problem. To meet this objective, the following four main hypotheses are tested:

(i) HIFF-M is difficult for RMHC to solve (section 5.1.1),

(ii) HIFF-M is more easily solved by GADC than RMHC (section 6.1),

(iii) HIFF-M is more or as easily solved by upGA than GADC (section 7.1), and

(iv) upGA solves HIFF-M more easily than HIFF-D (section 7.2).

If hypotheses (i) and (ii) above are confirmed, then: The changes made to the structure (inter-dependency matrix) of the original HIFF problems (HIFF-D and HIFF-C) that resulted in the HIFF-M problem has not fundamentally affected the HC < GA nature of the original HIFF problems. This implies that the requirements for the structure of the original HIFF problems as described in chapter 3 can be relaxed. Most notably, it is not necessary for all variables in the original HIFF problems to be inter-dependent on each other.

In addition, if hypotheses (iii) and (iv) above are also confirmed, then: HIFF-M is a distinct solution from HIFF-D to the HC < GA problem.

Further, because of the important role mutation plays in the success of upGA on HIFF-M (section 7.3), we conclude that HIFF-M is a solution to a more difficult version of the HC < GA problem. Having upGA rely on effective mutation for success on HIFF-M simultaneously increases the chance of success for mutation-only EAs such as RMHC. HIFF-D on the other hand works in the opposite direction: mutation becomes ineffective particularly for optimizing genotypes with high but suboptimal fitness. This reduces the chance of success for mutation-only evolutionary algorithms such as RMHC but it also creates diversity maintenance problems in genetic algorithms. Thus, a GA such as GADC which includes a mechanism for preserving genetic diversity in a population is required to solve HIFF-D (Watson, 2002).

Table II.A presents the roadmap for the experiments in Part II to test the above main hypotheses. A summary of findings from each experiment is also reported. The hypotheses are elaborated upon in the respective sections.

Table II.A Roadmap for the experiments in Part II

Hypothesis	Section	Summary of Findings
HIFF-M is difficult for RMHC to solve.	5.1.1	(i) HIFF-M is difficult for RMHC to solve. Therefore, HIFF-M's inter-dependency matrix, though different from that of the original HIFF problems, has not made HIFF-M an easy problem for RMHC to solve.
HIFF-M is difficult for MMHC to solve.	5.1.2	(i) HIFF-M is not as difficult as HIFF-D for MMHC to solve. HIFF-M is easier than HIFF-D for MMHC to solve. This demonstrates that HIFF-M is less mutation-hard than HIFF-D (section 4.2), and that the mechanics of crossover or macro-mutation on a single individual is insufficient for solving HIFF-M.
HIFF-M is difficult for multi-individual mutation-only EAs to solve.	5.2.1 5.2.2	(i) HIFF-M is difficult for mutation-only upGA and GARC to solve. Hence, HIFF-M is difficult for multi-individual mutation-only EAs to solve and the idea of crossover will be evolutionarily advantageous to EAs solving HIFF-M. This is confirmed in section 6.1.
	5.2.3	(ii) But a tailored version of GARC called dGARC is able to solve HIFF-M. Thus, for HIFF-M to be a HC < GA problem, the GA must also outperform dGARC. This is confirmed in section 6.1.
HIFF-M is more easily solved	6.1	(i) HIFF-M is more easily solved by crossover-only

by an EA that uses the idea of crossover than by an EA that does not.		GADC (PS = 1,000) than RMHC, MMHC, mutation-only upGA or dGARC. Hence HIFF-M is more easily solved by an EA that uses the idea of crossover than by an EA that does not; and the idea of crossover is evolutionarily advantageous to EAs solving HIFF-M. (ii) **HIFF-M's inter-dependency matrix, though different from that of the original HIFF problems, has not affected HIFF-M's ability to be a solution to the HC < GA problem.**
For both HIFF-D and HIFF-M, maintaining sufficient genetic diversity is important for good optimizing performance.	6.2	(i) For both HIFF-D and HIFF-M, maintaining sufficient genetic diversity is important for good optimizing performance. (ii) Homogenizing genetic drift is stronger in crossover-only upGA (PS=128) than crossover-only GADC (PS = 1,000). (iii) There is a positive relationship between higher levels of homogenizing genetic drift and poorer optimizing performance for both HIFF-D and HIFF-M. (iv) Homogenizing genetic drift in crossover-only upGA is too strong for solving either HIFF-D or HIFF-M.
Homogenizing genetic drift is stronger in HIFF-D than in HIFF-M populations.	6.3	(i) Homogenizing genetic drift is stronger in HIFF-D than in HIFF-M populations. Therefore, population diversity maintenance will be more important for EAs solving HIFF-D than HIFF-M.
Some degree of homogenizing genetic drift is beneficial for solving HIFF-M.	6.4	(i) Some degree of homogenizing genetic drift is beneficial for solving HIFF-M.

HIFF-M is more or as easily solved by upGA compared with crossover-only GADC (PS = 1,000).	7.1	(i) HIFF-M is as easily (efficiently) solved by upGA ($P_x = 0.5$, $P_m = 0.0625$) as by crossover-only GADC (PS = 1,000) and by implication dGARC (section 6.1). (ii) The genetic drift pattern produced by upGA ($P_x = 0.5$, $P_m = 0.0625$) in HIFF-M populations is different from that produced by crossover-only GADC (PS = 1,000) due to the inter-play between crossover and mutation (this point is examined further in section 7.3).
upGA solves HIFF-M more easily than HIFF-D.	7.2	(i) upGA solves HIFF-M more easily than HIFF-D. **(ii) HIFF-M is a distinct solution from HIFF-D to the HC < GA problem.** (iii) upGA solves HIFF-M more easily than HIFF-D because all other factors being the same, upGA's homogenizing genetic drift is stronger in HIFF-D than in HIFF-M populations. (iv) upGA's homogenizing genetic drift is stronger in HIFF-D than in HIFF-M populations because mutation is more effective in HIFF-M than in HIFF-D upGA populations, and fewer one-offspring crossovers succeed in HIFF-M than in HIFF-D upGA populations. (v) The synchronization problem (section 3.3.2) is detected in unsuccessful upGA HIFF-D runs.
Both crossover and mutation play an important role in the success of upGA on HIFF-M.	7.3	(i) upGA (with both mutation and crossover) solved HIFF-M more easily than both mutation-only upGA and crossover-only upGA. Therefore, both crossover and mutation play important roles in the success of upGA on HIFF-M. Crossover and

		mutation affect genetic drift. (ii) **Since mutation plays an important role in upGA success on HIFF-M, HIFF-M is a solution to a more difficult version of HC < GA.**

II.B Common methods and default parameter settings

The parameters of the hill climbers (HC) and genetic algorithms (GA) and their default values are listed in Table II.B. Default values for parameters are used unless otherwise stated.

Table II.B Parameters and their default values for the EA experiments

Parameter	HC	GA*
Problem size (N)	128	128
Maximum number of function evaluations (MaxEvals)	250,000	500,000
Number of runs per experiment	30	30
Population size (PS)	1	N
Mutation rate (P_m)	0.0625	0.0625
Mutation operator	Random	Random
Crossover rate (P_x)	-	0.5
Crossover operator	-	1-2 point

GA* includes mutation-only multi-individual EAs

The design of the experiments (algorithms, problem size, number of runs and implementation of random mutation as resetting of values instead of bit-flip) is influenced by Watson (2002). Other mutation rates (P_m) and crossover rates (P_x) are used in the experiments.

Random genotypes are genotypes whose genes are initialized with alleles chosen uniformly at random with replacement from the common alphabet. The alphabet for all genes is {0, 1}. Random genotypes are used at the start of all searchers. The Mersenne (pseudo-)random number generator[8] is used. All random choices are drawn independently from a uniform distribution.

Except for upGA, all HC and GA algorithms used in the experiments are either well-known in the EC community and/or used by Watson (2002) in his HIFF experiments. The upGA algorithm is outlined in Figure II.A.

```
Step  upGA
1.    Create a population of PS random genotypes and evaluate the genotypes.
2.    If MaxEvals > 0 and optimum not found
          Go to step 3, Else stop.
3.    Select 2 distinct genotypes p1 and p2 at random from the population as follows:
          d := random integer in [1, PS – 1]
          i := random integer in [0, PS – 1]
          p1 := population at i
          p2 := population at (i + d) % PS
4.    With probability $P_x$
          Perform a 1-2 point crossover on p1 and p2 to create 2 offspring, c1 and c2.
          Replace p1 with c1 if c1 is strictly fitter than both p1 and p2.
          Replace p2 with c2 if c2 is strictly fitter than both p1 and p2.
5.    Else
          Mutate p1 to create c1 and mutate p2 to create c2.
          Replace p1 with c1 if c1 is fitter than or as fit as p1.
          Replace p2 with c2 if c2 is fitter than or as fit as p2.
6.    MaxEvals := MaxEvals - 1
      Go to step 2.
```

Figure II.A Algorithm for upGA

```
1-2 point crossover
d := random integer in [1, N – 1]
x := random integer in [0,  N – 1]
y := (x + d) % N
if (x > y) swap x and y
c1 := p1[0,…,x-1] p2[x…y-1] p1[y,…,N-1]
c2 := p2[0,…,x-1] p1[x…y-1] p2[y,…,N-1]
```

Figure II.B 1-2 point crossover

[8] Implementation downloaded from http://www.agner.org/random/randomc.

The *1-2 point crossover* operation is outlined Figure II.B. $g[x,\ldots,y\text{-}1]$ denotes all genes located after position x and before position y in genotype g. Position 0 is the same as position N if the genotype is treated as a ring.

The 1-2 point crossover is a two point crossover operator that allows one-point crossovers ($x = 0$), does not allow cloning, produces two offspring and every (x, y) combination is equiprobable. 2-point crossover (with 1-point crossover excluded) reaches an equilibrium state much further away from Robbins equilibrium than 1-2 point crossover (de Jong, 2006 p. 146). The ability to reach Robbins equilibrium is preferred because it means any genotype in the representation space is reachable from a sufficiently diverse initial GA population via the application of the crossover operator (de Jong, 2006 p. 144).

Random mutation resets the values of k genes chosen at random with replacement from a genotype to values chosen randomly from the common alphabet. The integer k is chosen at random from $[1, P_m \times N]$.

Evaluation

The ability of an evolutionary algorithm (EA) to solve a problem is evaluated on the basis of the number of successful runs (SUCC) and the average number of function evaluations to find a globally optimal solution (MFPT). A successful run is one that finds a global optimum within the maximum number of function evaluations specified (`MaxEvals`). Standard deviations associated with averages are stated in parentheses. When comparing two evolutionary algorithms (EAs), the number of successful runs takes priority. When SUCC values are similar for two competing EAs, their MFPT values are compared to decide superiority.

An EA solves a problem easily if it is successful in more than 90% of runs within the maximum number of function evaluations. A problem is difficult for an EA if the EA is successful in less than 10% of runs within the maximum number of function evaluations.

Significance tests are conducted using Microsoft Office Excel's one-tailed Student's t-test function. There are three kinds of t-tests: paired, two-sample equal variance (homoscedastic) and two-sample unequal variance (heteroscedastic). All three kinds of t-tests are made and the largest value in percentage up to 6 decimal points is reported. The null hypothesis (H_0) claims that there is no difference between the averages of two data sets being compared. To accept the alternative hypothesis (H_1) that one sample mean is different from the other, the significance probability of H_0 should be less than 1% by the one-sided t-test. Alternatively, there should be no overlap between the 99% confidence intervals of two sample means.

Best-so-far fitness graphs are plotted to observe evolutionary behavior. Best-so-far fitness graphs give the best fitness value found so far by an EA during a run. Best-so-far fitness graphs for different EAs are compared to observe their relative evolutionary behaviors and how much success rates depend on when the comparison is made (Spears, 1998).

II.C Conclusion for Part II

All four hypotheses stated in section II.A are confirmed by the experiments in Part II. As such the following conclusions are made:

(i) That HIFF-M is a solution to the HC < GA problem, in spite of it having a different inter-dependency matrix from the original HIFF problems (section 6.1).

(ii) This implies that the requirements for the structure of the original HIFF problems as described in chapter 3 can be relaxed. Most notably, it is not necessary for all variables in the original HIFF problems to be inter-dependent on each other and have differentiated interaction weights.

(iii) HIFF-M is a distinct solution from HIFF-D to the HC < GA problem because HIFF-M is solved by a GA that does not rely on population diversity mechanisms other than mutation, i.e. upGA (section 7.1). This is not possible for HIFF-D and the GA of choice for HIFF-D is one that uses population diversity mechanisms in addition to or other than mutation, e.g. GADC (section 7.2).

(iv) Since mutation plays an important role in the success of upGA (section 7.3) on HIFF-M and HIFF-M is a less mutation-hard problem than HIFF-D (sections 4.2 and 5.1.2), HIFF-M is a solution to a more difficult version of the HC < GA problem.

Having upGA rely on effective mutation for success on HIFF-M simultaneously increases the chance of success for mutation-only EAs such as RMHC. HIFF-D on the other hand works in the opposite direction: mutation becomes ineffective particularly for optimizing genotypes with high but suboptimal fitness. This reduces the chance of success for mutation-only EAs such as RMHC but it also creates diversity maintenance problems in GAs. Thus, a GA such as GADC which includes a mechanism for preserving genetic diversity in a population is required to solve HIFF-D.

Since both mutation and crossover play important roles in the optimizing performance of upGA on the HIFF-M problem (section 7.3), the implication is that the combination of upGA (configured as it was in the experiments) and the HIFF-M problem produces a fitness landscape where crossover and mutation complement each other well in the search for a global optimum. The exploitative (population homogenizing) effect of crossover is balanced appropriately with the explorative (population diversifying) effect of mutation in the upGA, HIFF-M fitness landscape.

Table II.C summarizes the performances of all EAs in Part II which attained 100% success rate on HIFF-M. upGA, an EA that uses the idea of crossover and permits moderate genetic drift, is as efficient as crossover-only GARC but more efficient than both GIGA and dGARC. Figure II.C plots the 99% confidence intervals for MFPT values in Table II.C.

Table II.C Successful EAs for HIFF-M, N=128.

Evolutionary Algorithm	SUCC	MFPT	99% confidence interval for MFPT
dGARC (section 5.2.3)	30	364,710 (55,321)	364,710 ± 26,016
GIGA (section 6.3)	30	113,060 (28,889)	113,060 ± 13,586
Crossover-only GADC (PS = 1,000) (section 6.1)	30	94,921 (22,341)	94,921 ± 10,507
upGA (P_x = 0.5, P_m = 0.0625) (section 7.1)	30	80,225 (29,389)	80,225 ± 13,821

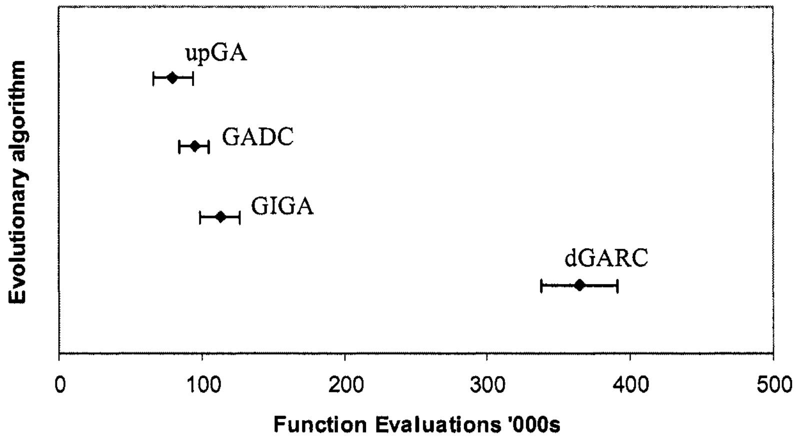


Figure II.C 99% confidence intervals for MFPT values in Table II.C

5. Mutation-only Evolutionary Algorithms

The overall objective of the experiments in this chapter is to show that HIFF-M is difficult for mutation-only evolutionary algorithms (EAs), in particular RMHC, to solve. The main hypotheses tested in this chapter are:

Table 5.1 Hypotheses for chapter 5

Hypothesis	Section
HIFF-M is difficult for RMHC to solve.	5.1.1
HIFF-M is difficult for MMHC to solve.	5.1.2
HIFF-M is difficult for multi-individual mutation-only EAs to solve.	5.2

5.1 Single-individual (Hill Climbing)

In this section, the HIFF problems are compared on the basis of the optimizing performance of mutation-only single-individual EAs, namely the Random Mutation Hill Climber (RMHC) for reasons mentioned in the preamble of Part II, and the Macro-Mutation Hill Climber (MMHC) (Jones, 1995) for reasons to be mentioned later in section 5.1.2.

5.1.1 Random Mutation Hill Climbing

Hypothesis

HIFF-M is difficult for RMHC to solve. The success rate of RMHC for HIFF-M should be comparable with that for HIFF-D, and less than 10%.

Method

Figure 5.1 gives the algorithm for Random Mutation Hill Climber (RMHC). RMHC uses random mutation (defined in section II.B) and replaces the current genotype with its mutant offspring if the current genotype is less fit than its offspring. Table II.B has the settings for the other parameters.

```
Step  Random Mutation Hill Climber
1.    Create a random genotype $p$. Calculate the fitness of $p$, $F(p)$.
2.    While MaxEvals > 0
3.        MaxEvals := MaxEvals – 1
4.        Create $c$ by reproducing $p$, and mutate $c$ as follows:
              $k$ := random integer in $[1, P_m \times N]$
              Select $k$ random genes and randomly assign 0 or 1 to each gene.
5.        Calculate the fitness of $c$, $F(c)$.
6.        If $F(c)$ is optimal
              Return $c$ as a solution and Exit.
7.        Else If $F(c) >= F(p)$
              Replace $p$ with $c$.
      End while
8.    Return $p$ as a best non-optimal solution found.
```

Figure 5.1 Algorithm for RMHC

Results

Table 5.2 reports the results for RMHC. It is difficult for RMHC to solve the original HIFF problems (HIFF-D and HIFF-C). RMHC's success rate on both HIFF-D and HIFF-C is 0%. It is also difficult for RMHC to solve the new HIFF problems (HIFF-II and HIFF-M). Out of the three mutation rates, RMHC's best success rate for HIFF-M is 3%.

Table 5.2 Number of successful RMHC runs (SUCC)

P_m	HIFF-D	HIFF-C	HIFF-II	HIFF-M
0.25	0	0	0	0
0.125	0	0	0	1
0.0625	0	0	0	1

Conclusion

Since the success rate of RMHC for HIFF-M is comparable with that for HIFF-D, and less than 10%, the hypothesis that HIFF-M is difficult for RMHC to solve is confirmed.

Discussion

HIFF-M is difficult for RMHC to solve for reasons discussed in section 3.3 namely modularity, modular-interdependency, wide fitness saddles and multiple local optima in the mutation fitness landscape.

Figure 5.2 portrays best-so-far fitness graphs for randomly chosen unsuccessful RMHC HIFF-M runs. The graphs show no sign of further fitness increase prior to the end of runs. However, to reveal the true difficulty for RMHC to reach the optimal fitness value of 127, genotypes at the end of the runs in these graphs are given below.

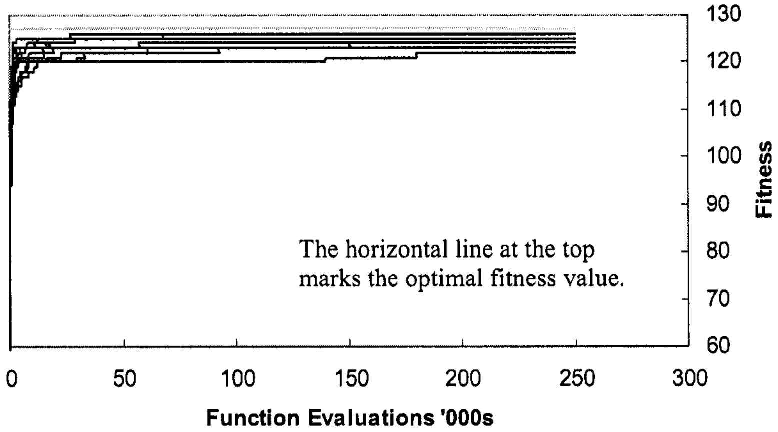


Figure 5.2 Best-so-far fitness graphs for unsuccessful RMHC HIFF-M runs.

Genotypes at the end of the RMHC runs graphed in Figure 5.2:

RMHC P_m=0.0625 HIFF-M N=128

```
11111111 11111111 11111111 00000000 11111111 11111111 11111111 11111111
11111111 00000000 11111111 11111111 00000000 00000000 00000000 00000000
Fitness: 124
00000000 00000000 00000000 00000000 00000000 11111111 00000000 00000000
11111111 11111111 00000000 00000000 11111111 00000000 11111111 11111111
Fitness: 123
11111111 11111111 11111111 11111111 00000000 00000000 00000000 11111111
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
Fitness: 125
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 11111111 11111111 11111111 11111111
Fitness: 126
```

RMHC P_m =0.25 HIFF-M N=128

```
11111111 11111111 11111111 11111111 11111111 11111111 00000000 00000000
11111111 11111111 00000000 00000000 00000000 00000000 00000000 00000000
Fitness: 124
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
11111111 11111111 11111111 00000000 11111111 11111111 11111111 11111111
Fitness: 126
00000000 00000000 00000000 00000000 11111111 11111111 11111111 11111111
00000000 00000000 00000000 11111111 11111111 11111111 11111111 00000000
Fitness: 123
11111111 11111111 11111111 11111111 11111111 00000000 11111111 11111111
00000000 00000000 11111111 11111111 00000000 00001111 11111111 11111111
Fitness: 122
```

Although the fitness of these genotypes are only 1 to 5 fitness points below the optimal fitness value of 127, to get these additional points RMHC has to mutate contiguous locations of a genotype to the same value. But RMHC selects genotype locations to mutate at random, and resets the value of a selected location to a random value of the alphabet. Thus it is very difficult for RMHC to solve HIFF-M.

Implication

The inference from the conclusion in this section is that it is possible to reduce the number of inter-dependencies between variables and homogenize the weights of the inter-dependencies without making the original HIFF problem easy for RMHC to solve. HIFF-M has fewer inter-dependencies between its variables than HIFF-D and all of HIFF-M's inter-dependencies are equally weighted (section 3.2.4). Yet, HIFF-M is difficult for RMHC to solve. In other words, HIFF-M's inter-dependency matrix, though different from that of the original HIFF problems, does not make HIFF-M an easy problem for RMHC to solve.

5.1.2 Macro-Mutation Hill Climbing

Hypothesis

HIFF-M is difficult for MMHC to solve. The success rate of MMHC for HIFF-M should be comparable with that for HIFF-D, and less than 10%.

Method

The Macro-Mutation Hill Climber (MMHC) (Jones, 1995) is used to test whether the repeated application of *macro-mutation* on a single genotype is sufficient to solve a problem. MMHC (Figure 5.3) is identical to RMHC (Figure 5.2) except for macro-mutation in step 4. Macro-mutation resets the values of k genes located between two positions x and y chosen at random, to values chosen randomly from the alphabet. The integer k is chosen at random from $[1, P_m \times N]$ and is the distance (number of locations) between x and y. The genotype is treated as a ring. An offspring or mutant genotype is kept if it is as fit as or fitter than its parent genotype.

```
Step  Macro-Mutation Hill Climber
1.    Create a random genotype $p$. Calculate the fitness of $p$, $F(p)$.
2.    While MaxEvals > 0
3.        MaxEvals := MaxEvals – 1
4.        Create c by reproducing $p$ and mutate $c$ as follows:
              $k$ := random integer in $[1, P_m \times N]$
              $x$ := random integer in $[0, N-1]$
              Randomly assign 0 or 1 to each gene between position $x$ and
              position $(x + k)$ mod N.
5.        Calculate the fitness of $c$, $F(c)$.
6.        If $F(c)$ is optimal
              Return $c$ as a solution and Exit.
7.        Else If $F(c) >= F(p)$
              Replace $p$ with $c$.
      End while
8.    Return $p$ as a best non-optimal solution found.
```

Figure 5.3 Algorithm for MMHC

Results

Table 5.3 reports on the results for MMHC. The original HIFF problems (HIFF-D and HIFF-C) are difficult for MMHC. HIFF-II is also difficult for MMHC. MMHC's success rates for HIFF-D, HIFF-C and HIFF-II are all less than 10%. MMHC has a higher success rate for HIFF-M than the other HIFF problems. Nevertheless, MMHC's success rate for HIFF-M is 50% at best. Further the successful MMHC runs have low (< 2.0) signal-to-noise ratio (average/standard deviation), suggesting that MMHC is a weak EA for the HIFF-M problem.

Table 5.3 Results for MMHC

P_m	HIFF-D	HIFF-C	HIFF-II	HIFF-M	
	SUCC	SUCC	SUCC	SUCC	MFPT
0.5	0	1	0	11	49,769 (60,084)
0.25	1	0	0	13	56,548 (69,014)
0.125	0	0	1	15	77,114 (76,843)
0.0625	0	0	0	9	50,247 (82,970)

Conclusion

Since the success rate of MMHC for HIFF-M is not comparable with that for HIFF-D and is not less than 10%, the hypothesis that HIFF-M is difficult for MMHC to solve is not confirmed. However, this does not mean that HIFF-M is easy for MMHC to solve. The highest success rate of MMHC for HIFF-M is 50%. The conclusion from the results in Table 5.3 is that HIFF-M is not as difficult as HIFF-D for MMHC to solve.

Discussion

Figure 5.4 portrays randomly chosen best-so-far fitness graphs for unsuccessful MMHC HIFF-M runs. The graphs show no sign of further fitness increase prior to the end of runs. However, to reveal the true difficulty for MMHC to reach the optimal fitness value of 127, genotypes at the end of the runs in these graphs are given below.

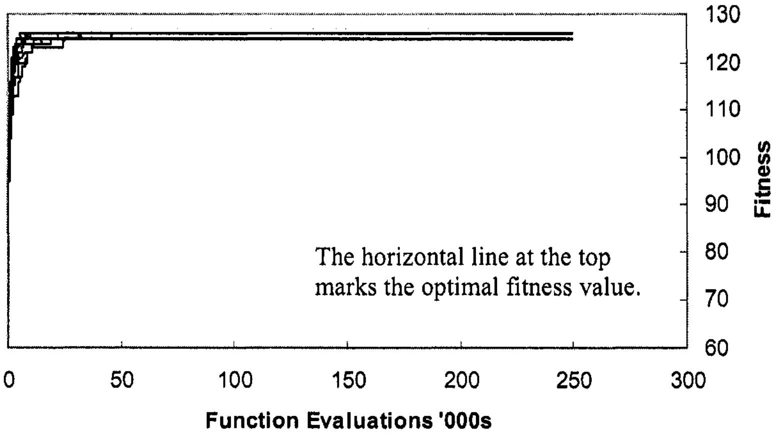


Figure 5.4 Best-so-far fitness graphs for unsuccessful MMHC HIFF-M runs

Genotypes at the end of the MMHC runs graphed in Figure 5.4:

MMHC P_m=0.5 HIFF-M N=128

```
11111111 11111111 11111111 11111111 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
Fitness: 125
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
Fitness: 126
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 11111111 11111111
Fitness: 126
```

MMHC P_m=0.25 HIFF-M N=128

```
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
Fitness: 126
11111111 11111111 00000000 00000000 11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
Fitness: 126
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111 00000000 00000000 00000000 00000000
Fitness: 126
```

MMHC P_m=0.125 HIFF-M N=128

```
11111111 11111111 00000000 00000000 11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
Fitness: 126
00000000 00000000 00000000 00000000 11111111 11111111 11111111 11111111
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
Fitness: 126
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
Fitness: 126
```

Although the fitness of these genotypes are only 1 or 2 fitness points below the optimal fitness value of 127, to get these additional points MMHC has to mutate the right contiguous locations of a genotype to the same value. But MMHC (Figure 5.3) resets the value of a selected location to a random value of the alphabet. Thus it is not easy for MMHC to solve HIFF-M.

Implication

The above results confirm that HIFF-M is easier than HIFF-D for MMHC to solve. This confirmation supports the statement made in section 4.2 that HIFF-M is less mutation-hard than HIFF-D.

Since MMHC was proposed by Jones (1995) to test the sufficiency of the mechanics of crossover or macro-mutation for an evolutionary algorithm (EA) to solve a problem, the MMHC results for HIFF-M in this section suggest that the mechanics of crossover or macro-mutation on a single individual is insufficient for solving HIFF-M.

5.2 Multi-individual

The objective of this section is to show that HIFF-M is difficult for multi-individual (population based) mutation-only EAs to solve.

A hill climber is a single-individual EA whereas a genetic algorithm is a multi-individual or population based EA. Having a population can be advantageous for search as discussed in section 2.2.2 under the topic of exploration versus exploitation. Therefore, to make a fairer comparison between hill climbers and genetic algorithms, genetic algorithms should also outperform their mutation-only counterpart. However, Jones (1995) proposed the Headless Chicken Test as a better way to compare a GA with a mutation-only multi-individual evolutionary algorithm. Both methods are used in this section.

Mutation-only Multi-individual Evolutionary Algorithms

The multi-individual or population based (population size is greater than one) mutation-only EAs used in this section are:

(i) Mutation-only upGA (section 5.2.1).

(ii) Crossover-only upGA with random-crossover (GARC) for the Headless Chicken Test (section 5.2.2), and

(iii) A deterministic version of GARC called dGARC (section 5.2.3). This is a version of GARC tailored after completing the Headless Chicken Test on HIFF-M. dGARC raises the bar for GAs since it solves HIFF-M with success rate > 90%.

Table 5.4 lists the main aspects in which the above three EAs differ.

Table 5.4 Comparison of the multi-individual mutation-only EAs

Mutation-only EA	Mutation	Parent selection
upGA	Random	Uniform stochastic
GARC	Macro	Uniform stochastic
dGARC	Macro	Uniform deterministic

5.2.1 Mutation-only upGA

Hypothesis

HIFF-M is difficult for mutation-only upGA to solve. The success rate of mutation-only upGA for HIFF-M should be less than 10%.

Method

Mutation-only upGA is upGA (Figure II.A) with crossover turned-off, i.e. $P_x = 0.0$. Mutation rates (P_m) 0.25, 0.125 and 0.0625 are tested. Other parameter settings are detailed in Table II.B.

Results

Table 5.5 reports the results. The success rate of mutation-only upGA for HIFF-M (N=128) is less than 10%. However on a smaller problem (N=64), mutation-only upGA ($P_m = 0.0625$) successfully optimized HIFF-M in all 30 runs. Nevertheless, its signal-to-noise ratio is low (< 2.0) indicating that even on HIFF-M N=64, mutation-only upGA is a weak solution. This result implies that mutation-only upGA algorithm is not a scalable solution for HIFF-M.

Table 5.5 Results for mutation-only upGA on HIFF-M

P_m	PS = N = 64		PS = N = 128	
	SUCC	MFPT	SUCC	MFPT
0.25	-	-	0	-
0.125	-	-	0	-
0.0625	30	95,835 (86,611)	2	365,150 (16,051)

Standard deviation values are reported in parentheses.

Conclusion

Since the success rate for mutation-only upGA is less than 10% when problem size N is larger than 64, the hypothesis that HIFF-M is difficult for mutation-only upGA to solve is confirmed when N > 64.

Discussion

Figure 5.5 portrays best-so-far fitness graphs for randomly chosen unsuccessful mutation-only upGA HIFF-M runs. To reveal the true difficulty for mutation-only upGA to reach the optimal fitness value of 127, genotypes at the end of the runs in these graphs are given below.

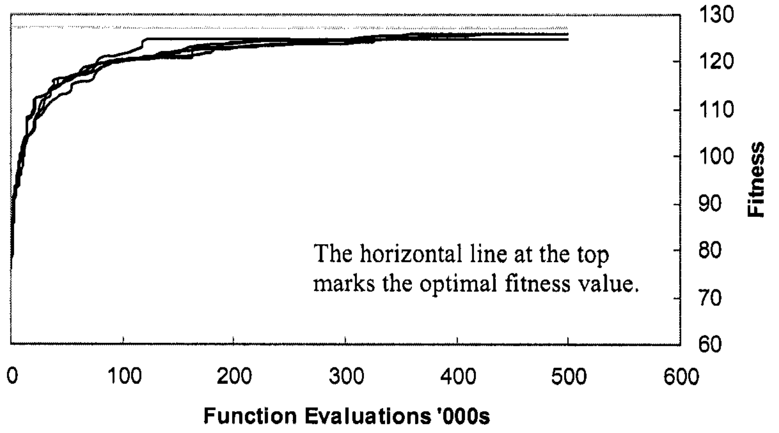


Figure 5.5 Best-so-far fitness graphs for unsuccessful mutation-only upGA HIFF-M runs

Genotypes at the end of the mutation-only upGA runs graphed in Figure 5.5:

Mutation-only upGA P_m=0.0625 HIFF-M N=128

```
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111 11111111 11111111 11110000 11111111
Fitness: 126
00000000 00000000 11111111 11111111 00000000 00000000 11111111 11111111
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
Fitness: 125
00000000 00000000 00000000 00000000 00000000 00110000 00000000 00000000
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
Fitness: 125
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000100 11111111 11111111 11111111 11111111
Fitness: 125
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11110000
Fitness: 126
```

Recall that mutation-only upGA uses random mutation (section II.B). Although the fitness of these genotypes are only 1 or 2 fitness points below the optimal fitness value of 127, to get these additional points mutation-only upGA has to mutate contiguous locations of a genotype to the same value. But mutation-only upGA selects genotype locations to mutate at random, and resets the value of a selected location to a random value of the alphabet. Thus it is difficult for mutation-only upGA to solve HIFF-M.

The best-so-far fitness graphs for mutation-only upGA show distinct evolutionary behavior from those for RMHC (Figure 5.2) and for MMHC (Figure 5.4). Fitness increases very rapidly at the beginning for RMHC and MMHC, but fitness increase for mutation-only upGA is more gradual. At the early stages in evolution, around the first 10,000 function evaluations, both RMHC and MMHC outperform mutation-only upGA in terms of the best fitness value attained so far. But this is a minor point. What matters, if HIFF-M is to be a HC < GA problem, is for an EA which uses the idea of crossover, i.e. a GA, with or without mutation, to outperform all the mutation-only EAs in this chapter.

Implication

The mutation-only upGA experiment on HIFF-M (N=128), serves as a baseline for upGA experiments in chapter 7. The degree to which a comparable upGA (which by default uses both and only crossover and mutation) outperforms the mutation-only upGA on a problem is a measure of the importance of crossover to the problem. Given the poor success rate of mutation-only upGA for HIFF-M, there is a lot for crossover to do.

5.2.2 Genetic algorithm with Random Crossover (GARC)

Hypothesis

HIFF-M is difficult for mutation-only GAs to solve. The success rate of crossover-only upGA with random crossover (GARC) for HIFF-M should be less than 10%.

Method

Genetic Algorithm with Random Crossover (GARC) (Jones, 1995 p.77) was proposed as a way to determine whether *the idea of crossover* and a population are necessary for a GA to solve a problem. Using the idea of crossover involves the exchange of segments of genetic material between at least two adapting individuals. The idea of crossover is contrasted with *the mechanics of crossover*. When using only the mechanics of crossover, segments of a genotype are also changed, but to random values instead (and not to values adapted in another individual). Hence, Jones (1995) calls a crossover operation that does not use the idea of crossover but still uses the mechanics of crossover, *random crossover*. Random crossover is essentially macro-mutation since a segment of a genotype is reset to random values. If GARC solves a problem more easily than a comparable GA (which uses the idea of crossover), then the idea of crossover is not contributing much to the performance of the GA and the problem fails the Headless Chicken Test[9].

In this section, GARC is crossover-only upGA ($P_x = 1.0$) and 1-2 point random crossover instead of 1-2 point crossover is used in step 4 (Figure II.A). Other parameter settings are detailed in section II.B.

1-2 point random crossover

```
d := random integer in [1, N – 1]
x := random integer in [0,  N – 1]
y := (x + d) % N
if (x > y) swap x and y
c1 := p1 and randomly assign 0 or 1 to c1[x...y-1]
c2 := p2 and randomly assign 0 or 1 to c2[x...y-1]
```

[9] The Headless Chicken Test is so named because claiming that crossover is useful for solving a problem when only the mechanism of crossover suffices is similar to calling a headless chicken a chicken (Jones, 1995).

Results

None of the GARC runs were successful on HIFF-M. The average best fitness at the end of the runs is 119.7 with a standard deviation of 1.2360.

Conclusion

Since the success rate of crossover-only upGA with random crossover (GARC) for HIFF-M is less than 10%, the hypothesis that HIFF-M is difficult for mutation-only GAs to solve is confirmed.

The GARC (this section) and mutation-only upGA (section 5.2.1) experiments confirm the general hypothesis of section 5.2 that HIFF-M is difficult for multi-individual mutation-only EAs to solve and suggest that the idea of crossover will be beneficial to EAs solving HIFF-M (this is confirmed in section 6.1).

Discussion

Figure 5.6 compares the best-so-far fitness graphs for randomly chosen unsuccessful GARC HIFF-M runs with the same for MMHC HIFF-M runs. The MMHC runs outperform GARC right from the beginning (Figure 5.6). The difference between GARC and MMHC is GARC uses a population of more than one whereas population size for MMHC is one. Thus, introducing multi-individuals into an EA to solve HIFF-M with macro-mutation has adversely impacted EA optimizing performance.

Random crossover is similar to macro-mutation with $P_m = 1.0$. Therefore GARC faces the same difficulties as MMHC (section 5.1.2) on HIFF-M.

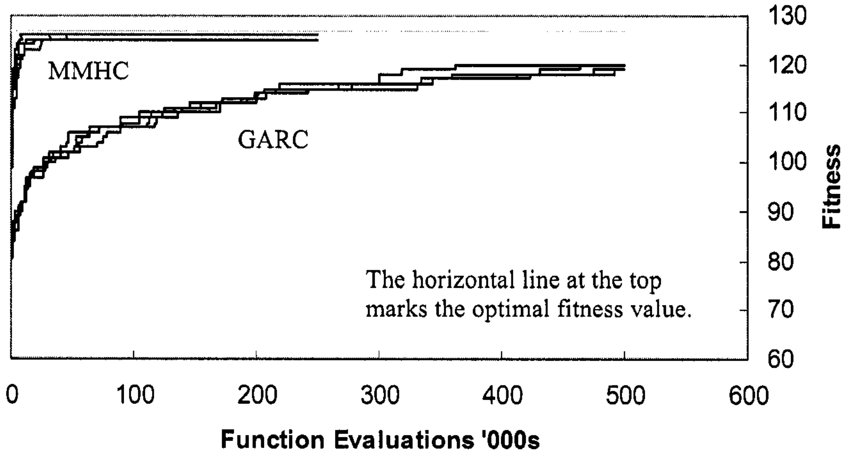


Figure 5.6 Best-so-far fitness graphs for unsuccessful GARC and MMHC HIFF-M runs

Genotypes at the end of the GARC runs graphed in Figure 5.6:

```
00001111 00001111 00000001 00000000 00000000 00010000 00000000 00000000
00000000 00000000 01000000 00000000 00000000 11111111 11111100 11111111
Fitness: 119
11101111 00000000 10110000 11111111 11111111 11111111 11111111 11111111
11111100 11111111 11111111 11111111 11111111 11111111 0000011 00000000
Fitness: 120
11110000 11110000 11111111 11110000 11111111 11111111 11111110 11111111
11110000 01000000 11111111 11111111 11111111 11111100 11111111 11111111
Fitness: 119
11111111 11111111 10111111 11110001 11111111 11111111 11111111 00000001
11111111 11111111 11111111 11101111 11101111 11111111 11111111 11111111
Fitness: 120
```

Implication

The RMHC (section 5.1.1) and mutation-only upGA experiments (section 5.2.1) confirm that random mutation alone, regardless of whether population size is 1 or more than 1, has difficulty solving the HIFF-M problem with a success rate > 90% within the given number of function evaluations.

The MMHC (section 5.1.2) and GARC (section 5.2.2) experiments confirm that macro-mutation alone, regardless of whether population size is 1 or more than 1, has difficulty solving the HIFF-M problem with a success rate > 90% when N > 64 within the given number of function evaluations.

The inference from all four of these experiments: RMHC, MMHC, mutation-only upGA and GARC, is: the idea of crossover, i.e. crossover proper, and by extension a population of at least two individuals will be necessary in an EA to solve the HIFF-M problem with a success rate > 90% when N > 64 within the given number of function evaluations.

In the next section, a version of GARC is formulated that is successful on HIFF-M. However, this version uses a constrained version of random crossover. Further, parent genotypes are deterministically and uniformly selected. Therefore, in principle HIFF-M is still difficult for multi-individual mutation-only EAs to solve and the idea of crossover will still be beneficial for EA optimizing performance on HIFF-M. The benefit of the idea of crossover to an EA optimizing HIFF-M is confirmed in chapter 6.

5.2.3 Deterministic GARC (dGARC)

Hypothesis

The GARC algorithm in section 5.2.2 can be modified so that it solves HIFF-M with success rate > 90%. If this is confirmed, then for HIFF-M to be a HC < GA problem, the GA must also outperform the modified GARC.

Method

Since MMHC reports some success in Table 5.3 for HIFF-M, two reasons are possible for the GARC failure in section 5.2.2:

(i) Random-crossover is ineffective because changes made to genotypes are too large.

(ii) The number of function evaluations (trials) per individual in GARC is insufficient or inappropriately distributed.

The deterministic version of GARC (dGARC) is a modification of GARC which addresses the two points above. First, instead of 1-2 point random-crossover, dGARC uses macro-mutation with $P_m = 0.125$ (mutation rate with the best success rate in Table 5.3). This means that the maximum distance between two cut-points (the number of genes mutated at one time) is now smaller than allowed in GARC. Macro-mutation with $P_m = 0.125$ is not crossover since crossover in a GA by definition does not restrict the distance between cut-points.

Second, in place of uniform stochastic parent selection (step 3 in upGA), dGARC allocates trials uniformly and in a deterministic manner as follows: place genotypes in a sequence and step through this sequence one genotype at a time going back to the head of the sequence when the end is reached. At each step, allocate a trial to the genotype at the current step. Allocating a trial means that the genotype undergoes macro-mutation and is replaced by its offspring if it is not fitter than its offspring.

Results

dGARC successfully optimized the HIFF-M problem in all 30 runs. MFPT was 364,710 with standard deviation of 55,321 which is a marked signal-to-noise ratio improvement over that for MMHC. Figure 5.7 give the best-so-far fitness graphs for randomly chosen successful dGARC HIFF-M runs. They show a more gradual fitness increase in the first 50,000 function evaluations than MMHC, but dGARC is more successful than MMHC on HIFF-M.

Conclusion

Since dGARC is 100% successful on HIFF-M, the GARC algorithm in section 5.2.2 can be modified so that it solves HIFF-M easily. dGARC, a mutation-only EA, is the most successful EA for HIFF-M so far. Therefore, for HIFF-M to be a HC < GA problem, the GA must also outperform dGARC.

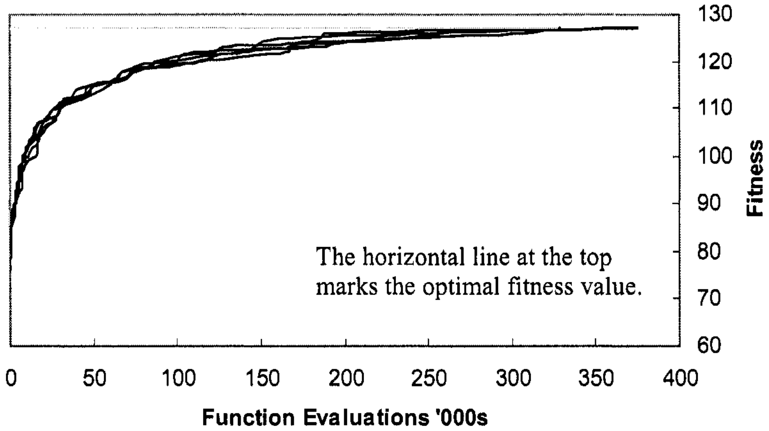


Figure 5.7 Best-so-far fitness graphs for successful dGARC HIFF-M runs

Discussion

Alternatives to dGARC were tested but they were much less successful than dGARC. For example, dGARC but with random mutation instead of macro-mutation, dGARC with macro-mutation $P_m = 1.0$ and GARC with macro-mutation $P_m = 0.125$ instead of 1-2 point random crossover all had a less than 10% success rate on HIFF-M. These less successful alternatives confirm that both macro-mutation at a smaller mutation rate than 1.0 (and therefore is not random-crossover) and uniformly deterministic parent selection are necessary elements for dGARC success on HIFF-M.

Implication

The conclusion of this section is not yet reason to reject the overall hypothesis that HIFF-M is a distinct solution from HIFF-D to the HC < GA problem if a GA is found that can outperform dGARC either by solving HIFF-M (of the same size or larger) more efficiently. upGA is one such GA (chapter 7).

5.3 Chapter Summary

The overall objective of the experiments in this chapter was to show that HIFF-M is difficult for mutation-only evolutionary algorithms (EAs), in particular RMHC, to solve. This objective is met in principle. HIFF-M is confirmed to be difficult for single-individual EAs, i.e. RMHC and MMHC, and multi-individual EAs, i.e. mutation-only upGA and GARC, to solve.

HIFF-M is difficult for RMHC to solve (section 5.1.1). Therefore, HIFF-M's inter-dependency matrix, though different from that of the original HIFF problems, does not make HIFF-M an easy problem for RMHC to solve.

HIFF-M is not as difficult as HIFF-D for MMHC to solve (section 5.1.2). HIFF-M is easier than HIFF-D for MMHC to solve. This demonstrates that HIFF-M is less mutation-hard than HIFF-D (section 4.2). Nevertheless, the mechanics of crossover or macro-mutation on a single individual is insufficient for solving HIFF-M.

Mutation-only upGA was more than 90% successful for HIFF-M when N=64 but less than 10% successful for HIFF-M when N=128 (section 5.2.1). This implies that mutation-only upGA algorithm is not a scalable solution for HIFF-M and that HIFF-M is difficult for mutation-only upGA to solve, when N > 64.

GARC was unable to solve HIFF-M (section 5.2.2). Hence, HIFF-M is difficult for multi-individual mutation-only EAs to solve and the idea of crossover will be evolutionarily advantageous to EA optimizing performance on HIFF-M. This is confirmed in section 6.1.

The RMHC (section 5.1.1) and mutation-only upGA experiments (section 5.2.1) confirm that random mutation alone, regardless of whether population size is 1 or more than 1, has difficulty solving the HIFF-M problem with a success rate > 90% within the given number of function evaluations.

The MMHC (section 5.1.2) and GARC (section 5.2.2) experiments confirm that macro-mutation alone, regardless of whether population size is 1 or more than 1, has difficulty solving the HIFF-M problem with a success rate > 90% when N > 64 within the given number of function evaluations.

The inference from all four of these experiments: RMHC, MMHC, mutation-only upGA and GARC, is: the idea of crossover, i.e. crossover proper, and by extension a population of at least two individuals will be necessary in an EA to solve the HIFF-M problem with a success rate > 90% when N > 64 within the given number of function evaluations.

A tailored version of GARC called dGARC is able to solve HIFF-M, even when N=128 (section 5.2.3). Hence, for HIFF-M to be a HC < GA problem, the GA must also outperform dGARC. This is confirmed in section 6.1 by the crossover-only GADC and later in section 7.1 by upGA.

6. Crossover-only Evolutionary Algorithms

The overall objective of the experiments in this chapter is to show that the idea of crossover, i.e. crossover proper, improves optimizing performance of evolutionary algorithms (EAs) on HIFF-M. Crossover (proper) has a homogenizing effect on genetic drift (section 2.2.2) and a pre-requisite to effective use of the idea of crossover is sufficient population genetic diversity as demonstrated clearly by the HIFF-D problem (preamble in Part II). Thus, the relative importance of maintaining genetic diversity in HIFF-D and HIFF-M genetic algorithm (GA) populations is examined in this chapter also. The main hypotheses tested in this chapter are:

Table 6.1 Hypotheses for chapter 6

Hypothesis	Section
HIFF-M is more easily solved by an evolutionary algorithm (EA) that uses the idea of crossover than by an EA that does not.	6.1
For both HIFF-D and HIFF-M, maintaining sufficient genetic diversity is important for good optimizing performance.	6.2
Homogenizing genetic drift is stronger in HIFF-D than in HIFF-M populations.	6.3
Some degree of homogenizing genetic drift is beneficial for solving HIFF-M.	6.4

Crossover-only Evolutionary Algorithms

The crossover-only EAs ($P_X = 1.0$) used in this section are:

(i) Genetic Algorithm with Deterministic Crowding or GADC (Mahfoud, 1995) (section 6.1); and

(ii) Crossover-only upGA (section 6.2); and

(iii) Gene Invariant Genetic Algorithm or GIGA (Culberson, 1994) (section 6.4).

Table 6.2 lists the relevant aspects in which the three crossover-only EAs listed above differ.

Table 6.2 Comparison of the crossover-only EAs

Crossover-only EA	Explicit diversity maintenance	Genetic drift possible
upGA	No	Yes
GADC	Yes	Yes
GIGA	Yes	No

upGA (Figure II.A) has no explicit diversification mechanism. Competition for survival replacement is based solely on objective fitness. Parent selection is performed in the same manner for both GADC and upGA – uniformly at random. *One-offspring crossovers*, where only one offspring from a crossover is accepted into the next population, are possible in both GADC and upGA. Hence, genetic drift is possible. The only difference between the algorithms for crossover-only upGA and crossover-only GADC (Figure 6.1) lies in survival selection (step 4).

Crossover-only GADC does explicit diversity maintenance by pairing parent and child genotypes so that a recombinant competes for survival replacement with the parent from whom it inherited most of its alleles. This pairing strategy reduces homogenizing genetic drift when only one offspring from a crossover is accepted into the next population. Genetic drift is prevented in GIGA (Figure 6.5) by uniform deterministic parent selection, and disallowing one-offspring crossovers and mutation.

Genetic drift

Genetic drift was defined in section 2.2.2 as 'the accidental and purposeful redistribution of alleles for each gene in a population'. Unlike mutation, crossover cannot introduce genes to new values (alleles). Crossover can only redistribute between recombinants, the alleles from their parent genotypes and all gene values of a recombinant must come from one of its parents. The process of allele redistribution is expected to lead to the discovery of new building-blocks (section 2.2.2) and fitter individuals which then, because survival selection favours the fit, replace less competitive building-blocks and less fit individuals in a (finite size) population. Such replacements, if uncontrolled, are likely to perturb the existing distribution of alleles per gene in the population and in the extreme, cause genes to fixate or converge on an allele.

Once a gene reaches fixation, there is nothing crossover can do to change the situation. In the extreme case, all genes converge and the entire population is taken over by a single individual. Let this individual be q. When this happens, the population is said to converge to q. If q is not a global optimum, then the population has prematurely converged. Hence crossover is a population homogenizing force. The degree to which a population is convergent is a measure of the strength of homogenizing genetic drift that the population experiences (assuming the initial population is randomly generated).

Measuring homogenizing genetic drift

The degree of homogenizing genetic drift in a population or population convergence is measured by the *Genome Convergence Ratio* (GCR). When GCR is not sensitive enough, *Averaged Site Convergence Ratio* (ASCR) is used to monitor genetic drift. ASCR and GCR are defined next

A gene reaches fixation when all individuals in the population have the same value for the gene. The extent to which a gene or site i is fixed to an allele a within a population of PS individuals at time t is measured by the *Site Convergence Ratio* (SCR) as follows: Let $G_t(g_i = a)$ be a multi-set containing all genotypes g in the population at time t that has a as the value of site i. Then SCR(t, $i = a$) = $|G_t(g_i = a)|$ / PS. *Averaged Site Convergence Ratio* (ASCR) at time t, is $\frac{1}{N}\sum_i SCR(t, i = a)$.

To calculate the SCR values in this thesis, *a* is set to '1'. Since the alphabet for all HIFF problems in this dissertation is {0, 1}, site convergence is indicated by SCR = 0.0 when the site has converged to the '0' allele, and by SCR = 1.0 when the site has converged to the '1' allele. A site is maximally diverse when its SCR is 0.5, (i.e. equal number of one and zero alleles for the gene in the population). Averaged Site Convergence Ratio (ASCR) monitors redistribution within each gene of the '1' allele in a population. If an initially random population experiences homogenizing genetic drift, then over time ASCR will diverge from the initial 0.5 mark towards either 0.0 or 1.0, and/or the variance associated with ASCR will increase.

The *genome convergence ratio* (GCR) is measured for a population at time *t* and it is the number of converged sites at time *t*, i.e. sites with SCR = 0.0 or SCR = 1.0 at time *t*, divided by the length of all genotypes in the population, i.e. the genome, N. So GCR × N yields the number of sites which have converged. GCR is bounded within [0.0, 1.0]. GCR = 0.0 for a population at time *t* when none of the genes in the population at time *t* have converged. GCR = 1.0 for a population at time *t* when all of the genes in the population at time *t* have converged.

GCR-ALL is end-of-run GCR values averaged over all 30 runs. GCR-SUCC is average end-of-run GCR values for successful runs only. GCR-FAIL is average end-of-run GCR values for unsuccessful runs only.

6.1 Crossover-only Genetic Algorithm with Deterministic Crowding (GADC)

Hypothesis

HIFF-M is more easily solved by an evolutionary algorithm (EA) that uses the idea of crossover than by an EA that does not. This hypothesis is confirmed if crossover-only GADC outperforms RMHC, MMHC, mutation-only upGA and dGARC. Specifically, crossover-only GADC should solve HIFF-M more successfully than RMHC, MMHC and mutation-only upGA, and should be as successful as dGARC but more efficient than dGARC.

Method

The EA that uses the idea of crossover is GADC because HIFF-D N=128 was solved by GADC with 1-point crossover, mutation and population size of 2,000 (Watson, 2002). Figure 6.1 gives the algorithm for crossover-only GADC. The GADC is this section is crossover-only and uses a smaller population size (1,000) than that used by Watson (2002) to solve HIFF-D. If it is possible for crossover-only GADC to solve HIFF-M with a smaller population size than that required in Watson's work for HIFF-D, then it is likely that homogenizing genetic drift is weaker in HIFF-M than in HIFF-D GA populations, as suggested in chapter 4, and confirmed in section 6.3.

In GADC, genetic drift is allowed but in a controlled manner. GADC's diversification mechanism pairs off parents and offspring which are more similar to each other (closer in Hamming space) in the competition for survival selection. Hence survival is no longer determined solely by objective fitness, and overhead is incurred for computing Hamming distances between 2 pairs of genotypes for each crossover event.

```
Step  Crossover-only GADC
1.    Create a population of PS random genotypes and evaluate the genotypes.
2.    If MaxEvals > 0 and optimum not found
         Go to step 3, Else stop.
3.    Select 2 distinct genotypes p1 and p2 at random from the population as follows:
            d := random integer in [1, PS – 1]
            i := random integer in [0, PS – 1]
            p1 := population at i
            p2 := population at (i + d) % PS
4.    Perform a 1-2 point crossover on p1 and p2 to create 2 offspring, c1 and c2.
      Evaluate c1 and c2.
      Calculate Hamming distance H, between the following pairs of genotypes:
      (p1, c1), (p1, c2), (p2, c2) and (p2, c1).
          If H(p1, c1) + H(p2, c2) < H(p1, c2) + H(p2, c1)
              Pair p1 with c1 and p2 with c2.
          Else
              Pair p1 with c2 and p2 with c1.
      For each pair of genotypes:
         If the offspring genotype is fitter than the parent genotype
            Replace the parent genotype with the offspring genotype
         Else
            Keep the parent genotype and discard the offspring genotype.
5.    MaxEvals := MaxEvals - 1
      Go to step 2.
```

Figure 6.1 Algorithm for crossover-only GADC

Results

Table 6.3 reports the optimizing performance of crossover-only GADC on HIFF-M. Crossover-only GADC solves HIFF-M more successfully than RMHC, MMHC and mutation-only upGA. The success rate of crossover-only GADC (PS=1,000) for HIFF-M is 100%. The success rates of RMHC and mutation-only upGA are less than 10% (Tables 5.2 and 5.5), while the best MMHC success rate is 50% (Table 5.3).

Table 6.3 Results for crossover-only GADC on HIFF-M, N=128

PS	SUCC	MFPT	Best fitness averaged over unsuccessful runs
1,000	30	94,921 (22,341)	-
500	25	76,382 (95,119)	126 (0)

Standard deviation values are in parentheses.

The evolutionary behavior of crossover-only GADC (Figure 6.2) is distinct from RMHC and MMHC. The RMHC and MMHC runs quickly reach a high fitness value while the fitness increase for crossover-only GADC runs is more gradual. Prior to approximately 60,000 function evaluations, both RMHC and MMHC outperform (attain higher best-so-far fitness values) crossover-only GADC. However, both RMHC and MMHC eventually fail to attain further fitness increase and are overtaken by crossover-only GADC (PS=1,000) as evolutionary time passes (function evaluations accumulate).

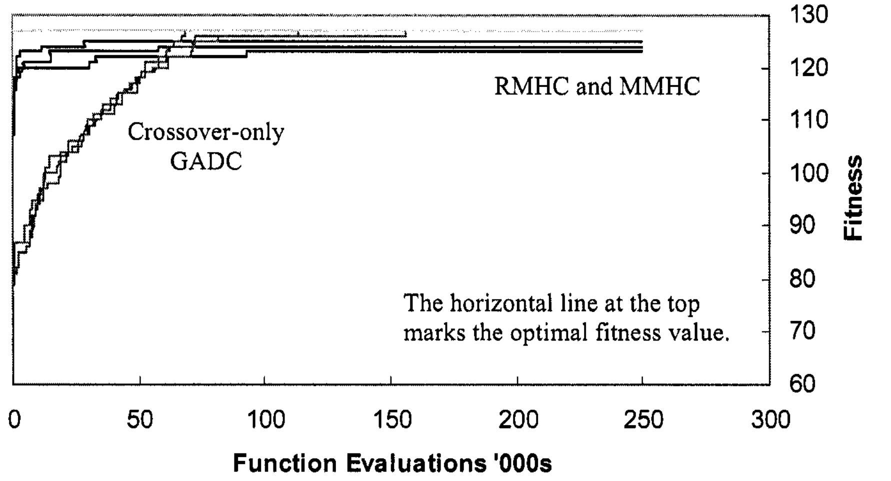


Figure 6.2 Best-so-far fitness graphs for RMHC and crossover-only GADC (PS=1,000) HIFF-M runs

Table 6.4 compares the optimizing performances of dGARC and crossover-only GADC. Crossover-only GADC (PS=1,000) is as successful as dGARC, but is more efficient than dGARC on the HIFF-M problem even though the population size (PS) of the crossover-only GADC is 1,000 and the PS of dGARC is only 128. The significance probability that dGARC's MFPT is the same as crossover-only GADC's MFPT is 0.000000% by the one-sided t-test which is less than 1%. Therefore, crossover-only

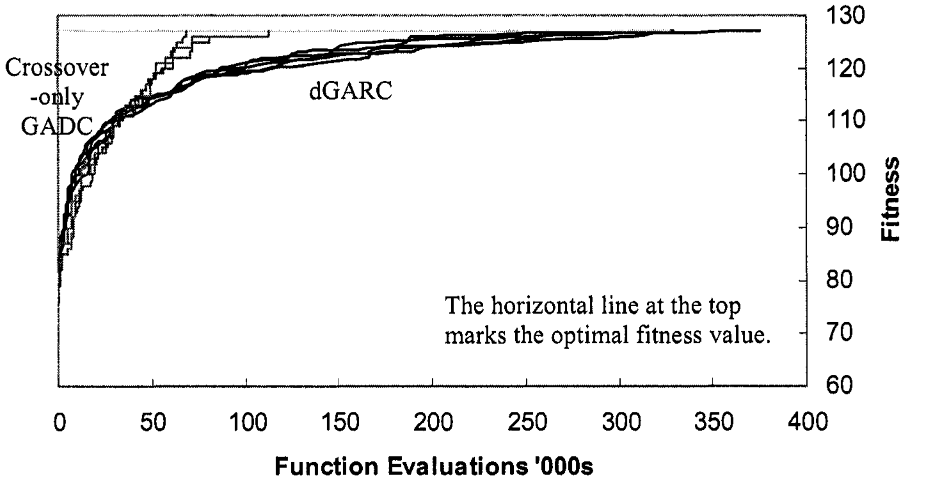
GADC (PS=1,000) is as successful as dGARC but more efficient (uses fewer function evaluations on average) than dGARC.

Table 6.4 dGARC vs. crossover-only GADC results for HIFF-M

Evolutionary Algorithm	SUCC	MFPT
dGARC (PS=128) (section 5.2.3)	30	364,710 (55,321)
Crossover-only GADC (PS=1,000)	30	94,921 (22,341)

Standard deviation values are in parentheses.

The best-so-far fitness graphs in Figure 6.3 show little difference between crossover-only GADC and dGARC runs before 50,000 function evaluations. After the 50,000 function evaluation mark though, the crossover-only GADC runs outperform the dGARC runs.



The horizontal line at the top marks the optimal fitness value.

Figure 6.3 Best-so-far fitness graphs for dGARC and for crossover-only GADC (PS=1,000) HIFF-M runs

Conclusion

Since crossover-only GADC outperforms RMHC, MMHC, mutation-only upGA and dGARC in terms of success rate and efficiency, the hypothesis that HIFF-M is more easily solved by an EA that uses the idea of crossover than by an EA that does not, is confirmed.

Discussion

The conclusion in this section demonstrates that the idea of crossover, i.e. crossover proper, and by extension a population of at least two individuals is advantageous to EA optimizing performance on HIFF-M. Crossover-only GADC uses the idea of crossover and solves HIFF-M with a 100% success rate and using significantly fewer function evaluations than an EA that does not use the idea of crossover, e.g. dGARC. None of RMHC, MMHC, mutation-only upGA or dGARC uses the idea of crossover. Therefore, the objective of this chapter to show that the idea of crossover is evolutionarily advantageous to EAs optimizing a HIFF-M problem is met.

Implication

Since HIFF-M is more easily solved by an EA that uses the idea of crossover and by extension a population of at least two individuals than by EAs which does not, HIFF-M is a solution to the HC < GA problem. The distinguishing feature of genetic algorithms is the use of the crossover operator (section 2.2.2). The GA in this instance is crossover-only GADC.

Since HIFF-M is a solution to the HC < GA problem, the changes made to the inter-dependency matrix of the original HIFF problems (HIFF-D and HIFF-C) that results in the HIFF-M function has not fundamentally affected the HC < GA nature of the original HIFF problems. This implies that the requirements for the original HIFF problems as specified in section 3.2 can be relaxed. Most notably, it is not necessary for all variables in the original HIFF problems to be inter-dependent on each other or for lower level interactions to weigh more than higher level interactions.

6.2 Crossover-only upGA

Hypothesis

For both HIFF-D and HIFF-M, maintaining sufficient genetic diversity is important for good optimizing performance. To confirm this hypothesis, the following sub-hypotheses need to be confirmed:

Sub-hypothesis 1: Homogenizing genetic drift should be stronger in crossover-only upGA than crossover-only GADC because crossover-only upGA does not explicitly maintain genetic diversity but crossover-only GADC does. Further, the population size (PS) for crossover-only upGA (128) is smaller than that for crossover-only GADC (1,000), and generally, homogenizing genetic drift is stronger in smaller than in larger populations. Crossover-only upGA's population size is set at 128 so that its results can be used in chapter 7. Crossover-only GADC's population size is 1,000 (section 6.1). The difference in homogenizing genetic drift between crossover-only upGA and crossover-only GADC should be reflected in GCR-ALL. Crossover-only upGA's GCR-ALL should be larger than crossover-only GADC's GCR-ALL for both HIFF-D and HIFF-M.

Sub-hypothesis 2: Crossover-only upGA performs worse than crossover-only GADC on both HIFF-D and HIFF-M.

If homogenizing genetic drift is stronger in crossover-only upGA than in crossover-only GADC (sub-hypothesis 1) and crossover-only upGA performs worse than crossover-only GADC on both HIFF-D and HIFF-M (sub-hypothesis 2), then the hypothesis that maintaining sufficient genetic diversity is important for good optimizing performance can be confirmed.

Method

The optimizing performance and Genome Convergence Ratio (GCR) values of crossover-only GADC and crossover-only upGA are compared. upGA (Figure II.A) has no diversification mechanism. Competition for survival is based solely on objective fitness. Parent selection is performed in the same manner for both GADC and upGA – uniformly at random. One-offspring crossovers, where only one offspring from a crossover is accepted into the next population, are possible in both GADC and upGA. The only difference between the algorithms for crossover-only upGA and crossover-only GADC lies in survival selection (step 4).

Results for sub-hypothesis 1

Table 6.5a reports the GCR values for crossover-only upGA and crossover-only GADC. Significance probabilities are summarized in Table 6.5b.

Table 6.5a GCR values for crossover-only EAs

Crossover-only EAs	HIFF-D			HIFF-M		
	GCR-ALL	GCR-SUCC	GCR-FAIL	GCR-ALL	GCR-SUCC	GCR-FAIL
upGA PS = 128	0.2133 (0.1428)	0.0578 (0.0487)	0.2444 (0.1349)	0.1245 (0.1123)	0.0104 (0.0040)	0.1530 (0.1080)
GADC PS = 1,000	0.0823 (0.1646)	0.0228 (0.0583)	0.4688 (0.0625)	0.0003 (0.0014)	0.0003 (0.0014)	-

Standard deviation values are in parentheses.

Table 6.5b Comparison of crossover-only upGA and crossover-only GADC GCR-ALL

H₀: There is no difference between the pair of GCR-ALL values compared.		Crossover-only GADC GCR-ALL	
		HIFF-D	HIFF-M
Crossover-only upGA GCR-ALL	HIFF-D	0.209027%	-
	HIFF-M	-	0.000068%

Significance probabilities by the one-sided t-test for the null hypothesis H_0

For the HIFF-D problem, the significance probability that there is no difference between crossover-only upGA's GCR-ALL (0.2133) and crossover-only GADC's GCR-

ALL (0.0823) is 0.209027% by the one-sided t-test (Table 6.5b). For the HIFF-M problem, the significance probability that there is no difference between crossover-only upGA's GCR-ALL (0.1245) and crossover-only GADC's GCR-ALL (0.0003) is 0.000068% by the one-sided t-test (Table 6.5b). Since these significance probabilities are less than 1%, the alternative hypothesis, that crossover-only upGA's GCR-ALL is larger than crossover-only GADC's GCR-ALL for both HIFF-D and HIFF-M is accepted.

Conclusion for sub-hypothesis 1

Since crossover-only upGA's GCR-ALL is larger than crossover-only GADC's GCR-ALL for both HIFF-D and HIFF-M, the hypothesis that homogenizing genetic drift is stronger in crossover-only upGA (PS=128) than crossover-only GADC (PS=1,000) is confirmed.

Results for sub-hypothesis 2

The optimizing performances of crossover-only upGA and crossover-only GADC are reported in Tables 6.6a and 6.6b respectively. Crossover-only upGA is less successful than crossover-only GADC on both HIFF-D and HIFF-M.

Table 6.6a Results for crossover-only upGA, PS = 128

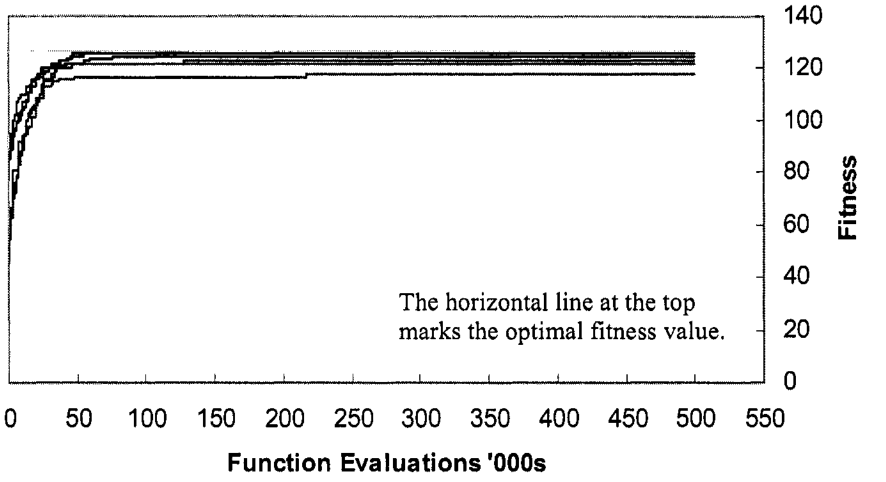
HIFF-D			HIFF-M		
SUCC	MFPT	Best fitness averaged over unsuccessful runs	SUCC	MFPT	Best fitness averaged over unsuccessful runs
5	133,740 (162,050)	122.32 (1.9087)	6	114,810 (49,657)	125.67 (0.4815)

Standard deviation values are in parentheses.

Table 6.6b Results for crossover-only GADC, PS = 1,000

HIFF-D			HIFF-M		
SUCC	MFPT	Best fitness averaged over unsuccessful runs	SUCC	MFPT	Best fitness averaged over unsuccessful runs
26	113,360 (56,892)	125.5 (0.5774)	30	94,921 (22,341)	-

Figure 6.4 depicts the best-so-far fitness graphs for randomly chosen unsuccessful crossover-only upGA runs for both HIFF-D and HIFF-M. These graphs do not show any fitness increase prior to end of runs. Crossover-only upGA is stuck at a local optimum because the population is insufficiently diverse to make further progress. Sites which converged do not converge to the same allele so an optimal genotype of all-ones or all-zeroes cannot be formed via crossover-only (this is verified by looking at the alleles of the converged sites at the end of the runs).



The horizontal line at the top marks the optimal fitness value.

Figure 6.4 Best-so-far fitness graphs for unsuccessful crossover-only upGA HIFF-D and HIFF-M runs

HIFF-M genotypes at the end of the crossover-only upGA runs graphed in Figure 6.4:

```
00000000 00000000 00000000 00000000 00001111 00000000 00000000 00000000
00000000 00000000 00000000 00000000 11111111 11111111 11111111 11111111
Fitness: 125
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111 00000000 00000000 00000000 00000000
Fitness: 126
00000000 00000000 00000000 00000000 11111111 11111111 11111111 11111111
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
Fitness: 126
```

HIFF-D genotypes at the end of the crossover-only upGA runs graphed in Figure 6.4:

```
00000000 00000000 00000000 00000000 11111111 11111111 00001111 11111111
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
```

```
Fitness: 122
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
11111111 00000000 00000000 00000000 11111111 11111111 11111111 11111111
Fitness: 123
11111111 11111111 11111111 11111111 11111111 11111111 00001111 00000000
11111111 11110000 11111111 11111111 11111111 11111111 11111111 11111111
Fitness: 118
```

Conclusion for sub-hypothesis 2

The results for sub-hypothesis 2 confirm that crossover-only upGA performs worse than crossover-only GADC on both HIFF-D and HIFF-M problems.

Conclusion

Since homogenizing genetic drift is stronger in crossover-only upGA than in crossover-only GADC, and crossover-only upGA performs worse than crossover-only GADC on both HIFF-D and HIFF-M, the hypothesis that maintaining sufficient genetic diversity is important for optimizing both HIFF-D and HIFF-M is confirmed.

Discussion

Since maintaining sufficient genetic diversity is important for optimizing for both HIFF-D and HIFF-M problems, it should follow that for a problem (either HIFF-D or HIFF-M) and an EA (either crossover-only upGA or crossover-only GADC), GCR-FAIL (the average GCR value at the end of unsuccessful runs) should be larger than GCR-SUCC (the average GCR value at the end of successful runs). Confirming this is further evidence of a positive relationship between low success rates and high levels of homogenizing genetic drift for both HIFF-D and HIFF-M.

Table 6.7 compares the GCR-SUCC and GCR-FAIL values of crossover-only upGA and crossover-only GADC reported in Table 6.5. For crossover-only GADC (PS = 1,000), the probability that there is no difference between HIFF-D's GCR-FAIL (0.4688) and HIFF-D's GCR-SUCC (0.0228) is 0.011310% by the one-sided t-test. All crossover-only GADC (PS = 1,000) HIFF-M runs were successful (Table 6.6b).

Table 6.7 Comparison of GCR-SUCC and GCR-FAIL values

H_0: There is no difference between the pair of GCR values compared.			GCR-FAIL		
			Crossover-only GADC PS=1000	Crossover-only upGA PS = 128	
			HIFF-D	HIFF-D	HIFF-M
GCR-SUCC	Crossover-only GADC PS=1,000	HIFF-D	0.011310%	-	-
		HIFF-M	-	-	-
	Crossover-only upGA PS = 128	HIFF-D	-	0.269717%	-
		HIFF-M	-	-	0.174016%

Significance probabilities by the one-sided t-test for the null hypothesis H_0

For crossover-only upGA, the probability that there is no difference between HIFF-D's GCR-FAIL (0.2444) and HIFF-D's GCR-SUCC (0.0578) is 0.269717% by the one-sided t-test. For crossover-only upGA, the probability that there is no difference between HIFF-M's GCR-FAIL (0.1530) and HIFF-M's GCR-SUCC (0.0104) is 0.174016% by the one-sided t-test.

Since these significance probabilities are less than 1%, the alternative hypothesis, that for a problem (either HIFF-D or HIFF-M) and an EA (either crossover-only upGA or crossover-only GADC), GCR-FAIL values is larger than GCR-SUCC values is accepted. This conclusion is further evidence of a positive relationship between low success rates and high levels of homogenizing genetic drift for both HIFF-D and HIFF-M. Thus maintaining sufficient population diversity is important for good optimizing performance for both HIFF-D and HIFF-M.

Implication

Given the low success rate of crossover-only upGA for HIFF-M (Table 6.6a) and the existence of a positive relationship between low success rates and high levels of homogenizing genetic drift for both HIFF-D and HIFF-M, it follows that homogenizing genetic drift in crossover-only upGA (PS = 128) is too strong for solving either HIFF-D or HIFF-M and that preserving sufficient genetic diversity in upGA is important for solving both HIFF-D and HIFF-M problems.

In genetic algorithms, mutation is the diversification or anti-homogenizing force. The degree to which a GA outperforms a comparable crossover-only GA on a problem is a measure of the importance of mutation to the problem. Whether mutation is up to task of supplying sufficient genetic diversity in upGA for both HIFF-D and HIFF-M is one of the questions investigated in chapter 7.

6.3 Crossover-only GADC and HIFF-D

Hypothesis

Homogenizing genetic drift is stronger in HIFF-D than in HIFF-M populations. This hypothesis is confirmed if with the population size the success rate of crossover-only GADC is higher for HIFF-M than for HIFF-D, and GCR-ALL for HIFF-D is larger than GCR-ALL for HIFF-M.

Method

Population sizes of 1,500, 1,000 and 500 are used. Since mutation is turned off in crossover-only GADC, any genetic diversity originates from the initial population and a larger population (of random genotypes) holds a greater wealth of genetic diversity than a smaller population.

Results

Table 6.8 reports the results of crossover-only GADC for HIFF-D and HIFF-M. HIFF-D is solved with 100% success rate by crossover-only GADC with a population of 1,500. In Watson (2002), the GADC that solved HIFF-D N=128, uses one-point crossover, mutation and a population size of 2,000. Crossover-only GADC is able to solve HIFF-D with a population size smaller than 2,000 because it uses 1-2 point crossover rather than 1-point crossover. However, HIFF-M is solved with 100% success rate by crossover-only GADC with a smaller population size, i.e. 1,000. As population size decreases from 1,000 to 500, the gap between crossover-only GADC success rates for HIFF-D and for HIFF-M widens.

Table 6.8 Results for crossover-only GADC

	HIFF-D			HIFF-M		
PS	SUCC	MFPT	Best fitness averaged over unsuccessful runs	SUCC	MFPT	Best fitness averaged over unsuccessful runs
1,500	30	133,168 (38,823)	-	-	-	-
1,000	26	113,360 (56,892)	125.5 (0.5774)	30	94,921 (22,341)	-
500	17	64,533 (38,713)	125.15 (0.8987)	25	76,382 (95,119)	126 (0)

Standard deviation values are in parentheses.

Crossover-only GADC GCR values

The GCR values of crossover-only GADC are reported in Table 6.5a. Table 6.9a compares the GCR values of crossover-only GADC for HIFF-D and HIFF-M. The significance probability that there is no difference between HIFF-D's GCR-ALL and HIFF-M's GCR-ALL is 0.534192% when population size (PS) is 1,000 and 0.032485% when PS = 500 by the one-sided t-test (Table 6.9a). Since both of these significance probabilities are less than 1%, the alternative hypothesis, that HIFF-D's crossover-only

GADC GCR-ALL is larger than HIFF-M's crossover-only GADC GCR-ALL is accepted.

Table 6.9a Comparison of HIFF-D and HIFF-M crossover-only GADC GCR values

H_0: There is no difference between the pair of GCR values compared.				HIFF-M					
				PS = 1,000			PS = 500		
				GCR-ALL	GCR-SUCC	GCR-FAIL	GCR-ALL	GCR-SUCC	GCR-FAIL
				0.0003 (0.0014)	0.0003 (0.0014)	-	0.1006 (0.1488)	0.0563 (0.1087)	0.3219 (0.1272)
HIFF-D	PS = 1,000	GCR-ALL	0.0823 (0.1646)	0.534192%	-	-	-	-	-
		GCR-SUCC	0.0228 (0.0583)	-	2.982734%	-	-	-	-
		GCR-FAIL	0.4688 (0.0625)	-	-	-	-	-	-
	PS = 500	GCR-ALL	0.3365 (0.3169)	-	-	-	0.032485%	-	-
		GCR-SUCC	0.0864 (0.1276)	-	-	-	-	21.594623%	-
		GCR-FAIL	0.6635 (0.1289)	-	-	-	-	-	0.060273%

Significance probabilities by the one-sided t-test for the null hypothesis H_0

HIFF-D's crossover-only GADC GCR-ALL is larger than HIFF-M's crossover-only GADC GCR-ALL because when HIFF-D crossover-only GADC runs are unsuccessful, they conclude with a larger GCR value than the HIFF-M crossover-only GADC unsuccessful runs.

The significance probability that there is no difference between HIFF-D's crossover-only GADC GCR-FAIL and HIFF-M's crossover-only GADC GCR-FAIL is 0.060273% (PS = 500) by the one-sided t-test (Table 6.9a). Since this significance probability is less than 1%, the alternative hypothesis, that HIFF-D's GCR-FAIL (0.6635) is larger than HIFF-M's GCR-FAIL (0.3219) is accepted. However, for the same PS, there is no difference at the 1% level of significance between HIFF-D's GCR-SUCC and HIFF-M's GCR-SUCC values (Table 6.9a).

Crossover-only upGA GCR values

Similar observations as those made for crossover-only GADC GCR values in the discussion before are found from the crossover-only upGA GCR values in Table 6.5a. Table 6.9b compares the GCR values of crossover-only GADC for HIFF-D and HIFF-M. The significance probability that there is no difference between HIFF-D's GCR-ALL and HIFF-M's GCR-ALL is 0.488345% by the one-sided t-test. Since this significance probability is less than 1%, the alternative hypothesis that HIFF-D's crossover-only upGA GCR-ALL (0.2133) is larger than HIFF-M's crossover-only upGA GCR-ALL (0.1245) is accepted.

Table 6.9b Comparison of crossover-only upGA GCR values

H_0: There is no difference between the pair of GCR values compared.			HIFF-M		
			GCR-ALL	GCR-SUCC	GCR-FAIL
			0.1245 (0.1123)	0.0104 (0.0040)	0.1530 (0.1080)
HIFF-D	GCR-ALL	0.2133 (0.1428)	0.488345%	-	-
	GCR-SUCC	0.0578 (0.0487)	-	4.743481%	-
	GCR-FAIL	0.2444 (0.1349)	-	-	0.604101%

Significance probabilities by the one-sided t-test for the null hypothesis H_0

HIFF-D's crossover-only upGA GCR-ALL is larger than HIFF-M's crossover-only upGA GCR-ALL because when crossover-only upGA HIFF-D runs are unsuccessful, they conclude with a larger GCR value than the crossover-only upGA unsuccessful HIFF-M runs. The significance probability that there is no difference between HIFF-D's crossover-only upGA GCR-FAIL and HIFF-M's crossover-only upGA GCR-FAIL is 0.604101% by the one-sided t-test (Table 6.9b). Since this significance probability is less than 1%, the alternative hypothesis, that HIFF-D's GCR-FAIL (0.2444) is larger than HIFF-M's GCR-FAIL (0.1530) is accepted. The significance probability that there is no difference between HIFF-D's crossover-only upGA GCR-SUCC and HIFF-M's crossover-only upGA GCR-SUCC is 4.743481% by the one-sided t-test (Table 6.9b). Since this significance probability is approximately 5%, there is no significant difference between HIFF-D's GCR-SUCC (0.0578) and HIFF-M's GCR-SUCC (0.0104) values.

Conclusion

Since with population sizes of 500 and 1,000 crossover-only GADC more easily solves HIFF-M than HIFF-D, and the GCR value at conclusion averaged over all runs (GCR-ALL) for HIFF-D is higher than that for HIFF-M, the hypothesis that homogenizing genetic drift is stronger in HIFF-D than in HIFF-M populations is confirmed. Further, for both crossover-only GADC and crossover-only upGA, unsuccessful HIFF-D runs conclude with larger GCR values on average than unsuccessful HIFF-M runs.

Discussion

In the absence of mutation to replenish genetic diversity, crossover-only GADC has to rely on the genetic diversity of the initial population. A larger population of random individuals is more diverse but also takes longer for an EA to process and may increase MFPT or require a larger `MaxEvals` needlessly. Since starting with the same amount of population diversity crossover-only GADC is more successful on HIFF-M than HIFF-D (Table 6.8), and there is a positive relationship between low success rates and high levels of homogenizing genetic drift (section 6.2), the implication is HIFF-M GA populations converge more slowly than HIFF-D GA populations. In other words, HIFF-M GA populations are better able to self-preserve their existing diversity than HIFF-D GA populations. In the absence of mutation, the ability of HIFF-M GA populations to better preserve their existing genetic diversity than HIFF-D GA populations is largely due to differences between the fitness distributions of HIFF-M and HIFF-D, as discussed in section 4.3.

Implication

The inference from the conclusions of sections 6.2 and 6.3 is that while maintaining sufficient population diversity is important for EA success on both HIFF-M and HIFF-D, maintaining sufficient population diversity will be a more serious issue for HIFF-D than for HIFF-M. This is confirmed in section 7.2.

6.4 Gene Invariant Genetic Algorithm (GIGA)

Hypothesis

Some degree of homogenizing genetic drift is beneficial for solving HIFF-M. To confirm this hypothesis, the following sub-hypotheses need to be confirmed:

Sub-hypothesis 1: For HIFF-M, homogenizing genetic drift should be stronger in crossover-only GADC than in GIGA populations. Crossover-only GADC in section 6.4 refers to crossover-only GADC with PS = 1,000 unless otherwise stated. Sub-hypothesis 1 is confirmed if the Averaged Site Convergence Ratio (ASCR) for GIGA hovers around 0.5 and remains invariant over evolutionary time (which it should by definition), while the ASCR for crossover-only GADC deviates either towards 0.0 or 1.0.

ASCR is used instead of GCR because both GADC and GIGA restrict site convergence to some extent. As a result, GCR is not sensitive enough to reflect differences between homogenizing genetic drift patterns in the GADC and GIGA populations.

Sub-hypothesis 2: For HIFF-M, crossover-only GADC should have a smaller MFPT value than GIGA, even though crossover-only GADC uses a larger population than GIGA.

If homogenizing genetic drift is stronger in crossover-only GADC than in GIGA populations (sub-hypothesis 1 is confirmed) and crossover-only GADC has a smaller MFPT value than GIGA (sub-hypothesis 2 is confirmed), then the hypothesis that some degree of homogenizing genetic drift is beneficial for solving HIFF-M is confirmed.

Method

The experiment is conducted by comparing the optimizing performance and homogenizing genetic drift of crossover-only GADC with GIGA. The algorithm for GIGA is outlined in Figure 6.5. GIGA is used since GIGA does not permit any genetic drift throughout a run, and so provide a clear contrast in terms of the effect of homogenizing genetic drift for a problem.

Genetic drift is prevented in GIGA by uniform deterministic parent selection, disallowing one-offspring crossovers and no mutation. The version of GIGA used in the experiments is elitist and keeps its population sorted by fitness in non-ascending order. The GIGA population is sorted so that genotypes are paired up with mates that are close in fitness value. Other variations of GIGA were tried out, but these were less successful (for the given parameters) for both HIFF-D and HIFF-M. The reason pairing genotypes that are close in fitness values works better for HIFF-D and HIFF-M is genotypes that are close in fitness values are complements of each other (Figure 3.7), i.e. both HIFF-D and HIFF-M have bit-flip symmetry (section 3.3.2) and therefore likely to produce fitter offspring when recombined with each other at the right cut-points.

```
Step  GIGA
1.    Create a population of PS random genotypes and evaluate the genotypes.
      Sort population by fitness in non-ascending order.
      offset := 0.
2.    If MaxEvals > 0 and optimum not found
          Go to step 3, Else stop.
3.    Pair up consecutive genotypes, p1 and p2, from the current population as follows:
          p1 := population at (offset)
          p2 := population at ((offset + 1) % PS)
          offset := (offset + 1) % PS
4.    Perform a 1-2 point crossover on p1 and p2 to create 2 offspring, c1 and c2.
      Evaluate c1 and c2.
      If the fitter offspring is strictly fitter than the fitter parent
        Replace the pair of parent genotypes with the pair of offspring genotypes.
        Sort population by fitness in non-ascending order.
5.    MaxEvals := MaxEvals - 1
      Go to step 2.
```

Figure 6.5 Algorithm for GIGA

Given a sufficiently diverse initial population (the initial distribution of alleles per gene is monitored to ensure uniform distribution on average – see Figure 6.6) and enough evolutionary time, GIGA should be 100% successful on both HIFF-D and HIFF-M. In principle, a population two complementary genotypes should suffice for GIGA to find a solution. A population size of 128 is used for GIGA in the experiments so that the results can be used in chapter 7 to compare with the upGA results.

Figure 6.6 plots the Site Convergence Ratio (SCR) for three randomly chosen initial populations (PS = 128) to confirm that the SCR for each site is around 0.5, i.e. the population size is large enough that random initialization of a population distributes '0's and '1's fairly uniformly for each gene. A site with SCR = 0.5 means half the genotypes in the population has '0'as the allele for the site while the other half of the genotypes in the population has '1'as the allele for the site.

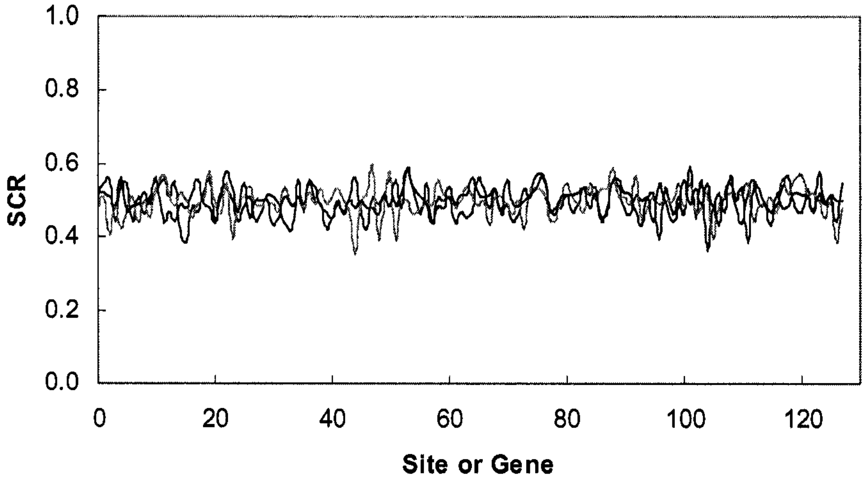


Figure 6.6 Site Convergence Ratio for each site in three initial populations, PS=128

Results for sub-hypothesis 1

Figure 6.7 plots the Averaged SCR (ASCR) for five randomly chosen crossover-only GADC HIFF-M runs over evolutionary time. The straight line in the middle marks ASCR for a GIGA run. ASCR for the GIGA run remains invariant at 0.5 over evolutionary time evincing the absence of homogenizing genetic drift, while the ASCR for crossover-only GADC HIFF-M runs deviate either towards 0.0 or 1.0 evincing the presence of homogenizing genetic drift over evolutionary time.

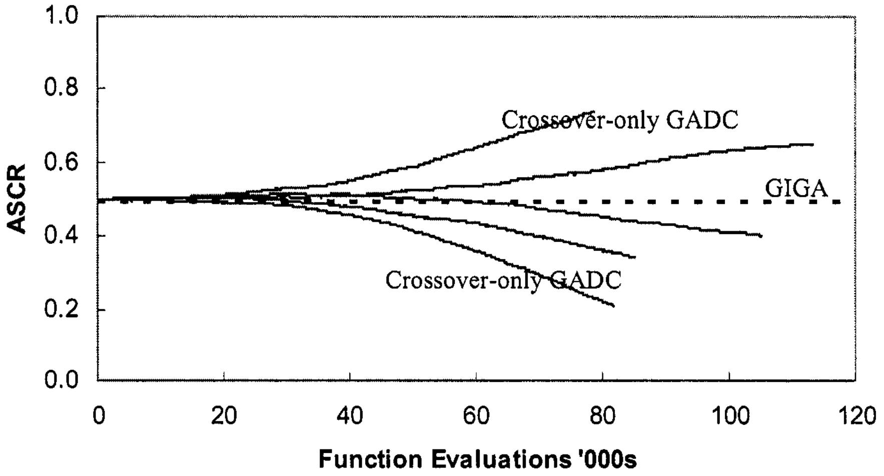


Figure 6.7 Homogenizing genetic drift for crossover-only GADC HIFF-M

Conclusion for sub-hypothesis 1

The results for sub-hypothesis 1 confirm that homogenizing genetic drift is stronger in crossover-only GADC than in GIGA populations.

Results for sub-hypothesis 2

Table 6.10 compares the optimizing performance of GIGA with that of crossover-only GADC. As expected, GIGA is 100% successful on both HIFF-D and HIFF-M.

Table 6.10 GIGA results compared with crossover-only GADC

EA	PS	HIFF-D		HIFF-M	
		SUCC	MFPT	SUCC	MFPT
GIGA	128	30	93,835 (21,790)	30	113,060 (28,889)
Crossover-only GADC	1,000	26	113,360 (56,892)	30	94,921 (22,341)
Crossover-only GADC	1,500	30	133,170 (38,823)	-	-

Standard deviation values are in parentheses.

For HIFF-M, GIGA is less efficient than crossover-only GADC (PS = 1,000) even though GIGA uses a population about an order smaller than crossover-only GADC (128 vs. 1,000). The significance probability that there is no difference between HIFF-M's GIGA MFPT and HIFF-M's crossover-only GADC MFPT is 0.436842% by the one-sided t-test. Since this significance probability is less than 1%, it is likely that HIFF-M's GIGA MFPT (113,060) is larger than HIFF-M's crossover-only GADC MFPT (94,921).

Conclusion for sub-hypothesis 2

The results for sub-hypothesis 2 confirm that for HIFF-M, crossover-only GADC has a smaller MFPT value than GIGA, even though crossover-only GADC uses a larger population than GIGA.

Conclusion

Since for HIFF-M, homogenizing genetic drift is stronger in crossover-only GADC than in GIGA populations (sub-hypothesis 1 is confirmed) and crossover-only GADC has a smaller MFPT value than GIGA (sub-hypothesis 2 is confirmed), the hypothesis that some degree of homogenizing genetic drift is beneficial for solving HIFF-M is confirmed.

Discussion

For HIFF-M, crossover-only GADC is as successful as but more efficient than GIGA (conclusion for sub-hypothesis 2). For HIFF-D, crossover-only GADC (PS = 1,500) is as successful as but less efficient than GIGA. The significance probability that there is no difference between HIFF-D's GIGA MFPT (93,835) and HIFF-D's crossover-only GADC (PS = 1,500) MFPT (133,168) is 0.003867% by the one-sided t-test which is less than 1%. Hence homogenizing genetic drift is more disadvantageous for an EA solving HIFF-D than for an EA solving HIFF-M. But, it will be advantageous for an EA solving HIFF-M to generate a moderate level of homogenizing genetic drift.

Implication

The homogenizing effect of crossover helps EAs focus their search efforts and increase exploitation of promising areas in the fitness landscape (section 2.2.2). Homogenizing genetic drift also helps EAs cut a more decisive path through neutral networks (section 4.3) because gene convergence reduces the number of genotypes reachable (that can be produced) by a population.

The focusing or exploitative effect of homogenizing genetic drift explains why crossover-only GADC needs significantly fewer function evaluations than both dGARC (section 6.1) and GIGA (section 6.4) to solve HIFF-M.

6.5 Chapter Summary

The overall objective of the experiments in this chapter was to demonstrate that the idea of crossover, i.e. crossover proper, is evolutionarily advantageous to (improves optimizing performance of) evolutionary algorithms solving HIFF-M as suggested in section 5.2.2. This objective is met in section 6.1. An evolutionary algorithm (EA) using the idea of crossover, i.e. crossover-only GADC, solved HIFF-M more successfully and efficiently than evolutionary algorithms that do not use the idea of crossover, i.e. RMHC, MMHC, mutation-only upGA and dGARC.

The usefulness of crossover in an evolutionary algorithm is intimately tied up with population diversity issues. Crossovers increase pressure on a population to become more homogenize, i.e. lose genetic diversity. If homogenizing genetic drift is too strong, a genetic algorithm (GA) population will converge prematurely and fail to locate a globally optimal solution. On the other hand, if homogenizing genetic drift is too weak, population convergence will be lethargic and the mean time to locate a globally optimum solution extended unnecessarily. Hence, an appropriate balance between homogenizing (exploitation) and diversifying (exploration) the genetic makeup of a population needs to be maintained to obtain optimal solutions efficiently. For this reason, chapter 6 examined the relative importance of maintaining genetic diversity in HIFF-D and HIFF-M GA populations.

For both HIFF-D and HIFF-M, a genetic algorithm which maintains sufficient levels of genetic diversity in its population (crossover-only GADC) is five times more successful than a genetic algorithm which does not (crossover-only upGA) (section 6.2, Tables 6.6a and 6.6b). Thus for both HIFF-D and HIFF-M, maintaining sufficient genetic diversity is important for good GA optimizing performance.

With all else being equal, homogenizing genetic drift is stronger in HIFF-D than in HIFF-M populations (section 6.3). Compared with HIFF-M, a larger population is needed by crossover-only GADC to solve HIFF-D with > 90% success rate. Since homogenizing genetic drift is stronger in HIFF-D than in HIFF-M populations, population diversity maintenance will be more important for solving HIFF-D than for HIFF-M.

HIFF-D runs concluded with larger GCR values on average than HIFF-M runs (section 6.3). This means a larger proportion of genes in HIFF-D populations converged. Since genes are free to converge to '0' or '1' and they do, composing a globally optimal solution of all-zeroes or all-ones becomes an impossibility for a crossover-only GA that has no means of reversing site convergence, i.e. no (effective) mutation.

Although there needs to be sufficient levels of genetic diversity in a GA population to solve HIFF-M (section 6.2), there can be too much of a good thing for HIFF-M (section 6.4). A GA that allows some genetic drift, i.e. crossover-only GADC, outperformed a GA that disallows genetic drift, i.e. GIGA, even though the population size of crossover-only GADC is about eight times bigger than GIGA's. The reverse is true for HIFF-D however. This shows that the HIFF-M problem is better able to take advantage of homogenizing genetic drift than the HIFF-D problem.

In short, maintaining sufficient levels of genetic diversity is important for GA success on both the HIFF-D and HIFF-M problems, but more critical for HIFF-D than HIFF-M.

The overall objective of the experiments in Part II is to show that HIFF-M is a distinct solution from HIFF-D to the HC < GA problem. To do this, the following four main hypotheses are tested:

(v) HIFF-M is difficult for RMHC to solve,

(vi) HIFF-M is more easily solved by GADC than RMHC,

(vii) HIFF-M is more or as easily solved by upGA than GADC, and

(viii) upGA solves HIFF-M more easily than HIFF-D.

Hypothesis (i) and (ii) are confirmed in chapters 5 and 6 respectively (Hypotheses (iii) and (iv) are confirmed in chapter 7). Thus, the changes made to the inter-dependency matrix of the original HIFF problems (HIFF-D and HIFF-C) that results in the HIFF-M function has not fundamentally affected the HC < GA nature of the original HIFF problems. This implies that the requirements for the original HIFF problems as stated in chapter 3 can be relaxed. In particular, the following are not necessary: (i) all variables to be directly inter-dependant on each other, (ii) fitness contributions to be scaled so that interactions at lower levels far outweigh interactions at higher levels, or (iii) the 'all-or-nothing' approach.

The best EA for HIFF-M found so far is crossover-only GADC (PS = 1,000) (section 6.1). For HIFF-M to be a distinct solution from HIFF-D to the HC < GA problem, HIFF-M needs to be solved more easily than HIFF-D by a different kind of GA. Given that HIFF-M populations are less susceptible to premature convergence (section 6.3) and are more able to capitalize on homogenizing genetic drift (section 6.4) than HIFF-D populations, a logical place to start is with a GA that maintains some level of genetic diversity (since this is still important for HIFF-M, section 6.2) but with less intensity than GADC. This different kind of GA is upGA which is the subject of study in chapter 7.

7. Mutation and Crossover

The overall objective of the experiments in this chapter is to show that HIFF-M is a distinct solution from HIFF-D to the HC < GA problem. So far, the best performing evolutionary algorithm (EA) for HIFF-M is the crossover-only GADC (PS=1,000) (section 6.1). Based on this finding, it was concluded that the idea of crossover is evolutionarily advantageous for solving HIFF-M (section 6.1), and that HIFF-M is more easily solved with a genetic algorithm that explicitly preserves genetic diversity, i.e. GADC, than a hill climber, i.e. RMHC. However, Watson (2006 chapter 6) also comes to the same conclusion for the HIFF-D problem using GADC and RMHC.

Hence, this chapter aims to distinguish HIFF-M's unique contribution to the HC < GA problem. Essentially, this is done by showing that it is easier for a genetic algorithm (GA) that does not explicitly preserve genetic diversity, i.e. upGA (Figure II.A), to solve HIFF-M than HIFF-D (section 7.2).

The main hypotheses tested in this chapter are:

Table 7.1 Hypotheses for chapter 7

Hypotheses	Section
HIFF-M is more or as easily solved by upGA compared with crossover-only GADC (PS=1,000).	7.1
upGA solves HIFF-M more easily than HIFF-D.	7.2
Both crossover and mutation play important roles in the success of upGA on HIFF-M.	7.3

In addition, section 7.4 explores the importance of upGA's survival selection strategy for crossover in step 4, and section 7.5 reports the optimizing performance of upGA on the HIFF-C and HIFF-II problems.

7.1 upGA on HIFF-M

Hypothesis

HIFF-M is more or as easily solved by upGA compared with crossover-only GADC (PS=1,000). All instances of crossover-only GADC in this section refer to crossover-only GADC which uses population size (PS) of 1,000. upGA should be as successful as crossover-only GADC, i.e. 100% success rate, and more efficient than or as efficient as crossover-only GADC, i.e. upGA's MFPT should be as small as or smaller than crossover-only GADC's MFPT (section 6.1).

Method

The algorithm for upGA is presented in Figure II.A. It incorporates crossover and mutation, and permits genetic drift. Parameter values in Table II.B are used unless stated otherwise.

Results

Table 7.2 reports the results for upGA on HIFF-M. For all three pairs of crossover-rate (P_x) and mutation-rate (P_m) settings, upGA is 100% successful, i.e. as successful as crossover-only GADC (Table 6.3).

Table 7.2 Results for upGA on HIFF-M

P_x	P_m	SUCC	MFPT	Significance probability of H_0
0.5	0.125	30	85,605 (68,689)	24.708758%
0.5	0.0625	30	80,225 (29,389)	2.675911%
0.25	0.0625	30	89,955 (15,606)	16.140505%

H_0: There is no difference between crossover-only GADC MFPT and upGA MFPT for HIFF-M.

However, the most efficient upGA is only as efficient as crossover-only GADC (PS = 1,000). The significance probability that there is no difference between the MFPT of upGA ($P_x = 0.5$, $P_m = 0.0625$) and the MFPT of crossover-only GADC is 2.675911% by the one-sided t-test which is more than the required 1%. The MFPT for crossover-only GADC is 94,921 with a standard deviation of 22,341 (Table 6.3).

The 95% confidence interval for MFPT of upGA ($P_x = 0.5$, $P_m = 0.0625$) is 80,225 ± 10,516. The 95% confidence interval for crossover-only GADC's MFPT is 94,921 ± 7,994. These two confidence intervals overlap each other so it is likely that the two MFPT values belong to the same underlying population. Thus, the difference between the two MFPT values is not statistically significant.

However, upGA's median FPT (72,048) is smaller than crossover-only GADC's median FPT (87,848). Medians are compared when averages may be affected by outliers. Figure 7.1 illustrates the distribution of MFPT values for crossover-only GADC and upGA ($P_x = 0.5$, $P_m = 0.0625$) HIFF-M runs.

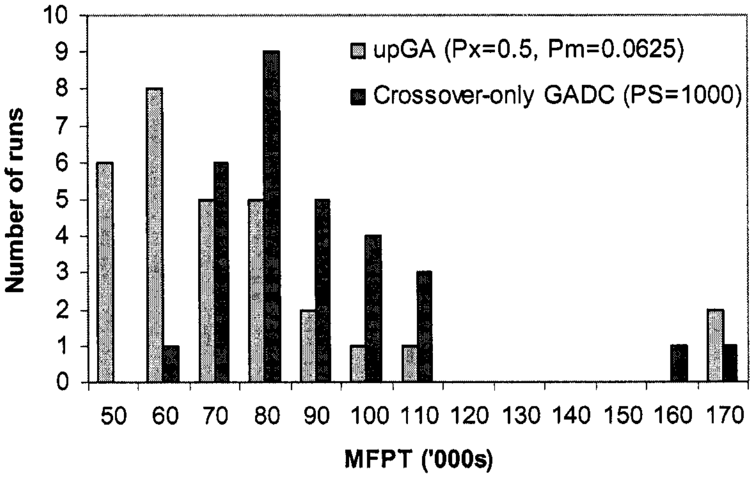


Figure 7.1 MFPT values for crossover-only GADC and upGA HIFF-M runs

In Figure 7.2, the best-so-far fitness graphs for crossover-only GADC and for upGA ($P_x = 0.5$, $P_m = 0.0625$) on the HIFF-M problem are compared. In general, the upGA runs outperform the crossover-only GADC runs throughout evolutionary time. That is, the statement that upGA solves HIFF-M as easily as crossover-only GADC is not dependent on the time (number of function evaluations) the observation is made.

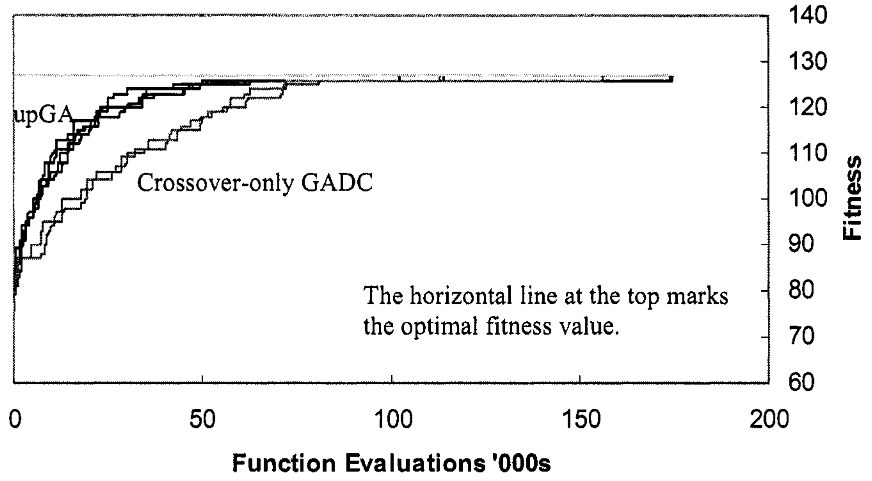


Figure 7.2 Best-so-far fitness graphs for crossover-only GADC and for upGA ($P_x = 0.5$, $P_m = 0.0625$) on HIFF-M

Mutation-only upGA solved HIFF-M N = 64 with 100% success rate and MFPT of 95,835 (86,611) (Table 5.5). upGA (PS = 64, $P_x = 0.5$, $P_m = 0.0625$) solves HIFF-M N = 64 also with 100% success rate but with MFPT of only 14,435 (2,726).

Conclusion

Since upGA is as successful as and as efficient as crossover-only GADC, the hypothesis that HIFF-M is more or as easily solved by upGA compared with crossover-only GADC is confirmed.

Additionally, since upGA is as efficient as crossover-only GADC and crossover-only GADC is more efficient than dGARC (section 6.1), it follows that upGA is also more efficient than dGARC at solving the HIFF-M problem, within the given parameters in Table II.B.

Discussion

Figure 7.3 plots the averaged site-convergence ratio (ASCR) (defined in chapter 6) over evolutionary time to observe genetic drift in four randomly chosen HIFF-M upGA ($P_x = 0.5$, $P_m = 0.0625$) runs.

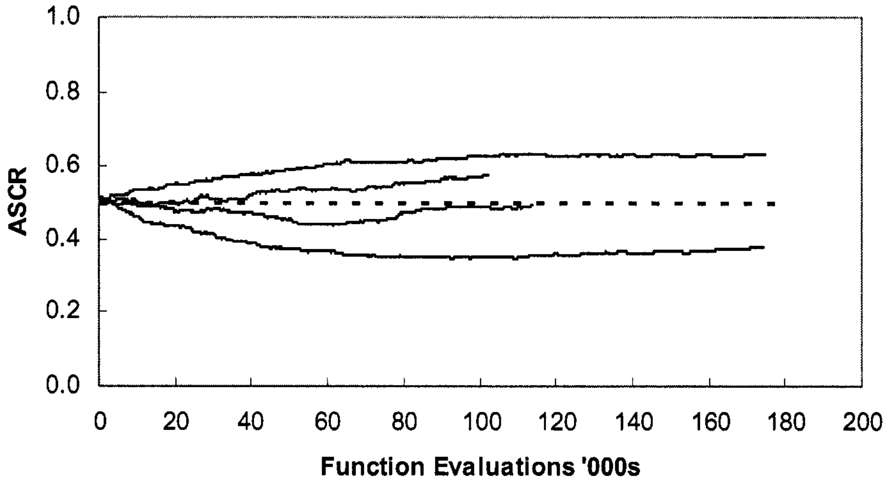


Figure 7.3 Genetic drift for upGA ($P_x = 0.5$, $P_m = 0.0625$) HIFF-M

Compared with the ASCR graphs for crossover-only GADC (Figure 6.7), the graphs in Figure 7.3 show earlier divergence from the 0.5 mark (denoted by the black-dotted horizontal line). This is partly due to upGA's smaller population size (128 for upGA compared with 1,000 for crossover-only GADC). Divergence from the 0.5 mark is indicative of homogenizing genetic drift. The ASCR for a population of 128 random genotypes is about 0.5 (Figure 6.6).

However, the divergence in Figure 7.3 is more constrained than in Figure 6.7, i.e. ASCR does not continually edge towards 0.0 or 1.0. Constrained ASCR divergence from the 0.5 mark indicates weakening homogenizing genetic drift (crossover stops being successful) and/or diversifying genetic drift (generated by successful mutation) that is sufficiently strong to counter the homogenizing genetic drift. In short, the genetic drift pattern produced by upGA ($P_x = 0.5$, $P_m = 0.0625$) in HIFF-M populations (Figure 7.3) is different from that produced by crossover-only GADC (PS=1,000) (Figure 6.7) due to the inter-play between crossover and mutation. The inter-play between crossover and mutation in upGA on the HIFF-M problem is studied in section 7.3.

7.2 upGA on HIFF-D

Hypothesis

upGA solves HIFF-M more easily than HIFF-D. upGA's success rate should be higher when solving HIFF-M than when solving HIFF-D. If upGA is as successful on HIFF-M as on HIFF-D, upGA should be more efficient solving HIFF-M than HIFF-D. That is, upGA's MFPT for HIFF-M should be smaller than upGA's MFPT for HIFF-D.

If this hypothesis is confirmed, then given the existing findings in Part II, the hypothesis that HIFF-M is a distinct solution from HIFF-D to the HC < GA problem is confirmed.

Method

upGA is run on HIFF-D and HIFF-M. The algorithm for upGA is presented in Figure II.A. It incorporates crossover and mutation, and permits genetic drift. Parameter values in Table II.B are used unless stated otherwise.

Results

Results are presented in Table 7.3. For all three pairs of crossover-rate (P_x) and mutation-rate (P_m), upGA's success rate is about twice higher when solving HIFF-M than when solving HIFF-D.

Table 7.3 Results for upGA

		HIFF-D		HIFF-M	
P_x	P_m	SUCC	MFPT	SUCC	MFPT
0.5	0.125	14	88,291 (28,340)	30	85,605 (68,689)
0.5	0.0625	18	78,848 (23,545)	30	80,225 (29,389)
0.25	0.0625	16	115,320 (31,171)	30	89,955 (15,606)

Standard deviation values in parentheses.

Best-so-far fitness graphs for unsuccessful HIFF-D upGA runs are given in Figure 7.4 and they show no sign of fitness improvement prior to the end of runs.

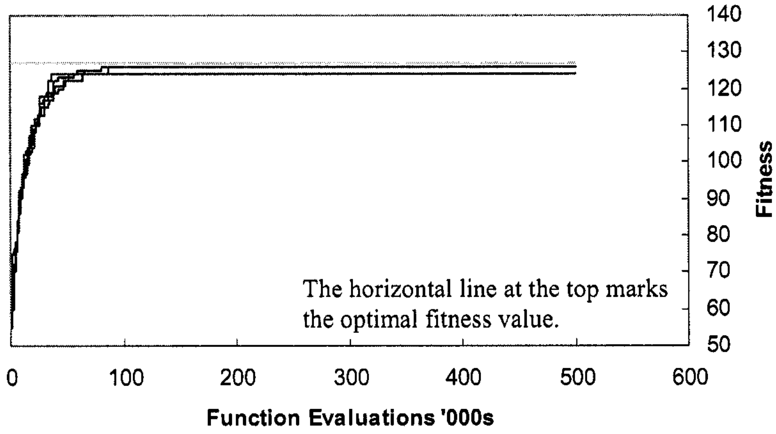


Figure 7.4 Best-so-far fitness graphs for unsuccessful HIFF-D upGA runs

Best-so-far fitness graphs for successful upGA HIFF-D and HIFF-M runs are compared in Figure 7.5. In general, the upGA HIFF-M runs outperform the upGA HIFF-D runs throughout evolutionary time. That is, the statement that upGA solves HIFF-M more easily than HIFF-D is not dependent on the time (number of function evaluations) the observation is made.

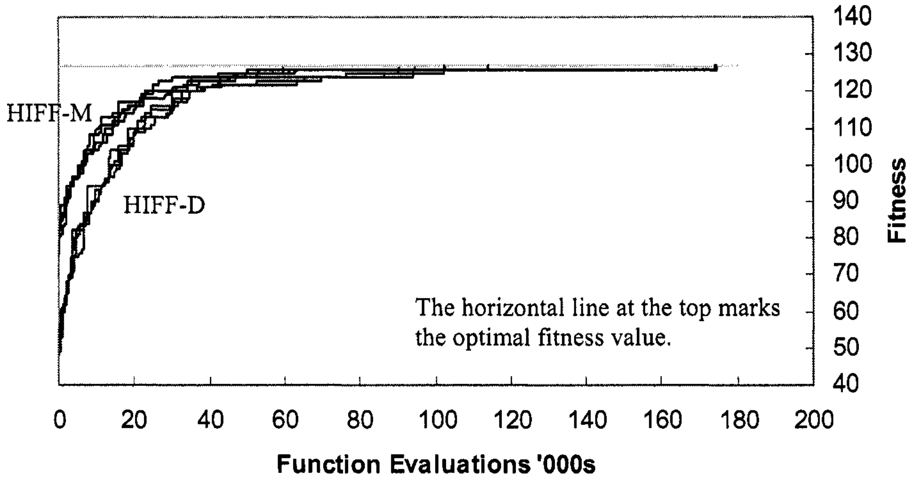


Figure 7.5 Best-so-far fitness graphs for successful HIFF-M and HIFF-D upGA runs

Conclusion

Since upGA's success rate is higher when solving HIFF-M than when solving HIFF-D and the best-so-far fitness graphs for unsuccessful HIFF-D runs show no sign of fitness improvement prior to the end of runs, the hypothesis that upGA solves HIFF-M more easily than HIFF-D is confirmed.

Previous work on HIFF-D (Watson, 2002) and findings in chapters 5 and 6 of this dissertation confirm that both HIFF-D and HIFF-M are solved more successfully by crossover-only GADC than RMHC. This makes both HIFF-D and HIFF-M solutions to the HC < GA problem. What differentiates HIFF-M from HIFF-D is upGA solves HIFF-M more easily than HIFF-D (this section) and upGA solves HIFF-M as successfully as, but more efficiently than crossover-only GADC (section 7.1). As such, one of the main hypotheses of this dissertation that HIFF-M is a distinct solution from HIFF-D to the HC < GA problem is confirmed.

Discussion

HIFF-M is a distinct solution from HIFF-D to the HC < GA problem because the genetic algorithm (i.e. upGA) that outperforms RMHC on HIFF-M can be a panmictic (single-population) GA without explicit diversification measures. Evidence that explicit diversification measures are necessary for a panmictic GA to solve HIFF-D has been investigated elsewhere by Watson (2002) and van Hoyweghen et al. (2002).

Why upGA solves HIFF-M more easily than HIFF-D

Next, the differences between HIFF-D and HIFF-M upGA runs are investigated to understand why upGA solves HIFF-M more easily than HIFF-D. The hypothesis forwarded is upGA solves HIFF-M more easily than HIFF-D because all other factors being equal, upGA's homogenizing genetic drift is stronger in HIFF-D than in HIFF-M populations, and consequently upGA converges prematurely for the HIFF-D problem. Evidence of stronger homogenizing genetic drift in HIFF-D than in HIFF-M populations was obtained for crossover-only GAs in section 6.2.

Table 7.4 GCR-ALL values for upGA runs

P_x	P_m	HIFF-D	HIFF-M	Significance probability of H_0.	H_0: There is no difference between corresponding HIFF-D and HIFF-M GCR-ALL values by the one-sided t-test.
0.5	0.125	0.1747 (0.1700)	0.0003 (0.0014)	0.000225 %	
0.5	0.0625	0.1190 (0.1425)	0.0016 (0.0052)	0.005286 %	
0.25	0.0625	0.0849 (0.0982)	0 (-)	0.002645 %	

Standard deviation values in parentheses.

Table 7.4 gives the GCR-ALL values for HIFF-D and HIFF-M runs. All significance probabilities in Table 7.4 is less than 1% which means HIFF-D upGA runs conclude with higher GCR-ALL values then HIFF-M upGA runs. Hence, all other factors being equal, upGA's homogenizing genetic drift is stronger in HIFF-D than in HIFF-M populations. Alternatively, HIFF-M populations converge more slowly than HIFF-D populations as suggested in section 4.3 and observed also in section 6.3.

Figure 7.6 plots the SCR values at the start and at the conclusion of five successful HIFF-M upGA and five successful HIFF-D upGA runs chosen at random. All SCR values are close to the 0.5 mark at the start of the runs (top figure). This is to be expected since all genes are assigned with values chosen uniformly at random from the common alphabet. However, at the conclusion of the runs, the SCR values have edged closer to either 0.0 or 1.0. This change in the distribution of alleles per gene (i.e. genetic drift) appears more pronounced and organized for the HIFF-D runs (middle figure) than for the HIFF-M runs (bottom figure) lending further support to the claim that upGA's homogenizing genetic drift is stronger in HIFF-D than in HIFF-M populations.

Figure 7.7 is a plot of the SCR values for one randomly chosen unsuccessful HIFF-D upGA (P_x=0.5, P_m=0.0625) run. In the middle of this figure, we see a chunk of sites converged to 0's and another chunk of sites converged to 1's. Because mutation is unable to compensate for this loss of non-inferior building-blocks (section 3.3.2), it is thus impossible for upGA to form a globally optimal genotype of all-zeroes or all-ones in this situation. The HIFF-D population depicted in Figure 7.7 has a synchronization problem (section 3.3.2). Thus, improving the success rate of upGA on HIFF-D is not just a matter of increasing MaxEvals.

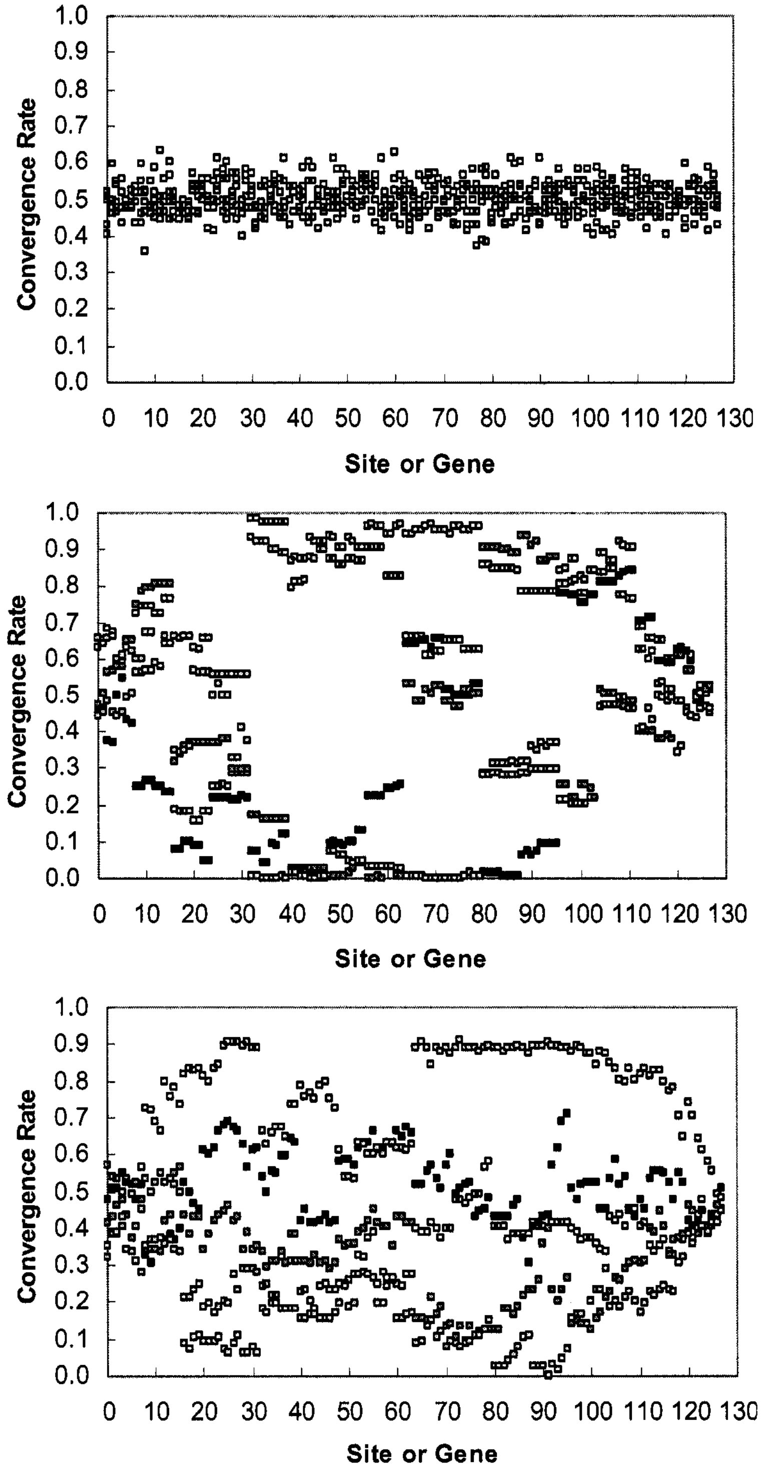


Figure 7.6 SCR values for randomly chosen upGA (P_x=0.5, P_m=0.0625) runs plotted at the start of the runs (top), at the end of HIFF-D runs (middle) and at the end of HIFF-M runs (bottom). Different colours distinguish different runs where necessary.

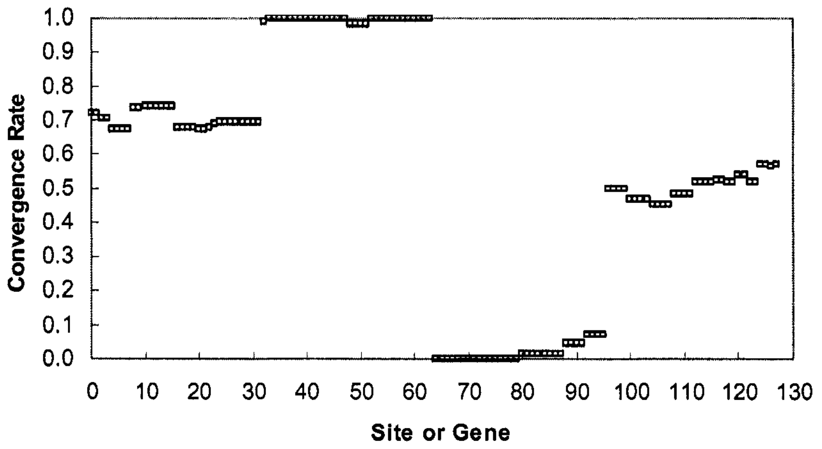


Figure 7.7 A HIFF-D population facing the synchronization problem

Why upGA's homogenizing genetic drift is stronger for HIFF-D than HIFF-M

Next, we look at the HIFF-D and HIFF-M upGA (P_x = 0.5, P_m = 0.0625) runs in more detail to understand why upGA's homogenizing genetic drift is stronger in HIFF-D than in HIFF-M populations. The hypothesis forwarded is upGA's homogenizing genetic drift is stronger in HIFF-D than in HIFF-M populations because mutation is more effective on HIFF-M than on HIFF-D. This is related to HIFF-M being less mutation-hard than HIFF-D (section 4.2), and HIFF-M's fitness distribution being less right-skewed than HIFF-D's (section 4.3).

Table 7.5 offers the first piece of evidence that mutation is more effective on HIFF-M than on HIFF-D. The addition of mutation increased upGA's success rate on both HIFF-D and HIFF-M but the increase is greater for HIFF-M. upGA (P_x = 0.5, P_m = 0.0625) is 2.6 times more successful than crossover-only upGA on HIFF-D. But upGA (P_x = 0.5, P_m = 0.0625) is 4.0 times more successful than crossover-only upGA on HIFF-M.

Table 7.5 Effect of mutation on upGA success rate for HIFF-D and HIFF-M

PS = N = 128	HIFF-D		HIFF-M	
	SUCC	MFPT	SUCC	MFPT
crossover-only upGA (Table 6.5a)	5	133,740 (162,050)	6	114,810 (49,657)
upGA ($P_x = 0.5, P_m = 0.0625$) (Table 7.3)	18	78,848 (23,545)	30	80,225 (29,389)

Standard deviation values in parentheses.

Next, the number of successful mutations and crossovers in upGA ($P_x = 0.5$, $P_m = 0.0625$) HIFF-D and HIFF-M runs are compared to obtain direct evidence that a larger number of mutations and a smaller number of crossovers succeed in HIFF-M runs than in HIFF-D upGA runs.

A mutation is successful if the mutant genotype is as fit as or fitter than and is different from its parent. A crossover is successful if a recombinant replaces its parent. The difference in the definition of a successful mutation and a successful crossover is due to the peculiarities of upGA (Figure II.A, step 4). A successful one-offspring crossover is a crossover where only one of the two parents is replaced by a recombinant. Successful one-offspring crossovers intensify homogenizing genetic drift because some of the genes of one parent are included twice into the population – once by the successful recombinant and once by the parent that is not replaced. Successful mutations intensify diversifying genetic drift.

Table 7.6a gives the number of successful mutations and crossovers in upGA ($P_x = 0.5$, $P_m = 0.0625$) for HIFF-M and HIFF-D averaged over all runs. Since only HIFF-D has unsuccessful runs and unsuccessful runs are longer (they run until MFPT is at least equal to MaxEvals) than successful runs, and therefore attempt more crossovers and mutations, Table 7.6b gives the number of successful mutations and crossovers in upGA

(P_x = 0.5, P_m = 0.0625) for HIFF-M and HIFF-D averaged over successful runs only. In either case, a larger number of mutations and a smaller number of crossovers succeed in HIFF-M than in HIFF-D upGA runs.

Table 7.6a Successful mutations and crossovers in upGA (P_x = 0.5, P_m = 0.0625) runs

Over all 30 runs	HIFF-D		HIFF-M		Significance probability of H_0
	Average	Median	Average	Median	
Successful mutations	2,125 (73.00)	2,136	3,284 (156.33)	3,254	0.000000%
Successful crossovers (includes one-offspring)	2,314 (270.19)	2,237	1,782 (132.09)	1,740	0.000000%
Successful one-offspring crossovers	2,298 (263.45)	2,229	1,767 (134.50)	1,725	0.000000%

H_0: There is no difference between the corresponding HIFF-D and HIFF-M averages by the one-sided t-test.

Table 7.6b Successful mutations and crossovers in upGA (P_x = 0.5, P_m = 0.0625) successful runs

Over successful runs only	HIFF-D		HIFF-M		Significance probability of H_0
	Average	Median	Average	Median	
Successful mutations	2,096 (68.76)	2,101	3,284 (156.33)	3,254	0.000000%
Successful crossovers (includes one-offspring)	2,128 (143.78)	2,148	1,782 (132.09)	1,740	0.000000%
Successful one-offspring crossovers	2,118 (143.98)	2,136	1,767 (134.50)	1,725	0.000000%

H_0: There is no difference between the corresponding HIFF-D and HIFF-M averages by the one-sided t-test.

Since all significance probabilities in Tables 7.6a and 7.6b are less than 1%, the following conclusions are made. Successful mutations averaged over all 30 runs for HIFF-M (3,284) is larger than that for HIFF-D (2,125). Successful crossovers averaged over successful runs only for HIFF-M (1,782) is smaller than that for HIFF-D (2,128). The number of successful one-offspring crossovers for HIFF-M is also smaller than that for HIFF-D. Successful one-offspring crossovers averaged over successful runs only for HIFF-M (1,767) is smaller than that for HIFF-D (2,118).

From the above analysis, upGA's homogenizing genetic drift is stronger in HIFF-D than in HIFF-M populations because (i) a larger number of mutations and a smaller number of crossovers succeed in HIFF-M than in HIFF-D upGA runs, (ii) the number of successful one-offspring crossovers for HIFF-M is smaller than that for HIFF-D, and (iii) successful crossovers particularly the one-offspring kind homogenize populations and successful mutations diversify populations.

Why a larger number of mutations succeed in HIFF-M than in HIFF-D upGA runs

A larger number of mutations succeed in HIFF-M than in HIFF-D upGA runs because HIFF-M's fitness function allows random mutation (upGA uses random mutation) to succeed even when genotypes are highly fit. Regardless of level (in a HIFF-M problem), each HIFF-M fitness point depends only on the satisfaction of one *iff* constraint between two variables.

This is not the case with HIFF-D. It becomes more difficult for random mutation to have any effect as HIFF-D genotypes become fitter. At higher levels, each HIFF-D fitness point depends on larger and larger string segments having the same value. This is hard for random mutation to deliver (section 5.1.1). For example, there are several HIFF-M genotypes in the Figure 7.8 that random mutation would be able to transform to all-zeroes or all-ones. But for HIFF-D, random mutation would need to flip either a segment of four zeroes or ones to obtain any increase in fitness.

Why a larger number of (one-offspring) crossovers succeed in HIFF-D than in HIFF-M upGA runs

For both HIFF-M and HIFF-D, at least 99% of successful crossovers are one-offspring crossovers (Tables 7.6a and 7.6b). A larger number of (one-offspring) crossovers succeed in HIFF-D than in HIFF-M upGA runs because once a segment of a

HIFF-D genotype that has the same allele emerges in a population and the size of this segment is large enough that it cannot be destroyed by random mutation, this segment will propagate through the population via crossover as a large segment of all-zeroes or all-ones is a highly fit segment, i.e. a good building-block.

In contrast, random mutation has some chance of destroying large segments of similarly valued genes in HIFF-M genotypes because substituting the satisfaction of lower level *iff* constraints with the satisfaction of higher level *iff* constraints is more possible due to HIFF-M's equally weighted interactions (section 3.4.3).

HIFF-D
1111 0000
0000 1111

HIFF-M	
0000 0001	1111 1110
0000 0011	1111 1100
0000 0100	1111 1011
0001 0000	1110 1111
0011 0000	1100 1111
0100 0000	1011 1111
0000 1111	1111 0000

Figure 7.8 Equally fit HIFF-D (left) and HIFF-M (right) genotypes

Implication

Since upGA solves HIFF-M more easily than HIFF-D, and upGA is a different kind of GA from GADC, given the previous findings in Part II, HIFF-M is a distinct solution from HIFF-D to the HC < GA problem.

7.3 Mutation and crossover in upGA on HIFF-M

The objective of this section is to examine the inter-play between crossover and mutation in upGA on the HIFF-M problem. Compared with the optimizing performances of mutation-only upGA (section 5.2.1) and crossover-only upGA (section 6.2), a hypothesis can be made that both mutation and crossover play important roles in the success of upGA solving HIFF-M.

Hypothesis

Both mutation and crossover play important roles in upGA solving HIFF-M successfully. The upGA in this section refers to upGA (P_x = 0.5, P_m = 0.0625). If upGA significantly outperforms both mutation-only upGA and crossover-only upGA, then the hypothesis can be confirmed.

Method

The optimizing performances of mutation-only upGA (section 5.2.1), crossover-only upGA (6.2) and upGA (section 7.2) are compared.

Results

Table 7.7 summarizes the effects of introducing mutation to crossover-only upGA and crossover to mutation-only upGA for the HIFF-M problem. In both instances, upGA (P_x = 0.5, P_m = 0.0625) is the resulting EA and the change increased success rate dramatically.

Table 7.7 Effect of mutation and crossover on upGA for HIFF-M

Evolutionary Algorithm	SUCC	MFPT	GCR-ALL
Mutation-only upGA (P_m = 0.0625) (Table 5.5)	2	365,150 (16,051)	0 (-)
Crossover-only upGA (Tables 6.5a and 6.6a)	6	114,810 (49,657)	0.1245 (0.1123)
upGA (P_x = 0.5, P_m = 0.0625) (Tables 7.3 and 7.4)	30	80,225 (29,389)	0.0016 (0.0052)

Standard deviation values in parentheses.

The introduction of mutation to crossover-only upGA increased its success rate on HIFF-M five-fold. The introduction of crossover to mutation-only upGA is more dramatic – it increased success rate by fifteen times (more evidence that crossover is evolutionarily advantageous for solving HIFF-M).

Conclusion

The results of this section confirm that both mutation and crossover play important roles in upGA solving HIFF-M successfully.

Discussion

The introduction of mutation to crossover-only upGA (which results in upGA) reduced homogenizing genetic drift. The significance probability that there is no difference between crossover-only upGA and upGA GCR-ALL values is 0.000096% by the one-sided t-test (Table 7.8) which is less than 1%. Therefore, GCR-ALL for crossover-only upGA (0.1245) is larger than GCR-ALL for upGA (0.0016).

Table 7.8 Genetic drift effect of mutation and crossover on upGA for HIFF-M

Evolutionary Algorithm	Significance probability of H_0.	H_0.
Mutation-only upGA ($P_m = 0.0625$)	5.498483%	There is no difference between mutation-only upGA and upGA GCR-ALL values by the one-sided t-test.
Crossover-only upGA	0.000096%	There is no difference between crossover-only upGA and upGA GCR-ALL values by the one-sided t-test.

The effect of crossover on mutation-only upGA is not evident by comparing the GCR-ALL value of mutation-only upGA and that of upGA. The significance probability that there is no difference between mutation-only upGA and upGA GCR-ALL values is 5.498483% by the one-sided t-test (Table 7.8) which more than 5%. Therefore, there is no significant difference between GCR-ALL for mutation-only upGA (0) and GCR-ALL for upGA (0.0016). Nevertheless, the homogenizing genetic drift effect of crossover in upGA is detected by observing the Averaged Site Convergence Ratio (ASCR) (chapter 6) over evolutionary time (Figure 7.7).

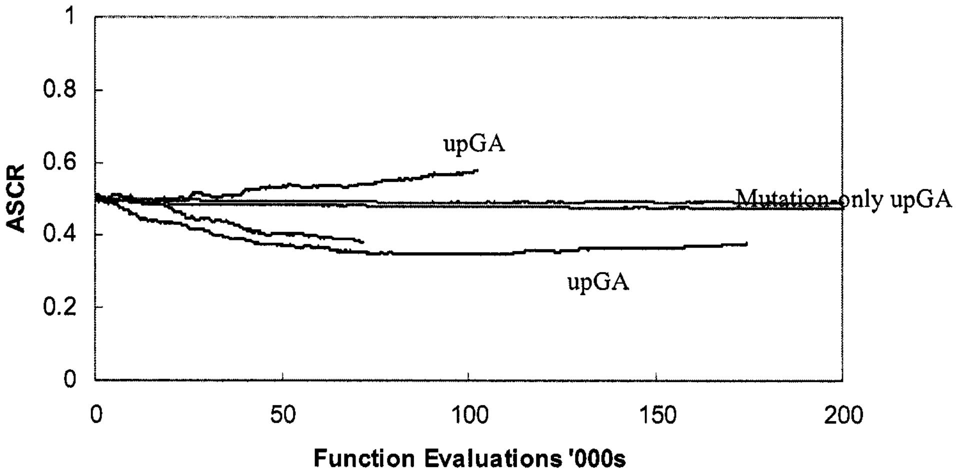


Figure 7.9 Genetic drift pattern of successful mutation-only upGA and upGA HIFF-M runs

In Figure 7.9, ASCR does not change for mutation-only upGA as expected since mutation is not a population homogenizing force. However, ASCR value for upGA runs do change and they deviate from 0.5 after a while indicating homogenizing forces of crossover is stronger than diversifying forces of mutation which as it should be since a moderate amount of homogenizing genetic drift (not so strong that sites converge) is evolutionarily advantageous for solving HIFF-M efficiently (section 6.4).

Figure 7.10 displays the best-so-far fitness graph for randomly chosen successful mutation-only upGA, crossover-only upGA and upGA HIFF-M runs. In general, the crossover-only upGA HIFF-M runs outperform the mutation-only upGA HIFF-M runs while the upGA HIFF-M runs outperform the other runs throughout evolutionary time.

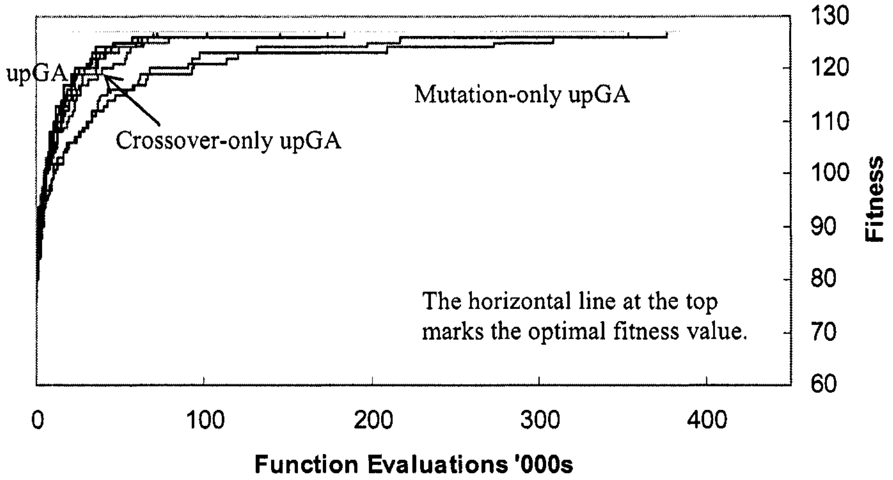


Figure 7.10 Best-so-far fitness graphs for successful mutation-only upGA, crossover-only upGA and upGA HIFF-M runs

Implication

Since mutation plays an important role in upGA success on HIFF-M, HIFF-M is a solution to a more difficult version of the HC < GA problem. Relying on (successful or effective) mutation to maintain sufficient genetic diversity in a genetic algorithm (GA) population also reduces problem difficulty for hill climbers. HIFF-M is a less mutation-hard problem than HIFF-D (section 4.2). Thus, HIFF-M is a more subtle solution to the HC < GA problem than HIFF-D.

7.4 upGA's survival selection strategy for crossover

This section takes a look at an important feature of upGA – its survival selection strategy for crossover in step 4 which allows a recombinant to survive only if it is strictly fitter than both its parents. upGA's optimizing performance for HIFF-M is negatively affected without this survival selection strategy.

Table 7.9 reports upGA's optimizing performance on HIFF-M with three other survival selection strategies. upGA does not perform as well with any of these alternative survival selection strategies. upGA's success rate declines, MFPT variance increases and homogenizing genetic drift intensifies (larger GCR-ALL) as the condition for survival becomes less restrictive.

Table 7.9 Performance of upGA ($P_x = 0.5$, $P_m = 0.0625$) with different survival selection strategies on HIFF-M, N=128

Survival selection strategies for upGA		SUCC	MFPT	GCR-ALL
Original (as in Figure II.A)	If F(c1) > F(p1) and F(c1) > F(p2) replace p1 with c1, otherwise discard c1 If F(c2) > F(p1) and F(c2) > F(p2) replace p2 with c2, otherwise discard c2	30	80,225 (29,389)	0.0016 (0.0052)
Alternative 1	If F(c1) > F(p1) or F(c1) > F(p2) replace p1 with c1, otherwise discard c1 If F(c2) > F(p1) or F(c2) > F(p2) replace p2 with c2, otherwise discard c2	4	161,600 (187,530)	0.8102 (0.1460)
Alternative 2	If F(c1) ≥ F(p1) and F(c1) ≥ F(p2), replace p1 with c1, otherwise discard c1 If F(c2) ≥ F(p1) and F(c2) ≥ F(p2), replace p2 with c2, otherwise discard c2	24	76,511 (59,993)	0.1438 (0.2491)
Alternative 3	If F(c1) > F(p1) replace p1 with c1, otherwise discard c1 If F(c2) > F(p2) replace p2 with c2, otherwise discard c2	8	64,387 (106,550)	0.7279 (0.1877)

The list of alternative survival selections strategies for upGA in Table 7.9 is not meant to be exhaustive, but to demonstrate to some extent, the importance of upGA having the survival selection strategy that it does in Figure II.A. More study is required to tie this observation with schema construction and destruction theory (Spears, 1998).

The upGA algorithm is actually a by-product of the $\mathcal{J}$ algorithm described in Appendix B, and as such shares similarities with it. In particular upGA's survival selection strategy works on the same principle as $\mathcal{J}$'s criterion for successful joins and exchanges – joins and exchanges succeed only if some extra fitness is the result (synergistic cooperation). This author speculates that there might be an interesting relationship between problem structure (Part III), specificity (Khor, 2007d) and GA difficulty. Specificity or isolation of parts has also been noted as an important feature of naturally evolved systems (Maslov and Sneppen, 2002).

7.5 HIFF-C and HIFF-II

This section looks at how upGA performs on the HIFF-C and HIFF-II problems. Since HIFF-C is one of the original HIFF problems and shares the same inter-dependency matrix as HIFF-D, HIFF-C is expected to be easier for genetic algorithms than hill climbers to solve. HIFF-II has a different inter-dependency matrix from HIFF-C. Like HIFF-M, HIFF-II's inter-dependency matrix (section 3.2.3) is sparser than HIFF-C. But it was found in section 5.1.1 that a sparser inter-dependency matrix did not make HIFF-M an easy problem for RMHC to solve, and in section 6.1 that HIFF-M is still a solution to the HC < GA problem. Therefore, HIFF-II is also expected to be easier for genetic algorithms than hill climbers to solve (this was confirmed with the GADC and RMHC algorithms in Khor, 2007a). Further, HIFF-II is more similar to HIFF-C than HIFF-M in terms of *SWO* values (Table 4.1), averaged FDC (Figures 4.2 and 4.3), fitness distribution (Figure 4.4) and MMHC success rate (Table 5.3).

Like HIFF-D and HIFF-M, HIFF-C and HIFF-II are difficult for RMHC to solve (Table 5.2). Table 7.10 reports upGA's optimizing performance on HIFF-C and HIFF-II. GIGA results are also reported for comparison with a genetic algorithm that does not allow genetic drift. In general, there is no significant difference between the MFPT values within each evolutionary algorithm; their 99% confidence intervals (CI) overlap. Thus, SUCC values form the main basis of comparison.

Table 7.10 GIGA and upGA optimizing performance for HIFF-C and HIFF-II

N = 128	GIGA			upGA $(P_x = 0.5, P_m = 0.0625)$			upGA $(P_x = 0.25, P_m = 0.0625)$		
	SUCC	MFPT	99% CI	SUCC	MFPT	99% CI	SUCC	MFPT	99% CI
HIFF-D	30	93,835 (21,790)	±10,247	18	78,848 (23,545)	±14,295	16	115,320 (31,171)	±20,073
HIFF-C	30	113,720 (32,736)	±15,395	15	52,418 (24,568)	±16,340	27	86,159 (30,116)	±14,929
HIFF-II	30	131,870 (44,697)	±21,020	16	70,273 (36,700)	±23,633	25	91,594 (43,116)	±22,212
HIFF-M	30	113,060 (28,889)	±13,586	30	80,225 (29,389)	±13,821	30	89,955 (15,606)	±7,339

Like HIFF-D, upGA is less successful than GIGA on both HIFF-C and HIFF-II. But, the success rate of upGA on both HIFF-C and HIFF-II improves considerably when the crossover rate (P_x) is reduced from 0.5 to 0.25. This observation, combined with the GIGA results, support the hypothesis that both HIFF-C and HIFF-II, like HIFF-D, induce stronger homogenizing genetic drift in an upGA population, than does HIFF-M. In contrast, the success rate for HIFF-M in Table 7.10 is 100% regardless of evolutionary algorithm and crossover rate, and there is no significant difference between HIFF-M's MFPT values. Thus, although both HIFF-C and HIFF-II can be easier for upGA to solve than RMHC, the conditions under which they do so are more specific than HIFF-M.

The HIFF-C and HIFF-II results in Table 7.10 are remarkably similar to one another but distinct from the results of both HIFF-D and HIFF-M. Further, the result for HIFF-D is distinct from that for HIFF-M. These similarities and differences are inline with the analysis made in chapter 4, particularly by the averaged FDC metric (Figures 4.2 and 4.3) and by the fitness distributions (Figure 4.4).

7.6 Chapter Summary

The overall objective of the experiments in this chapter was to show that HIFF-M is a distinct solution from HIFF-D to the HC < GA problem. This objective is met. Section 7.1 found that HIFF-M N=128 is as efficiently solved by upGA (P_x=0.5, P_m=0.0625) as crossover-only GADC (PS=1,000). Further, section 7.2 concludes that it is easier for a GA that does not explicitly preserve genetic diversity, i.e. upGA (Figure II.A), to solve HIFF-M than HIFF-D.

In addition, because of the important role mutation plays in the success of upGA on HIFF-M, section 7.3 concluded that HIFF-M is a solution to a more difficult version of the HC < GA problem. Having upGA rely on effective mutation for success on HIFF-M simultaneously increases the chance of success for mutation-only EAs such as RMHC. HIFF-D on the other hand works in the opposite direction: mutation becomes ineffective particularly for optimizing genotypes with high but suboptimal fitness. This reduces the chance of success for mutation-only EAs such as RMHC but it also creates diversity maintenance problems in GAs, i.e. the synchronization problem (Figure 7.7). Thus, a GA such as GADC which includes a mechanism for preserving genetic diversity in a population is required to solve HIFF-D (Watson, 2002).

Section 7.4 confirmed the importance of upGA's survival selection strategy for crossover in step 4 to upGA's high success rate on HIFF-M. Survival selection strategies with less restrictive conditions allowed too much homogenizing genetic drift. As a result, upGA's success rate declined and MFPT variance increased.

Finally, section 7.5 looked at upGA's performance on HIFF-C and HIFF-II. upGA did not perform as well on either HIFF-C or HIFF-II as on HIFF-M. Like HIFF-D, HIFF-

C and HIFF-II induced stronger homogenizing genetic drift in upGA populations than did HIFF-M.

That HIFF-II exhibited similar upGA optimizing performance as HIFF-C is anticipated by the analysis in chapter 4, but is also interesting in its own right because HIFF-II has a different inter-dependency matrix from HIFF-C and both HIFF-II and HIFF-C do not utilize the 'all-or-nothing' approach in their fitness calculation (section 3.2.2). HIFF-M's inter-dependency matrix is even more different from HIFF-II's and HIFF-C's, and upGA performs differently on HIFF-M than on either HIFF-II or HIFF-C. Like HIFF-II and HIFF-C, HIFF-M also calculates fitness of string by summing up the fitness contributions of all satisfied *iff* constraints. Thus, there may be certain aspects of how the variables of a problem interact (which is completely captured by the respective inter-dependency matrices of the HIFF-C, HIFF-II and HIFF-M problems) which are indicative of evolutionary difficulty for upGA. Differences between the inter-dependency matrices of HIFF-C, HIFF-II and HIFF-M did not affect RMHC's success rate (Table 5.2).

In Part III, properties of inter-dependency matrices are examined from three angles: degree distribution type, network transitivity and modularity, to determine which of these structural characteristics most influence the ability of a problem to distinguish the evolutionary capability of hill climbers from genetic algorithms.

Part III

Watson (2002) credits HIFF-D's modular structure (section 3.3.1) for its ability to distinguish the evolutionary capability of hill climbers from genetic algorithms. That is, HIFF-D is a solution to the HC < GA problem because of its modular structure. Having a modular structure does not imply that a problem is separable (section 3.3.2) or that inter-module links are unimportant (Watson and Pollack, 2005). Satisfying the inter-dependencies between modules is imperative for solving the HIFF-D problem.

HIFF-M also has a modular structure (section 3.3.1) and this is reinforced by the *Q* metric (a measure of modularity defined in section 8.2.3) in section 9.2.3 (Table 9.5) where it is revealed that HIFF-C (and by implication HIFF-D since *Q* is measured on the inter-dependency matrix **M** and HIFF-D has the same **M** as HIFF-C) and HIFF-M have the same *Q* value for the same problem size N.

But, the experiments in Part II confirm that HIFF-M and HIFF-D are distinct solutions to the HC < GA problem when the GA is upGA. Hence the motivation to find a problem characteristic other than modularity that better captures the ability of a problem to distinguish the evolutionary capability of hill climbers from genetic algorithms, particularly when the genetic algorithm does not explicitly maintain population genetic diversity and relies on mutation for genetic diversity, e.g. upGA.

Problem structure

Modularity is a structural characteristic of a network's topology. It describes the degree to which a network has identifiable subsets of nodes with more links within subsets than between subsets (section 8.2.3). A s*tructural characteristic* describes a property of a network's topology. A network's topology can be represented by an adjacency matrix **A** where $\mathbf{A}_{ij}$ is the number of links between nodes *i* and *j*. Structural characteristics of a network are then computed on the adjacency matrix for the network.

Test problems in Part III are viewed as networks of nodes denoting problem variables, and links representing *iff* constraints between problem variables. The topology of the network for a test problem is derived by transforming the test problem's inter-dependency matrix **M** (Part I) into an adjacency matrix **A** (this is illustrated in Figures 8.4a and 8.4b). Structural characteristics of a test problem are then computed on the adjacency matrix for the test problem. *Problem structure* refers to structural characteristics of test problems.

Common structural characteristics of real-world networks

The structural characteristics that will be examined in Part III are modularity, degree distribution type and transitivity. These three structural characteristics are chosen as candidates because they are commonly used to distinguish the topology of real-world networks from the topology of random networks (defined in section 8.1). Compared with random networks, real-world networks are more modular, exhibit more transitivity and have non-Poisson degree distributions (Newman, 2003). Definitions of these three structural characteristics and examples of real-world networks together with what constitutes their nodes and links are given in chapter 8.

Since real-world networks, in particular biological ones, have evolved on their own accord independent of global design and central control; and genetic algorithms more so than hill climbers (because genetic algorithms have multi-individual populations and use crossover) emulate the process of biological evolution, might there not be a connection between the structure of problems genetic algorithms are better than hill climbers at solving and the structure of real-world networks? After all, the process which brought real-world biological networks into being, i.e. biological evolution, is the same process that genetic algorithms (at their conception by John Holland in 1975) is based upon. That this connection exists forms the underlying assumption of the investigation in Part III and the reason why the influence of the three aforementioned structural characteristics of problems on evolutionary algorithm difficulty (in the sense of the HC < GA problem) is studied.

This dissertation is not alone in suggesting that structural characteristics of problems have an impact on the performance of search algorithms. For example, Walsh (1999) examined the small-world effect (section 8.2.1) in search problems such as graph coloring, time-tabling, quasigroup problems and satisfaction constraint problems on the cost of solving these problems and concluded that: "search problems met in practice may be neither completely structured nor completely random...simple topology features can have a large impact on the cost of solving search problems." However, the search algorithm in Walsh's study was not an evolutionary algorithm.

Objective and method overview

The objective of Part III is to find a structural characteristic other than modularity that better captures the ability of a problem to distinguish the evolutionary capability of hill climbers from genetic algorithms that do not explicitly maintain population genetic diversity. In specific terms, is there a structural characteristic which is a better indicator than modularity of when a problem will be more easily solved by upGA (Figure II.A) than a hill climber (HC)? If so, what is it?

The hill climber (HC) competing with upGA in Part III is MMHC (Figure 5.3), but the optimizing performance of RMHC (Figure 5.1) is also reported and analyzed for comparison. MMHC is more successful than RMHC on all test problems in Part III (Table 9.8). Comparing upGA's SUCC value (Part II) with MMHC's is more demanding quantitatively, but not much different qualitatively for the objective at hand, i.e. the same conclusions can be made regardless of whether the hill climber is RMHC or MMHC.

A good indicator of when a problem will be more easily solved by upGA than MMHC is one that correlates strongly with upGA – MMHC values, i.e. the difference in SUCC values between upGA and MMHC; or alternatively, one that categorizes problems by their upGA – MMHC values.

The answer to the questions posed in Part III may very well involve a combination of structural characteristics, e.g. a right-skewed heavy-tailed degree distribution and high modularity, but the study carried out in Part III only looks at one structural characteristic at a time. This is an acknowledged limitation of the current study that will be remedied in future work once the basic lay of the land is done in Part III.

Section 9.3 describes the methodology employed to meet the objective of Part III. Part of the methodology involves the creation of a larger sample of HIFF-like problems (section 9.1) with varied structural characteristics and amounts of epistasis (section 9.2). *SWO* values (section 2.5.1) are included in the list of candidate problem properties that influence evolutionary algorithm difficulty (in the sense of the HC < GA problem) because epistasis is intimately linked with how problem variables are inter-dependent on each other.

Main Conclusion

Of the four problem properties considered – degree distribution type, network transitivity, modularity and epistasis – degree distribution type is the best indicator of when a problem will be more easily solved by upGA than a hill climber (section 9.5.5). The hill climber may be MMHC or RMHC.

Problems with exponential and power-like degree distributions showed more discrimination between upGA and MMHC (or RMHC) than problems with single-point degree distributions (section 9.5.1). Exponential and power-like degree distributions are more typical of degree distributions found in real-world networks. Thus, in general, problems with real-world type degree distributions were more easily solved by upGA than by the random mutation hill climber (RMHC) or the macro-mutation hill climber (MMHC).

In light of this conclusion, there might be a connection after all between the structure of problems genetic algorithms are better than hill climbers at solving, and the structure of real-world networks. Section 9.6 discusses the limitations of this study and makes suggestions for future work.

8. Background: Structure of Real-World Networks

This chapter presents the necessary background on the structure of real-world networks and defines the three structural properties of interest: modularity, transitivity and degree distribution.

8.1 Real-World Networks

A network G is a set of nodes (vertices) $V(G)$ and a set of links (edges) $E(G)$ connecting elements in $V(G)$. Many real-world systems can be viewed as networks (see Table 8.1 for examples).

The nodes and/or links of a network may be associated with any number of attributes and there may be more than one type of node and/or link in a network. The links may be directed or undirected, weighted non-uniformly, able to connect more than two nodes (hyper-edge) and restricted to the types of nodes that can be connected (e.g. bipartite graph). Any number of these stated features of a network may be dynamic; e.g. an attribute of a node or link may change over time, nodes and links may be added to or removed from a network and links may change the nodes they connect.

Table 8.1 Examples of real-world networks by domain (summarized from Newman, 2003)

Domain	Network	Node	Link
Social	Movie	Actors	Acted in a movie together.
	Collaboration	Authors, scholars.	Co-authorship, i.e. written an article together.
Information	Citation	Published articles	Directed edge referencing a previously published article.
	World Wide Web	Web pages	Hyperlinks (URL) between web pages.
	Preference (bipartite network)	Individuals and objects (e.g. books, movies, hotels, etc.)	Link an individual to an object to indicate strength of like/dislike.

Technological	Internet	Domains. Computers, routers, etc.	A route that connects two domains. Wires and cables that physically connect computers, routers and other devices.
	Electric power grid	Generators, transformers and substations.	High-voltage transmission lines.
Biological	Metabolic pathways e.g. glycolysis and Krebs cycle.	Metabolites, enzymes and products.	Chemical interactions.
	Protein interactions	Proteins.	Physical protein-protein interactions (e.g. binding ability).
	Signaling networks e.g. genetic regulation.	Gene (DNA segment), mRNA, and proteins.	How the products of one gene affect the expression of another gene, whether inductive or inhibitory.
	Food web	Species in an ecosystem	Directed edge from predator to prey (or from prey to predator).
	Neural networks	Neurons.	Existence of a synapse or a gap junction between two neurons.

What is interesting about these diverse real-world networks is that they share common structural characteristics which are divergent from the structural characteristics of random networks.

A *random network* is defined as a network created by placing undirected edges independently at random with some small probability *p* between a fixed number of *n* nodes (see Albert and Barabási, 2002 for a discussion of classical random graph generation and associated phase transitions, e.g. the critical threshold for giant components to appear). Classical random graphs, first studied in the 1950s by Rapoport, and Erdös and Rényi, are random networks. Random graphs can also be generated to have non-Poisson degree distributions (see Newman, 2003 for examples of such generating models). Hence the distinction is made between classical random graphs and random graphs in general.

Three common structural characteristics real-world networks share are:

(i) Right-skewed degree-distribution, typically following a power-law or exponential form (section 8.2.1);

(ii) Higher network transitivity or clustering than random networks with similar number of nodes and links (section 8.2.2); and

(iii) Stronger community structure or higher modularity than random networks of similar number of nodes and links (section 8.2.3).

8.2 Structural characteristics

Structural characteristics describe properties of a network topology. Structural characteristics are particularly useful when networks are too large for the human eye to examine. There are a growing number of ways to describe networks by their structural characteristics (see Costa et al., 2007 for a survey). The structural characteristics of interest in Part III are: degree distribution, transitivity and modularity.

8.2.1 Degree distribution

The *degree* of a node is the number of edges or links incident on the node. The degree of a node also gives the number of nearest neighbours for the node. A network's degree distribution gives the probability $P(k)$ of a randomly selected node having degree k. In directed networks, nodes can have in- and out- degrees to distinguish incoming from outgoing links. Accordingly, directed networks can have in- and out- degree distributions. All networks under study in Part III are undirected so the discussion in this chapter is confined to undirected networks.

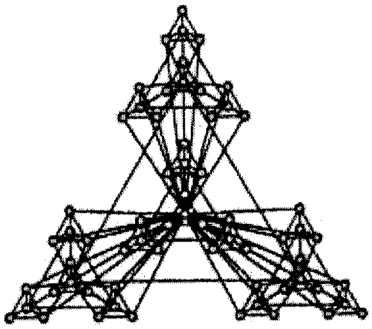
A network is *regular* if all of its nodes have the same degree. All nodes in a k-regular graph have degree k. Regular networks have single-point degree distributions.

Random networks have a bell-shaped Poisson degree distribution $P(k) \sim \frac{z^k e^{-z}}{k!}$ in the limit of n, which approximates a Gaussian distribution when p is such that the mean degree $z = p(n -1)$ is large. In words, the majority of nodes in random networks have degree close to the mean degree.

Real-world networks tend to have right-skewed degree distribution. The term '*scale-free network*' has come to be associated with networks exhibiting power-law degree distributions (see Newman, 2003; Watts, 2004 p.111; Keller, 2005; Dorogovtsev and Mendes, 2004 for why this association is misleading and problematic). Although 'scale-free networks' were 'discovered' in the 1990's, networks of this kind were already known in the 1920s and 30s by Yule, Lotka and Lévy (Keller 2005).

In a 'scale-free network' the probability of a randomly selected node having degree k follows $P(k) \sim k^{-\gamma}$, where γ is called the degree exponent. Real-world scale-free networks which have been analyzed typically have constant degree exponents not larger than three. In contrast to Gaussian, Poisson and exponential degree distributions where the number of nodes with a given degree decay exponentially fast as we move away from the mean degree, degree distributions following a power-law have a much slower rate of decay, i.e. they are fat or heavy-tailed.

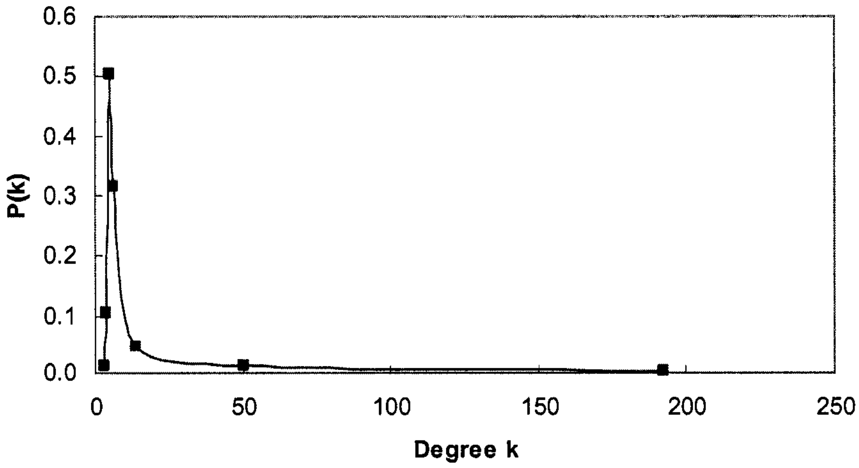
The presence of a power-law in a network's degree distribution is inferred when the degree distribution of a network plot on a log-log scale shows a significant negatively sloped straight line. This, argues Keller (2005), is a weak method for power-law detection. A more discerning method (and the method adopted here in Part III) is to plot the (complementary) cumulative degree distribution on a log-log scale for the straight line detection (Keller, 2005; Newman, 2006a). Figure 8.1 below illustrates this method for a small network of N = 256 nodes linked as hierarchical network (Ravasz et al., 2002; Ravasz and Barabási, 2003).



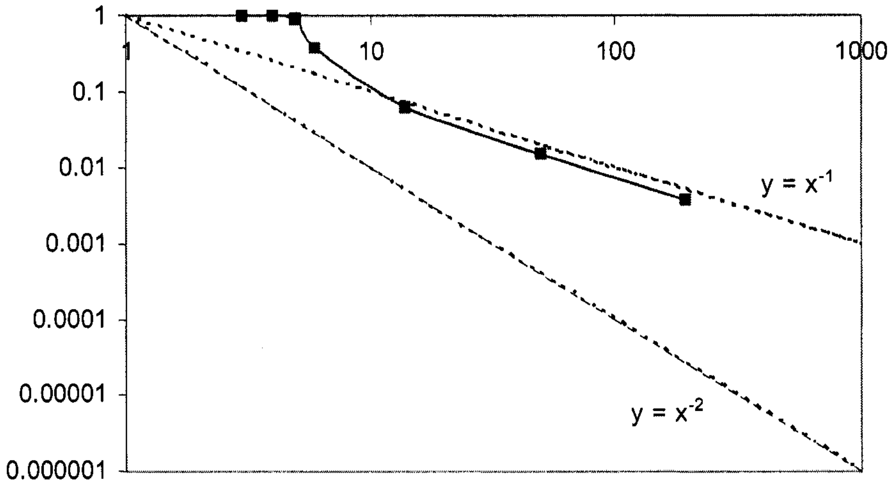
The diagram on the left is taken from Barabási (2003, p. 233). The size of this network (N) is 256.

The table below provides the node degree information for this network which is the underlying data for the following two graphs.

Degree *k*	Number of nodes with degree *k*	P(*k*) = *k* / N	Cumulative P(*k*)
192	1	0.003906	0.003906
50	3	0.011719	0.015625
14	12	0.046875	0.062500
6	81	0.316406	0.378906
5	129	0.503906	0.882813
4	27	0.105469	0.988281
3	3	0.011719	1.000000



The degree distribution is highly right-skewed and has a heavy-tail.



The cumulative degree distribution on a log-log plot. This graph has a negatively sloped straight line section which indicates that the network has a power-law degree distribution and is scale-free in this sense. The degree exponent γ is between 1 and 2.

Figure 8.1 Illustration of scale-free network detection

Both random networks and networks with power-law degree distribution are *small-worlds*. A network exhibits the *small-world effect* (Watts and Strogatz, 1998) if there is a short path connecting most pairs of nodes in the network; more precisely, if the average distance (path length) between nodes, grows logarithmically or slower with network size for fixed mean degree. Average path length is the average distance between all node pairs in a network. The distance between two nodes is the number of edges or length of the shortest (geodesic) path connecting them. The average path length of real-world networks is close to that of random networks. Some adjustment to the definition of distance between two nodes is required to accommodate disconnected networks, e.g. ignore disconnectedness or use infinite distances and a harmonic mean (Newman, 2003). A *small-world network* (Watts and Strogatz, 1998) is the result of a "superposition of a lattice and a classical random graph" (Dorogovtsev and Mendes, 2004). A network may exhibit the small-world effect and not be a small-world network itself, e.g. random networks.

Although small-worldliness by itself is insufficient to distinguish real-world networks from random networks, real-world networks exhibiting the small-world effect have been studied and classified by degree distribution type (Amaral et al., 2000). Further, the small-world effect in search problems such as graph coloring, time-tabling, quasigroup problems and satisfaction constraint problems has been of interest to computer scientists as well (Walsh, 1999). Walsh (1999): "search problems met in practice may be neither completely structured nor completely random...simple topology features can have a large impact on the cost of solving search problems."

8.2.2 Network transitivity or clustering

This property refers to the cliquishness of a neighbourhood. What is the likelihood that the (nearest) neighbours of a node are also linked directly to each other? The presence of triangles (and longer loops) in the topology of a network are signs of transitivity.

The *clustering coefficient* C quantifies the degree of network transitivity in a network. A common method to calculate C involves finding the ratio of the number of complete triangles to the number of incomplete triangles (paths of length two) in a network. A more local approach (and which is adopted here in Part III) is to use the formula $C_i = \frac{2E_i}{k_i(k_i - 1)}$ to calculate the clustering coefficient for each node in a network, and then take the average of these C_i values to give the clustering coefficient for the network (Watts and Strogatz, 1998). E_i is the number of links between node i's k neighbours. k (k-1) / 2 is the total number of possible (undirected) links between k nodes.

Thus C_i is the ratio of actual to possible links amongst a set of nodes. Figure 8.2 illustrates.

There may be discrepancies between C values calculated using different methods, but regardless of the method used to calculate network transitivity, "...the values [the clustering coefficient of many real-world networks] tend to be considerably higher than for a random graph with a similar number of vertices and edges" (Newman, 2003).

Transitivity is low in random networks and the clustering coefficient for random networks tends to 0 in the limit of large n. This is unsurprising since link placement during generation of a random network is a random and independent event while high network transitivity implies a heightened probability of a link between the nearest neighbours of a node.

	0	1	2	3	4	5	6	7
0	0	1	1	1	1	0	0	0
1	1	0	1	1	0	0	0	0
2	1	1	0	1	0	0	0	0
3	1	1	1	0	0	0	0	0
4	1	0	0	0	0	1	1	1
5	0	0	0	0	1	0	1	1
6	0	0	0	0	1	1	0	1
7	0	0	0	0	1	1	1	0

Adjacency matrix A for a network with eight nodes: $a_{ij} > 0$ if there nodes i and j are linked and $a_{ij} = 0$ otherwise.

i	k_i	E_i	C_i
0	4	3	0.5
1	3	3	1.0
2	3	3	1.0
3	3	3	1.0
4	4	3	0.5
5	3	3	1.0
6	3	3	1.0
7	3	3	1.0
		C	0.875

Node 0 has four direct neighbours, i.e. nodes 1, 2, 3 and 4, and there are three direct links amongst them, i.e. (1, 2), (2, 3) and (1, 3). Therefore the clustering coefficient for node 0 is (2×3) / (4×3) = 0.5. The clustering coefficient for the network is 0.875.

Figure 8.2 Illustration of clustering coefficient

For the calculation of network clustering coefficient in Part III, there can be at most one link between two nodes. So if node i's neighbours are connected to each other by more than one link, for example in a multi-graph, the E value for node i is incremented by one only.

8.2.3 Community structure or modularity

Community structure and modularity refer to the same concept of there being within a network, identifiable subsets of nodes where there is a higher density of links amongst nodes within a subset than between nodes of different subsets. Such subsets of nodes are called communities or modules, and a network exhibiting such characteristic is said to have community structure or a modular organization. Figure 8.3 is an example of a network with three communities or modules.

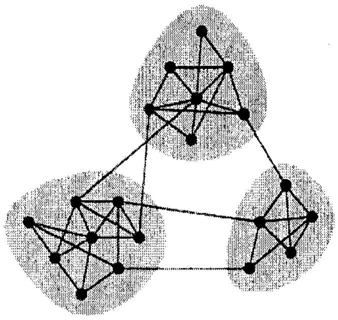


Figure 8.3 A network with three communities or modules (shaded). There is a larger number of links between nodes in the same community than between nodes in different communities. Source: Newman, (2006).

Community structure or modularity is a common characteristic of real-world networks (Girvan and Newman, 2002). Ravasz and Barabási (2003): "Indeed, many networks are fundamentally modular: one can easily identify groups of nodes that are highly interconnected with each other, but have only a few or no links to nodes outside of the group to which they belong to." This does not however necessarily imply that the inter-group or inter-module links are unimportant (Watson and Pollack, 2005). The inter-module links are very important in the case of the HIFF problems. If the *iff* constraints represented by the inter-modular links are not satisfied, a globally optimal solution to a HIFF problem cannot emerge.

There are a number of ways to detect community structure (Girvan and Newman, 2002; Dorogovtsev and Mendes, 2004). One of which is that proposed by Newman (2006). This method uses the Q metric to guide the top-down recursive partitioning or division of a network into its communities. At each step in this method, a network is divided into two if possible, and the Q metric is used to evaluate candidate node groupings. A good node grouping is one that divides a network so that links are denser within a group than between groups. A node grouping with a higher Q value is better than a node grouping with a lower Q value. However, the purpose of using the Q metric here in Part III is not to find where the modules are in a HIFF problem, as this is already known. The purpose here is to quantify modularity so that the HIFF problems can be compared on the basis of modularity. This is why Newman's method is of interest. The Q metric is defined next.

According to Newman (2006), density and sparseness of links should be measured relative to what can be expected by random chance: "if the number of edges between groups is significantly less than we expect by chance – or equivalently if the number within groups is significantly more – then it is reasonable to conclude that something interesting is going on." On the basis of this argument, Newman defines Q to be $\mathbf{s}^{\mathrm{T}}\mathbf{B}\mathbf{s}$ / $2m$ where $\mathbf{s}$ is a column vector of n ±1 elements representing a particular division of a network into two modules, $\mathbf{s}^{\mathrm{T}}$ is the transpose of $\mathbf{s}$, and $\mathbf{B}$ is a real symmetric matrix called the *modularity matrix* with elements $B_{ij} = A_{ij} - \frac{k_i k_j}{2m}$. $\mathbf{A}$ is the adjacency matrix for the network where $A_{\mathrm{ij}} = e$ means e links exists between nodes i and j, k_i and k_j are the respective degrees of nodes i and j, and $2m$ is the total number of edges in the network.

A Q value close to one means that the division defined by **s** is a good one and that the network has high modularity (in the sense that two identifiable modules exists). A network with a close to zero Q value for most combinations of **s** has low modularity (in the sense that it is unlikely that two identifiable modules exists).

Since the best **s** column vector for the HIFF problems is known, the Q metric quantifies the maximum degree of modularity in a HIFF problem. For the HIFF problems studied in Part III, Q values increase with increase in problem size (Figure 9.10), so the Q value measured at the highest level, i.e. considering the whole network, is maximal.

Example

Below are two examples of Q calculation for two HIFF problems. The elements of column vector **s** for all Q calculations in Part III are +1 in the top half and –1 in the bottom half. Thus, $\mathbf{s}^{\mathrm{T}}$ is the row vector with +1 in the left half and –1 in the right half; and $\mathbf{s}^{\mathrm{T}}\mathbf{B}\mathbf{s}$ is essentially the sum of the entries in the top-left and bottom-right quadrants in **B** minus the sum of the entries in the top-right and bottom-left quadrants in **B**. Thus to obtain high positive Q values, the sum of the entries in the top-left and bottom-right quadrants in **B** need to be larger than the sum of the entries in the top-right and bottom-left quadrants in **B**. For this to happen, there needs to be more non-zero entries in the top-left and bottom-right quadrants in **A** than in the top-right and bottom-left quadrants in **A**, which in essence means link density is higher between nodes of the same module than between nodes of different modules.

For all Q calculations in Part III, **A** is derived by multiplying the inter-dependency matrix **M** for a problem with the largest denominator found in **M**. For example, the inter-dependency matrix for HIFF-M with eight variables (Figure 3.8) is multiplied by 2 to obtain the adjacency matrix (Figure 8.4a), and the inter-dependency matrix for HIFF-II with eight variables (Figure 3.6) is multiplied by 8 (Figure 8.4b).

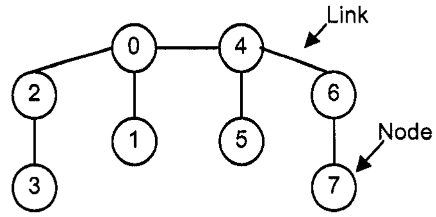
$$\begin{array}{c|cccc|cccc|}
 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
0 & 0 & 1/2 & 1/2 & 0 & 1/2 & 0 & 0 & 0 \\
1 & 1/2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
2 & 1/2 & 0 & 0 & 1/2 & 0 & 0 & 0 & 0 \\
3 & 0 & 0 & 1/2 & 0 & 0 & 0 & 0 & 0 \\
\hline
4 & 1/2 & 0 & 0 & 0 & 0 & 1/2 & 1/2 & 0 \\
5 & 0 & 0 & 0 & 0 & 1/2 & 0 & 0 & 0 \\
6 & 0 & 0 & 0 & 0 & 1/2 & 0 & 0 & 1/2 \\
7 & 0 & 0 & 0 & 0 & 0 & 0 & 1/2 & 0 \\
\hline
\end{array}
\qquad
\begin{array}{c|cccc|cccc|}
 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
2 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
3 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
\hline
4 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
5 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
6 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\
7 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
\hline
\end{array}$$

Figure 8.4a The inter-dependency matrix M for HIFF-M, N = 8 from Figure 3.8 is on the left. On the right is the adjacency matrix A for HIFF-M, N = 8.

$$\begin{array}{c|cccc|cccc|}
 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
 & 0 & 1/2 & 1/4 & 0 & 1/8 & 0 & 0 & 0 \\
1 & 1/2 & 0 & 0 & 1/4 & 0 & 1/8 & 0 & 0 \\
2 & 1/4 & 0 & 0 & 1/2 & 0 & 0 & 1/8 & 0 \\
3 & 0 & 1/4 & 1/2 & 0 & 0 & 0 & 0 & 1/8 \\
\hline
4 & 1/8 & 0 & 0 & 0 & 0 & 1/2 & 1/4 & 0 \\
5 & 0 & 1/8 & 0 & 0 & 1/2 & 0 & 0 & 1/4 \\
6 & 0 & 0 & 1/8 & 0 & 1/4 & 0 & 0 & 1/2 \\
7 & 0 & 0 & 0 & 1/8 & 0 & 1/4 & 1/2 & 0 \\
\hline
\end{array}
\qquad
\begin{array}{c|cccc|cccc|}
 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
0 & 0 & 4 & 2 & 0 & 1 & 0 & 0 & 0 \\
1 & 4 & 0 & 0 & 2 & 0 & 1 & 0 & 0 \\
2 & 2 & 0 & 0 & 4 & 0 & 0 & 1 & 0 \\
3 & 0 & 2 & 4 & 0 & 0 & 0 & 0 & 1 \\
\hline
4 & 1 & 0 & 0 & 0 & 0 & 4 & 2 & 0 \\
5 & 0 & 1 & 0 & 0 & 4 & 0 & 0 & 2 \\
6 & 0 & 0 & 1 & 0 & 2 & 0 & 0 & 4 \\
7 & 0 & 0 & 0 & 1 & 0 & 2 & 4 & 0 \\
\hline
\end{array}$$

Figure 8.4b The inter-dependency matrix M for HIFF-II, N = 8 from Figure 3.6 is on the left. On the right is the adjacency matrix A for HIFF-II, N = 8.

Example 1: Q value calculation for the HIFF-M problem with eight variables (N=8). The HIFF-M problem with eight variables is represented by a network with eight nodes (one per variable) and seven undirected equally weighted links (one per inter-dependency) as shown in Figure 8.5 below. The adjacency matrix **A** representing this network (which was derived in Figure 8.4a), and the modularity matrix **B** multiplied by the total number of links is given in Figure 8.5. The *Q* value for this example is 0.7143.



The HIFF-M problem with eight variables represented as a network.

	0	1	2	3	4	5	6	7	*k*
0	0	1	1	0	1	0	0	0	3
1	1	0	0	0	0	0	0	0	1
2	1	0	0	1	0	0	0	0	2
3	0	0	1	0	0	0	0	0	1
4	1	0	0	0	0	1	1	0	3
5	0	0	0	0	1	0	0	0	1
6	0	0	0	0	1	0	0	1	2
7	0	0	0	0	0	0	1	0	1

Adjacency matrix **A** for HIFF-M, N=8.

If nodes *i* and *j* are linked $A_{ij} > 0$, otherwise $A_{ij} = 0$. This adjacency matrix is *un-weighted* because all non-zero entries have the same value.

k denotes the degree for a node.

The total number of links (*2m*) for this network is 14 (each link is counted twice when calculating *Q*).

	0	1	2	3	4	5	6	7
0	-9	11	8	-3	5	-3	-6	-3
1	11	-1	-2	-1	-3	-1	-2	-1
2	8	-2	-4	12	-6	-2	-4	-2
3	-3	-1	12	-1	-3	-1	-2	-1
4	5	-3	-6	-3	-9	11	8	-3
5	-3	-1	-2	-1	11	-1	-2	-1
6	-6	-2	-4	-2	8	-2	-4	12
7	-3	-1	-2	-1	-3	-1	12	-1

Modularity matrix **B** multiplied by 14.

$$B_{ij} = A_{ij} - \frac{k_i k_j}{2m}$$

E.g. $B_{00} = 0 - (3 \times 3)/14$, so $B_{00} \times 14 = -9$.

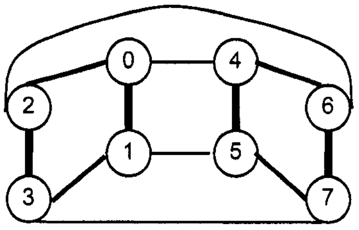
$\mathbf{S}^T\mathbf{B} = [14\ 14\ 28\ 14\ -14\ -14\ -28\ -14] \times 1/14$

$\mathbf{s}^T\mathbf{Bs} = 140/14 = 10$

$Q = \mathbf{s}^T\mathbf{Bs}\ /\ 2m = 10/14 = 0.71429$

Figure 8.5 Illustration of *Q* value calculation for an un-weighted network

Example 2: *Q* value calculation for the HIFF-II problem with eight variables (N=8). The HIFF-II problem with eight variables is represented by a network with eight nodes (one per variable) and 12 undirected *un-equally* weighted links (one per inter-dependency) as shown in Figure 8.6 below. The adjacency matrix **A** representing this network (which was derived in Figure 8.4b), and the modularity matrix **B** multiplied by the total number of links is given on in Figure 8.6. The Q value for this example is 0.7143.



The HIFF-II problem with eight variables represented as a network.
The weight of a line corresponds to the weight of the link represented by the line. Heavier lines denote links with heavier weights or stronger node-to-node inter-dependencies.

	0	1	2	3	4	5	6	7	*k*
0	0	4	2	0	1	0	0	0	7
1	4	0	0	2	0	1	0	0	7
2	2	0	0	4	0	0	1	0	7
3	0	2	4	0	0	0	0	1	7
4	1	0	0	0	0	4	2	0	7
5	0	1	0	0	4	0	0	2	7
6	0	0	1	0	2	0	0	4	7
7	0	0	0	1	0	2	4	0	7

Adjacency matrix **A** for HIFF-II, N=8.

If nodes i and j are linked $A_{ij} > 0$, otherwise $A_{ij} = 0$. This adjacency matrix is *weighted* because all non-zero entries do not have the same value.

k denotes the degree for a node.

The total number of links ($2m$) for this network is 56 (each link is counted twice when calculating Q).

	0	1	2	3	4	5	6	7
0	-49	175	63	-49	7	-49	-49	-49
1	175	-49	-49	63	-49	7	-49	-49
2	63	-49	-49	175	-49	-49	7	-49
3	-49	63	175	-49	-49	-49	-49	7
4	7	-49	-49	-49	-49	175	63	-49
5	-49	7	-49	-49	175	-49	-49	63
6	-49	-49	7	-49	63	-49	-49	175
7	-49	-49	-49	7	-49	63	175	-49

Modularity matrix **B** multiplied by 56.

$$B_{ij} = A_{ij} - \frac{k_i k_j}{2m}$$

E.g. $B_{23} = 4 - (7 \times 7)/56$, so $B_{23} \times 56 = 175$.

$\mathbf{S}^T\mathbf{B}$ = [280 280 280 280 -280 -280 -280 -280] × 1/56

$\mathbf{s}^T\mathbf{B}\mathbf{s} = 2240/56 = 40$

$Q = \mathbf{s}^T\mathbf{B}\mathbf{s} \,/\, 2m = 40/56 = 0.71429$

Figure 8.6 Illustration of Q value calculation for a weighted network

8.2.4 Hierarchical architecture

Hierarchical architecture is also included in the list of common structural characteristics of real-world systems in section 8.1 above by some researchers. Hierarchical architecture is claimed to resolve a fundamental conflict between a power-law degree distribution (scale-free topology) and a modular organization in a network. Ravasz and Barabási (2003): "In order to bring modularity, the high degree of clustering, and the scale-free topology under a single roof, we need to assume that modules combine into each other in a hierarchical manner, generating what we call a *hierarchical network*". The network in Figure 8.1 is an example of a hierarchical network.

A network with a power-law degree distribution means there are many more nodes with small degrees (poorly connected to other nodes) and only a few nodes with large degrees (richly connected to other nodes). But the few richly connected nodes or hubs link a large proportion of the nodes in a network thus precluding a network topology with fully isolated or clearly delineated non-trivial sub-networks or modules (Ravasz et al., 2002). High transitivity (degree of clustering) in real-world networks suggests the existence of a modular organization, but a modular organization imposes severe restrictions on the degree distribution of a network, i.e. most nodes will have the same degree (Ravasz et al., 2002). Thus, according to Ravasz et al. (2002) and Ravasz and Barabási (2003), there is a fundamental conflict between a power-law degree distribution (scale-free topology) and a modular organization.

Regardless of whether one agrees with the above explanation (in particular about high transitivity suggesting modular organization), many real-world systems have a hierarchical architecture. In the chapter entitled "The Architecture of Complexity", Herbert A. Simon (1969) argues that "Hierarchy ... is one of the central structural schemes that the architect of complexity uses." This is because "complex systems will evolve from simple systems much more rapidly if there are stable intermediate forms than if there are not. The resulting complex forms in the former case will be hierarchic."

Simon defines a hierarchic system or hierarchy as "a system that is composed of interrelated subsystems, each off the latter being, in turn, hierarchic in structure until we reach some lowest level of elementary subsystem". Simon also makes a special point regarding the importance of the relations between subsystems: "The mathematical term "partitioning" will not do for what I call here a hierarchy; ... By "hierarchy" I mean the partitioning in conjunction with the relations that hold among its parts."

To exemplify his definition of hierarchy and the prevalence of hierarchical systems, Simon appeals to a broad-range of real-world systems: (i) the formal organization of social systems where subordination is implied in the hierarchic structure, e.g. business-firms, governments, universities, (ii) the informal organization of social systems, e.g. families are grouped within villages, villages within counties and so on; (iii) biological and physical systems, e.g. atoms are composed into molecules and molecules into macro-molecules, cells are organized into tissues, tissues into organs and organs into systems; and finally (iv) symbolic systems, e.g. words are organized into sentences, sentences into paragraphs, paragraphs into sections, sections into chapters and chapters into books. The availability of empirical data and computer power to processes large amounts of data have revealed more hierarchical structures in the real-world. For example, the metabolic system of *Escherichia coli* (Ravasz et al., 2002) and the Internet at the Autonomous System level (Trusina et al., 2003) can be added to the list of hierarchic architecture in the real-world.

For a more mechanistic way of identifying hierarchical architecture in a network, the concept of a *hierarchical path* (Gao, 2001) has been used (Trusina et al., 2003). Dorogovtsev and Mendes (2004): "A path between nodes *a* and *b* is hierarchical if (1) the degrees of vertices along this path vary monotonously from one vertex to the other, or (2) vertex degrees first monotonously grow, reach maximum value, and then monotonously decrease along the path. Let the fraction H of the shortest paths in a network be hierarchical. Then this number H can be used as a metric of a hierarchical topology [27]. If H of a network is sufficiently close to 1, the network has a pronounced hierarchical organization."

By the above definition, HIFF-C and HIFF-II are hierarchical since all nodes have the same degree and therefore the degrees of vertices along any path in either a HIFF-C or HIFF-II network vary monotonously from one vertex to the other. A more interesting example to illustrate hierarchical path is HIFF-M. In Figure 8.7, the shortest paths between all pairs of nodes in a HIFF-M network of size eight is enumerated along with the corresponding degrees of vertices along each path. (The HIFF-M network of size eight is illustrated in Figure 8.5.). All paths in this example are hierarchical paths. Therefore this network has a pronounced hierarchical organization.

Nested within every HIFF-M network of size 2^i are two HIFF-M networks of size 2^{i-1} each. A link between nodes 0 of its two nested networks connects the two halves of a HIFF-M network. Therefore, using induction on i, it can be shown that all HIFF-M networks of size 2^i have hierarchical organization.

	1	2	3	4	5	6	7
0	0, 1 3, 1	0, 2 3, 2	0, 2, 3 3, 2, 1	0, 4 3, 3	0, 4, 5 3, 3, 1	0, 4, 6 3, 3, 2	0, 4, 6, 7 3, 3, 2, 1
1		1, 0, 2 1, 3, 2	1, 0, 2, 3 1, 3, 2, 1	1, 0, 4 1, 3, 3	1, 0, 4, 5 1, 3, 3, 1	1, 0, 4, 6 1, 3, 3, 2	1, 0, 4, 6, 7 1, 3, 3, 2, 1
2			2, 3 2, 1	2, 0, 4 2, 3, 3	2, 0, 4, 5 2, 3, 3, 1	2, 0, 4, 6 2, 3, 3, 2	2, 0, 4, 6, 7 2, 3, 3, 2, 1
3				3, 2, 0, 4 1, 2, 3, 3	3, 2, 0, 4, 5 1, 2, 3, 3, 1	3, 2, 0, 4, 6 1, 2, 3, 3, 2	3, 2, 0, 4, 6, 7 1, 2, 3, 3, 2, 1
4					4, 5 3, 1	4, 6 3, 2	4, 6, 7 3, 2, 1
5						5, 4, 6 1, 3, 2	5, 4, 6, 7 1, 3, 2, 1
6							6, 7 2, 1

Figure 8.7 Illustration of all shortest and hierarchical paths in a HIFF-M network of size eight.

In Figure 8.7, the shortest paths between all pairs of nodes in a HIFF-M network of size eight is given as a sequence of numbers at the top of each cell, and the corresponding degrees of nodes along each path is given as a sequence of numbers at the bottom of each cell. For example, the shortest path between nodes 2 and 6 is the one that traverses the path <2, 0, 4, 6>. The degrees of nodes along this path are 2, 3, 3, 2 which makes it a hierarchical path by the definition given above. Observe that all shortest paths in the upper-right quadrant traverse the (0, 4) edge which connects the two halves of this network, and for shortest paths longer than three (found in the upper-right quadrant), degree of nodes increase in the first (second) half of a path, and then decrease in the second (first) half of a path.

In general, the nesting of smaller modules within larger modules in a hierarchical manner is intrinsic to the HIFF problems studied in Part III (the problems are defined in section 9.1.). Thus, attention in Part III is paid to the other three structural characteristics: degree distribution, modularity and transitivity.

8.3 Chapter Summary

This chapter has presented the necessary background on the structure of real-world networks and defined the three structural properties of interest: modularity, transitivity and degree distribution.

The common structural characteristics real-world networks share in the main are:

(i) Right-skewed degree-distribution, typically following a power-law or exponential form, (section 8.2.1);

(ii) Higher network transitivity or clustering than random networks with similar number of nodes and links, (section 8.2.2); and

(iii) Stronger community structure or higher modularity than random networks of similar number of nodes and links (section 8.2.3).

Another common structural property of real-world networks, hierarchical organization, was also discussed (section 8.2.4).

This chapter has also illustrated the detection of power-law degree distribution for a network (section 8.2.1), calculation of network clustering coefficient C values (section 8.2.2) and calculation of Q value to quantify degree of modularity (section 8.2.3).

9. Problem Structure and the HC < GA Problem

This chapter forms the heart of Part III. Recall that the specific objective of Part III is to find a structural characteristic which is a better indicator than modularity of when a problem will be more easily solved by upGA (Figure II.A) than a hill climber (HC). The hill climber competing with upGA in Part III is MMHC (Figure 5.3). The structural characteristics of interest are modularity, degree distribution type and network transitivity (chapter 8). Epistasis in terms of *SWO* values (section 2.5.1) is also considered.

To meet this objective, a set of HIFF-like problems with varied structural characteristics and amounts of epistasis is created. These test problems (described in section 9.1) build on the HIFF problems defined in chapter 3. Section 9.2 records the structural characteristics of interest and amounts of epistasis in these test problems. Section 9.3 lays out the methodology for the experiments, and section 9.4 reports the results of the experiments. Section 9.5 uses the results in section 9.4 to select the best indicator of when a problem will be more easily solved by upGA than MMHC. Section 9.6 discusses some limitations of the study in this chapter and section 9.7 concludes with a summary.

9.1 Test Problems

In Part I (chapter 3), different HIFF problems were generated by varying the set of *iff* constraints and their fitness contributions. In Part III (this section), in addition to the set of *iff* constraints and their fitness contributions, the size of blocks is also varied to create different HIFF problems. However, the basic HIFF problem remains as it was in Part I, that is to transform a $\{0, 1\}^N$ string to a string of either all-zeroes $\{0\}^N$ or all-ones $\{1\}^N$.

All HIFF fitness functions in this chapter do not use the all-or-nothing approach. Fitness of a string is the accumulation of fitness contributions of satisfied *iff* constraints between variable pairs: $F(S) = \sum_i w_i c_i(S)$ where $c_i(S) = 1$ if S satisfies c_i *iff* constraint and 0 otherwise, and w_i is the fitness contribution or weight of c_i. Unless otherwise stated, the *iff* constraints for problems in this chapter are all equally weighted, i.e. $\forall i\ w_i = 1.0$.

Problem Decomposition

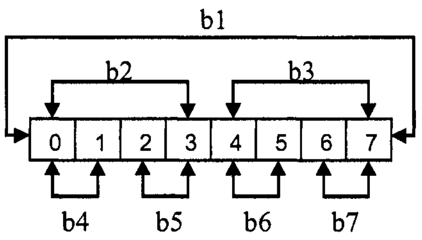
Part I introduced the concept of a block, and the size of all blocks at level λ is 2^λ. In Part III (this section), blocks at level 1 are called *basic blocks*, and blocks at levels above level 1 are called *non-basic blocks*. As in Part I, a (basic or non-basic) block is optimal when the values of its variables satisfy all the block's inter-dependencies. But, the optimal or maximum block fitness in this chapter need not be 1.0.

The notation X-*p*-*q* is used to describe a HIFF problem in this chapter. As an example, the HIFF-M problem using this notation is M-2-2.

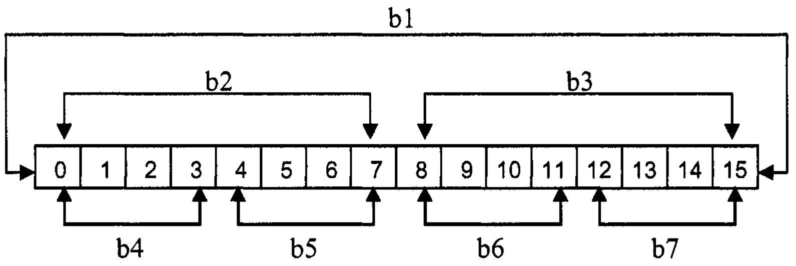
(i) X represents the connectivity or variable-to-variable inter-dependency pattern for non-basic blocks. For example, the variable-to-variable inter-dependency pattern of M is: for all non-basic blocks b there is a *iff* constraint between variable 0 and variable $|b|/2$;

(ii) q represents the size of basic blocks. The size of all basic blocks for a problem is the same, and in this chapter, either $q = 2$ or $q = 4$. The variables of a basic block are fully connected, that is there is a *iff* constraint between every distinct pair of variables in a basic block; and

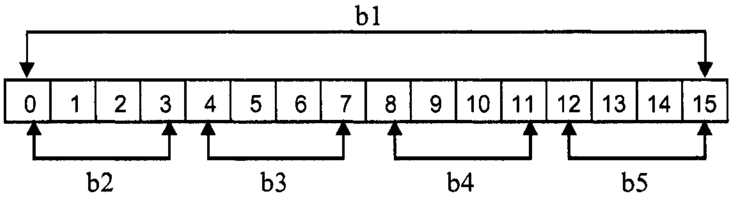
(iii) p represents the number of blocks directly nested within a non-basic block. The number of blocks directly nested within a non-basic block is the same for all non-basic blocks of a problem. In this chapter, either $p = 2$ or $p = 4$, and $p \leq q$. So only three combinations are permissible: X-2-2, X-2-4 and X-4-4. Figure 9.1 below illustrates.



The diagram on the left shows the blocks for a X-2-2 HIFF problem, N = 8. The non-basic blocks b1, b2 and b3 each have 2 blocks directly nested within them. The size of the basic blocks b4, b5, b6 and b7 are all 2.



The diagram above shows the blocks for a X-2-4 HIFF problem, N = 16. The non-basic blocks b1, b2 and b3 each have 2 blocks directly nested within them. The size of the basic blocks b4, b5, b6 and b7 are all 4.



The diagram above shows the blocks for a X-4-4 HIFF problem, N = 16. The non-basic block b1 has 4 blocks directly nested within it. The size of the basic blocks b2, b3, b4 and b5 are all 4.

Figure 9.1 Example of X-2-2, X-2-4 and X-4-4 HIFF problems

For a HIFF problem X-*p*-*q*:

(i) The size of non-basic blocks at level λ is $p^{\lambda+\delta}$ where $\delta = \log_p (q / p)$; and

(ii) The total number of levels is $\log_p (N / q) + 1$. N is assumed to be p^i where $i \in \mathbf{Z}^+$.

Table 9.1 exemplifies.

Table 9.1 Blocks size calculation for X-*p*-*q*, N=16

X-*p*-*q*	Number of levels	δ	Block size			
			λ = 4	λ = 3	λ = 2	λ = 1
X-2-2	4	0	16	8	4	2
X-2-4	3	1	-	16	8	4
X-4-4	2	0	-	-	16	4

Inter-dependency patterns (X)

An inter-dependency pattern describes the way variables of a non-basic block interact with each other, i.e. which variable pair is linked with an *iff* constraint. The variables of a basic block for all basic blocks are fully connected, that is there is an *iff* constraint between every distinct pair of variables in a basic block. Unless otherwise stated, the fitness contribution of every *iff* constraint in this chapter is 1.0. The inter-dependency patterns used in this chapter are: C, II, M, M1, M2 and R.

The C, II and M inter-dependency patterns are respectively, that of the HIFF-C, HIFF-II and HIFF-M problems in chapter 3.

The C inter-dependency pattern places an *iff* constraint between or links every distinct pair of variables within each block. The C-2-2 problem is the same as the HIFF-C problem defined in section 3.2.1. The inter-dependency matrix **M** for C-2-2 given in Figure 9.2a is identical to that for HIFF-C in Figure 3.1.

The Ce-2-2 problem is the same as the C-2-2 problem except that all interactions in Ce-2-2 contribute 1.0 fitness points when satisfied. The inter-dependency matrix **M** for Ce-2-2 is given in Figure 9.2b.

	0	1	2	3	4	5	6	7
0	0	1/2	1/8	1/8	1/32	1/32	1/32	1/32
1	1/2	0	1/8	1/8	1/32	1/32	1/32	1/32
2	1/8	1/8	0	1/2	1/32	1/32	1/32	1/32
3	1/8	1/8	1/2	0	1/32	1/32	1/32	1/32
4	1/32	1/32	1/32	1/32	0	1/2	1/8	1/8
5	1/32	1/32	1/32	1/32	1/2	0	1/8	1/8
6	1/32	1/32	1/32	1/32	1/8	1/8	0	1/2
7	1/32	1/32	1/32	1/32	1/8	1/8	1/2	0

Figure 9.2a Inter-dependency matrix M for C-2-2, N=8

	0	1	2	3	4	5	6	7
0	0	1/2	1/2	1/2	1/2	1/2	1/2	1/2
1	1/2	0	1/2	1/2	1/2	1/2	1/2	1/2
2	1/2	1/2	0	1/2	1/2	1/2	1/2	1/2
3	1/2	1/2	1/2	0	1/2	1/2	1/2	1/2
4	1/2	1/2	1/2	1/2	0	1/2	1/2	1/2
5	1/2	1/2	1/2	1/2	1/2	0	1/2	1/2
6	1/2	1/2	1/2	1/2	1/2	1/2	0	1/2
7	1/2	1/2	1/2	1/2	1/2	1/2	1/2	0

Figure 9.2b Inter-dependency matrix M for Ce-2-2, N=8

The II inter-dependency pattern links variable pairs (i, $|b|/2 + i$) of each block b where $0 \leq i < |b|/2$. The II-2-2 problem is the same as the HIFF-II problem defined in section 3.2.3. The inter-dependency matrix **M** for II-2-2 given in Figure 9.3a is identical to that for HIFF-II in Figure 3.3.

The IIe-2-2 problem is the same as the II-2-2 problem except that all interactions in IIe-2-2 contribute 1.0 fitness points when satisfied. The inter-dependency matrix **M** for IIe-2-2 is given in Figure 9.3b.

	0	1	2	3	4	5	6	7
0	0	1/2	1/4	0	1/8	0	0	0
1	1/2	0	0	1/4	0	1/8	0	0
2	1/4	0	0	1/2	0	0	1/8	0
3	0	1/4	1/2	0	0	0	0	1/8
4	1/8	0	0	0	0	1/2	1/4	0
5	0	1/8	0	0	1/2	0	0	1/4
6	0	0	1/8	0	1/4	0	0	1/2
7	0	0	0	1/8	0	1/4	1/2	0

Figure 9.3a Inter-dependency matrix for II-2-2, N=8

	0	1	2	3	4	5	6	7
0	0	1/2	1/2	0	1/2	0	0	0
1	1/2	0	0	1/2	0	1/2	0	0
2	1/2	0	0	1/2	0	0	1/2	0
3	0	1/2	1/2	0	0	0	0	1/2
4	1/2	0	0	0	0	1/2	1/2	0
5	0	1/2	0	0	1/2	0	0	1/2
6	0	0	1/2	0	1/2	0	0	1/2
7	0	0	0	1/2	0	1/2	1/2	0

Figure 9.3b Inter-dependency matrix for IIe-2-2, N=8

The M inter-dependency pattern links variables 0 and |*b*|/2 of each block *b*. The M-2-2 problem is the same as the HIFF-M problem defined in section 3.2.4. Figure 9.4 illustrates the M-2-4 problem, N=16.

The M1 inter-dependency pattern is similar to the M inter-dependency pattern except the interactions above level 1 are shifted to the right by ε = (level – 2) variables. That is, the M1 inter-dependency pattern links variable pair (0 + ε, |*b*|/2 + ε) of each block *b*. There is no a priori reason for choosing level 3 as the level to begin the displacement except that M1 produces a different degree distribution than M (section 9.2.1). For the same *p* and *q* values, there is no distinction between M1-*p*-*q* and M-*p*-*q* problems with fewer than 3 levels. Figure 9.5 illustrates for M1-2-2, N=16.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0		1/2	1/2	1/2	1/2				1/2							
1	1/2		1/2	1/2												
2	1/2	1/2		1/2												
3	1/2	1/2	1/2													
4	1/2					1/2	1/2	1/2								
5					1/2		1/2	1/2								
6					1/2	1/2		1/2								
7					1/2	1/2	1/2									
8	1/2									1/2	1/2	1/2	1/2			
9									1/2		1/2	1/2				
10									1/2	1/2		1/2				
11									1/2	1/2	1/2					
12									1/2					1/2	1/2	1/2
13													1/2		1/2	1/2
14													1/2	1/2		1/2
15													1/2	1/2	1/2	

Figure 9.4 Inter-dependency matrix for M-2-4, N=16. This problem has 3 levels. Basic blocks are shaded. Only non-zero entries are shown.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0		1/2	1/2													
1	1/2					1/2										
2	1/2			1/2							1/2					
3			1/2													
4						1/2	1/2									
5		1/2			1/2											
6					1/2			1/2								
7							1/2									
8										1/2	1/2					
9									1/2					1/2		
10			1/2						1/2			1/2				
11											1/2					
12														1/2	1/2	
13										1/2			1/2			
14													1/2			1/2
15															1/2	

Figure 9.5 Inter-dependency matrix for M1-2-2, N=16. This problem has four levels. Basic blocks are shaded. Only non-zero entries are shown.

The M2 inter-dependency pattern links every distinct pair of variables from the set *v* of each block *b*. The set *v* for a block *b* is $\{x \mid x = i \times |b|/p \wedge i \in \{0, 1, \ldots, p\text{-}1\}\}$. For example, if $p = 4$ and the block *b* has 16 variables, then *v* for a block $b = \{0, 4, 8, 12\}$. M2-*p*-*q* problems are the same as M-*p*-*q* problems when *p* is 2. Figure 9.6 illustrates for M2-4-4, N=16.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0		1/2	1/2	1/2	1/2				1/2				1/2			
1	1/2		1/2	1/2												
2	1/2	1/2		1/2												
3	1/2	1/2	1/2													
4	1/2					1/2	1/2	1/2	1/2				1/2			
5					1/2		1/2	1/2								
6					1/2	1/2		1/2								
7					1/2	1/2	1/2									
8	1/2				1/2					1/2	1/2	1/2	1/2			
9									1/2		1/2	1/2				
10									1/2	1/2		1/2				
11									1/2	1/2	1/2					
12	1/2				1/2				1/2					1/2	1/2	1/2
13													1/2		1/2	1/2
14													1/2	1/2		1/2
15													1/2	1/2	1/2	

Figure 9.6 Inter-dependency matrix for M2-4-4, N=16. This problem has 2 levels. Basic blocks are shaded. Only non-zero entries are shown.

The R inter-dependency pattern is based on the hierarchical network model in Figure 8.1. Thus *p* must be equal to *q*. To be compatible with the HIFF problems in terms of N, R-4-4 (Barabási, 2003 p. 233) will be used. Figure 9.7 illustrates for R-4-4, N=16.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0		1/2	1/2	1/2		1/2	1/2	1/2		1/2	1/2	1/2		1/2	1/2	1/2
1	1/2		1/2	1/2												
2	1/2	1/2		1/2												
3	1/2	1/2	1/2													
4						1/2	1/2	1/2	1/2				1/2			
5	1/2				1/2		1/2	1/2								
6	1/2				1/2	1/2		1/2								
7	1/2				1/2	1/2	1/2									
8					1/2					1/2	1/2	1/2	1/2			
9	1/2								1/2		1/2	1/2				
10	1/2								1/2	1/2		1/2				
11	1/2								1/2	1/2	1/2					
12					1/2				1/2					1/2	1/2	1/2
13	1/2												1/2		1/2	1/2
14	1/2												1/2	1/2		1/2
15	1/2												1/2	1/2	1/2	

Figure 9.7 Inter-dependency matrix for R-4-4, N=16. This problem has 2 levels. Basic blocks are shaded. Only non-zero entries are shown.

Table 9.2 lists the test problems relevant to this chapter along with their optimal fitness values, for problem size N = 256.

Table 9.2 Optimal fitness values for test problems, N=256

HIFF Problem	Optimal Phenotype (highest to lowest level)	Optimal Fitness
M-2-2, M1-2-2	1, 2, 4, 8, 16, 32, 64, 128	255
M-2-4, M1-2-4	1, 2, 4, 8, 16, 32, 384	447
M2-4-4	6, 24, 96, 384	510
R-4-4	147, 156, 192, 384	879
Ce-2-2	16384, 8192, 4096, 2048, 1024, 512, 256, 128	32640
IIe-2-2	128, 128, 128, 128, 128, 128, 128, 128	1024
C-2-2, II-2-2	1, 2, 4, 8, 16, 32, 64, 128	255

9.2 Structural characteristics and epistasis

In this section, HIFF problems are viewed as networks where problem variables correspond to the nodes of a network and the inter-dependencies between variables are represented by the links in a network.

9.2.1 Degree distribution

Degree distribution was explained in section 8.2.1. C-2-2, Ce-2-2, II-2-2 and IIe-2-2 have single point degree distributions. The degree for all nodes in a C-2-2, Ce-2-2, II-2-2 or IIe-2-2 network is uniform and is the mean degree. Figure 9.8a plots the degree distribution for the M* problems, i.e. M-2-2, M-2-4, M1-2-2, M1-2-4 and M2-4-4, when N = 256. The degree distribution for R-4-4 is displayed in Figure 8.1.

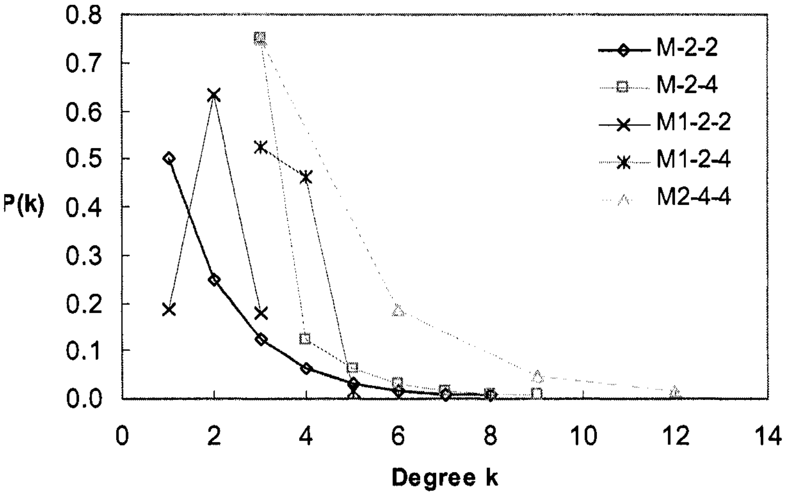


Figure 9.8a Degree distributions for M* problems, N=256

The problems studied in this chapter can be divided by their degree distribution into three categories:

(i) Problems with single-point degree distribution. Problems in this category are C-2-2, Ce-2-2, II-2-2 and IIe-2-2;

(ii) Problems with degree distribution types typical of real-world networks, i.e. highly right-skewed – few nodes have high degrees and markedly more nodes have low degrees. Problems in this category are M-2-2, M-2-4, M2-4-4 and R-4-4; and

(iii) Problems with other types of degree distributions not covered by the first two categories. Problems in this category include M1-2-2 and M1-2-4.

Thus, the problem set is varied with respect to degree distribution.

Problems in category (ii) can be further classified by the function type that best fits the degree distribution curve. The degree distribution curves of M-2-2 and M2-4-4 are best fitted by exponential functions, while power functions fit best the degree distribution curves of M-2-4 and R-4-4.

Figure 9.8b plots the degree distributions on a log-linear scale. Figure 9.8c plots the cumulative degree distributions on a log-log scale. Trendlines are shown (using Microsoft Excel) with their respective coefficient of determination (R^2 values).

The R^2 values for M-2-2 and M2-4-4 are closer to 1.0 (showing better fit) when the trendlines are exponential functions (Figure 9.8b). The R^2 values for M-2-4 and R-4-4 are closer to 1.0 (showing better fit) when the trendlines are power functions (Figure 9.8c). Table 9.3 compares the coefficient of determination (R^2) values for problems in category (ii).

Table 9.3 Coefficient of determination values

X-*p*-*q* problem	Exponential R^2	Power R^2	Polynomial R^2
M2-4-4	**0.9971**	0.9608	-
M-2-2	**0.9837**	0.9189	-
M-2-4	0.9282	**0.9930**	-
M1-2-4	0.7771	0.8273	1.0000
R-4-4	0.4480	**0.9685**	-
M1-2-2	0.0009	0.7155	1.0000

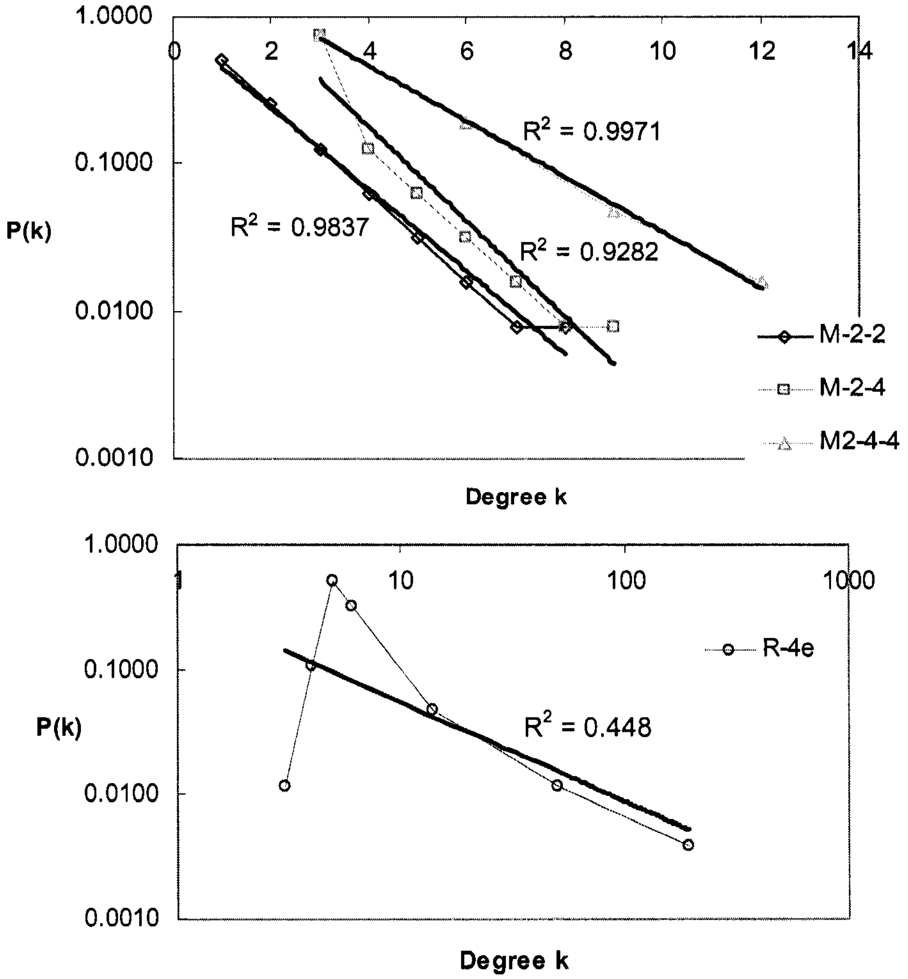


Figure 9.8b Degree distributions plotted on a log-linear scale.

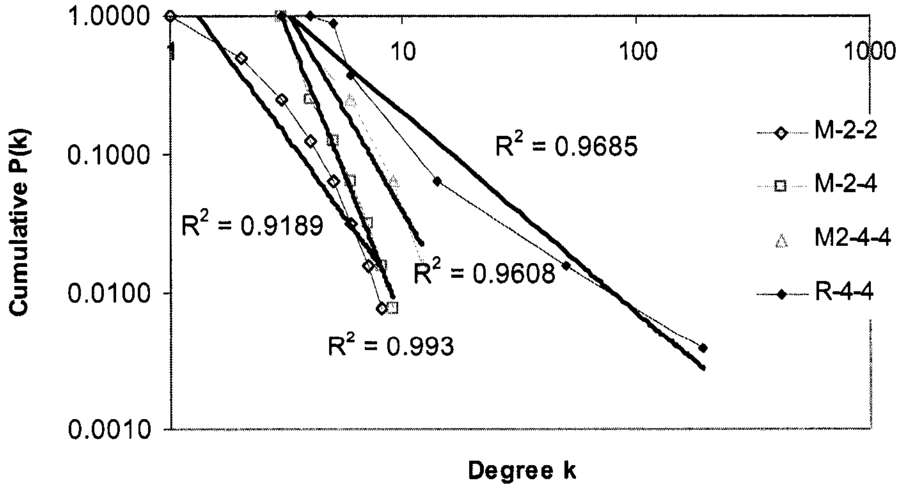


Figure 9.8c Degree distributions plotted on a log-log scale

9.2.2 Network transitivity or clustering

Network transitivity or clustering was explained in section 8.2.2. Table 9.4 lists the network clustering coefficient C for each of the problems studied in this chapter. The network clustering coefficients are calculated by the method of averaging C_i values (section 8.2.2). The C values in Table 9.4 range from minimum 0.0 to maximum 1.0. Therefore, the problem set is varied with respect to network transitivity.

Table 9.4 Network clustering coefficients *C*, N=256

X-*p*-*q* problem	*C*
II-2-2	0.0000
IIe-2-2	0.0000
M-2-2	0.0000
M1-2-2	0.0000
R-4-4	0.5931
M1-2-4	0.7586
M2-4-4	0.8396
M-2-4	0.8412
C-2-2	1.0000
Ce-2-2	1.0000

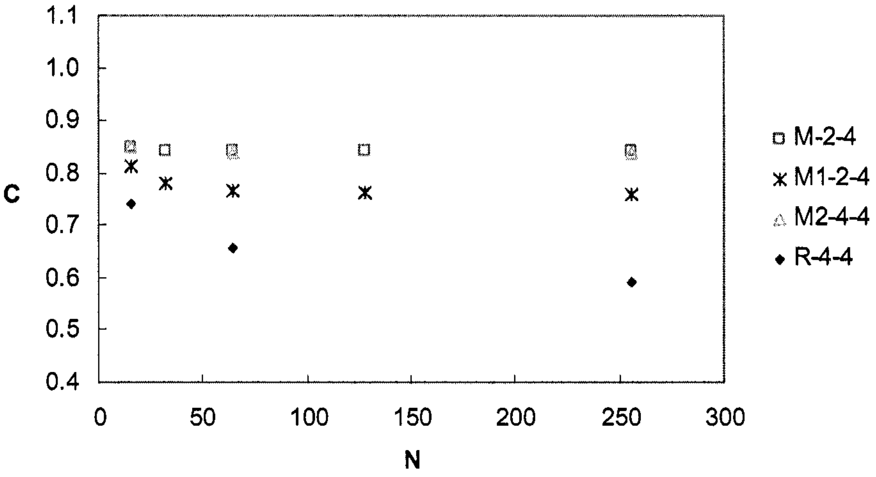


Figure 9.9 Network clustering coefficients for different problem sizes

Figure 9.9 shows how network clustering coefficient *C* values change with problem size, N. Amongst networks with transitivity, R-4-4 problems have the lowest *C* values, and these values decrease faster with increase in problem size compared to the other test problems.

9.2.3 Modularity

Modularity was explained in section 8.2.3. Table 9.5 lists the *Q* values for each of the problems studied in this chapter. The *Q* values are calculated by the method described in section 8.2.3. The *Q* values in Table 9.5 range from minimum 0.0 and come close to the maximum 1.0. Therefore, the problem set is varied with respect to modularity.

Table 9.5 Modularity *Q* values

X-*p*-*q* problem	*Q*	
	N = 128	N = 256
Ce-2-2	-	0.0000
IIe-2-2	-	0.7500
R-4-4	-	0.7742
M2-4-4	-	0.9843
C-2-2 (HIFF-C)	0.9843	0.9922
II-2-2 (HIFF-II)	0.9843	0.9922
M-2-2 (HIFF-M)	0.9843	0.9922
M1-2-2	0.9843	0.9922
M-2-4	-	0.9955
M1-2-4	-	0.9955

Figure 9.10 shows how *Q* values for R-4-4 and M* problems change with problem size. When N=256, *Q* values for M* problems converge but R-4-4 remains consistently below the *Q* values for M* problems. R-4-4 is less modular than the M* problems.

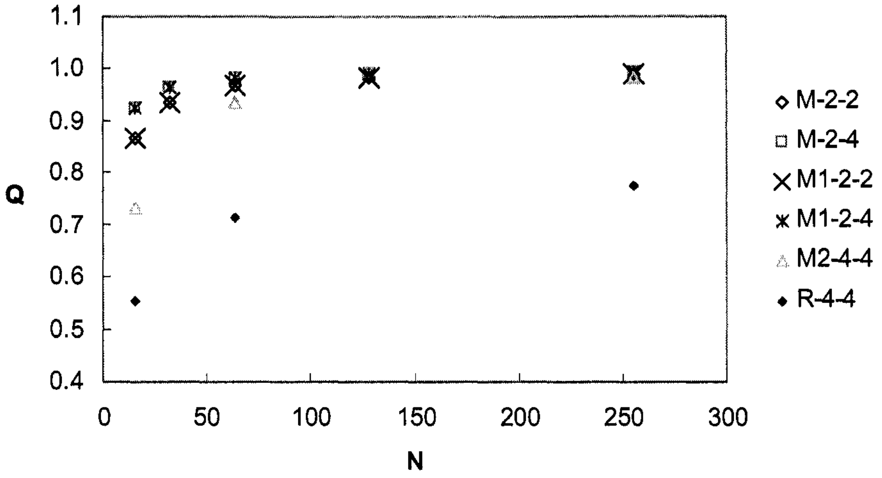


Figure 9.10 Modularity *Q* values for different problem sizes

9.2.4 Epistasis

Epistasis is calculated by the SWO method (section 2.5.1). Table 9.6 lists the *SWO* values for each of the problems studied in this chapter and Figure 9.11 plots their 99% confidence interval. The *SWO* values in Table 9.6 range from close to minimum 0.0 and come close to the maximum 1.0. Therefore, the problem set is varied with respect to epistasis.

Table 9.6 Epistasis *SWO* values, N=256

X-*p*-*q* problem	*SWO*	99% confidence interval
Ce-2-2	0.0931 (0.0226)	± 0.02603
R-4-4	0.9104 (0.0085)	± 0.00979
IIe-2-2	0.9834 (0.0010)	± 0.00115
M2-4-4	0.9892 (0.0008)	± 0.00092
M1-2-4	0.9909 (0.0004)	± 0.00046
M-2-4	0.9915 (0.0004)	± 0.00046
M-2-2	0.9923 (0.0007)	± 0.00081
M1-2-2	0.9933 (0.0002)	± 0.00023
C-2-2	0.9973 (0.0005)	± 0.00058
II-2-2	0.9973 (0.0005)	± 0.00058

Standard deviation values in parentheses.

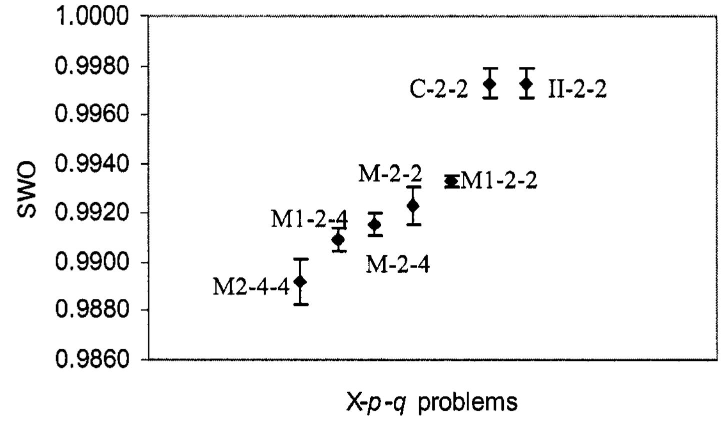


Figure 9.11 ***SWO*** **values and their 99% confidence interval compared**

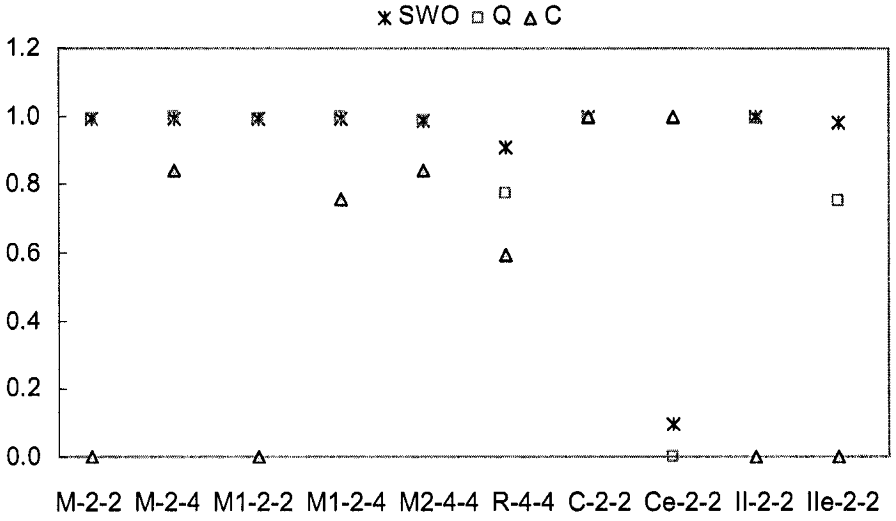


Figure 9.12 ***SWO*****,** ***Q*** **and** ***C*** **values compared, N=256**

Figure 9.12 compares the *SWO*, *Q* and *C* values of the HIFF problems studied in this chapter. The correlation between *SWO* and *Q* is 0.9676, between *Q* and *C* is -0.3104 and between *SWO* and *C* is -0.3965. This implies that *Q* conveys different information from *C*, and *SWO* conveys different information from *C*. There is however a strong direct relationship between *SWO* and *Q* for the problems studied in this chapter. This is not surprising given the importance of modularity and modular-interdependency in the design of the HIFF problems (section 3.3).

A strong direct relationship between *SWO* and *Q* suggests that if one variable is a good indicator of when a problem will be more easily solved by upGA than a hill climber, the other will be too. However, the experiments in Part II confirm that problems with the same *Q* values can be equally difficult for RMHC to solve (section 5.1) but produce distinct upGA optimizing performances (chapter 7). Hence, it is unlikely that *Q* or *SWO* is a good indicator of when a problem will be more easily solved by upGA than a hill climber. This hypothesis is evaluated in section 9.5.3.

9.3 Methodology

The objective of conducting the experiments in this chapter is to find a structural characteristic which best indicates when a problem will be more easily solved by upGA (Figure II.A) than MMHC (Figure 5.3). RMHC (Figure 5.1) results are also reported for comparison. The findings of this chapter are robust to whether the hill climber is RMHC or MMHC, within the given parameters.

To meet the objective stated above, RMHC, MMHC and upGA are run on the X-*p*-*q* test problems defined in section 9.1 with the parameter settings listed in Table 9.7 below. The results of these runs are reported in section 9.4 and analyzed in section 9.5.

A good indicator of when a problem will be more easily solved by upGA than MMHC is one that correlates strongly with upGA – MMHC values, i.e. the difference in SUCC values between upGA and MMHC; or alternatively, one that categorizes problems by their upGA – MMHC values. SUCC values are compared for the reasons explained in the section 9.4.

Table 9.7 Parameter settings for experiments in chapter 9

Parameter	RMHC and MMHC	upGA
Problem size (N)	256	256
MaxEvals	0.5 million	1.0 million
Number of runs per experiment	30	30
Population size (PS)	1	N
Mutation rate (P_m)	0.0625	0.0625
Crossover rate (P_x)	-	0.25

9.4 Results

Table 9.8 reports the number of successful runs (SUCC) RMHC, MMHC and upGA made on the X-*p*-*q* test problems defined in section 9.1 using the parameter settings listed in Table 9.7.

Table 9.8 SUCC for RMHC, MMHC and upGA on X-*p*-*q* test problems, N=256

X-*p*-*q* problem	RMHC	MMHC	upGA	upGA –RMHC	upGA –MMHC
Ce-2-2	30	30	30	0	0
IIe-2-2	25	26	30	5	4
R-4-4	6	21	30	24	9
M-2-2	0	6	27	27	21
M2-4-4	0	7	26	26	19
M1-2-4	0	3	17	17	14
M-2-4	0	3	12	12	9
M1-2-2	0	1	10	10	9
C-2-2	0	0	8	8	8
II-2-2	0	0	7	7	7

In general, all problems except Ce-2-2, IIe-2-2 and R-4-4 are difficult for both RMHC and MMHC to solve. The most significant difference between MMHC and

RMHC success rates is on the R-4-4 problem. Problems which are easier for RMHC and MMHC to solve are also easier for upGA to solve. This is expected since upGA relies on mutation for keeping its population sufficiently diverse and mutation can only do this if it can be successful, i.e. a mutant genotype can replace its parent because it is as fit as or fitter than its parent. But the more mutation is allowed to be successful for a problem, the easier the problem becomes for mutation to solve. Thus a positive relationship between the SUCC values of upGA and the SUCC values of MMHC (and RMHC) is observed in Table 9.8. Nonetheless, this need not mean that a mutation-only evolutionary algorithm such as a hill climber will solve the problem easily. For example, in Part II, HIFF-M is less mutation-hard than HIFF-D, but HIFF-M is sill more easily solved by upGA than a mutation-only evolutionary algorithm.

What is more interesting however, are the instances where RMHC and MMHC have low success rates and yet upGA has high success rates. This gap in SUCC values may then be attributed to the use of the idea of crossover (section 5.2.2) and a multi-individual population in an evolutionary algorithm. However this explanation for the gap in SUCC values is not sufficient since as witnessed in Part II, the use of crossover has the attendant difficulty of premature loss of genetic diversity in a GA that does not explicitly manage its population diversity, e.g. upGA.

This chapter proposes that a more complete explanation for the gap in SUCC values includes the structure of the problems themselves. Hence, the investigation in this chapter to find relevant structural characteristics that would indicate when upGA will be more successful than MMHC (and RMHC) at solving a problem.

MFPT values

Next, MFPT values are examined to see whether they rank the evolutionary algorithms (upGA, RMHC and MMHC) differently from SUCC values. Table 9.9 reports the average number of function evaluations needed to find a (the first) globally optimal string (MFPT).

Table 9.9 MFPT for RMHC, MMHC and upGA on X-*p*-*q* test problems, N=256

X-*p*-*q* problem	upGA-MMHC Max = 30	RMHC		MMHC		upGA	
		MFPT	95% confidence interval for MFPT	MFPT	95% confidence interval for MFPT	MFPT	95% confidence interval for MFPT
M-2-2	21	-	-	138,560 (156,530)	± 125,248	488,040 (180,900)	± 68,235
M2-4-4	19	-	-	193,890 (128,510)	± 95,200	391,140 (124,570)	± 47,882
M1-2-4	14	-	-	428,900 (58,412)	± 66,098	528,160 (161,370)	± 76,709
R-4-4	9	7,119 (1,791)	± 1,433	9,542 (3,808)	± 1,629	243,850 (21,740)	± 7,779
M-2-4	9	-	-	339,790 (197,220)	± 146,100	523,510 (182,230)	± 103,104
M1-2-2	9	-	-	56,919 (-)	-	606,540 (155,200)	± 96,192
C-2-2	8	-	-	-	-	313,516 (66,384)	± 46,001
II-2-2	7	-	-	-	-	403,508 (186,163)	± 137,909
IIe-2-2	4	9,127 (2,273)	± 891	11,177 (5,790)	± 2,226	315,095 (23,585)	± 8,440
Ce-2-2	0	7,375 (2,053)	± 735	8,013 (2,334)	± 835	216,620 (16,292)	± 5,830

Note: The test problems in Table 9.9 are not in the same order as in Table 9.8.

Figure 9.13 plots the MFPT values in Table 9.9 with their respective 95% confidence intervals. Since the 95% confidence interval for the upGA MFPT values of the M* problems overlap each other, there are no significant differences between their upGA MFPT values. Hence, differences in upGA's SUCC values amongst the M* problems (which ranges from 10 to 27 in Table 9.8) is not attributable to differences in upGA's MFPT values.

The higher success rate of upGA on Ce-2-2, IIe-2-2 and R-4-4 than on the M* problems, is also not attributable to larger MFPT values. upGA's MFPT values for these problems are significantly smaller than those for the M* problems. As explained above, the Ce-2-2, IIe-2-2 and R-4-4 problems are easy for upGA because they are easy for MMHC, and in some cases also RMHC. In fact, the Ce-2-2 and IIe-2-2 problems are easier for hill climbers than genetic algorithms to solve. Their MFPT values for RMHC and MMHC are significantly smaller than that for upGA and their upGA – MMHC and upGA – RMHC values are small (≤ 5). The same cannot be said unconditionally about the R-4-4 problem because its upGA – RMHC value is large (21).

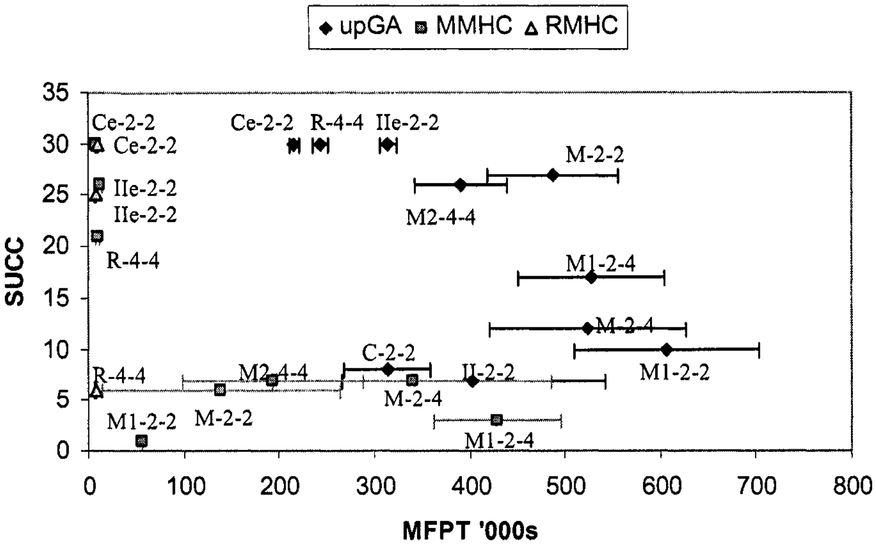


Figure 9.13 MFPT values with 95% confidence intervals

Both RMHC and MMHC have `MaxEvals` values of 0.5 million and the `MaxEvals` value for upGA is 1.0 million (Table 9.7). Only MMHC's MFPT values for M-2-4 and M1-2-4 come close to 0.5 million (Figure 9.13). All other MFPT values in Table 9.9 are less than 70% of their respective `MaxEvals` values. Without incurring a significant increase in MFPT values, upGA is at least four times more successful than MMHC on both M-2-4 and M1-2-4. MMHC has a 10% success rate on both M-2-4 and M1-2-4 while upGA's success rates for M1-2-4 and M-2-4 respectively are 56% and 40% (Table 9.8).

The previous discussion on MFPT values imply that increasing the `MaxEvals` value of the respective evolutionary algorithms is unlikely to change the upGA – MMHC (or upGA – RMHC) SUCC values reported in Table 9.8 dramatically. Most runs either complete successfully well before they reach `MaxEvals`, or do not complete successfully at `MaxEvals`. In instances where runs completed successfully and came close to `MaxEvals`, e.g. MMHC M-2-4 and MMHC M1-2-4, increasing `MaxEvals` is unlikely to change the upGA – MMHC SUCC values reported in Table 9.8 substantially because many more upGA runs completed successfully within the same number of function evaluations used by MMHC. The MFPT values for upGA and MMHC on M-2-4 and M1-2-4 respectively, do not differ significantly (Figure 9.13). Appendix A displays randomly chosen genotypes at the end of unsuccessful MMHC runs to paint a more concrete picture of the difficulty posed by the X-*p*-*q* problems to MMHC.

Best-so-far fitness graphs

For a qualitative comparison of optimizing performance, Figure 9.14 displays the best-so-far fitness graphs of randomly chosen successful MMHC and upGA runs. Fitness (the y-axis) of the best-so-far fitness graphs is normalized (fitness values are divided by the maximum fitness value of the respective problem) so runs from different X-*p*-*q* problems can be included in one graph. As expected, fitness increases more rapidly early in the MMHC runs than in the upGA runs. Prior to around 200,000 function evaluations, successful MMHC runs outperform (reach higher levels of fitness) successful upGA runs.

Discussion

The choice of any performance measure, in this case SUCC, is not a straightforward matter. Different performance measurements may rank the same set of evolutionary algorithms on the same set of test problems differently. There are two possible reactions to this: (i) since there appears to be no one definitive objective way of comparing the performance of evolutionary algorithms, all efforts to do so is futile; (ii) since there appears to be no one definitive objective way of comparing the performance of evolutionary algorithms, all efforts to do so should be encouraged to obtain information from multiple points of view. The second reaction also acknowledges that practitioners may have different resources and priorities when solving problems and thus be interested in different performance measurements.

Part III, and in general this dissertation, adopts the more constructive second reaction. It makes clear that the findings of this study are based on differences in SUCC values of evolutionary algorithms configured with a particular set of parameter values, but it also briefly examines other forms of performance measurements, i.e. MFPT and best-so-far fitness graphs. Since from this examination, neither MFPT nor best-so-far fitness curves are found to contradict the evaluation made with SUCC values in a substantially relevant manner, SUCC values are taken to represent the differences between optimizing performances of evolutionary algorithms on the test problems in this chapter.

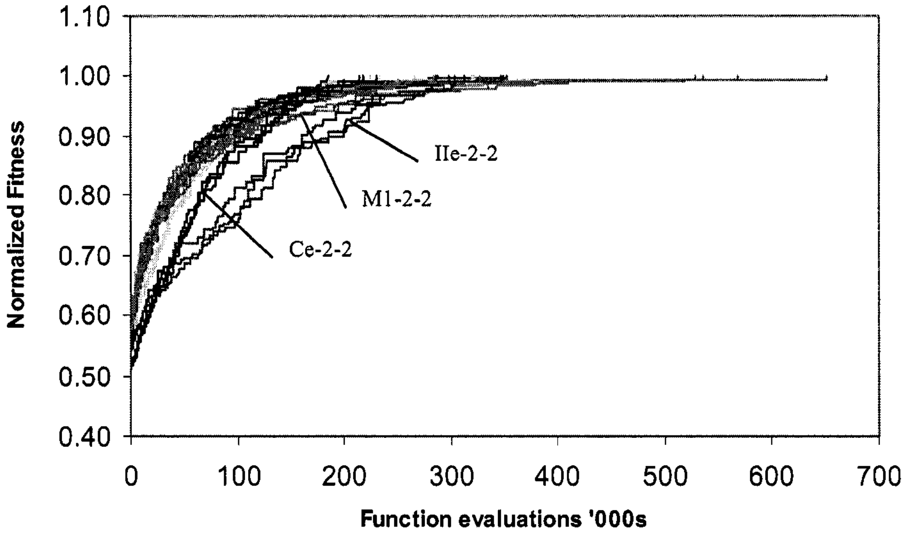


Figure 9.14a Best-so-far fitness (normalized) graphs for upGA. Three successful runs are randomly chosen for each test problem, N=256. Three sets of upGA runs marginally distinguishable from the pack are marked: Ce-2-2, IIe-2-2 and M1-2-2.

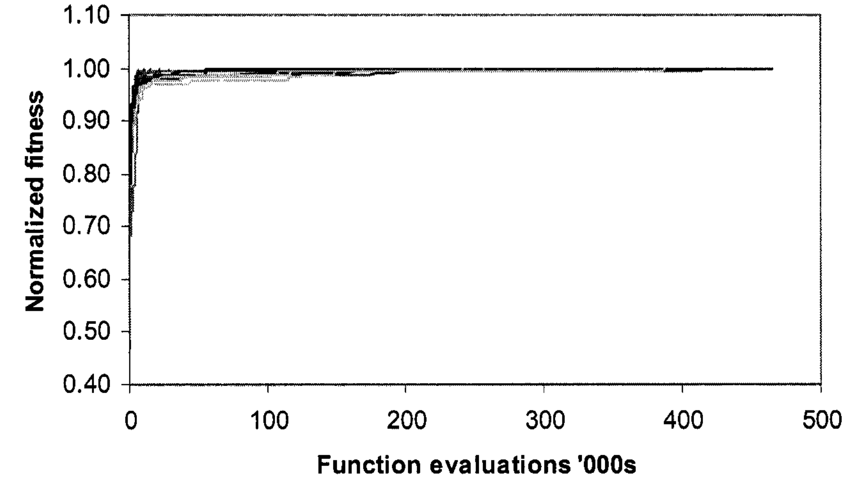


Figure 9.14b Best-so-far fitness (normalized) graphs for MMHC. Three successful runs where available are randomly chosen for each test problem, N=256. M1-2-4 and M-2-4 runs took much longer to complete.

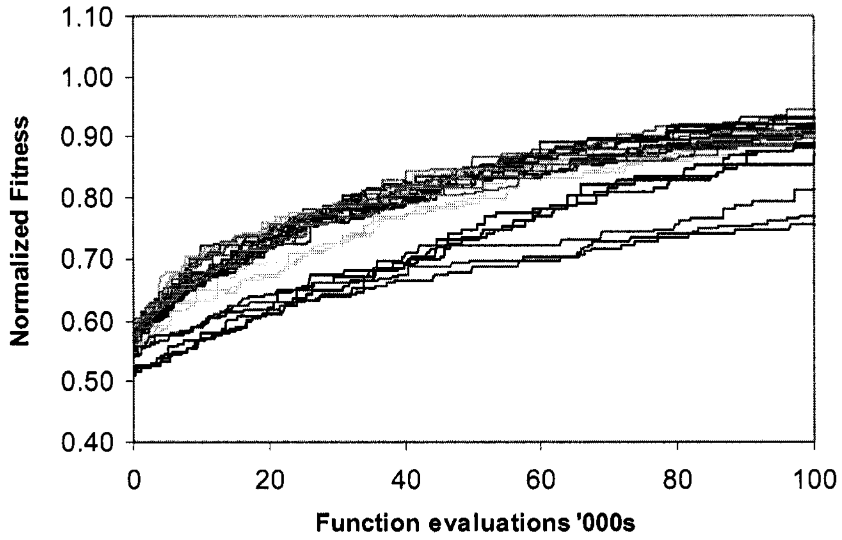


Figure 9.14c A closer look at the first 100,000 function evaluations of the upGA runs in Figure 9.14a.

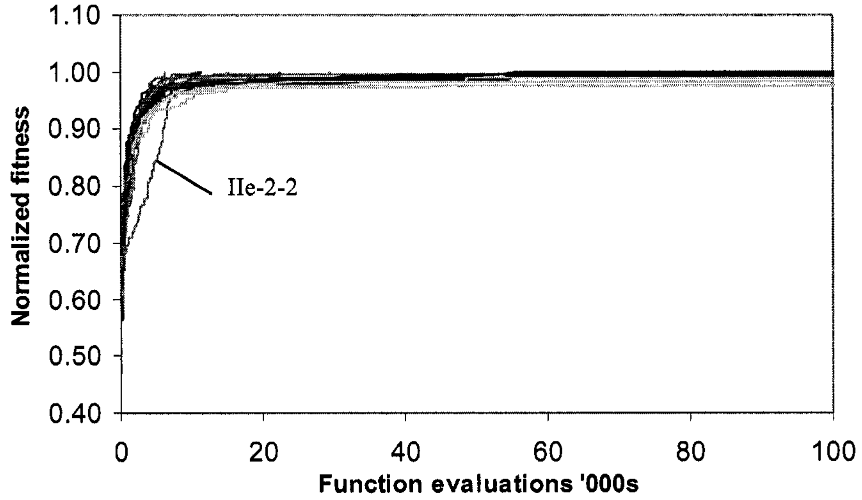


Figure 9.14d A closer look at the first 100,000 function evaluations of the MMHC runs in Figure 9.14b.

9.5 Finding the best indicator

Using the results in section 9.4, this section considers one structural characteristic at a time to find the best indicator from amongst degree distribution type, modularity, transitivity and epistasis, of when a problem will be more easily solved by upGA (Figure II.A) than MMHC (Figure 5.3).

A good indicator of when a problem will be more easily solved by upGA than MMHC is one that correlates strongly with upGA – MMHC SUCC values, i.e. the difference in SUCC values between upGA and MMHC; or alternatively, one that categorizes problems by their upGA – MMHC SUCC values.

9.5.1 Degree distribution type

This subsection investigates whether degree distribution type (section 8.2.1) is a good indicator of when a problem will be more easily solved by upGA than MMHC. The investigation concludes with an affirmative answer.

Tables 9.10a and 9.10b arrange the X-*p*-*q* problems in Table 9.8 by descending order of upGA – MMHC and upGA – RMHC values respectively. The only positional difference between the two arrangements in these tables is the placement of R-4-4. The R-4-4 problem has a larger upGA – RMHC than upGA – MMHC value.

Table 9.10a X-*p*-*q* problems in Table 9.8 sorted by descending order of upGA – MMHC

Degree distribution type	upGA – MMHC SUCC		*Q* (Table 9.5)	*C* (Table 9.4)	*SWO* (Table 9.6)
(ii) Exponential	**M-2-2**	**21**	**0.9922**	**0.0000**	0.9923 ± 0.00081
(ii) Exponential	M2-4-4	19	0.9843	0.8396	0.9892 ± 0.00092
(iii) Other	M1-2-4	14	0.9955	0.7586	0.9909 ± 0.00046
(ii) Power	**R-4-4**	9	0.7742	0.5931	0.9104 ± 0.00979
(ii) Power	M-2-4	9	0.9955	0.8412	0.9915 ± 0.00046
(iii) Other	**M1-2-2**	**9**	**0.9922**	**0.0000**	0.9933 ± 0.00023
(i) Single-point	C-2-2	8	0.9922	1.0000	0.9973 ± 0.00058
(i) Single-point	II-2-2	7	0.9922	0.0000	0.9973 ± 0.00058
(i) Single-point	IIe-2-2	4	0.7500	0.0000	0.9834 ± 0.00115
(i) Single-point	Ce-2-2	0	0.0000	1.0000	0.0931 ± 0.02603

The categories for degree distribution type were explained in section 9.2.1. *Q*, *C* and *SWO* values are reproduced here from section 9.2 for the convenience of the reader.

Table 9.10b X-*p*-*q* problems in Table 9.8 sorted by descending order of upGA – RMHC

Degree distribution type	upGA – RMHC SUCC		*Q* (Table 9.5)	*C* (Table 9.4)	*SWO* (Table 9.6)
(ii) Exponential	**M-2-2**	**27**	**0.9922**	**0.0000**	0.9923 ± 0.00081
(ii) Exponential	M2-4-4	26	0.9843	0.8396	0.9892 ± 0.00092
(ii) Power	**R-4-4**	24	0.7742	0.5931	0.9909 ± 0.00046
(iii) Other	M1-2-4	17	0.9955	0.7586	0.9104 ± 0.00979
(ii) Power	M-2-4	12	0.9955	0.8412	0.9915 ± 0.00046
(iii) Other	**M1-2-2**	**10**	**0.9922**	**0.0000**	0.9933 ± 0.00023
(i) Single-point	C-2-2	8	0.9922	1.0000	0.9973 ± 0.00058
(i) Single-point	II-2-2	7	0.9922	0.0000	0.9973 ± 0.00058
(i) Single-point	IIe-2-2	5	0.7500	0.0000	0.9834 ± 0.00115
(i) Single-point	Ce-2-2	0	0.0000	1.0000	0.0931 ± 0.02603

In both arrangements in Tables 9.10a and 9.10b, problems with single-point degree distribution type (C-2-2, II-2-2, IIe-2-2 and Ce-2-2) occupy the last four places, and problems with exponential degree distribution type (M-2-2 and M2-4-4) occupy the first two places. The remaining places in the middle are occupied by problems with power (R-4-4 and M-2-4) or 'other' (M1-2-2 and M1-2-4) degree distribution types. Section 9.2.1 classifies M1-2-2 and M1-2-4 as problems with 'other' degree distribution type because their degree distributions (Figure 9.8a) are not single-point and are not right-skewed. But, their degree distribution curves are better fitted with power than exponential functions (Table 9.3).

The arrangement in Table 9.10a suggests that degree distribution type can categorize problems by their upGA – MMHC values. Therefore degree distribution type is a good indicator of when a problem will be more easily solved by upGA than MMHC. Specifically, problems with an exponential degree distribution show most discrimination between upGA and MMHC while problems with a single-point degree distribution are least discriminating. Problems with degree distributions that are more power-like than exponential are less discriminating than problems with exponential degree distribution type, but are more discriminating than problems with single-point degree distribution type. These same observations can be made with the arrangement in Table 9.10b.

A comparison between the M-2-2 and M1-2-2 test problems and their SUCC values demonstrates clearly the effect degree distribution type can have on the ability of a problem to discriminate between upGA and MMHC. For problems of the same size, M-2-2 and M1-2-2 have the same level of modularity or Q value (Table 9.3), the same level of network transitivity or C value (Table 9.4), and similar *SWO* values (Figure 9.12). But M-2-2 and M1-2-2 have distinctly different degree distribution types (Figure 9.8a). M-2-2's degree distribution type is exponential and fits the profile of degree distribution types found in the real-world. M1-2-2's degree distribution does not. The upGA – MMHC value for M-2-2 (21) is larger than that for M1-2-2 (9) (Table 9.8). Further, for problems of the same size, M-2-2 and M1-2-2 have identical fitness distributions (Figure 4.4) and FDC and FDC-X values (section 4.2). This observation also diminishes somewhat the likelihood of fitness distribution or FDC and FDC-X values as good indicators of when a problem will be more easily solved by upGA than a hill climber.

Thus, other factors being equal, a change in degree distribution type can dramatically affect the ability of a problem to discriminate between upGA and MMHC. These same observations can be made with RMHC in place of MMHC.

9.5.2 Network transitivity

This subsection investigates whether network transitivity (section 8.2.2) is a good indicator of when a problem will be more easily solved by upGA than MMHC. The investigation does not conclude with an affirmative answer.

Table 9.11 arranges the X-*p*-*q* problems in order of increasing clustering coefficient *C* values. Three categories of problems are found: (i) those without network transitivity, *C* = 0.000; (ii) those with maximum network transitivity, *C* = 1.000; and (iii) those with network transitivity in between the two extremes.

Table 9.11 Clustering coefficient *C* and SUCC values

X-*p*-*q* problem	*C*	RMHC SUCC	MMHC SUCC	upGA SUCC	upGA – RMHC SUCC	upGA – MMHC SUCC
II-2-2	0.0000	0	0	7	7	7
IIe-2-2	0.0000	25	26	30	5	4
M-2-2	**0.0000**	0	6	27	**27**	**21**
M1-2-2	0.0000	0	1	10	10	9
R-4-4	0.5931	6	21	30	24	9
M1-2-4	0.7586	0	3	17	17	14
M2-4-4	**0.8396**	0	7	26	**26**	**19**
M-2-4	0.8412	0	3	12	12	9
C-2-2	1.0000	0	0	8	8	8
Ce-2-2	1.0000	30	30	30	0	0
	Correlation	0.0617	0.0935	0.0371	-0.0364	-0.1100

The resulting arrangement in Table 9.11 do not suggests that network transitivity can categorize problems by their upGA – MMHC or upGA – RMHC values. Therefore it is unlikely that network transitivity is a good indicator of when a problem will be more easily solved by upGA than a hill climber. The two problems with the largest upGA – MMHC values (M-2-2 and M2-4-4) appear in the first and third categories, and have widely different clustering coefficient *C* values. Conversely, it is possible for problems with the same *C* values to have vastly different SUCC values.

In addition, the correlation between *C* and any of the SUCC values in Table 9.11 is low, i.e. between -0.1100 and 0.0935 (Table 9.11). Figure 9.15 give the scatter-plots of *C* values against SUCC values in Table 9.11. As the correlation values indicate, no clear relationship is discernable between *C* and SUCC values.

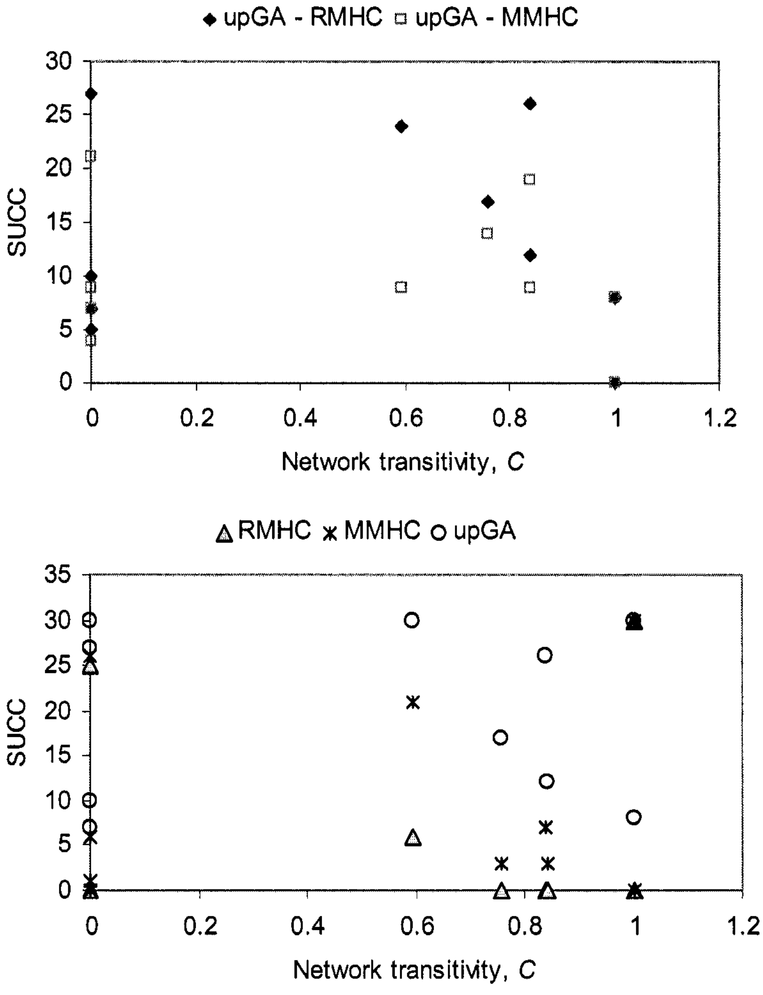


Figure 9.15 *C* vs. SUCC scatter-plots

A comparison between the C-2-2 and II-2-2 test problems and their SUCC values clearly demonstrates that network transitivity has only weak influence over the ability of a problem to discriminate between upGA and a hill climber (RMHC or MMHC). For problems of the same size, C-2-2 and II-2-2 have the same degree distribution type, the same level of modularity or *Q* value and identical *SWO* values (Table 9.10a). But their *C* values lie on opposite ends of the spectrum. Network transitivity is maximal for C-2-2 and is minimal for II-2-2. But the upGA – MMHC and upGA – RMHC values for C-2-2 and II-2-2 are essentially the same. Thus, other factors being equal, a change in network transitivity is unlikely to affect the ability of a problem to discriminate between upGA and a hill climber (RMHC or MMHC).

9.5.3 Modularity

This subsection investigates whether modularity (section 8.2.3) is a good indicator of when a problem will be more easily solved by upGA than MMHC. The investigation does not conclude with an affirmative answer. But, it does suggest that modularity is a good indicator of when a problem will be difficult for a hill climber (MMHC or RMHC) to solve.

Table 9.12 arranges the X-*p*-*q* problems in order of increasing *Q* values. The problems can be divided into two categories: (i) those with low modularity, *Q* values < 0.9000 and (ii) those with high modularity, *Q* values ≥ 0.9000.

The resulting arrangement in Table 9.12 does not suggest that modularity can categorize problems by their upGA – MMHC (or upGA – RMHC) values. Problems that have the same *Q* values (C-2-2, II-2-2, M-2-2 and M1-2-2) have wide ranging upGA –

MMHC (7 to 21) and upGA – RMHC (7 to 27) values. Therefore modularity is unlikely to be a good indicator of when a problem will be more easily solved by upGA than a hill climber (MMHC or RMHC). Figure 9.16 give plots the *Q* and SUCC values in Table 9.12.

Table 9.12 Modularity *Q* and SUCC values

X-*p*-*q* problem	*Q*	RMHC SUCC	MMHC SUCC	upGA SUCC	upGA – RMHC SUCC	upGA – MMHC SUCC
Ce-2-2	0.0000	30	30	30	0	0
IIe-2-2	0.7500	25	26	30	5	4
R-4-4	0.7742	6	21	30	24	9
M2-4-4	0.9843	0	7	26	**26**	**19**
M-2-2	0.9922	0	6	27	**27**	**21**
M1-2-2	0.9922	0	1	10	10	9
C-2-2	0.9922	0	0	8	8	8
II-2-2	0.9922	0	0	7	7	7
M-2-4	0.9955	0	3	12	12	9
M1-2-4	0.9955	0	3	17	17	14
Correlation		-0.8740	-0.8250	-0.5439	0.4976	0.6393

Table 9.12 reports the presence of a strong negative correlation between *Q* and MMHC SUCC (-0.8250) and a moderately strong negative correlation between *Q* and upGA SUCC (-0.5439). These negative correlations imply that high levels of modularity adversely impact the ability of both MMHC and upGA to solve the problems, but the negative impact is more strongly felt by MMHC than upGA. So problems with high modularity most likely produce low MMHC SUCC values, but they may or may not be difficult for upGA to solve. Thus, a mixed bag of upGA – MMHC values are seen in Table 9.12 for the X-*p*-*q* problems with *Q* values ≥ 0.9000.

The stronger negative correlation between *Q* and MMHC SUCC than between *Q* and upGA SUCC help explain the positive moderately strong correlation between *Q* and upGA – MMHC values (0.6393). In short, higher levels of modularity can increase the ability of a problem to discriminate between upGA and MMHC, but within limits. The same analysis applies if the RMHC results are used in place of MMHC.

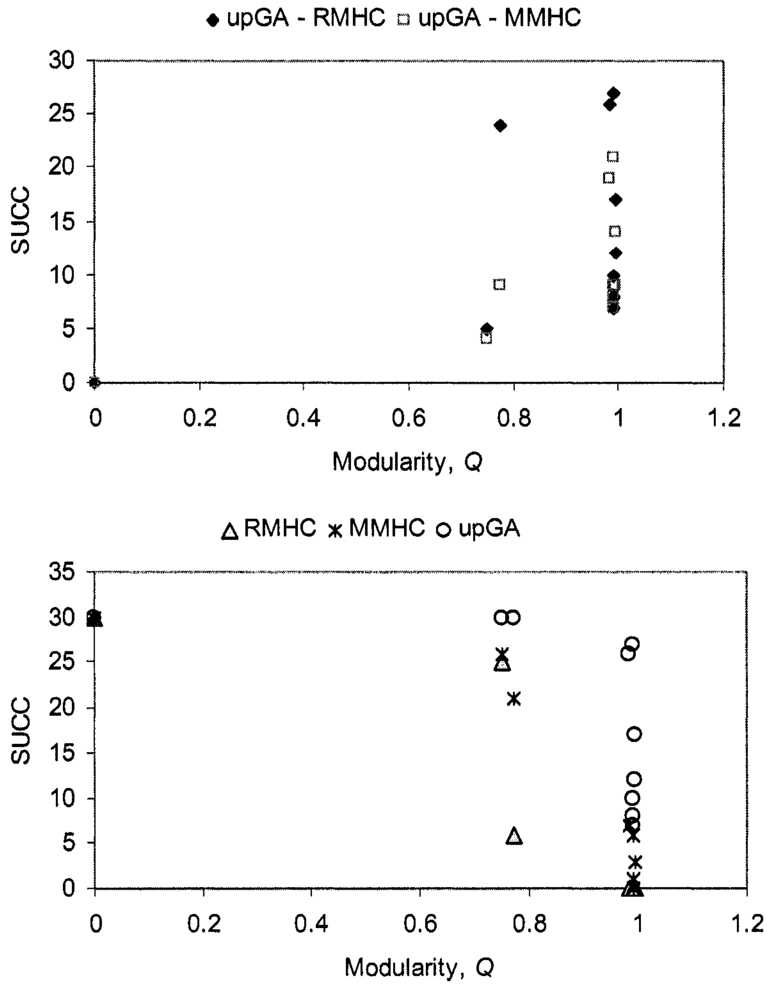


Figure 9.16 *Q* vs. SUCC scatter-plots

Comparison with related work

The adverse impact of high levels of modularity (in problems) on the ability of the hill climbers (RMHC and MMHC) to solve problems is consistent with Watson's work (2002) and a subsequent study by Mills and Watson (2007). In both of these studies, the genetic algorithm is GADC (section 6.1), and the variables of the test problems are fully connected with each other in the manner of HIFF-C (section 3.2.1).

The studies by Watson (2002) and Mills and Watson (2007) emphasize the importance of modularity in distinguishing mutation (hill climbers) from crossover (genetic algorithms). However, the results in this section do not suggest that modularity by itself is a good indicator of when a problem will be more easily solved by a genetic algorithm than a hill climber. This inconsistency is due to differences between GADC and upGA, and the importance of mutation to the success of these genetic algorithms.

GADC manages population diversity explicitly with the deterministic crowding strategy, and therefore relies less on mutation for reintroducing loss alleles into its population. This means a problem can be extremely difficult for hill climbers without adversely impacting GADC SUCC values. Thus, problems with high levels of modularity also have GADC SUCC values which are much larger than RMHC or MMHC SUCC values. With GADC, it is possible to reduce homogenizing genetic drift that its success on a problem becomes a given. When genetic algorithm success on a problem is a given, the difference between genetic algorithm and hill climber SUCC values becomes dependent only on hill climber SUCC values.

In contrast, upGA does not manage population diversity explicitly and relies on mutation for reintroducing loss alleles into its population. This means a problem cannot be extremely difficult for hill climbers without adversely impacting upGA SUCC values. To solve a problem successfully, upGA relies on mutation to be effective to avoid premature convergence. This reliance creates a conundrum because upGA is a steady-state genetic algorithm, so for mutation to be effective and have a diversifying effect on a population, mutant genotypes must be at least as fit as their parents in order to replace them in the population. But a problem that allows such fit mutants also increases the probability of success for hill climbers, or other mutation-only evolutionary algorithms. Hence, the test problems in the present study are subjected to a more delicate balancing act than in the studies by Watson (2002) and Mills and Watson (2007).

To summarize, modularity is a better indicator of evolutionary difficulty for hill climbers than genetic algorithms. For the same problem size N, C-2-2, II-2-2 and M-2-2 have identical *Q* values (Table 9.5), but yield vastly different upGA SUCC values (Table 9.8).

Structurally, an M-2-2 network is distinct from a C-2-2 or II-2-2 network. Links in an M-2-2 network are equally weighted and distributed heterogeneously, while links in a C-2-2 or II-2-2 network are unequally weighted and distributed homogeneously. Thus, how modularity is attained in a problem is relevant to evolutionary difficulty for upGA. The structural difference between an M-2-2 network and a C-2-2 or II-2-2 network is reflected in degree distribution type. C-2-2 and II-2-2 have the same *SWO* value (Table 9.6), and II-2-2 and M-2-2 have the same clustering coefficient (Table 9.4).

9.5.4 Epistasis

This subsection investigates whether (the amount of) epistasis (section 2.5.1) is a good indicator of when a problem will be more easily solved by upGA than MMHC. The investigation does not conclude with an affirmative answer. In general, the effect of epistasis on evolutionary difficulty mirrors that of modularity, which is not surprising given the strong correlation between *Q* and *SWO* values (section 9.2.4).

Table 9.13 arranges the X-*p*-*q* problems in order of increasing *SWO* values. Problems with low modularity, *Q* values < 0.9000, also have the smallest *SWO* values. Figure 9.17 give plots the *SWO* and SUCC values in Table 9.13.

The resulting arrangement in Table 9.13 does not suggest that epistasis can categorize problems by their upGA – MMHC (or upGA – RMHC) values. Problems with not significantly different *SWO* values (Figure 9.11) yield very different upGA – MMHC values. For example: M-2-4 and M-2-2; and M1-2-4 and M-2-4. Conversely, problems with significantly different *SWO* values yield the same upGA – MMHC values. For example: R-4-4, M-2-4 and M1-2-2. Therefore it is unlikely that epistasis is a good indicator of when a problem will be more easily solved by upGA than MMHC. The same observations can be made with RMHC results in place of MMHC (although with different example test problems).

Table 9.13 *SWO* and SUCC values

X-*p*-*q* problem	*SWO*	RMHC SUCC	MMHC SUCC	upGA SUCC	upGA – RMHC SUCC	upGA – MMHC SUCC
Ce-2-2	0.0931	30	30	30	0	0
R-4-4	0.9104	6	21	30	24	9
IIe-2-2	0.9834	25	26	30	5	4
M2-4-4	0.9892	0	7	26	**26**	**19**
M1-2-4	0.9909	0	3	17	17	14
M-2-4	0.9915	0	3	12	12	9
M-2-2	0.9923	0	6	27	**27**	**21**
M1-2-2	0.9933	0	1	10	10	9
C-2-2	0.9973	0	0	8	8	8
II-2-2	0.9973	0	0	7	7	7
Correlation		-0.7420	-0.6665	-0.4135	0.4726	0.5562

Additionally, the correlation between *SWO* and SUCC values in Table 9.13 are only moderately strong, and follow the same pattern as the correlation between *Q* and SUCC values in Table 9.12. Thus, the effect of epistasis on evolutionary difficulty is akin to that explained for modularity (section 9.2.3).

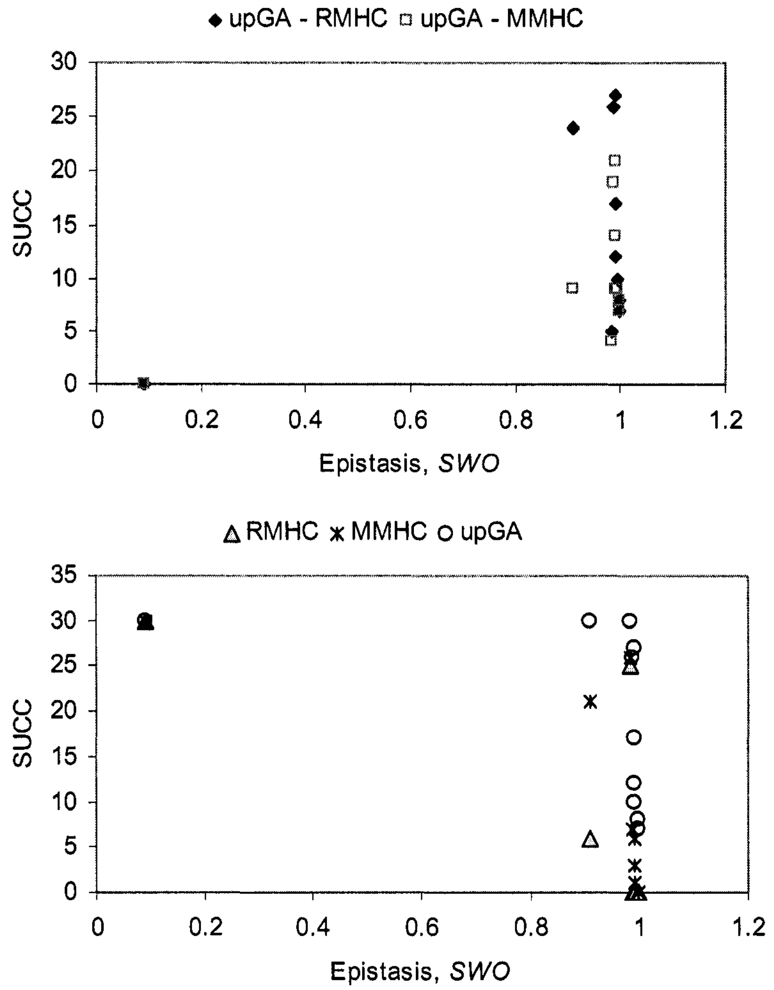


Figure 9.17 *SWO* vs. SUCC scatter-plots

9.5.5 Section conclusion

Of the four problem properties considered in this section – degree distribution type, network transitivity, modularity and epistasis – degree distribution type is the best indicator of when a problem will be more easily solved by upGA than a hill climber. The hill climber may be MMHC or RMHC.

9.6 Discussion

The parameter settings in Table 9.7 have not been researched extensively (although this need not be a bad thing). Thus the conclusions derived from the results in section 9.4 are valid for only a small set of parameter settings. Nevertheless, the findings of this chapter can still be regarded as a proof of concept that problem structure is relevant to evolutionary algorithm difficulty. While this assertion might seem obvious to some, an actual study of the kind conducted in this chapter has not been done thus far.

What has been done in terms of network structure and evolutionary algorithms is the explicit use of network topology to study the effectiveness of multi-populations structured in different ways (e.g. Bryden et al. 2006; Payne and Eppstein, 2007). In contrast to these studies, the approach in this dissertation is to study structure as a component of the problem itself, and not as a component of the solution (evolutionary algorithm).

To increase the robustness and precision of the findings in this chapter, additional experiments and analyses with the help of an automatic test problem generator that incorporates problem structural characteristics as parameters is suggested for future work. Such an automated test problem generator would need to be able to create networks with user-specified degree distribution type, modularity level and/or network transitivity extent. In addition, knowing what the globally optimal solutions are a priori will also be helpful, e.g. calculating distance to a globally optimal string.

The structure of problems produced by automated test problem generators has hitherto been random. For example, automated test problem generators such as Kauffman's NK model (1989) and Random Boolean Networks (Kauffman, 1993), and those proposed by de Jong et al. (1997) and Naudts (1998) to study epistasis and by Spears (1998) to study multimodality, and the Hierarchical Problem Generator (HPG) (de Jong et al. 2005) to study hierarchical problems, are all designed to create test problems with random variable interactions. The element of randomness in test problems insures against over-tuning evolutionary algorithms to a specific set of ad hoc test problems and inadvertent exploitation of hidden biases in a set of test problems, and thereby increases the predictive power of empirical results for the class of problems studied (Spears, 1998).

In addition to this argument for randomness, random constrained optimization problems (e.g. graph-coloring, TSP – traveling sales person and SAT) are also frequently used as benchmark tests for evolutionary algorithms because they are viewed as more realistic than toy problems such as HIFF and many others. But how well do random optimization problems correspond to real-world optimization problems? Walsh (1999): "search problems met in practice may be neither completely structured nor completely random...simple topology features can have a large impact on the cost of solving search problems."

Perhaps because of this argument for randomness, problem structure beyond the amount of epistasis, has not garnered much attention as a factor of significance to evolutionary difficulty. This dissertation hopes to change this situation by exciting research in this direction.

At first glance, studying the effect of problem structure on evolutionary algorithm difficulty without considering the fitness component may appear odd, even illogical. Spears (1998, p.221): "This illustrates the danger in attempting to characterize fitness functions with purely syntactic measures that ignore fitness to a large extent." The fitness functions referred to by Spears are random Boolean satisfiability problems produced by a test problem generator designed to investigate epistasis, multimodality and crossover. The amount of epistasis is controlled by varying three parameters: the number of Boolean variables (V), the number of clauses (C) and the number of Boolean variables in each clause (L). The amount of epistasis in a problem is measured syntactically using the V, C and L values, and hence do not involve the fitness component at all. The amount of epistasis in a problem is used to control the number of peaks (in some fitness landscape), while the height of peaks is determined separately by a fitness function. Since fitness of solutions is not an intrinsic part of a random Boolean satisfiability problem, one could change the fitness function without changing the amount of epistasis in a problem. This design produced unexpected results, which led Spears (1998, p.221) to conclude: "Syntactic measures ... are simply inadequate. In order for such characterizations of fitness functions to be useful, they must include fitness information to a much larger extent."

In contrast, the structural characteristics of the X-*p*-*q* problems discussed in this chapter include fitness information. The structural characteristics are computed on the adjacency matrices of the X-*p*-*q* problems, the adjacency matrices are derived from the inter-dependency matrices of the X-*p*-*q* problems, and fitness of an X-*p*-*q* solution (string or genotype) is computed from the inter-dependency matrix of the X-*p*-*q* problem.

That M-2-2 and M1-2-2 have the same fitness distribution but different upGA SUCC (section 9.5.1) is also an interesting find because it suggests a counter-example to the proposal by Borenstein and Poli (2004) to classify GA-hardness of problems (how difficult it would be for a GA to solve a problem) by their fitness distributions. The genetic algorithm (GA) in that study is the Simple Genetic Algorithm (SGA) (Figure 2.4)

A comment received from a reviewer about the work in this chapter is upGA is not a 'standard' genetic algorithm by which is meant the SGA. This author's response to this criticism is: why should the SGA be given such credence when, except for academic convention, history and existing theory on SGA, the SGA is not used in practice as evident from the variety of genetic algorithms proposed over the years by researchers in academia and in industry. That upGA may also not be of practical use is no reason for favouring the use of SGA over upGA, or judging a study which uses SGA more authoritative than one which uses upGA.

In this author's opinion, the purpose of GA theory (if it is to be more than an academic exercise) is to suggest factors and conditions that could influence the performance of evolutionary algorithms so that practitioners can make educated-guesses about which evolutionary algorithm to use on a problem. Ultimately, the practitioners are the experts of their domains and their problems, as should be the case.

9.7 Chapter Summary

This chapter sought to find a problem characteristic which is a better indicator than modularity of when a problem will be more easily solved by upGA (Figure II.A) than a hill climber. Results of the MMHC and the RMHC hill climbers were considered. The problem characteristics of interest were degree distribution type (section 9.5.1), network transitivity (section 9.5.2), modularity (section 9.5.3) and amount of epistasis (section 9.5.4). To conduct the investigation, a set of test problems modeled on the HIFF problems was prepared (section 9.1). As such, the problems are frustrating due to modular-interdependency and bit-flip symmetry (section 3.3.2). The problems in this test set were confirmed to be varied in terms of degree distribution type, network transitivity, modularity and amount of epistasis (section 9.2).

From the results reported in section 9.4, the following conclusions were made:

(i) Degree distribution type is a good indicator of when a problem will be more easily solved by upGA than a hill climber (section 9.5.1). Specifically, problems with an exponential degree distribution showed most discrimination between upGA and a hill climber (MMHC or RMHC) while problems with a single-point degree distribution were least discriminating.

(ii) It is unlikely that network transitivity is a good indicator of when a problem will be more easily solved by upGA than a hill climber (section 9.5.2).

(iii) It is unlikely that modularity is a good indicator of when a problem will be more easily solved by upGA than a hill climber (section 9.5.3). Modularity is a better indicator of evolutionary difficulty for hill climbers than genetic algorithms. This conclusion differs slightly from previous work as discussed in section 9.5.3.

(iv) The effect of epistasis on evolutionary difficulty mirrors that of modularity (section 9.5.4). Therefore, it is also unlikely that the amount of epistasis is a good indicator of when a problem will be more easily solved by upGA than a hill climber. This agrees with previous work which found that the *kind* of epistasis is more important than the amount of epistasis for evolutionary difficulty (section 2.5.1).

Of the four problem properties considered in this section – degree distribution type, network transitivity, modularity and epistasis – degree distribution type is the best indicator of when a problem will be more easily solved by upGA than a hill climber (MMHC or RMHC).

To evaluate the generality and robustness of the above conclusions, a similar study with a larger problem set and a wider range of parameter settings aided by an automated test problem generator to create HIFF problems with user-specified degree distribution type, modularity level, network transitivity and other structural characteristics of interest is suggested for future work. Such a study might also consider analyzing more than one structural characteristic at a time.

10. Conclusion

Evolutionary algorithms (section 2.2.2) have proven themselves to be of practical use. The smorgasbord that is the designs of evolutionary algorithms is an exhibition of human creativity harnessed to pragmatic ends. But it is also its Achilles' heel, leaving it vulnerable to criticisms of lack of scientific foundation, or at least a science of how to use evolutionary algorithms effectively.

General research problem

This dissertation responds to these criticisms by working towards an understanding of what makes a problem more difficult for one evolutionary algorithm to solve as opposed another evolutionary algorithm. The evolutionary algorithms of interest in this dissertation are *hill climbers* and *genetic algorithms*.

Hill climbers and genetic algorithms occupy opposite ends of the evolutionary algorithm complexity spectrum, in the sense that hill climbers (e.g. Figure 2.3) use only one variation operator (mutation), and store only one solution in memory at all times (single-individual population); while genetic algorithms (e.g. Figure 2.4) typically use more than one variation operator (mutation and crossover), and store more than one solution in memory at all times (multi-individual population).

For the sake of brevity, the problem of identifying when a genetic algorithm outperforms a hill climber is referred to in this dissertation as *the HC < GA problem* (section 2.4). A solution to the HC < GA problem takes the form of an abstract problem specially designed to highlight aspects of the problem which makes it easier for genetic algorithms than hill climbers to solve.

The HC < GA problem was first addressed by Mitchell, et al. in 1992 with the *Royal-Roads* problem (section 2.3.3). But their solution was subsequently discovered to be unsatisfactory (section 2.4.3). Later in 1998, another solution to the HC < GA problem was proposed by Watson, et al. in the form of the *Hierarchical-If-And-Only-If* (HIFF) problem (chapter 3). The HIFF problem, of which there are two versions – discrete (HIFF-D) and continuous (HIFF-C), highlights the importance of the right understanding of modularity as the key to distinguishing the evolutionary capability of hill climbers from that of genetic algorithms. Modularity is to be understood in terms of subsets of problem variables that are not only inter-dependent on other problem variables within their own respective subsets, but are also inter-dependent on problem variables outside their own respective subsets; and inter-dependencies are stronger within subsets than between subsets (sections 3.3.1 and 3.3.2). This understanding of modularity is exactly that found in the area of complex networks (section 8.2.3). A problem variable x is inter-dependent on another problem variable y if the effect x has on fitness of a solution depends also on the value of y, and vice versa (chapter 1).

Motivation

The inter-dependencies in the HIFF problem can be represented in an inter-dependency matrix as in Figure I.C. Referring to the matrix in Figure I.C, Watson (2006, p. 14) says: "But significantly, the values of the matrix that are off the diagonal, representing the strength of dependencies of variables across modules, are nonzero." This sentence served as catalyst for the research in this dissertation because it was not accompanied with further explanation with regards to its necessity. Much of this dissertation is a study of HIFF problems from the view point of differences in their inter-dependencies.

First research objective

The first research objective of this dissertation is to determine whether it is necessary for the entries in a HIFF inter-dependency matrix which are off the main diagonal to be nonzero. It is not necessary if the resulting HIFF problem is still a solution to the HC < GA problem. This research objective is met. The conclusion is: it is *not* necessary that all the entries off the main diagonal of a HIFF inter-dependency matrix be nonzero for the resulting HIFF problem to be a solution to the HC < GA problem.

To meet this first research objective, two new HIFF problems (HIFF-II and HIFF-M) were introduced in Part I (chapter 3). The new HIFF problems are derived from the original HIFF problems (HIFF-D and HIFF-C) by selectively reducing the number of inter-dependencies between variables but keeping the optimal phenotype (fitness for each level) unchanged. Both HIFF-II (section 3.2.3) and HIFF-M (section 3.2.4) have off diagonal zero entries in their inter-dependency matrices. In addition, all nonzero entries in HIFF-M's inter-dependency matrix have the same value.

Chapter 3 (section 3.3) also compared the new HIFF problems in terms of the design of the original HIFF problems, and found that in general, the design of the new HIFF problems are compatible with that of the original HIFF problems. However, notable differences were found between HIFF-M and the original HIFF problems in terms of widest fitness saddle width (section 3.3.3), relative speed at which hierarchical levels could optimize (section 3.4.1), and inter-level inter-dependency (section 3.4.2). Section

3.4.3 discussed the consequence of HIFF-M's equally weighted inter-dependencies which explained why HIFF-M has a narrower widest fitness saddle.

Further analysis of, and comparisons between the HIFF problems defined in chapter 3 (i.e. HIFF-D, HIFF-C, HIFF-II and HIFF-M) were made in chapter 4 in terms of amount of epistasis (SWO), mutation and crossover easiness (FDC), and fitness distribution (neutrality). The largest differences found were between the HIFF-M and HIFF-D problems. Compared with HIFF-D, for problems of the same size, HIFF-M has a smaller amount of epistasis (section 4.1), is less mutation-hard and less crossover-easy (section 4.2), and has a less right-skewed fitness distribution (section 4.3). These differences led to the overall expectation that an evolutionary algorithm would exhibit different optimizing performance (section 2.4) on HIFF-M, than on the other HIFF problems. In particular, combating premature loss of genetic diversity may be less difficult to do in a genetic algorithm population for the HIFF-M problem than for the other HIFF problems (this is confirmed in section 7.5).

Based on the analyses in chapter 4 and to meet the first research objective, Part II investigates whether HIFF-M is a solution to the HC < GA problem and if so, how HIFF-M is distinct from HIFF-D as a solution to the HC < GA problem. Together, the experiments in Part II (chapters 5, 6 and 7) confirm that HIFF-M is a distinct solution from HIFF-D to the HC < GA problem (section 7.2). It is easier for upGA (Figure II.A) to solve HIFF-M than HIFF-D because upGA does not explicitly maintain population diversity and relies on mutation to supply genetic diversity, maintaining sufficient levels of genetic diversity is important for solving HIFF-D and HIFF-M (section 6.2), but homogenizing genetic drift is weaker in HIFF-M than in HIFF-D upGA populations (section 6.3). Table II.A summarizes the experiments conducted and conclusions found in Part II.

Second research objective

The second research objective is born out of the results from Part II and the observation that although all of the four HIFF problems in Part I have the same level of modularity (section 9.2.3), their upGA success rates differ widely (section 7.5) and thus so too their ability to distinguish the evolutionary capability of hill climbers from genetic algorithms.

The second research objective is to find a problem characteristic other than modularity that better captures the ability of a problem to distinguish the evolutionary capability of hill climbers from genetic algorithms, particularly when the genetic algorithm does not explicitly maintain population genetic diversity and relies on mutation for genetic diversity, e.g. upGA. This research objective is met. The problem characteristic is degree distribution type (section 9.5.1).

To meet this second research objective, additional HIFF-like problems were introduced in Part III (section 9.1) and viewed as networks. Problems in this test set are varied in terms of degree distribution type (section 9.2.1), network transitivity (section 9.2.2), modularity (section 9.2.3) and amount of epistasis (section 9.2.4). Degree distribution type, network transitivity and modularity are structural characteristics commonly used to distinguish real-world networks from random networks (chapter 8).

A structural characteristic describes a property of a network's topology. A network's topology can be represented by an adjacency matrix **A** where $\mathbf{A}_{ij}$ is the number of links between nodes *i* and *j*. Structural characteristics of a network are then computed on the adjacency matrix for the network. Problem structure refers to structural characteristics of test problems (Part III).

Of the four problem properties considered – degree distribution type, network transitivity, modularity and epistasis – degree distribution type is the best indicator of when a problem will be more easily solved by upGA than a hill climber because the problems in chapter 9 could be categorized according to differences in their upGA and hill climber SUCC values by degree distribution type (section 9.5.1). The hill climber may be MMHC or RMHC.

Problems with exponential and power-like degree distributions showed more discrimination between upGA and MMHC (or RMHC) than problems with single-point degree distributions (section 9.5.1). Exponential and power-like degree distributions are more typical of degree distributions found in real-world networks (chapter 8). Thus, in general, problems with real-world type degree distributions were more easily solved by upGA than by the random mutation hill climber (RMHC) or the macro-mutation hill climber (MMHC).

Main contribution

The main contribution of this dissertation is the connection it makes between the structure of problems genetic algorithms are better than hill climbers at solving, and the structure of real-world networks. In specific terms, problems with real-world type degree distributions were found to be more easily solved by upGA than by a hill climber (section 9.5.1).

While it might seem obvious that problem structure is relevant to evolutionary algorithm difficulty, expect for epistasis (e.g. Naudts, 1998), an actual study of the kind conducted in this dissertation has not been done thus far. Section 9.6 discusses this point further.

Limitations

Discussions in sections 9.4 and 9.6 respectively address the issue of using SUCC values for comparison, and of using the upGA algorithm instead of the Simple Genetic Algorithm.

The parameter settings in Table 9.7 have not been researched extensively (although this need not be a bad thing). Thus the conclusions derived from the results in section 9.4 are valid for only a small set of parameter settings. But this is a commonly acknowledged limitation akin to assumptions theoretical studies make. Naudts and Kallel (2000): "whether a given fitness function belongs to one convergence quality class or not can depend heavily on the choice of parameters of the algorithm." (Part II).

A better answer to the second research objective may very well involve a combination of structural characteristics, e.g. a right-skewed heavy-tailed degree distribution and high modularity, but the study carried out in Part III only looks at one structural characteristic at a time.

Suggestions for future work

To evaluate the generality and robustness of the findings in Part III, a similar study with a larger problem set and a wider range of parameter settings aided by an automated test problem generator to create test problems with user-specified degree distribution type, modularity level, network transitivity and other structural characteristics of interest is suggested for future work. Such a study might also consider analyzing more than one structural characteristic at a time.

Another suggestion is to investigate the relationship between the structure of real-world optimization or combinatorial problems and evolutionary algorithm difficulty. Walsh (1999) used benchmark problems from well-known OR libraries to study the topological features of problems and their impact on the cost of search. A similar study could be done but with evolutionary algorithms. It would also interesting to see whether there is a relationship between problem structure and when evolutionary algorithms outperform more deterministic type search algorithms.

The upGA algorithm is actually a by-product of the $\mathcal{J}$ algorithm described in Khor (2007d), and as such shares similarities with it. In particular upGA's survival selection strategy works on the same principle as $\mathcal{J}$'s criterion for successful joins and exchanges – joins and exchanges succeed only if some extra fitness is the result. This author speculates that there might be an interesting relationship between problem structure (Part III), specificity (Appendix B) and GA difficulty. Specificity or isolation of parts has been noted as an important feature of naturally evolved systems (Maslov and Sneppen, 2002).

Finally, the study in Parts II and III could be extended to other genetic algorithms and even other types of evolutionary algorithms.

References

Albert, R. and A.-L Barabási (2002). Statistical mechanics of complex networks. *Reviews of Modern Physics* 74(1): 47-97.

Altenberg, L. (1994). Evolving better representations through selective genome growth. *IEEE World Congress on Computational Intelligence*: 182-187. IEEE Press.

Altenberg, L. (1995). The Schema Theorem and Price's Theorem. *Foundations of Genetic Algorithms* 3: 23-49. Morgan Kaufmann.

Altenberg, L. (1997). Fitness Distance Correlation analysis: An instructive counterexample. 7th *International Conference on Genetic Algorithms*: 57-64. Morgan Kaufmann.

Altenberg, L. (2005). Modularity in evolution: Some low-level questions. In W. Callebaut and D. Raskin-Gutman (Eds.) *Modularity: Understanding the development and evolution of complex natural systems.* Chapter 5. MIT Press.

Amaral, L. A. N., Scala, A., Barthelemy, M. and H. E. Stanley (2000). Classes of small-world networks. *Proceedings of the National Academy of Sciences* 97 (21): 11149-11152.

Baker, J. (1987). Reducing bias and inefficiency in the selection algorithm. 2nd *International Conference on Genetic Algorithms*: 14-21. Morgan Kaufmann.

Barabási, A.-L. (2003). *Linked: How everything is connected to everything else and what it means for business, science and everyday life.* Plume.

Borenstein, Y. and R. Poli (2004). Fitness distributions and GA hardness. *Parallel Problem Solving in Nature*: 11-20. Springer.

Bryden, K. M., Ashlock, D. A., Corns, S. and S. J. Willson (2006). Graph-based evolutionary algorithms. *IEEE Transactions on Evolutionary Computation* 10 (5): 550-567.

Costa, L. da F., Rodrigues, F. A., Traviesco, G., and P. R. Villas Boas (2007). Characterization of complex networks: A survey of measurements. *Adv. Phys.* 56: 67-242.

Culberson, C. (1994). Mutation-crossover isomorphisms and the construction of discriminating functions. *Evolutionary Computation* 2 (3): 279-311. MIT Press.

Davidor, Y. (1991). Epistasis variance: a viewpoint of GA-hardness. *Foundations of Genetic Algorithms 1*: 23-35. Morgan Kaufmann.

Dawkins, R. (2006). *The Selfish Gene.* Oxford University Press.

De Jong, E. D., Watson, R. A. and D. Thierens (2005). A generator for hierarchical problems. *Genetic and Evolutionary Computation Conference*: 321-326. ACM Press.

De Jong, K. A. (1992). Genetic Algorithms are NOT function optimizers. *Foundations of Genetic Algorithms 2*: 5-17. Morgan Kaufmann.

De Jong, K. A. (2006). *Evolutionary Computation: A Unified Approach.* MIT Press.

De Jong, K. A., Potter, M. A. and W. M. Spears (1997). Using problem generators to explore the effects of epistasis. 7th *International Conference on Genetic Algorithms*: 338-345. Morgan Kaufmann.

Dong-II, S. and M. Byung-Ro (2007). An information-theoretic analysis on the interactions of variables in combinatorial optimization problems. *Evolutionary Computation* 15(2):169-198. MIT Press.

Dorogovtsev, S. N. and J. F. F. Mendes (2004). The shortest path to complex networks.

arXiv:cond-mat/0404593v4.

Ebner, M., Shackleton, M. and R. Shipman (2001). How neutral networks influence evolvability. *Complexity* 7(2): 19-33. Wiley Periodicals.

Eiben, A. E. and J. E. Smith (2003). *Introduction to Evolutionary Computing.* Natural Computing Series, Springer.

Fischer, S. and I. Wegener (2005). The one-dimensional Ising model: mutation versus recombination. *Theoretical Computer Science* 344: 208-225.

Fogel, D. B. (1995). *Evolutionary Computation.* IEEE Press.

Fogel, L., Owens, A. and M. Walsh (1966). *Artificial Intelligence through Simulated Evolution.* New York: John Wiley & Sons.

Forrest, S. and M. Mitchell (1993). Relative building-block fitness and the building-block hypothesis. *Foundations of Genetic Algorithms 2*: 109-126. Morgan Kaufmann.

Gao, L (2001). On inferring autonomous system relationships in the Internet. *IEEE/ACM Transactions on Networking* 9: 733-745.

Girvan, M. and M. E. J. Newman (2002). Community structure in social and biological networks. *Proceedings of the National Academy of Sciences* 99: 7821-7826.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Longman Inc.

Guo, H. and W. H. Hsu (2003). GA-Hardness revisited. *Genetic and Evolutionary Computation Conference*: 211. Springer Berlin / Heidelberg.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems.* The University of Michigan Press, Ann Arbor, MI.

Holland, J. H. (1992). *Adaptation in Natural and Artificial Systems.* The MIT Press, Cambridge, MA.

Jansen, T. and I. Wegener (2005). Real Royal Road functions – where crossover provably is essential. *Discrete Applied Mathematics* 149: 111-125.

Jones, T. (1995). *Evolutionary Algorithms, Fitness Landscapes and Search.* PhD Dissertation, University of New Mexico, New Mexico, USA.

Jones, T. and S. Forrest (1995). Fitness Distance Correlation as a measure of problem difficulty for genetic algorithms. 6th *International Conference on Genetic Algorithms*: 184-192. Morgan Kaufmann.

Kauffman, S. A. (1989). Adaptation on rugged fitness landscapes. *Lectures in the Sciences of Complexity* 1 SFI studies: 619-712. Addison-Wesley.

Kauffman, S. A. (1993). *The Origins of Order: Self-organization and Selection in Evolution.* Oxford University Press.

Keller, E. F. (2005). Revisiting "scale-free" networks. *BioEssays* 27: 1060-1068. Wiley Periodicals, Inc.

Khor, S. (2006). Constructional selection, methylation and adaptable representation: Preliminary findings. *Genetic and Evolutionary Computation Conference Late-breaking paper.* ACM Press.

Khor, S. (2007). Rethinking the adaptive capability of accretive evolution on hierarchically consistent problems. *IEEE Symposium on Artificial Life*: 409-416. IEEE Press.

Khor, S. (2007a). HIFF-II: A hierarchically decomposable problem with inter-level interdependency. *IEEE Symposium on Artificial Life*: 274-281. IEEE Press.

Khor, S. (2007b). Hill climbing on discrete HIFF: Exploring the role of DNA transposition in long-term artificial evolution. *Genetic and Evolutionary Computation Conference*: 277-284. ACM Press.

Khor, S. (2007c). On solving hierarchical problems with top down control. *Genetic and Evolutionary Computation Conference Companion*: 2543-2548. ACM Press.

Khor, S. (2007d). How Different Hierarchical Relationships Impact Evolution. *Australian Conference on Artificial Life*. LNAI 4828: 119 – 130. Springer.

Khor, S. (2008). Where genetic drift, crossover and mutation play nice in a free-mixing single-population genetic algorithm. *IEEE World Congress on Computational Intelligence*: 62-69. IEEE Press.

Koza, J. (1992). *Genetic Programming.* Cambridge. MIT Press, MA.

Mahfoud, S. (1995). *Niching Methods for Genetic Algorithms.* Ph.D. Dissertation, University of Illinois.

Maslov, S. and K. Sneppen (2002). Specificity and stability in topology of protein networks. *Science* 296: 910-913.

Maynard-Smith, J. and E. Szathmáry (1995). *The Major Transitions in Evolution.* W.H. Freeman & Co. Ltd.

Michod, R. E. (1999). *Darwinian Dynamics, Evolutionary Transitions in Fitness and Individuality.* Princeton University Press.

Michod, R. E. (2003). Cooperation and conflict in the evolution of complexity. Computational Synthesis: From basic building blocks to high level functionality: Papers from the *AAAI Spring Symposium*: 3-10. Technical Report SS-03-02, The AAAI Press.

Mills, R. and R. A. Watson (2007). Variable discrimination of crossover versus mutation using parameterized modular structure. *Genetic and Evolutionary Computation Conference*: 1312-1319. ACM Press.

Mitchell, M. (1998). *An Introduction to Genetic Algorithms.* The MIT Press, Cambridge, MA.

Mitchell, M., Forrest, S. and J. H. Holland (1992). The Royal Road for genetic algorithms: fitness landscapes and GA performance. 1st *European Conference on Artificial Life*: 245-254. MIT Press.

Mitchell, M., Holland, J. H. and S. Forrest (1994). When will a genetic algorithm outperform hill climbing? *Advances in Neural Information Processing Systems* 6: 51-58.

Naudts, B. (1998). *Measuring GA-Hardness.* PhD Dissertation, University of Antwerp, Belgium.

Naudts, B. and J. Naudts (1998). The effect of spin-flip symmetry on the performance of the simple GA. *Parallel Problem Solving in Nature* V: 67-76. Springer-Verlag.

Naudts, B. and L. Kallel (2000). A comparison of predictive measures of problem difficulty in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation* 4(1): 1-15.

Naudts, B., Suys, D. and A. Verschoren (1997). Epistasis as a basic concept in formal landscape analysis. 7th *International Conference on Genetic Algorithms*: 65-72. Morgan Kaufmann.

Nedelcu, A. M. and R. E. Michod (2003). Evolvability, modularity and individuality during the transition to multicellularity in Volvocalean green algae. In G. Schlosser and G. Wagner (Eds.) *Modularity in Development and Evolution.* University of Chicago Press.

Newman, M. E. J. (2003). The structure and function of complex networks. *SIAM Review* 45: 167-256.

Newman, M. E. J. (2006). Modularity and community structure in networks. *arXiv:physics/0602124v1*

Newman, M. E. J. (2006a). Power laws, Pareto distributions and Zipf's law. *arXiv:cond-mat/0412004v3.*

Payne, J. L. and M. J. Eppstein (2007) Takeover times on scale-free topologies. *Genetic and Evolutionary Computation Conference*: 308-315. ACM Press.

Prügel-Bennett, A. (2004). When a genetic algorithm outperforms hill-climbing. *Theoretical Computer Science* 320(1): 135-153.

Ravasz, E. and A.-L. Barabási (2003). Hierarchical organization in complex networks. *Physical Review E* 67: 026112

Ravasz, E., Somera, A. L., Mongru, D. A., Oltvai, Z. N. and A.-L. Barabási (2002). Hierarchical organization of modularity in metabolic networks. *Science* 297: 1551-1555.

Rechenberg, I. (1973). *Evolutionsstrategie.* Frommann-Holzboog, Stuttgart.

Reeve, H. K. and L. Keller (1999). Levels of selection: Burying the units-of-selection debate and unearthing the crucial new issues. In L. Keller (Ed.) *Levels of Selection in Evolution.* Princeton University Press.

Rosé, H., Ebeling, W. and T. Asselmeyer (1996). The density of states – a measure of the difficulty of optimization problems. *Parallel Problem Solving in Nature* IV: 208-217. Springer Berlin / Heidelberg.

Rothlauf, F. (2002). *Representations for genetic and evolutionary algorithms.* Physica-Verlag Heidelberg.

Schaffer, D. and L. Eshelman (1991). On crossover as an evolutionarily viable strategy. 4th *International Conference on Genetic Algorithms*: 61-68. Morgan Kaufmann.

Shapiro, J., Prügel-Bennett, A., and M. Rattray (1994). A statistical mechanical formulation of the dynamics of genetic algorithms. Selected Papers from *AISB Workshop on Evolutionary Computing*, LNCS 865: 17-27. Springer-Verlag, London, UK.

Sherrington, D. (1997). Landscape paradigms in physics and biology: Introduction and overview. *Physica D* 107: 117-121.

Simon, H. A. (1969). *The Sciences of the Artificial.* The MIT Press.

Spears, W. M. (1998). *Evolutionary Algorithms: The role of mutation and recombination.* PhD Dissertation, George Mason University, Virginia, USA.

Sudholt, D. (2005). Crossover is provably essential for the Ising model on trees. *Genetic and Evolutionary Computation Conference*: 1161-1167. ACM Press.

Trusina, A., Maslov, S., Minnhagen, P., and K. Sneppen (2003). Hierarchy measures in complex networks. *arXiv:cond-mat/0308339.*

Van Hoyweghen, C., Goldberg, D. E. and B. Naudts (2001). Building-block superiority, multimodality and synchronization problems. *IlliGAL Report No. 2001020*, University of Illinois at Urbana-Champaign, March.

Van Hoyweghen, C., Goldberg, D. E. and B. Naudts (2002). From TwoMax to the Ising model: Easy and hard symmetrical problems. *Genetic and Evolutionary Computation Conference*: 626-633. Morgan Kaufmann.

Van Hoyweghen, C., Naudts, B. and D. E. Goldberg (2002a). Spin-flip symmetry and synchronization. *Evolutionary Computation* 10(4): 317-344. MIT Press.

Van Nimwegen, E. and J. P Crutchfield (2000). Metastable evolutionary dynamics: Crossing fitness barriers or escaping via neutral paths? *Bulletin of Mathematical Biology* 62 (5): 799-848.

Vose, M. D. and G. E Liepins (1991). Punctuated equilibria in genetic search. *Complex Systems* 5: 31-44.

Wagner, G. P. and L. Altenberg (1996). Complex adaptations and the evolution of evolvability. *Evolution* 50(3): 967-976.

Walsh, T. (1999). Search in a small world. *International Joint Conference on Artificial Intelligence*: 1172-1177. Morgan Kaufmann.

Watson, R. A. (2001). An analysis of recombinative algorithms on a non-separable building-block problem. *Foundations of Genetic Algorithms* 6: 69-90. Morgan Kaufmann.

Watson, R. A. (2002). *Compositional Evolution: Interdisciplinary investigations in evolvability, modularity and symbiosis.* Ph.D. Dissertation, Brandeis University, Massachusetts, USA.

Watson, R. A. (2006). *Compositional Evolution: The Impact of sex, symbiosis and modularity on the gradualist framework of evolution.* The MIT Press, Cambridge MA.

Watson, R. A. and J. B. Pollack (1999). Hierarchically consistent test problems for genetic algorithms. *IEEE Congress on Evolutionary Computation*: 1406-1413. IEEE Press.

Watson, R. A. and J. B. Pollack (2005). Modular interdependency in complex dynamical systems. *Artificial Life* 11: 445-457. The MIT Press.

Watson, R. A., Hornby, G. S. and J. B. Pollack (1998). Modeling building-block interdependency. *Parallel Problem Solving in Nature*: 97-108. Springer-Verlag, London, UK.

Watson, R.A. and T. Jansen (2007). A building-block Royal Road where crossover is provably essential. *Genetic and Evolutionary Computation Conference*: 1452-1459. ACM Press.

Watts, D. J. (2004). *Six degrees: The science of a connected age.* W. W. Norton & Co. Inc.

Watts, D. J. and S. H. Strogatz (1998). Collective dynamics of 'small-world' networks. *Nature* 393: 440-442.

Whitley, D. (1993). A Genetic Algorithm Tutorial. *Tech. Rep. CS-93-103*, Dept. of Comp. Sci., Colorado State University.

Whitley, D., Mathias, K., Rana, S., and J. Dzubera (1995). Building better test functions. 6th *International Conference on Genetic Algorithms*: 239-246. Morgan Kaufmann.

Wolpert, D. H. and W. G. Macready (1997). No Free Lunch Theorems for optimization. *IEEE Transactions on Evolutionary Computation* 1(1): 67-82. IEEE Press.

Wright, S. (1967). Surfaces of selective value. *Proceedings of the National Academy of Sciences* 58: 165-179.

Appendix A

Listed below is a random sample of genotypes produced at the end of unsuccessful MMHC runs for test problems in chapter 9. The mutation rate for the MMHC is 0.0625 (Table 9.7), which means 1 to 16 consecutive genes may have their values reassigned uniformly at random to 0 or 1, at one time. The genotypes below give a concrete idea of the level of difficulty posed by each X-*p*-*q* problem to MMHC (P_m = 0.0625). Recall that for MMHC to be successful, it needs to transform the genotypes below either to an all-zeroes genotype or all-ones genotype.

The distinct arrangement of ones and zeroes in the first two IIe-2-2 genotypes below are due to top-down inter-level conflict (section 3.4.2) which is able to come into effect occasionally because higher-level *iff* constraints make the same fitness contribution as lower-level *iff* constraints in IIe-2-2.

C-2-2 (HIFF-C)

```
11111111 11111111 11111111 11111111 00000000 00000000 00000000 00000000
11111111 11111111 00000000 00000000 11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111 00000000 00000000 00000000 00000000
11111111 11111111 00000000 00000000 11111111 11111111 11111111 11111111
Phenotype: <0.53125 1 1 6 16 32 64 128> Total fitness: 248.53
```

```
11111111 11111111 11111111 11111111 11111111 11111111 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 11111111 11111111 11111111 11111111 00000000 00000000
11111111 11111111 00000000 00000000 00000000 00000000 11111111 11111111
Phenotype: <0.5 0.75 2.5 3 16 32 64 128> Total fitness: 246.75
```

II-2-2 (HIFF-II)

```
00000000 00000000 00000000 00000000 11111111 11111111 11111111 11111111
00000000 00000000 00000000 00000000 11111111 11111111 11111111 11111111
00000000 00000000 11111111 11111111 00000000 00000000 11111111 11111111
00000000 00000000 11111111 11111111 00000000 00000000 00000000 00000000
Phenotype: <0.375 1.75 1.5 5 16 32 64 128> Total fitness: 248.63
```

```
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
11111111 11111111 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 11111111 11111111 00000000 00000000
```

```
11111111 11111111 00000000 00000000 00000000 00000000 00000000 00000000
Phenotype: <0.875 1.25 2.5 5 16 32 64 128> Total fitness: 249.63
```

IIe-2-2

```
11001100 11001100 11001100 11001100 11001100 11001100 11001100 11001100
11001100 11001100 11001100 11001100 11001100 11001100 11001100 11001100
11001100 11001100 11001100 11001100 11001100 11001100 11001100 11001100
11001100 11001100 11001100 11001100 11001100 11001100 11001100 11001100
Phenotype: <128 128 128 128 128 128 0 128> Total fitness: 896

11111111 10101010 11111111 00000000 11111111 10101010 11111111 00000000
11111111 10101010 11111111 00000000 11111111 10101010 11111111 00000000
11111111 10101010 11111111 00000000 11111111 10101010 11111111 00000000
11111111 10101010 11111111 00000000 11111111 10101010 11111111 00000000
Phenotype: <128 128 128 96 32 128 128 96> Total fitness: 864

11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
Phenotype: <0 128 128 128 128 128 128 128> Total fitness: 896
```

M-2-2 (HIFF-M)

```
00000000 00000000 00000000 00000000 11111111 11111111 11111111 11111111
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
Phenotype: <1 2 3 8 16 32 64 128> Total fitness: 254

11111111 11111111 11111111 11111111 00000000 00000000 00000000 00000000
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111 00000000 00000000 00000000 00000000
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
Phenotype: <1 2 2 8 16 32 64 128> Total fitness: 253

11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
Phenotype: <0 2 4 8 16 32 64 128> Total fitness: 254
```

M1-2-2

```
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111 00001011 00000000 00000000 00000000
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
00001111 00000000 00000000 00000000 11111111 11111111 11111111 11111111
Phenotype: <1 2 4 8 16 31 64 127> Total fitness: 253

11001111 00000000 00000000 00000000 11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111 00001011 00000000 00000000 00000000
Phenotype: <1 2 4 8 16 32 63 127> Total fitness: 253

11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
```

```
11111111 11111111 11111111 11111111 00001011 00000000 00000000 00000000
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
Phenotype: <1 2 4 8 16 32 64 127> Total fitness: 254
```

M-2-4

```
00000000 00000000 00000000 00000000 00000000 00000000 11111111 11111111
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
Phenotype: <1 1 4 7 16 32 384> Total fitness: 445

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
Phenotype: <1 1 4 8 16 32 384> Total fitness: 446

11111111 11111111 11111111 11111111 00000000 00000000 00000000 00000000
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
Phenotype: <0 2 3 8 16 32 384> Total fitness: 445
```

M1-2-4

```
11111111 11111111 11111111 11111111 00000000 00000000 00000000 00000000
11111111 11111111 11111111 11111111 00000000 00000000 00000000 00000000
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
Phenotype: <1 1 2 8 16 32 384> Total fitness: 444

00001111 00000000 00000000 00000000 00000000 00000000 00000000 00000000
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
Phenotype: <1 2 4 8 16 31 384> Total fitness: 446
```

M2-4-4

```
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
11111111 11111111 00000000 00000000 00000000 00000000 11111111 11111111
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
Phenotype: <6 20 96 384> Total fitness: 506

11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
11111111 11111111 00000000 00000000 00000000 00000000 11111111 11111111
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
Phenotype: <3 20 96 384> Total fitness: 503
```

R-4-4

```
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
Phenotype: <97 156 192 384> Total fitness: 829
```

```
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
11110000 00000000 11111111 11111111 11111111 11111111 11111111 11111111
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
Phenotype: <106 156 183 384> Total fitness: 829

11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
00001111 11111111 00000000 00000000 00000000 00000000 00000000 00000000
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
Phenotype: <106 156 183 384> Total fitness: 829
```

Appendix B

This chapter (Appendix B) describes a modified version of the $\mathscr{J}$ model introduced in Khor (2007d), and reports on experiments conducted with $\mathscr{J}$ and test problems defined in chapter 3 (HIFF-C, HIFF-II and HIFF-M) to understand how structure affects the usefulness of conflict mediation to bottom-up evolution. Bottom-up evolution denotes the process of self-directed self-assembly of more complex forms from less complex forms. Conflict mediation was proposed by Richard E. Michod (1999) as a way to increase stability of collectives, and stability of collectives is regarded as fundamental to successful bottom-up evolution (Simon, 1969). Sexual reproduction is a prime example of a conflict mediation function. Structure refers to how entities interact with (are inter-dependent on) one another, and is determined by interaction rules or relationships, e.g. HIFF-C, HIFF-II and HIFF-M.

Research problem addressed

What makes self-directed evolution of complexity from simplicity possible according to Herbert A. Simon (1969) is *stability of intermediate forms*. Simon (1969) illustrates this point with his parable of the two watchmakers, Tempus and Hora, and says (p.93): "… the theory assumes no teleological mechanism. The complex forms can arise from the simple ones by purely random processes … Direction is provided to the scheme by the stability of the complex forms, once these come into existence. But this is nothing more than survival of the fittest – that is, of the stable." Richard Dawkins (2006, p.12) concurs: "Darwin's 'survival of the fittest' is really a special case of a more general law of *survival of the stable*. The universe is populated by stable things. A stable thing is a collection of atoms that is permanent enough or common enough to deserve a name …"

One difficulty with self-directed bottom-up evolution is explaining why (how), in the absence of some universal blueprint, would (could) entities (used here in the most general sense) which have evolved for their own ends, become permanent functional parts of some other entity and contribute towards the survival of the other entity, sometimes at a cost to their own ends. Understanding how composite entities or collectives – entities made up of other entities – remain in existence within a dynamic environment and in light of possible conflicts of interest between a composite entity and its parts, and amongst parts of a composite entity, is called in this chapter *the stability problem* (section B.1).

One solution to the stability problem is *conflict mediation* (section B.1). Michod (1999) proposes that conflict mediation is necessary to manage inter-level conflict within composite entities and to help collectives transition into integrated evolutionary individuals. However, because the formation of a conflict mediation function is itself subject to evolutionary pressures (which emphasize short-term survival advantages), a conflict mediation function need not be advantageous to the survival of a composite entity in the long-term.

A prime example of a conflict mediation function is sexual reproduction. Separation of the germ (reproductive) from the soma (non-reproductive) line induces cooperation amongst soma cells to manifest a phenotype (body, genotype carrier) fit enough for the co-habitant germ cells to participate in sexual reproduction. Successful sexual reproduction through germ cells enables genetic material in soma cells to occupy another phenotype.

Research objective and hypothesis investigated

This chapter seeks to understand how structure influences the usefulness of conflict mediation to stability of collectives and ultimately to the success and speed of bottom-up evolution. Structure refers to how entities interact with (are inter-dependent on) one another, and is determined by interaction rules or relationships (section B.2). Conflict mediation in this chapter refers to conflict mediation in favour of higher levels. Therefore it is expected that one of the necessary conditions for effective conflict mediation is the absence of top-down inter-level conflict (section 3.4.2) in the structure. Conflict mediation is useful if it improves stability of collectives and enhances (increases success rate and/or speeds up) bottom-up evolution.

The hypothesis is that certain types of structures or ways of connecting parts of a composite entity together, are more suited to conflict mediation. For certain types of structures, conflict mediation improves stability of composite entities and enhances (increases success rate and/or speeds up) bottom-up evolution

Overview of the model

To test this hypothesis, an evolutionary algorithm called $\mathscr{J}$ (section B.3) is designed to simulate a bottom-up self-assembly process. Starting with an initial pool of entities (randomly generated genotypes of atomic size), $\mathscr{J}$ creates interaction opportunities between randomly selected entities in a population. These interaction opportunities may result in the construction or destruction of larger entities. The outcome of an interaction between entities is determined by the self-interest of entities, that is the maximization of their own fitness. Entities are also under mutation pressure. The interaction rules between entities are governed by the fitness functions defined in chapter 3, i.e. HIFF-C, HIFF-II and HIFF-M. Section B.2 describes entities and their interactions in more detail.

In the main part of this dissertation, the size of genotypes in an evolutionary algorithm is fixed at N, the size of the HIFF problem to solve, and genotypes evolve in terms of fitness. In contrast, genotypes in this chapter evolve in terms of both fitness and size. Due to its bottom-up thrust, level fitness plays an important role in $\mathcal{J}$. Hence, this chapter examines problems from the viewpoint of levels and hierarchy. The general research objective is to understand how the bottom-up self-assembly process of evolving larger and fitter entities is affected by the way entities interact with one another, i.e. structure.

B.1 The stability problem

The stability problem has occupied much thought and debate in biology (Reeve and Keller, 1999), and has also been addressed in the form of understanding the emergence and continued presence of cooperation and altruistic behaviors. Maynard-Smith and Szathmary (1995, p.7): "Why did not natural selection, acting on entities at the lower level (replicating molecules, free-living prokaryotes, asexual protists, single cells, individual organisms), disrupt integration at the higher level (chromosomes, eukaryotic cells, sexual species, multicellular organisms, societies)?"

Reeve and Keller (1999) use an analogy from particle physics to paint a picture of the "myriad interlocking and concatenated selective forces acting simultaneously at different levels of biological organization" (p. 3). In this picture, higher level entities are composites of lower level entities, and there is a constant evolutionary force called the *centrifugal* force in their terminology that works to break apart higher level entities. The strength of this centrifugal force is in terms of the inclusive fitness of an entity that leaves its collective, i.e. is an entity more fecund within a collective or outside it? *Inclusive fitness* takes into account genetic relatedness (i.e. if I help my relatives to be more fecund, I am contributing positively to my own fecundity also since I and my relatives have genes in common.) and ecological contingencies (i.e. the costs and benefits of competitive or non-cooperative and cooperative behaviors depends on the current conditions of the environment which determine which traits are adaptive and which are not). There are two other evolutionary forces which Reeve and Keller call the *repulsive* and *attractive* forces. These two forces are at play between entities and their same level partners within a collective. The strengths or magnitudes of these two forces are also in terms of inclusive fitness, i.e. is an entity more fecund within a collective by competing or cooperating with its partners (the other entities within the same collective that it interacts with).

Both centrifugal and repulsive forces threaten the stability of a collective (an entity that does poorly within a collective would find leaving the collective more attractive). Hence, Reeve and Keller surmise that for a collective to be stable, the attractive forces must exceed the repulsive and centrifugal forces for each unit within a collective. Reeve and Keller calculated that this requirement becomes more difficult to satisfy as entities become larger. So a question relevant to collectives (composite or higher level entities) at any level is what conditions contribute to the satisfaction of this requirement (i.e. attractive force exceeds repulsive and centrifugal forces), i.e. what conditions help keep parts together as wholes.

Genetic relatedness is one such condition for biological entities that can recognize their own kin (kin can be recognized via cue-based mechanisms e.g. imprinting and phenotype matching). Other conditions that have been suggested include mutualism of some kind to induce or coerce 'voluntary' cooperation or cooperation out of necessity, nullification of disintegrative forces, reduced opportunity for conflicts and cheating, and forced cooperation through dominance, control or contingent irreversibility (Reeve and Keller, 1993 pp. 11-13; Maynard-Smith and Szathmáry, 1995 p. 9). At this juncture it is useful to note the opportunistic nature of evolution and the dynamic nature of biological and ecological systems. Inclusive fitness is a dynamic value and so also the relative strength of cooperative and non-cooperative forces within a composite entity. Hence the stability that is achieved within composite entities may be transient and context dependent. Collectives may be able to form under one set of conditions, but their stability may require constant vigilance using another set of conditions (Maynard-Smith and Szathmáry, 1995 p. 9).

Cooperation is a common theme in the conditions mentioned above. Cooperation can be altruistic or synergistic. In altruistic cooperation, the entity that cooperates is disadvantaged in some way, so the advantages of cooperation need to outweigh the disadvantages for the entity to cooperate and continue cooperating. In synergistic cooperation the entity which cooperates benefits from its cooperation without being disadvantaged. Even when cooperation is cost-free, entities may find it more beneficial to be in one collective as opposed to another, creating competition between collectives and a threat to the stability of composite entities.

However, Richard E. Michod (1999) proposes that cooperation between entities, even when the entities are related by common descent (and therefore can have non-negligible inclusive fitness), is insufficient for a collective to transition to a state of individuality that can be considered a unit of Darwinian selection. "An organism is more than a group of cooperating cells related by common descent and requires adaptations that regulate conflict within itself. Otherwise their individuality and continued evolvability is frustrated by the creation of within-organism variation and conflict between levels of selection. Conflict leads to greater individuality and harmony for the organism, through the evolution of adaptations that reduce it" (Michod, 2003).

In addition to cooperation, Richard E. Michod proposes that there needs to be conflict mediation within a collective that either favours the adaptation of higher level entities of the collective and/or reduces uncooperative adaptations in the lower level entities. Conflict mediation is a new (heritable) function that needs to emerge at the level of the collective for the transition to a new individual to be complete.

In fact, Michod views the conflict induced by cooperation as the instigator for cohesion in a collective and for integrity of a new evolutionary unit. Conflict modifiers mold the way in which levels interact to reduce conflict within collectives. As such, conflict modifiers may influence future evolutionary trajectory or evolvability of a collective or new evolutionary individual (Nedelcu and Michod, 2003). Since conflict modifiers are evolved functions and therefore may suffer from the short-sightedness of natural selection, the conflict mediation function that emerges in course of a collective's transition to individuality may hinder the new individual's evolutionary potential in the long term. An example of this evolutionarily pathological outcome is found in the evolution of multi-cellularity in the Volvox (Nedelcu and Michod, 2003).

The bottom-up evolutionary model described in sections B.2 and B.3 captures salient features of the discussion in this section. Namely: (i) entities and inter-entity interactions, (ii) synergistic and altruistic cooperation, and (iii) inter-level conflict.

B.2 Entities and Their interactions

Entities

An entity e is an individual in a $\mathcal{J}$ population. An entity comprises a genotype and an age attribute. The age of an entity (e.age) is monitored for estimating its stability. Greater longevity implies more stability. An entity's age is modified by the exchange operation in $\mathcal{J}$ (section B.3). The size of an entity (e.size) refers to the length of its genotype (e.genotype). All genotypes are binary strings of length q^n where n is an integer and $1 \leq n \leq \log_q N$, and N is the target size. The smallest and largest possible entity sizes are respectively q and N.

Entities of size q are *atomic entities*. All entities are either atomic or composite. A *composite entity* or *collective* is an entity constituted from other atomic or composite entities. Entities constituting a composite entity are referred to as *part-entities* with respect to the encompassing composite entity. A composite entity provides a *context* for its constituent entities or part-entities. The general rule in $\mathcal{J}$ is part-entities stay in their current context until there is clear incentive, in the form of increase to their own fitness, to change context. An entity which is not part of a composite entity is a *free-living entity*. Free-living entities are their own context. An atomic entity has no part-entities by definition.

Inter-entity interactions

Inter-entity interactions are interactions that happen between entities when they encounter each other. The *interaction rules* or *relationships* between entities are governed by the HIFF fitness functions defined in chapter 3. From a network perspective, the genes of entities make up the nodes of a network, and inter-dependencies between genes make up the links of a network. Inter-entity interactions are essentially the inter-dependencies between genes belonging to different entities, i.e. "the relations that hold among its parts" which Simon (1969) mentions (section 8.2.4).

𝒥 provides two types of opportunities for inter-entity interactions: join and exchange. A *join* is potentially constructive in the sense that the entities which interact may form a collective if it is beneficial for them to do so. An *exchange* is potentially destructive in the sense that one or more of the interacting entities may disintegrate if it is beneficial for one or more part-entities to leave their respective collectives for another context.

Bonus-fitness and association-fitness

Benefit to an entity from an interaction with another entity depends on the amount of fitness the inter-entity interaction produces. The amount of fitness an inter-entity interaction produces is called *bonus-fitness*. The amount of bonus-fitness is the residual fitness after subtracting the sum fitness of the interacting entities.

The amount of bonus-fitness is governed by the HIFF fitness function at play, and is determined by the genotypes of the interacting entities. In 𝒥, bonus-fitness is shared equally amongst the entities participating in an interaction. The amount of bonus-fitness an entity receives as a result of its participation in an inter-entity interaction is called *association-fitness*.

To illustrate, consider two entities *a* and *b* with respective genotypes and fitness values as given in Table B.1. A join between *a* and *b* can yield either *c (a + b)* or *d (b + a)*. The bonus-fitness produced by this interaction between *a* and *b* is 3.625 – (1.5 + 1.5) = 0.625 under HIFF-C, 3.5 – (1.5 + 1.5) = 0.5 under HIFF-II, and 3.0 – (2.0 + 1.0) = 0.0 under HIFF-M. Thus, the amount of association-fitness is 0.3125 (0.625 ÷ 2) under HIFF-C, 0.25 (0.5 ÷ 2) under HIFF-II and 0.0 (0.0 ÷ 2) under HIFF-M.

Table B.1 Fitness details for entities to illustrate joins

Entity	Genotype	HIFF-C	HIFF-II	HIFF-M
a	0001	1.5	1.5	2.0
b	1000	1.5	1.5	1.0
c	0001 1000	⟨0.625, 1.0, 2.0⟩ 3.625	⟨0.5, 1.0, 2.0⟩ 3.5	⟨0.0, 1.0, 2.0⟩ 3.0
d	1000 0001	⟨0.625, 1.0, 2.0⟩ 3.625	⟨0.5, 1.0, 2.0⟩ 3.5	⟨0.0, 1.0, 2.0⟩ 3.0

Synergistic cooperation through joins

In the previous example, only the joins under HIFF-C and HIFF-II succeed because under these relationships both *a* and *b* increase their fitness by remaining in composite entity *c* or *d*. Under HIFF-C, the increase is from 1.5 to 1.8125 (1.5 + 0.3125) and under HIFF-II, it is from 1.5 to 1.75 (1.5 + 0.125). This is an example of *synergistic cooperation* – entities *a* and *b* cooperate to form composite entity *c* or *d* to their mutual benefit and without any cost to themselves. In contrast, there is no incentive for entity *a* or *b* to form composite entity *c* or *d* under HIFF-M because bonus-fitness and accordingly association-fitness is zero for doing so.

Part-fitness

However, the increase in *a*'s and *b*'s fitness under HIFF-C and HIFF-II in the previous example is context-dependent. To distinguish between context-independent and context-dependent fitness values, the term *part-fitness* is used when referring to an entity's context-dependent fitness value. Part-fitness is fitness of a part-entity within some context, and its value is the sum of the part-entity's (context-independent) fitness and a share of the bonus-fitness generated by the composite entity (context) the part-entity belongs to. For instance, under HIFF-C, *a*'s (context-independent) fitness is 1.5 but *a*'s context-dependent fitness or part-fitness is 1.8125 (when *a*'s context is *c* or *d*). The difference between an entity's fitness and part-fitness is its association-fitness.

Synergistic cooperation through exchanges

Composite entity *c* or *d* in Table B.1 could also be produced by poaching *a* and *b* parts from other composite entities. In $\mathscr{J}$, this is made possible by the exchange operation. Let the entities 'donating' the *a* and *b* parts be entities *e* and *f* respectively (Table B.2a). A poach or exchange attempt succeeds if *a*'s and *b*'s respective part-fitness values increases in the new context. That is, if a is fitter in *c* or *d* than in *e*, and *b* is fitter in *c* or *d* than in *f*. Table B.2b summarizes the changes to *a*'s and *b*'s respective part-fitness values when their context is changed from *e* and *f* respectively to *c* (or *d*).

Table B.2a Fitness details for entities to illustrate exchange

Entity	Genotype	HIFF-C	HIFF-II	HIFF-M
a	0001	1.5	1.5	2.0
b	1000	1.5	1.5	1.0
c	0001 1000	⟨0.625, 1.0, 2.0⟩ 3.625	⟨0.5, 1.0, 2.0⟩ 3.5	⟨0.0, 1.0, 2.0⟩ 3.0
d	1000 0001	⟨0.625, 1.0, 2.0⟩ 3.625	⟨0.5, 1.0, 2.0⟩ 3.5	⟨0.0, 1.0, 2.0⟩ 3.0
e	0001 1001	⟨0.5, 1.0, 1.0 ⟩ 2.5	⟨0.75, 0.5, 1.0⟩ 2.25	⟨0.0, 1.0, 1.0⟩ 2.0
f	1111 1000	⟨0.25, 1.5, 3.0⟩ 4.75	⟨0.25, 1.5, 3.0⟩ 4.75	⟨1.0, 1.0, 3.0⟩ 5.0
g	1001	0.5	0.0	0.0
h	1111	3.0	3.0	3.0

Composite entity *e* comprises part-entities *a* and *g*. The bonus-fitness produced in *e* from interaction of its parts is 2.5 – (1.5 + 0.5) = 0.5 under HIFF-C, 2.25 – (1.5 + 0.0) = 0.75 under HIFF-II, and 2.0 – (2.0 + 0.0) = 0.0 under HIFF-M. The bonus-fitness generated in *f* from interaction between its *h* and *b* part-entities is 4.75 – (3.0 + 1.5) = 0.25 under HIFF-C, 4.75 – (3.0 + 1.5) = 0.25 under HIFF-II, and 5.0 – (3.0 + 1.0) = 1.0 under HIFF-M. Association fitness is bonus-fitness divided by the number of part-entities which is two in this example. So *a*'s association-fitness for being part of *e* under HIFF-II is 0.375, which gives it a part-fitness of 1.5 + 0.375 = 1.875.

Table B.2b Change in association-fitness for *a* and *b* as they move from composite entities *e* and *f* respectively to form composite entity *c*

	HIFF-C				HIFF-II				HIFF-M			
		e	*f*	*c*		*e*	*f*	*c*		*e*	*f*	*c*
a	1.5	+0.25	-	+0.3125	1.5	+0.375	-	+0.25	2.0	0.0	-	0.0
b	1.5	-	+0.125	+0.3125	1.5	-	+0.125	+0.25	1.0	-	+0.5	0.0

From Table B.2b, only under HIFF-C do both *a* and *b* increase their part-fitness when moving from *e* and *f* respectively, to *c* (or *d*). Under HIFF-II, *a* suffers a decrease in part-fitness of 0.125 (0.375 – 0.25) if it changes context from *e* to c (or *d*). Under HIFF-M, *b* suffers a decrease in part-fitness of 0.5 (0.5 – 0.0) if it changes context from *f* to *c*. Thus, only under HIFF-C does the 'poach' or exchange succeed. In the process of forming *c* (or *d*) under HIFF-C, both *e* and *f* disintegrate, releasing *h* and *g* to exist as free-living entities. This is another example of synergistic cooperation between entities *a* and *b* who cooperate to form composite entity *c* (or *d*) for their mutual benefit (part-fitness increase) and without any cost to themselves.

Even though entity *c* is fitter than *e* and less fit than entity *f* under all relationships (Table B.2a), the exchange succeeds under HIFF-C (i.e. *f* is destroyed to create *c*) and fails under HIFF-II and HIFF-M (i.e. *e* is not destroyed to create *c*) because only local fitness or fitness from the perspective of individual entities, i.e. part-fitness, matters.

Under HIFF-II or HIFF-M, both *e* and *f* survive the disintegration attempt, and $\mathcal{G}$ increases the age of both *e* and *f* by one to reflect this.

Inter-level conflict and altruistic cooperation through mutation

Inter-level conflict occurs when changes which are good (adaptive or fitness improving) for one level is not so for another level. Such changes are made in $\mathcal{G}$ though the mutation operation. Inter-level conflict involves transference of fitness from one level to another, and usually involves some degree of *altruistic cooperation*. An entity cooperating altruistically suffers some loss in fitness as a result of the interaction.

Bottom-up inter-level conflict

In *bottom-up inter-level conflict*, changes which are adaptive for lower levels are maladaptive for higher levels. Due to modular inter-dependency (section 3.3.2) (and existence of more than one global optima), bottom-up inter-level conflict is expected in HIFF relationships with high modularity. This is confirmed with RMHC's low success rates on HIFF problems with high modularity (Table 9.12). For a given problem size N, all three HIFF-C, HIFF-II and HIFF-M relationships have the same high level of modularity (Table 9.5) and low RMHC success rates (Table 5.2).

Bottom-up inter-level conflict threatens the existence of a composite entity because it can reduce the amount of bonus-fitness generated by the interaction between its part-entities. Put another way, bottom-up inter-level conflict can transfer fitness at higher levels which is shared by all part-entities within a composite entity, to fitness at lower levels which benefits only some part-entities, and thereby weaken the bonds that bind part-entities together in a composite entity.

For example, suppose $\mathcal{G}$ mutates entity *e* by flipping the rightmost bit. This mutation transforms entity *e* into entity *c* and part-entity *g* into *b* (Table B.3a) if it increases part-fitness of any part-entity in *e*.

Table B.3a Fitness for composite entities *e* and *c*

Entity	Genotype	HIFF-C	HIFF-II	HIFF-M
$e = a + g$	0001 1001	⟨0.5, 1.0, 1.0 ⟩ 2.5	⟨0.75, 0.5, 1.0⟩ 2.25	⟨0.0, 1.0, 1.0⟩ 2.0
$c = a + b$	0001 1000	**⟨0.625, 1.0, 2.0⟩ 3.625**	**⟨0.5, 1.0, 2.0⟩ 3.5**	**⟨0.0, 1.0, 2.0⟩ 3.0**

Table B.3b Fitness and association-fitness of part-entities (*a*, *g*, *b*) in different contexts (*e*, *c*)

Part-entity	HIFF-C			HIFF-II			HIFF-M		
		e	*c*		*e*	*c*		*e*	*c*
a 0001	1.5	+0.25	+0.3125	1.5	+0.375	+0.25	2.0	0.0	0.0
g 1001	0.5	+0.25	-	0.0	+0.375	-	0.0	0.0	-
b 1000	1.5	-	+0.3125	1.5	-	+0.25	1.0	-	0.0

Table B.3b illustrates the changes to part-fitness of *e*'s part-entities *a* and *g* if *e* mutates to *c*. Under all three relationships, this mutation increases *g*'s part-fitness by increasing *g*'s (context-independent) fitness. Under HIFF-C, the mutation also increases *g*'s association-fitness from 0.25 to 0.3125. As such, the mutation succeeds under all three relationships.

Although *c* is fitter than *e* (Table B.3a), under HIFF-II, the increase in fitness at lower levels is accompanied by a decrease in fitness at a higher level. Fitness at level three under HIFF-II drops from 0.75 to 0.5 (Table B.3a). This is an instance of bottom-up inter-level conflict. Fitness is transferred from higher to lower levels in a sense that lower levels are able to increase their fitness because higher levels give up some of their fitness.

A consequence of bottom-up inter-level conflict is a smaller association-fitness, and a smaller association-fitness lowers the barrier for part-entities to switch context when the opportunity arises. For instance, a new context only needs to provide association-fitness of more than 0.25 to either of *c*'s part-entities (i.e. *a* and *b*) under HIFF-II to lure them out of *c* and destroy *c* in the process. The context-switch barrier prior to the mutation was 0.375. This is what is meant by bottom-up inter-level conflict threatening the existence of a composite entity.

Another consequence of bottom-up inter-level conflict is involuntary altruistic cooperation. Under HIFF-II in this example, the mutation decreases association-fitness for both *a* and *g* (now *b*) with the result that *a*'s part-fitness is decreased by the mutation from 0.375 to 0.25. But *a* remains in composite entity *e* (now *c*) because the association-fitness is still positive (*a* is still better off in *c* than as a free-living entity) and there is no incentive (a better context) for *a* to leave *c*. In this case, *a* has no choice but cooperate with *g* (now *b*) and this cooperation is altruistic because *a* suffers some fitness loss while *g* transforms to *b*. Entity *g*'s increase in part-fitness under HIFF-II from 0.375 to 1.75 comes at the expense of *a*'s decrease in part-fitness from 1.875 to 1.75. This is an instance of altruistic cooperation.

Conflict mediation in favour of higher levels

From the previous example, since increasing association-fitness raises the context-switch barrier of part-entities, and increases in fitness at higher levels produces larger association-fitness values, it follows that emphasizing fitness increase at higher levels should result in more stable composite entities. The transfer of fitness from lower levels to higher levels (conflict mediation in favour of higher levels) where it is shared by all part-entities would strengthen the bonds (increase association-fitness) that bind part-entities in a composite entity. The RMHC2 algorithm where phenotypes are compared level by level from the highest level down (section 3.4.2) is an example of an evolutionary algorithm that is biased towards adaptation (fitness improvement) of higher levels, i.e. implements conflict mediation in favour of higher levels.

For example, suppose $\mathcal{G}$ mutates entity *j* by flipping the rightmost bit. This mutation transforms entity *j* into entity *k* and part-entity *s* into *t* (Table B.4a) if it increases fitness at any level λ without decreasing fitness at any level higher than level λ. This follows the RMHC2 multi-level selection scheme described in section 3.4.2. Therefore, the mutation succeeds under the HIFF-M relationship only.

Table B.4a Fitness for composite entities *j* and *k*

Entity	Genotype	HIFF-C	HIFF-II	HIFF-M
$j = r + s$	0000 0010	**⟨0.75, 1.5, 3.0⟩ 5.25**	**⟨0.75, 1.5, 3.0⟩ 5.25**	⟨1.0, 1.0, 3.0⟩ 5.0
$k = r + t$	0000 0011	⟨0.5, 1.0, 4.0⟩ 5.5	⟨0.5, 1.0, 4.0⟩ 5.5	**⟨1.0, 1.0, 4.0⟩ 6.0**

Table B.4b Fitness and association-fitness of part-entities (*r*, *s*, *t*) in different contexts (*j*, *k*)

Part-entity	HIFF-C			HIFF-II			HIFF-M		
		j	*k*		*j*	*k*		*j*	*k*
r 0000	3.0	+0.375	+0.25	3.0	+0.375	+0.25	3.0	0.5	0.5
s 0010	1.5	+0.375	-	1.5	+0.375	-	1.0	0.5	-
t 0011	2.0	-	+0.25	2.0	-	+0.25	2.0	-	0.5

Table B.4b illustrates the changes to part-fitness of *j*'s part-entities *r* and *s* if *j* mutates to *k*. Under all three relationships, this mutation increases *s*'s (context-independent) fitness, but only under HIFF-M does *s*'s association-fitness not decrease (when *s* becomes *t* as a result of the mutation). As such, since inter-level conflict is mediated in favour of higher levels (increasing or maintaining association-fitness), the mutation succeeds only under HIFF-M. The mutation fails under the HIFF-C and HIFF-II relationships even though *k* is fitter than *j* (Table B.4a). In this example, under both HIFF-C and HIFF-II, part-entity *s* is acting altruistically since if the mutation were to succeed, *s*'s part-fitness would increase from 1.875 (1.5 + 0.375) to 2.25 (2.0 + 0.25).

Part-entity *s* forgoes an increase in part-fitness to maintain association-fitness at 0.375 for itself and part-entity *r*.

If instead, a right-most bit-flip mutation was made on entity *k*, this mutation would succeed under HIFF-C and HIFF-II but not under HIFF-M. Under HIFF-C and HIFF-II, *k* would be transformed to *j* even though *j* is less fit than *k* because *j* is fitter than *k* at level three; and part-entity *t* would be transformed to *s* even though *t* would suffer a decrease in part-fitness because association-fitness is increased. This mutation illustrates fitness transfer from lower levels to higher levels, and is an instance of altruistic cooperation on the part of *t* (now *s*).

Top-down inter-level conflict

However given the blind nature of evolution, there is no guarantee that fitness transfer from lower to higher levels will lead to optimal composite entities in the long term. This is most evident when the structure or relationship has *top-down inter-level conflict*. In top-down inter-level conflict, changes which are adaptive for higher levels are maladaptive for lower levels.

A test using RMHC2 (section 3.4.2) revealed that of the three relationships studied in this chapter, only HIFF-II has the potential for top-down inter-level conflict. The RMHC2 results suggest that conflict mediation in favour of higher levels could be beneficial for evolving HIFF-C or HIFF-M entities from the bottom-up, but not for HIFF-II.

B.3 The $\mathcal{J}$ evolutionary algorithm

The $\mathcal{J}$ evolutionary algorithm (Figure B.1) is designed to simulate a bottom-up self-assembly process. Starting with an initial pool of atomic entities with randomly generated genotypes, $\mathcal{J}$ creates interaction opportunities between randomly selected entities in a population. The interaction opportunities take the form of joins or exchanges. Entities are also under mutation pressure. $\mathcal{J}$ follows the rationale that when two or more entities come in close proximity with one another, they either repel (nothing happens), are attracted to each other as wholes (a join is made) or there is partial attraction (an exchange of part-entities is made).

The interaction between entities may result in the construction or destruction of larger entities; and strengthening or weakening of bonds between part-entities. The outcome of an interaction between entities or a mutation is determined by the self-interest of entities that is the maximization of their own fitness or part-fitness, as illustrated in section B.2. The total amount of genetic material in a $\mathcal{J}$ population remains constant. In every iteration, $\mathcal{J}$ attempts a join, an exchange or a mutation operation until either an optimal entity of the target size N is formed, or the maximum number of iterations (`MaxIters`) is reached. Table B.5 lists $\mathcal{J}$'s parameters.

Table B.5 Parameters for $\mathcal{J}$

Parameter	Symbol
Atomic entity size	q
Target entity size	N
Number of entities per join or exchange	p
Maximum number of iterations	`MaxIters`
Initial population size	PS
Mutation rate	P_m
Join rate	P_j

```
𝒥
Create PS atomic entities each with a random genotype.
While number of iterations < MaxIters
    Increment number of iterations by 1
    If number of iterations is divisible by 50
        Record statistics.
        If fittest entity is optimal and of the target size, N
            Stop.
    With probability $P_J$
        Chose p distinct entities at random.
        If the p entities are all the same size and their combined size is $\leq N$
                                                    and with probability 0.5
            Attempt a join with the p entities.
        Else
            Attempt an exchange with the p entities.
    Otherwise
        Select an entity e using fitness-proportionate (roulette-wheel) selection.
        Attempt a mutate on e.
End.
```

Figure B.1 The algorithm for 𝒥

The join operation (Figure B.2) enables p randomly chosen distinct entities of the same size to form a new composite entity *e* not larger than N. A join succeeds if positive bonus-fitness is produced as explained in section B.2. A successful join reduces population size by p – 1, and exerts downward pressure on the average age of the population because a single entity with age = 0 replaces p entities with ages ≥ 0.

```
Join p entities.
Create a new entity e.
    e.genotype is the concatenation of the genotypes of all p entities.
    e.age := 0
If e is fitter than the combined fitness of all p entities (bonus-fitness is positive)
    Remove the p entities from the population.
    Add e to population.
Else
    Discard e.
```

Figure B.2 𝒥's join operation

The exchange operation (Figure B.3) enables part-entities belonging to p distinct entities chosen at random, to form a new composite entity *e*. At least one of the p distinct entities must be a composite entity, all part-entities exchanged are the same size (for simplicity), and the size of the new composite entity is the size of the largest of p the entities.

```
Exchange between p entities.
Determine smallest, the size of the smallest entity in the p entities.
Determine largest, the size of the largest entity in the p entities.
If every one of the p entities is atomic
    Stop.
Determine levels, the number of levels in the smallest entity. levels is $\log_q$ smallest.
Determine part size, the size of all part-entities, by chosing an integer $n$ at random
from within [1, levels]. Part size is $q^n$.
Use part size to split the p entities into their part-entities.
Determine part-fitness for each part-entity. Let this part-fitness be old.

From the pool of part-entities, randomly select enough part-entities to create a new
composite entity $e$ such that $e$.size = largest.
$e$.genotype is the concatenation of the genotypes of the selected part-entities.
$e$.age := 0.
Determine part-fitness for each part-entity in $e$. Let this part-fitness be new.

For each part-entity in $e$, compare new with old.
If every new > old,
    Remove the p entities from the population.
    Add $e$ and the unused part-entities (as new entities with age = 0) to population.
Else
    Increment the age of every one of the p entities by 1.
    Discard $e$.
```

Figure B.3 𝔍's exchange operation

The exchange succeeds if every part-entity that comprises *e* has a larger part-fitness value in *e* than in their respective original context as explained in section B.2. If an exchange succeeds, the new composite entity *e*, and the remaining part-entities not in *e* are added to the population. The remaining part-entities not in *e* are added to the population as new free-living entities, i.e. age = 0. Thus, successful exchanges exert downward pressure on the average entity age of the population, and may increase the size of the population. If an exchange fails, the age of the p entities are incremented by one respectively. Failed exchanges exert upward pressure on the average entity age of the population.

The mutate operation is identical to random mutation in Part I except flip-mutation (described in section 3.3.3) is used. Mutant genotypes compete with their parents for survival (a place) in the population. This ensures that all increases in entity size are the result of successful joins, and not because larger (and therefore fitter) genotypes replace smaller genotypes in the population. The outcome of the competition between parent and offspring depends on whether conflict mediation is used. By default, there is no conflict mediation. When conflict mediation in the form of RMHC2 as described in section B.1 is used, the 𝒥 runs are labeled 'R2'. 𝒥 runs labeled 'R1' do not make use of conflict mediation. Mutation does not affect the average entity age or size of the population.

B.4 Methodology

The objective of the experiments reported in section B.5 is to understand how structure affects the usefulness of conflict mediation. Conflict mediation is useful if its presence increases stability of composite entities and enhances (increases success rate and/or speeds up) bottom-up evolution. Structure refers to how entities interact with one another, and is defined by the interaction rules or relationships. The hypothesis is that certain types of structures or ways of connecting parts of a composite entity together, are more amenable to conflict mediation (in favour of higher levels).

To test this hypothesis, the 𝒥 evolutionary algorithm defined in section B.3 was configured with the values in Table B.6 and run with a different random number seed each time. For each of the three relationships: HIFF-C, HIFF-II, and HIFF-M, thirty 𝒥 runs were made without conflict mediation (these runs are denoted R1), and another thirty with conflict mediation (these runs are denoted R2).

Table B.6 Parameter values

Parameter	Value
Atom entity size (q)	2
Target entity size (N)	128
Number of entities per join or exchange (p)	2
Maximum number of iterations (MaxIters)	0.5 million
Initial population size (PS)	512
Mutation rate (P_m)	0.03125
Join rate (P_j)	0.5
Number of runs per experiment	30

The end-of-run statistics recorded are: (i) number of runs which evolved an optimal entity of size N (SUCC), and (ii) number of iterations taken to evolve an optimal entity of size N (MFPT). The during-a-run statistics collected every 50 iterations are: (i) best-fitness or fitness of the fittest entity in the current population, (ii) average entity age of the current population, (iii) cumulative number of attempted and successful joins, and (iv) cumulative number of attempted and successful exchanges.

Best-fitness during a run is collected to monitor evolutionary behavior. Average age of entities in a population is collected during a run to get a sense of the stability of entities in the population. Steady increase in average age over evolutionary time means the number of exchanges that fail exceeds the number of exchanges that succeed and the number of successful joins combined. In order words, more entities are surviving disintegration attempts by the exchange operation. Therefore, steady increase in average age over evolutionary time is associated with high levels of stability. Join and exchange events are monitored since they affect the age of entities and by inference, stability.

B.5 Results and Discussion

Table B.7 reports the results of the experiments and Figure B.4 plots the 99% confidence intervals for MFPT.

Table B.7 Results for 𝒥

	R1				R2			
	SUCC	MFPT	Median	99% CI	SUCC	MFPT	Median	99% CI
HIFF-C	30	10,397 (773)	10,500	± 364	30	14,633 (1,967)	14,850	± 925
HIFF-II	30	10,407 (963)	10,350	± 453	1	381,000	381,000	-
HIFF-M	30	11,643 (911)	11,700	± 428	30	10,778 (666)	10,800	± 313

R1 = no conflict mediation (uses RMHC selection scheme)
R2 = with conflict mediation (uses RMHC2's selection scheme)
CI = confidence interval

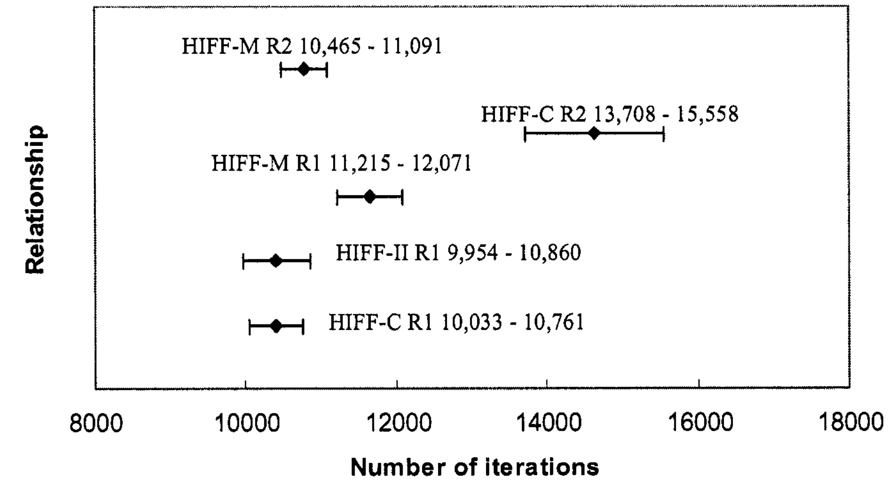


Figure B.4 MFPT and 99% confidence interval

R1

When there is no conflict mediation (R1), all 𝒥 runs were success. 𝒥 performed equally well for the HIFF-C and HIFF-II relationships, but took significantly more iterations for HIFF-M. The 99% confidence intervals for HIFF-C and HIFF-II overlap (Figure B.4), indicating no significant difference between their MFPT values. But, HIFF-

M's 99% confidence interval does not overlap those for HIFF-C and HIFF-II. HIFF-M's MFPT is significantly larger than both HIFF-C's and HIFF-II's.

R2

The results are dramatically different when there is conflict mediation in favour of higher levels (R2). Most apparent is the difference in $\mathscr{J}$'s success rate for HIFF-II. However, this is to be expected since RMHC2's conflict mediation function triggers the top-down inter-level conflict behavior inherent in the HIFF-II relationship (section 3.4.2).

$\mathscr{J}$ also fared worse in the R2 situation for the HIFF-C relationship. Significantly more iterations were used on average by $\mathscr{J}$ to evolve optimal HIFF-C entities of size 128 when conflict mediation is present (R2) than when it is not (R1). The 99% confidence interval for HIFF-C's R2 MFPT is clearly to the right of, and does not overlap the 99% confidence interval for HIFF-C's R1 MFPT (Figure B.4).

The above two results suggest that R2 does not benefit self-directed bottom-up evolution when entities interact following either the HIFF-II or the HIFF-C rules. Under HIFF-II, R2 lowered $\mathscr{J}$'s SUCC. Under HIFF-C, R2 increased $\mathscr{J}$'s MFPT. For both HIFF-C and HIFF-II, the introduction of conflict mediation reduced entity stability (Figure B.6). The HIFF-C R2 outcome is interesting because unlike HIFF-II, HIFF-C has no top-down inter-level conflict (section 3.4.2). As such, another explanation needs to be found for the difference in $\mathscr{J}$'s performance for HIFF-C. This explanation is specificity (section B.6).

Nevertheless, R2 does enhance bottom-up evolution for the HIFF-M relationship. R2 helped $\mathscr{J}$ evolve optimal HIFF-M entities of the target size more efficiently. HIFF-M's R2 MFPT is significantly smaller than HIFF-M's R1 MFPT (Figure B.4). R2 reduces $\mathscr{J}$'s

MFPT for HIFF-M to within the range of HIFF-C's R1 and HIFF-II's R1 MFPT. HIFF-M's R2 MFPT 99% confidence interval overlaps those for HIFF-C's R1 and HIFF-II's R1 (Figure B.4). The introduction of conflict mediation into $\mathscr{J}$ increased entity stability for HIFF-M (Figure B.6).

The R1 and R2 results in Table B.7 reveal a further point about the robustness of the HIFF-M relationship. Compared with HIFF-C and HIFF-II, HIFF-M's results showed least sensitivity to changes in the conflict mediation situation.

The results in Table B.7 confirm that conflict mediation can enhance bottom-up evolution, but that its usefulness is influenced by how entities relate to or interact with one another. Blindly giving higher levels priority over lower levels to adapt need not enhance bottom-up evolution, even when the relationship has no top-down inter-level conflict (e.g. HIFF-C). The absence of top-down inter-level conflict in a relationship is insufficient to ensure that conflict mediation will be useful for bottom-up evolution.

Best-so-far fitness graphs

Figure B.5 plots the best-so-far fitness graphs for five (when available) of the longest successful runs in each of the six categories in Table B.7. All three relationships show a step-like pattern when $\mathscr{J}$ is run without conflict mediation (R1). A step-like pattern in a best-so-far fitness graph is evidence of optimal entities aggregating to form larger optimal composite entities. The reason why all three relationships show step-like evolutionary behavior (Figure B.5 left) is because R1 encourages formation of optimal part-entities and $\mathscr{J}$'s exchange operation (Figure B.3) permits composite entities to split into the smallest possible part size. Consequently, it is possible to exchange optimal part-entities and form an optimal composite entity.

In contrast, using conflict mediation in favour of higher levels (R2) does not encourage the formation of optimal part-entities. R2 works to increase association-fitness, sometimes by reducing fitness of part-entities, as illustrated in section B.2. Therefore, it becomes harder to exchange optimal part-entities and form an optimal composite entity in the R2 situation. The step-like pattern detected in the HIFF-C and in the HIFF-II R1 best-so-far fitness graphs is much less evident or disappears completely when conflict mediation is introduced. But surprisingly, this is not so for the HIFF-M relationship, which is the most robust of the three relationships not only in terms of SUCC and MFPT, but also in the sense that HIFF-M's best-so-far fitness graph is least changed by the presence or absence of conflict mediation.

The HIFF-C R2 best-so-far fitness graphs suggest evolutionary behavior of a more gradual kind, which is not surprising since RMHC2 is a mutation-only evolutionary algorithm and it is capable of solving HIFF-C using only the *iff* constraints at the highest level (section 3.4.2).

Stability

Next, the effect of conflict mediation on stability is examined. Section B.1 asserted that conflict mediation is one way to increase stability of composite entities, and that stability of composite entities enhances (makes possible and/or speeds up) bottom-up evolution. To assess the stability of composite entities, Figure B.6 plots the average age of entities in a population over evolutionary time for five (when available) of the longest successful runs in each of the six categories in Table B.7.

Recall from section B.4 that steady increase in average age over evolutionary time is associated with high levels of average entity stability. The HIFF-C and HIFF-II graphs for R1 (Figure B.6 left) show much steadier increases in average age over evolutionary time than the corresponding graphs for R2 (Figure B.6 right). However, the situation is reversed for HIFF-M, though only slightly. HIFF-M's stability graph for R2 is smoother and steeper (note differences in y-axis) than for R1. Thus, HIFF-C and HIFF-II entities are more stable in the R1 than the R2 situation, and HIFF-M entities are stable in both situations, but slightly more so in the R2 situation.

Since, bottom-up evolution in $\mathcal{J}$ under both HIFF-C and HIFF-II were more successful in the R1 than the R2 situation, and bottom-up evolution in $\mathcal{J}$ under HIFF-M was more efficient in the R2 than the R1 situation (Table B.7), this confirms a direct relationship between higher levels of entity stability and enhanced bottom-up evolution as suggested in section B.1.

The graphs in Figure B.6 also confirm that mediating inter-level conflict in favour of higher levels does not automatically improve entity stability. The usefulness of conflict mediation to entity stability depends also on structure – how parts relate within entities.

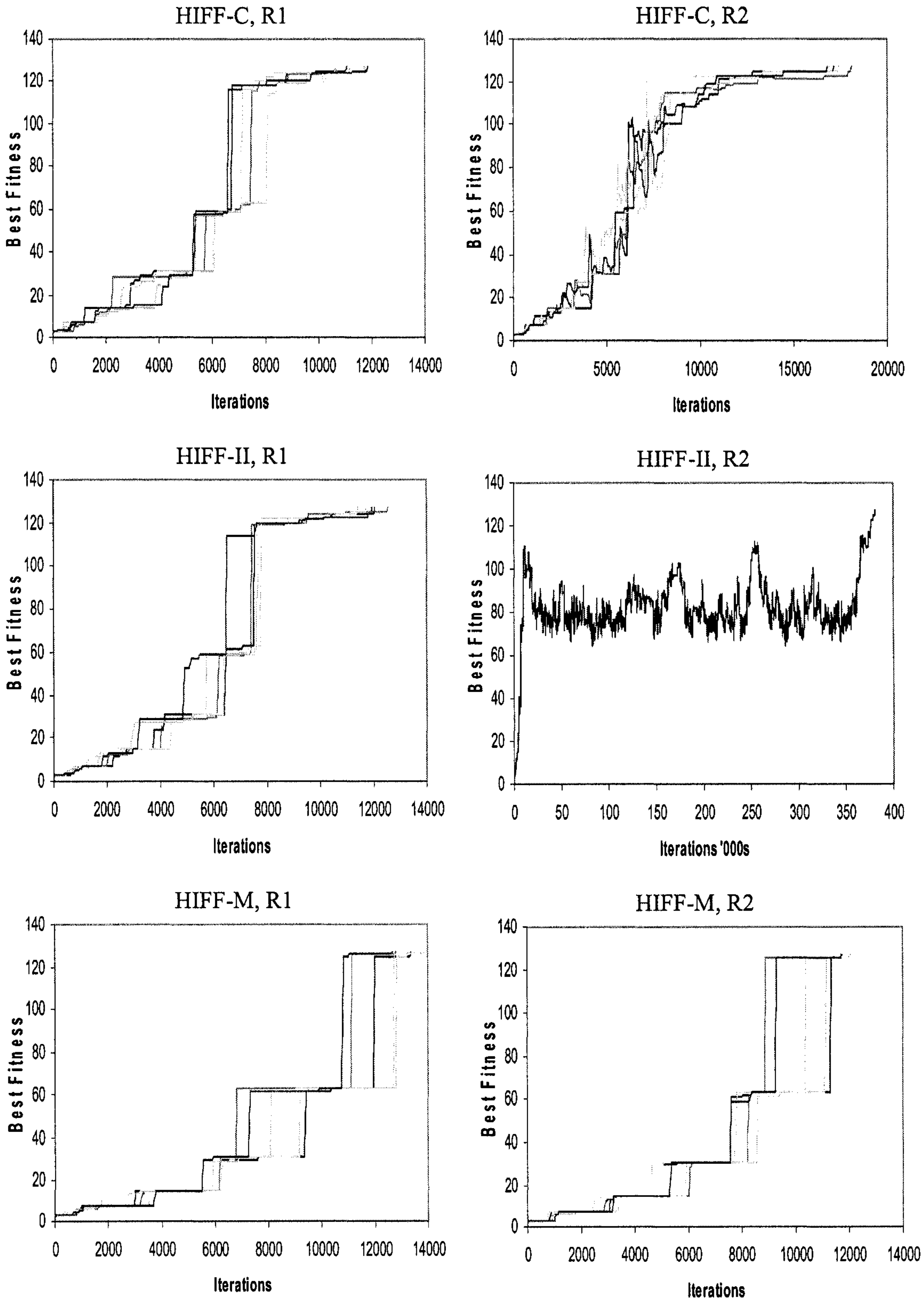


Figure B.5 Evolutionary behavior: Best fitness over time. Note the difference in scale of the x-axis and y-axis.

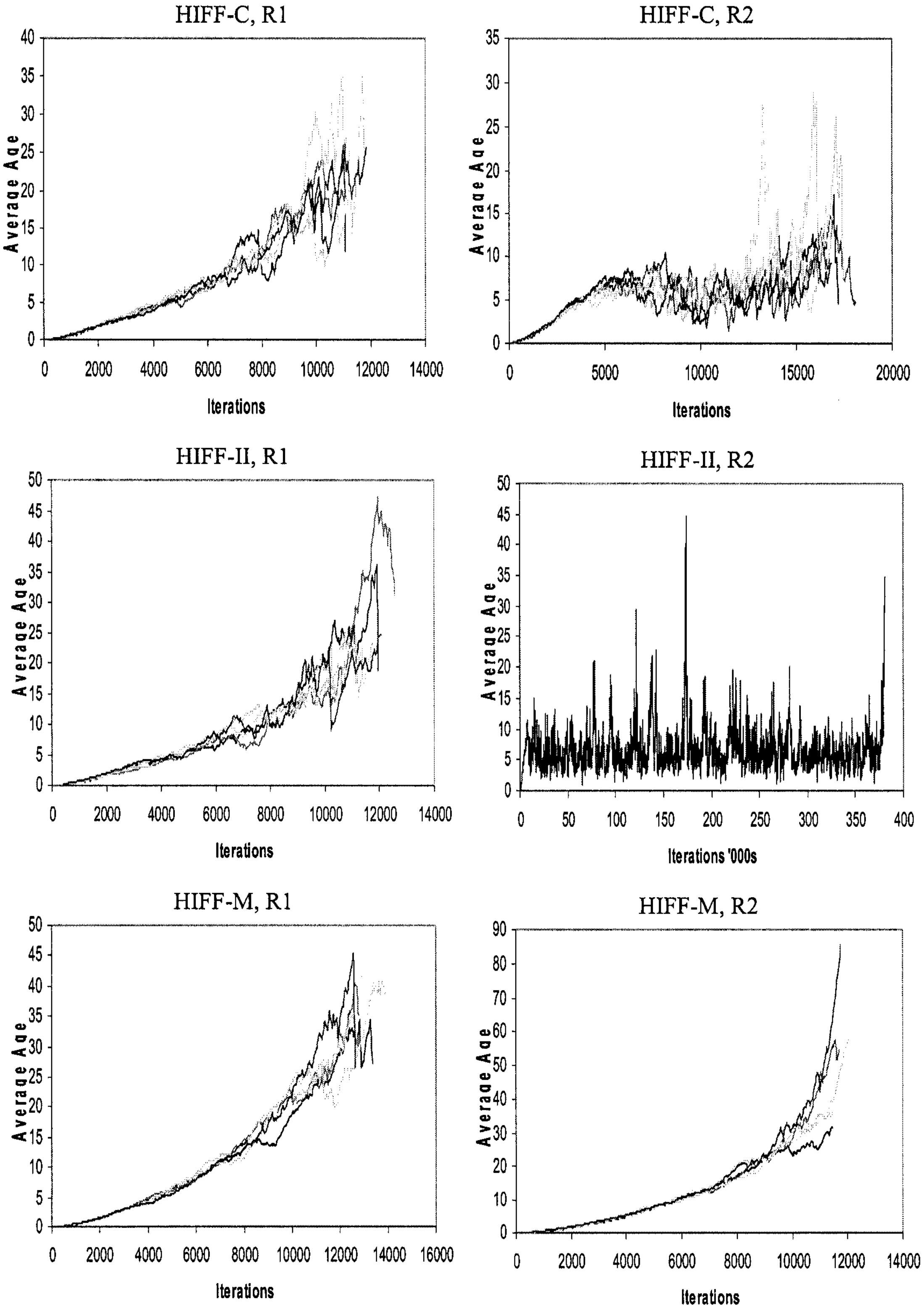


Figure B.6 Stability graphs: Average age over evolutionary time. Note the difference in scale of the x-axis and y-axis.

Summary of results

A direct relationship is confirmed between entity stability and bottom-up ($\mathcal{J}$'s) evolutionary performance (Table B.7 and Figure B.6). $\mathcal{J}$ runs which exhibited more stability (HIFF-C R1, HIFF-II R1, HIFF-M R1 and HIFF-M R2) also displayed more step-like evolutionary behavior (Figure B.5).

Conflict mediation can improve entity stability and enhance bottom-up evolution, e.g. HIFF-M, but this depends on structure. Conflict mediation is not useful when top-down inter-level conflict is present in a relationship, e.g. HIFF-II. But the mere absence of top-down inter-level conflict is insufficient to guarantee that conflict mediation will be useful, e.g. HIFF-C.

B.6 Specificity

Specificity is a property of inter-entity interactions or relationships. Relationships are more specific when there are fewer interactions between entities, or alternatively when part-entities are more, but not completely, isolated from each other.

For the HIFF relationships studied in this chapter, specificity for a given level λ is measured by counting the number of unique binary strings or genotype configurations whose level fitness is 0.0. Relationships with more genotype configurations whose level fitness is 0.0 are more specific. Table B.8 illustrates this calculation. Fitness of level λ is the sum of fitness of all blocks belonging to level λ (Part I). Thus fitness of level λ is 0.0 when block fitness of all blocks belonging to level λ is 0.0.

For HIFF-C, there are only two situations when a block's fitness is zero: either its p is 0 and its q is 1 or its p is 1 and its q is 0. p and q is the proportion of one bits in the left and right halves of a block respectively (section 3.2.1). There are $N/2^{\lambda}$ blocks at level λ (Part I). Since there are only two possible combinations for each block, the number of genotypes with zero fitness is $2^{N/2^{\lambda}}$ at level λ, and 2 at the highest level.

Table B.8 Specificity of HIFF relationships

	HIFF-C	HIFF-II	HIFF-M
Conditions for when a block *b* has zero fitness: L and R are respectively the left and right halves of *b*. *i* is an integer in [0, |*b*| -1]. L_i (R_i) is the *i*-th variable of L (R).	$\forall i\ (L_i = 0 \ \wedge R_i = 1)$ $\vee$ $\forall i\ (L_i = 1 \ \wedge R_i = 0)$	$\forall i\ L_i \neq R_i$	$L_0 \neq R_0$
Example of genotypes with zero fitness at level 2.	1100 0011 0011 0011	1100 0110 1001 0011	0111 0010 1000 0110
Number of genotypes with zero fitness at level λ.	$2^{N/2^{\lambda}}$	$2^{N/2}$	$2^{N\,(1\,-\,1/2^{\lambda})}$
Number of genotypes with zero fitness at the highest level, i.e. $\lambda = \log_2 N$.	2	$2^{N/2}$	2^{N-1}

HIFF-II (section 3.2.3) has N/2 distinct *iff* constraints per level and these *iff* constraints spread out over all genes in a genotype. A level has zero fitness when none of its *iff* constraints are satisfied. Since there are only two possible combination of values that is dissatisfies an *iff* constraint, the number of genotypes with zero fitness is independent of level and is $2^{N/2}$.

HIFF-M (section 3.2.4) has $N/2^{\lambda}$ distinct *iff* constraints per level and each *iff* constraint involves the first and middle variables of a block. This leaves $(2^{\lambda} - 2) \cdot N/2^{\lambda}$ variables free to take on any value without affecting the fitness of level λ. Therefore the number of genotypes with zero fitness at level λ is $2^{N/2^{\lambda}} \cdot 2^{(2^{\lambda} - 2)\, N/2^{\lambda}}$ which simplifies to $2^{N\,(1\,-\,1/2^{\lambda})}$. At the highest level, the number of genotypes with zero fitness is 2^{N-1}.

Therefore, for $\lambda > 1$, HIFF-C < HIFF-II < HIFF-M where '<' means is less specific than. In words, specificity is highest in the HIFF-M relationship and lowest in the HIFF-C relationship and the specificity of the HIFF-II relationship lies between that of HIFF-M and HIFF-C. The relative isolationism of parts can also be deduced from the inter-dependency matrices of the HIFF relationship. Sparser inter-dependency matrices produce more specific relationships.

Effect of specificity on entity stability

High specificity reduces the number of ways, but not necessary the amount, bonus-fitness can be generated from an inter-entity interaction. Generation of positive bonus-fitness (any amount over 0.0) is the criterion for successful joins in $\mathfrak{J}$. Therefore, high specificity has the effect of discouraging joins, or at least delaying the formation of composite entities.

High specificity also discourages exchanges by making it more difficult to generate bonus-fitness. Exchanges in $\mathfrak{J}$ succeed if the association-fitness generated in the new composite entity (context) is large enough to overcome the context-switch fitness barrier for all part-entities looking to form the new composite entity (section B.2).

Since average entity age of a population is pushed down by successful joins and successful exchanges, but pushed up by unsuccessful exchanges; and steady increase in average entity age of a population is indicative of entity stability, it follows that high specificity should be conducive for producing stable composite entities. This hypothesis is confirmed as explained next.

In Figure B.7, the number of successful joins and successful exchanges accumulated over evolutionary time is plotted for HIFF-C and HIFF-M under the R1 and the R2 situation. No successful exchanges made in any of the HIFF-M R2 runs. In both the R1 and the R2 situation, joins and exchanges were more successful under HIFF-C than under HIFF-M. This implies, since successful joins and successful exchanges reduce entity stability (depresses average entity age), that HIFF-C entities are less stable than HIFF-M entities, in general. Since HIFF-M is a more specific relationship than HIFF-C, the implication is that high specificity is conducive for producing stable composite entities.

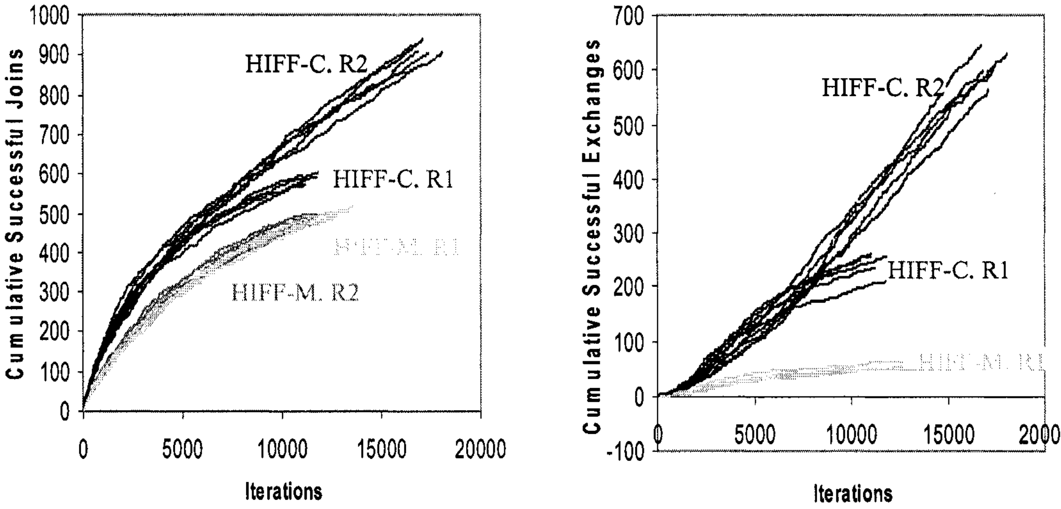


Figure B.7 Number of join and exchange successes over time

When conflict mediation is used (R2), the number of successful joins and successful exchanges increase for HIFF-C. Since successful joins and exchanges reduce entity stability (depresses average entity age), it follows that HIFF-C entities are less stable in the R2 than in the R1 situation as indicated by the HIFF-C graphs in Figure B.6. In contrast, the number of successful joins and successful exchanges did not increase for HIFF-M when conflict mediation is used. The number of successful exchanges in fact decreased. Consequently, stability of HIFF-M's entities improved in the R2 situation as indicated by the HIFF-M graphs in Figure B.6.

The differences detected between HIFF-C and HIFF-M with respect to the number of successful joins and successful exchanges, and accordingly usefulness of conflict mediation to entity stability and bottom-up evolution, are due to specificity. HIFF-M is a more specific relationship than HIFF-C. Specificity delays the formation of composite entities, giving them time to optimize themselves and thus making exchanges of optimal part-entities possible (as revealed by the step-like evolutionary behavior in Figure B.5 for HIFF-M). But once composite entities form under relationships with high specificity, they are more stable because a fitter alternative context is harder to find.

Specificity and modularity

The importance of specificity in a relationship is more evident when conflict mediation is used because prioritizing the optimization of higher levels at the expense of lower levels can override the modular structure of a relationship. $\mathcal{J}$ works well (exhibits step-like evolutionary behavior in Figure B.5) under all three relationships in the R1 situation because R1 respects the modular structure of the HIFF relationships, and high modularity encourages optimal part-entities to form. When the modular structure of a HIFF relationship is weakened, the probability of dealing with optimal part-entities in exchanges is reduced. For a relationship with low specificity like HIFF-C, this makes joins and exchanges easier to succeed, which in turn is destabilizing for composite entities.

Even though HIFF-C and HIFF-M have the same amount of modularity for problems of the same size (Table 9.5), HIFF-M's modular nature is more robust than HIFF-C's because HIFF-M is a more specific relationship than HIFF-C. High specificity is why HIFF-M's $\mathscr{J}$ results are more robust than the other two HIFF relationships. Low specificity is why R2 was not helpful for HIFF-C. R2 was not helpful for HIFF-II because of top-down inter-level conflict.

B.7 Conclusion

This chapter has identified two factors necessary for conflict mediation to be useful to bottom-up evolution: (i) absence of top-down inter-level conflict, and (ii) high specificity. These two factors depend on how genes from different genotypes (entities) interact with each other, i.e. structure. Thus, the hypothesis that certain types of structures or ways of connecting parts of a composite entity together, are more suited to conflict mediation is confirmed.

The positive relationship between specificity and stability confirmed in section B.5 is also supported by empirical studies of protein networks (Maslov and Sneppen, 2002). High specificity is related to modularity since isolation of parts implies a connectivity pattern that is stingy with links between parts, but high modularity need not imply high specificity as exemplified by the HIFF-C relationship.

Specificity is also independent of degree distribution type and network clustering. HIFF-C and HIFF-II share the same degree distribution type (section 9.2.1), but HIFF-II is more specific than HIFF-C. HIFF-II and HIFF-M have the same clustering coefficient for problems of the same size (Table 9.4), but HIFF-M is more specific than HIFF-II.

That conflict mediation is beneficial for HIFF-M is an interesting conclusion because, of the three relationships tested, HIFF-M is the most structurally similar to structures in the real-world (chapter 8), particularly in terms of degree distribution type.

A suggestion for future research is to run $\mathscr{J}$ with a wider set of inter-entity interaction rules (e.g. the X-*p*-*q* problems defined in section 9.1) to better understand the relationship between structure, conflict mediation and bottom-up evolution.