Multivariate Markov Networks for Fitness Modelling in an Estimation of Distribution Algorithm

Alexander Edward Ian Brownlee

A thesis submitted in partial fulfilment
of the requirements of
The Robert Gordon University
for the degree of Doctor of Philosophy

May 2009

Abstract

A well-known paradigm for optimisation is the evolutionary algorithm (EA). An EA maintains a population of possible solutions to a problem which converges on a global optimum using biologically-inspired selection and reproduction operators. These algorithms have been shown to perform well on a variety of hard optimisation and search problems.

A recent development in evolutionary computation is the Estimation of Distribution Algorithm (EDA) which replaces the traditional genetic reproduction operators (crossover and mutation) with the construction and sampling of a probabilistic model. While this can often represent a significant computational expense, the benefit is that the model contains explicit information about the fitness function.

This thesis expands on recent work using a Markov network to model fitness in an EDA, resulting in what we call the Markov Fitness Model (MFM). The work has explored the theoretical foundations of the MFM approach which are grounded in Walsh analysis of fitness functions. This has allowed us to demonstrate a clear relationship between the fitness model and the underlying dynamics of the problem. A key achievement is that we have been able to show how the model can be used to predict fitness and have devised a measure of fitness modelling capability called the fitness prediction correlation (FPC). We have performed a series of experiments which use the FPC to investigate the effect of population size and selection operator on the fitness modelling capability. The results and analysis of these experiments are an important addition to other work on diversity and fitness distribution within populations.

With this improved understanding of fitness modelling we have been able to extend the framework Distribution Estimation Using Markov networks (DEUM) to use a multivariate probabilistic model. We have proposed and demonstrated the performance of a number of algorithms based on this framework which lever the MFM for optimisation, which can now be added to the EA toolbox. As part of this we have investigated existing techniques for learning the structure of the MFM; a further contribution which results from this is the introduction of precision and recall as measures of structure quality.

We have also proposed a number of possible directions that future work could take.

Acknowledgments

I would like to thank the many individuals who have helped me over the years in various ways.

First I want to thank my darling wife Jay for being patient, loving, understanding and humbling, as well as putting up with many days of my working particularly when writing up. I also want to thanks my countless friends and family. Particularly my parents Alasdair and Edna Brownlee for their encouragement, support and nurturing of my thirst for learning. My brother Ian for being a best friend, helping me to understand better my priorities and for many good times. Also my sisters Heather and Allison and their families for their support and inspiration. I would also like to thank my friends at Oldmachar and St Kanes, New Deer churches, my network group at Oldmachar and friends within North East Scotland Youth for Christ for their prayers and support. I would also like to thank Tom Bell in particular for taking me into the hills to recuperate and along with my other friends for always being there.

I want to extend my deep gratitude to my principal supervisor, John McCall, for his expert advice, constant stream of ideas and suggestions and encouragement. Without John I wouldn't have discovered how fun research can be, nor reached the end of writing my thesis. Thanks also to my other supervisors Deryck Brown and Qingfu Zhang for their helpful suggestions and guidance. I have also appreciated the help of many other people within RGU's School of Computing: Sid Shakya in particular for his help while getting started in this subject area. Also to Ulises, Thierry, Malcolm, Amandine, Sudha, David, Yanghui, Francois and others in CTC for many hours of sanity-preserving distractive activities and also enlightening discussions as well as mutual encouragement. Thanks also to the other staff and students in the department who I had the privilege of knowing over the past three and a half years, and others at RGU in the various societies and committees I was involved in. I would also like to thank the teachers and staff of Mintlaw Academy and New Deer School for their encouragement in my early years.

Thanks to others in the EDA and general research community, in particular Jose Lozano and Martin Pelikan for their feedback on my work. Also particular thanks to Paul

Godley and Lloyd Oteniya of the University of Stirling for thought-provoking discussions and encouragement.

I would like to thank the Carnegie Trust for the summer research grant which ignited my interest in research and RGU for providing the grant that allowed me to work on my PhD, as well as giving me employment on numerous occasions. I am grateful also to the IEEE, ACM, BCS and SIET for providing travel grants that allowed me to attend interesting conferences that helped me greatly with the research. I would also like to thank ODS-Petrodata and Peter Barclay for giving me a job and having patience while I finished writing up.

Thanks to my viva committee, Prof. Xin Yao, Dr. Andrei Petrovski, Dr Ines Arana and Dr John McCall for providing me with helpful feedback and interesting suggestions.

Finally I would like to thank God through whom all things are possible for changing my life and making me whom I am.

Published Papers

Some parts of the work presented in this thesis have appeared in following publications, which are sorted by date of publication.

- Brownlee, A. E. I., McCall, J. A. W. & Brown, D. F. (2007). Solving the MAXSAT problem using a multivariate EDA based on Markov networks, *Proceedings of the Genetic and Evolutionary Computation COnference (GECCO 2007) (Late Breaking Papers)*, London, UK. ACM Press, New York, NY, USA, pp. 24232428.

- Brownlee, A. E. I., McCall, J. A. W., Zhang, Q. & Brown, D. (2008). Approaches to Selection and their effect on Fitness Modeling in an Estimation of Distribution Algorithm, *Proceedings of the IEEE World Congress on Computational Intelligence (CEC 2008)*, Hong Kong, China, pp. 2621-2628. IEEE Press, Piscataway, NJ, USA.

- Brownlee, A. E. I., Wu, Y., McCall, J. A. W., Godley, P. M., Cairns, D. E. & Cowie, J. (2008). Optimisation and fitness modelling of bio-control in mushroom farming using a Markov network EDA, *Proceedings of the Genetic and Evolutionary Computation COnference (GECCO 2008)*, Atlanta, Georgia, USA, pp. 465466. ACM Press, New York, NY, USA.

- Brownlee, A. E. I., McCall, J. A. W., Shakya, S. K. & Zhang, Q. (2009). Structure Learning and Optimisation in a Markov-network based Estimation of Distribution Algorithm, *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2009)*, Trondheim, Norway, pp. 447-454. IEEE Press, Piscataway, NJ, USA.

In addition, some experiments performed for the work in this thesis were used as part of the results presented in the following publications.

- Wu, Y., McCall, J., Godley, P., Brownlee, A. & Cairns, D. (2008). Bio-control in mushroom farming using a Markov network EDA, *Proceedings of the IEEE World Congress on Computational Intelligence (CEC 2008)*, Hong Kong, China, pp. 29963001. IEEE Press, Piscataway, NJ, USA.

- Brownlee, A. E. I., Pelikan, M., McCall, J. A. W. & Petrovski, A. (2008). An application of a multivariate estimation of distribution algorithm to cancer chemotherapy, *Proceedings of the Genetic and Evolutionary Computation COnference (GECCO 2008)*, Atlanta, Georgia, USA, pp. 463464. ACM Press, New York, NY, USA.

- Shakya, S. K., Brownlee, A. E. I., McCall, J. A. W., Fournier, F. & Owusu, G. (2009). A fully multivariate DEUM algorithm, *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2009)*, Trondheim, Norway, pp. 479-486. IEEE Press, Piscataway, NJ, USA.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Evolutionary computation (EC) is a long standing area of research in computer science. Inspired by concepts taken from Darwinian biological evolution, it encompasses a class of general purpose algorithms which can be applied to a large range of different problems. The framework typically includes a population of candidate solutions to a problem, a means of selection to distinguish between high and low quality solutions and a means of variation or reproduction to create new solutions using information learned from those previously seen. A number of different paradigms exist within the concept of EC - some of the most well-known are Genetic Algorithms (GAs) (Holland 1975, Goldberg 1989c), Evolution Strategies (Rechenberg 1973, Schwefel 1981, Back, Hoffmeister & Schwefel 1991) and Evolutionary Programming (Fogel 1962, Fogel 1964, Koza 1992). Further to these are a number of related techniques such as the Learnable Evolution Model (LEM) (Michalski 2000), Ant Colony Optimisation (Dorigo & Gambardella 1997), Particle Swarm Optimisation (Eberhart & Kennedy 1995), Differential Evolution (Price, Storn & Lampinen 2005), Artificial Immune Systems (De Castro & Timmis 2002, Dasgupta 1998) and the Estimation of Distribution Algorithm (Larrañaga & Lozano 2002) or EDA.

EDAs retain the selection and variation concepts common to genetic and other evolutionary algorithms, but replace the biologically inspired versions of variation with the construction and sampling of a probabilistic model. A topic which has recently become of widespread interest is fitness modelling (Schmidt & Lipson 2008, Lima, Pelikan, Sastry, Butz, Goldberg & Lobo 2006, Orriols-Puig, Bernadó-Mansilla, Sastry & Goldberg 2007, Sastry, Lima & Goldberg 2006, Pelikan & Sastry 2004, Jin 2005), which has been used in a number of different contexts.

Fitness models have been used as a surrogate fitness function for problems where evaluation of a solution is time-consuming (Lim, Jin, Ong & Sendhoff 2008, Sastry et al. 2006, Ong, Nair, Keane & Wong 2004). They are also becoming more commonly used to improve traditional genetic operators in hybrid algorithms (Lima, Sastry, Goldberg & Lobo 2005, Abboud & Schoenauer 2002, Jin & Sendhoff 2004, Rasheed & Hirsh 2000, Rasheed, Vattam & Ni 2002, Zhang & Sun 2006, Zhang, Sun & Tsang 2005). A further approach is taken by the EDA *Distribution Estimation using Markov networks* (DEUM) (Shakya 2006, Shakya, Brownlee, McCall, Fournier & Owusu 2009, Shakya & McCall 2007) in the optimisation of discrete problems. In DEUM, a probabilistic fitness model based on a Markov network (or Markov random field) is built and directly sampled to generate individuals with a high probability of being high in fitness.

In general terms fitness modelling may be viewed as a tool to improve the performance of the EC approach to problem solving. This thesis will explore the concept in detail by looking at a number of strands related to fitness modelling, including factors which affect the quality of fitness model, how the model may be used to reveal underlying information about the fitness function, and how it may be used for function optimisation. The work here focuses on the Markov fitness model as used by the DEUM framework but will be of interest to the wider EC community for a number of reasons:

- The impact of genetic operators, population size and model complexity on fitness modelling capability reinforces similar work using alternative fitness models, taking us closer to an understanding of the factors which affect general fitness models
- A new set of algorithms for optimisation which may be added to the EC toolbox
- Investigation of techniques for learning model structure and measures for the accuracy of the learned structure will be of relevance to any work on algorithms incorporating learning of the structure for an undirected graphical model

We now move on to discuss the key objectives for this research.

1.1 Research Objectives

The primary objective of this research is to explore the use of Markov network fitness modelling in improving the evolutionary approach to problem solving. This can be broken into several parts:

1. Extending the fitness model of the existing DEUM algorithm to use a multivariate structure
2. Development of fitness prediction as a measure of quality for Markov network fitness models
3. Investigation of the factors affecting fitness model quality - particularly relationships between variables, population size and selection operators
4. Manual analysis of the fitness model for different fitness functions
5. Direct sampling of the fitness model for optimisation
6. Learning the model structure from data and the effect of structure learning on fitness modelling and optimisation

1.2 Thesis Outline

The thesis is divided into nine chapters.

Chapter 2 describes the existing work which forms the background to this research. It first looks at evolutionary computation and estimation of distribution algorithms, particularly those using Markov networks. It then goes on to look at fitness modelling and different applications which make use of it. The chapter concludes with a summary of the benchmark fitness functions where are referred to throughout the thesis.

Chapter 3 investigates three aspects of the Markov network fitness model (MFM). First, it describes in detail how the MFM is constructed. It then describes the fitness prediction capability of the MFM and outlines a measure of fitness model quality based on statistical correlation. The chapter then looks in detail at the relationship of the model with the fitness function and how it may be used to understand underlying patterns in the fitness function.

Chapter 4 investigates the effect of population size on the quality of model and the tradeoff between model quality and complexity of the model structure.

Chapter 5 investigates the effect of different approaches to selection on the quality of model and draws conclusions as to the effect of selection on EC in terms of the information about fitness distributed throught the population.

Chapter 6 continues and extends existing work sampling the fitness model directly to optimise a number of benchmark fitness functions.

Chapter 7 builds on the previous chapter on optimisation by using models with a structure learned from the population rather than supplied by the user.

Chapter 8 discusses possible directions of new research which could continue from this starting point. This includes the use of Markov network fitness model in hybrid algorithms such as a guided hillclimber, possible efficiency improvements and further lines of investigation into the effective use of fitness modelling.

Chapter 9 draws together and sums up the original work contained in this thesis.

Chapter 2

Literature Review

Estimation of Distribution Algorithms (EDAs) and fitness modelling have been receiving an increasing amount of interest over the past decade. In this chapter, we explore the background to our research by presenting a summary of existing work in evolutionary computation and EDAs. This will include some definitions of terminology which we will use throughout the thesis. We will then discuss recent advances in fitness modelling and its applicability. The chapter concludes with a review of benchmark functions which we will be using in experiments described in subsequent chapters.

2.1 Evolutionary Computation

The field of evolutionary computation has seen a great deal of interest and growth over the past few decades. It brings together a number of biologically inspired ideas including Genetic Algorithms (GAs) (Holland 1975, Goldberg 1989c), Evolution Strategies (Rechenberg 1973, Schwefel 1981, Back et al. 1991) and Evolutionary Programming (Fogel 1962, Fogel 1964, Koza 1992). Further to these are a number of related techniques such as the Learnable Evolution Model (LEM) (Michalski 2000), Ant Colony Optimisation (Dorigo & Gambardella 1997), Particle Swarm Optimisation (Eberhart & Kennedy 1995), Differential Evolution (Price et al. 2005) and Artificial Immune Systems (De Castro & Timmis 2002, Dasgupta 1998). Interesting results have also been achieved by combining genetic operators with other methods such as local search in the memetic algorithm

(Moscato 1989) and, more recently, by adaptation to multiobjective problems (Deb 2001).

The principle of a genetic algorithm is based on that of Darwinian biological evolution – a population of chromosomes (each a candidate solution to the problem being solved) is maintained within which evolves a good chromosome. The classic genetic algorithm (Goldberg 1989c, Mitchell 1998, Vose 1999) has each chromosome encoding a solution as a string of bits which yields a particular fitness value when applied to the problem. It is helpful to think of a genetic algorithm as searching a landscape where the fitness translates to height and the different chromosome values determine coordinates.

Based on the terminology used in (Larrañaga, Etxeberria, Lozano & Penã 1999), a fitness function $F(X)$ for a set of discrete random variables $X_1, X_2...X_n$ is represented as $F(X) = F(X_1, X_2...X_n)$ with the fitness for a single solution or chromosome x being represented as $f(x) = f(x_1, x_2...x_n)$.

The evolution starts with a random population and moves through a sequence of *generations* by selecting a number of chromosomes or individuals (normally biased towards those with a high fitness) and generating new ones by mutating (randomly altering) and recombining parts of the old ones. Gradually the population begins to *converge*, meaning that the individual chromosomes in the population are highly similar to each other. The algorithm is normally stopped once convergence is detected, or a particular number of fitness evaluations have been completed, or once a chromosome of a certain fitness level has been found.

2.1.1 Example

This is best illustrated with an example. A commonly used benchmark problem for genetic algorithms is onemax; the fitness $F(X)$ is simply equal to the number of 1s in the individual. This is explicitly defined in (2.1).

$$F(X) = \sum_{i=1}^{n} X_i \tag{2.1}$$

The optimal solution for this problem over n variables is an individual of all 1s giving a fitness of n. For a problem size of 5 variables, a GA could run as shown in Figure 2.1.1.

1. Generate a population of individuals at random and evaluate the fitness of each.
   `11001` $F(x) = 3$
   `00111` $F(x) = 3$
   `10100` $F(x) = 2$
   `10000` $F(x) = 1$
   `00101` $F(x) = 2$

2. Select two of the fittest individuals:
   `11001` $F(x) = 3$
   `00111` $F(x) = 3$

3. Pick a random point to cross over the individuals (here we choose between the third and fourth bits) and combine them to generate two offspring:
   `11001` $\Rightarrow$ `11011`
   `00111` $\Rightarrow$ `00101`

4. Give each bit a small probability of being mutated (inverted) - here the third bit of the second individuals is the only one to be mutated (this may increase or decrease the individual's overall fitness)
   `11011`
   `00001`

5. Insert the offspring into the population in place of the two parent individuals and evaluate their fitness:
   `11011` $F(x) = 4$
   `00001` $F(x) = 1$
   `10100` $F(x) = 2$
   `10000` $F(x) = 1$
   `00101` $F(x) = 2$

Figure 2.1: An example run of a genetic algorithm on the onemax problem

We can see that each individual is a string of bits, each bit representing a variable in the problem. This is is referred to as a bit string *encoding.* The set of possible solutions to the problem is known as the *search space.*

The example shows one iteration of the genetic algorithm with a very small population; this iteration results in an improvement to the best fitness within the population (3 increases to 4) but also the replacement of a individual with fitness 3 with one of fitness 1. This is due to the stochastic nature of the algorithm. The general trend over many iterations would be the population filling with higher and higher fitness chromosomes. At the same time poorer ones will also be created and then have a high chance of being discarded by the selection operator. Much work has been done to find selection, recombination and mutation operators which give better performance for given problems.

2.2 Estimation of Distribution Algorithms

The evolutionary concept has been adapted and optimised in many ways from this basic model. One direction which has grown in recognition over the past decade has been to remove the stochastic recombination and mutation operations for generating a population. Instead, a probabilistic model of the fitter members of the population is built and sampled to generate a new population. This technique has become known as Estimation of Distribution Algorithms (EDAs) or Probabilistic Model Building Genetic Algorithms (PMBGAs). In this section we present a survey of work in this area.

2.2.1 Terminology

EDAs are typically categorised by the complexity of their probabilistic model structure (Pelikan, Goldberg & Lobo 1999). Probabilistic Graphical Models (PGMs) represent a powerful means of describing the probabilistic models employed by EDAs and it is worth defining some terminology related to EDAs and PGMs before moving on. A PGM can be separated into two components; *structure* and *parameters*.

The *structure* represents the interactions between variables in the problems. Figure 2.2 illustrates some simple PGMs for a problem with four variables. The variables are represented by nodes on the graph and connected by edges which represent direct interactions between them. The rightmost graph in the figure also shows three-way interactions

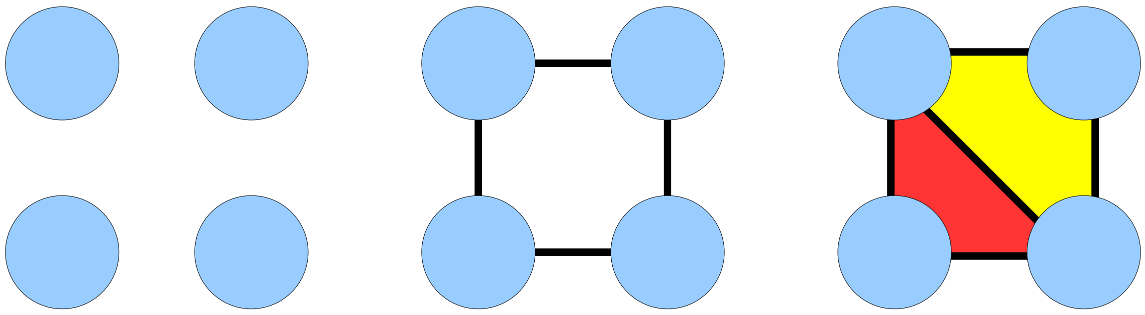


Figure 2.2: EDAs are typically classed as univariate, bivariate or multivariate

in colour. A *clique* is a set of mutually connected nodes on the graph, which represents a set of variables which all directly interact with each other. A *maximal clique* is any clique whose component variables do not also form part of a larger clique. Clique sizes of one and zero are permitted; these represent single independent variables and parameters not associated with a variable respectively. The *parameters* are a set of *potential functions* which are associated with cliques on the graph. Generally these represent the importance of particular interactions or variables and consist of a set of conditional or marginal probabilities.

EDAs are typically grouped by the order of the maximum clique sizes which occurs in their graphical model (Larrañaga & Lozano 2002, Pelikan, Goldberg & Lobo 1999). These groupings are represented by the three graphs in Figure 2.2 and are univariate (no interactions), bivariate (interactions between variable pairs only) and multivariate (higher order interactions). PGMs use two broad groups of graphs; those illustrated are undirected graphs specify interaction or mutual dependency between variables. These are known are Markov Random Fields or Markov Networks. Directed acyclic graphs, known as Bayesian Networks, specify directed dependencies between variables. In Markov networks variables depend only on their immediate neighbours on the graph, a property known as *Markovianity.* This is in contrast to directed models such as Bayesian networks where one variable depends on a hierarchy of ancestors (this property requires the model to be acyclic to allow sampling). In the context of Markov networks the concept of a clique becomes important because a variable's neighbours are those with which it shares membership of cliques.

Rather than the conventional univariate, bivariate and multivariate classes, we propose that an alternative means of classifying the algorithms is into those using univariate, directed and undirected probabilistic models. We will adopt this latter approach as the work in this thesis focuses on undirected approaches to probabilistic modelling and it is logical to group other algorithms using an undirected model together.

2.2.2 Univariate EDAs

2.2.2.1 Population Based Incremental Learning (PBIL)

Population Based Incremental Learning (Baluja & Caruana 1995) computes and samples marginal probabilities to move from one generation to the next. The marginal probability of each variable taking the value 1 is updated (or *incremented*) each generation. These probabilities are then sampled to generate the next population. The set of probabilities is called a *probability vector*. PBIL is defined in Algorithm 2.1.

Algorithm 2.1 Population Based Incremental Learning (PBIL)

```
1: Set all values in probability vector (PV) to 0.5
2: while Stopping criteria is not met do
3:     Sample PV to generate population $p$
4:     Evaluate $p$
5:     Select a subset $\sigma$ of $p$
6:     for each variable $X_i$ do
7:         Calculate marginal probability $p(X_i)$ within $\sigma$
8:     end for
9:     Increment each $p_i$ in the probability vector according to (2.2)
10: end while
```

Each probability p_i in the probability vector is updated according to (2.2), where α is a constant known as the learning rate, $p'(X_i)$ is the new marginal probability of variable X_i and $\rho(X_i)$ is the marginal probability for X_i computed from the set of sampled individuals.

$$p'(X_i) = (1-\alpha)p(X_i) + \alpha\rho(X_i) \tag{2.2}$$

Convergence occurs when the probabilities in the PV reach 0 or 1, after which there will be no variation in the generated individuals. The combination of probability vector and learning rate allows for control over the rate of convergence. Calculation of marginal probabilities is a lightweight task and consequently PBIL has little algorithm overhead. However, it can be easily trapped by deceptive problems (de Bonet, Isbell Jr. & Viola 1997) owing to its lack of consideration for interactions between variables.

2.2.2.2 Univariate Marginal Distribution Algorithm (UMDA)

The Univariate Marginal Distribution Algorithm was proposed in (Mühlenbein & Paaß 1996). It differs from PBIL in having no probability vector or learning rate parameter, simply generating the individuals using the marginal probabilities from the preceding generation. It has been observed that UMDA can be viewed as a specific instance of PBIL, with the learning rate set to 1.0. (Larrañaga & Lozano 2002).

2.2.2.3 Compact Genetic Algorithm (cGA)

In developing the compact Genetic Algorithm (Harik, Lobo & Goldberg 1997), Harik observed that a genetic algorithm's population can itself be viewed as a probabilistic model. Similar to PBIL, a probability vector is maintained but rather than generating a whole population only two individuals are generated from it. These are then subjected to a tournament (similar to tournament selection (Goldberg & Deb 1991)). This is one of the most space-efficient evolutionary algorithms, with no requirement for storing a large population. It has been shown to compare favourably with a simple genetic algorithm in the paper referenced above.

2.2.2.4 Distribution Estimation Using Markov Random Fields (DEUM)

Algorithm 2.2 DEUM$_{\text{pv}}$

```
1: Set all values in probability vector (PV) to 0.5
2: while stopping criteria is not met do
3:    Sample PV to generate population $p$
4:    Evaluate $p$
5:    Select a subset $\sigma$ of $p$
6:    Form an equation for each individual in $\sigma$ as in (2.3)
7:    Solve the resulting system of equations to determine each $\alpha_i$
8:    Use zero temperature Metropolis algorithm to sample distribution and generate an
      individual
9:    Update PV in the same way as for PBIL
10: end while
```

One further EDA which has recently been developed is DEUM which will be looked at in more detail as it forms the foundation this research. DEUM uses a Markov Random Field or Markov network (Li 1995) to model the energy distribution across the fitness function, rather than marginal probabilities of selected individuals. Two major univariate variants of DEUM have been publicised: DEUM$_{\mathrm{pv}}$ (Shakya, McCall & Brown 2004b) uses a probability vector similar to PBIL and DEUM$_{\mathrm{d}}$ (Shakya 2006, Shakya & McCall 2007, Shakya, McCall & Brown 2005b, Shakya, McCall & Brown 2005c) uses a Gibbs sampler to directly sample the MRF and generate a new population.

Both approaches use a Markov network to model the distribution of energy across the set of variables in the problem (energy has a negative log relationship with fitness). The set of individuals selected from the population in a generation are used to build a set of equations (*energy functions*) relating fitness to the variables, as in (2.3).

$$\sum_{i=0}^{n} \alpha_i x_i = -\ln(f(x)) \tag{2.3}$$

To maintain mathematical symmetry in the energy function variable values 0 and 1 are substituted by -1 and +1. The system of equations is solved using singular value decomposition (Lucey 1984) to find the unknown α values which specify the distribution. DEUM$_{\mathrm{pv}}$ uses a zero-temperature Metropolis method to sample the distribution and generate an individual which the probability vector is incremented towards in the same way as PBIL. The sampler is biased to minimise the energy of the generated individual; this means that the individual has an increased probability of being high in fitness compared to a randomly generated one. In this respect the algorithm moves the probability vector and hence the population towards areas of the search space that have a high probability of containing the global optimum. DEUM$_{\mathrm{pv}}$ runs as shown in Algorithm 2.2.

DEUM$_{\mathrm{d}}$ runs in a similar fashion but removes the probability vector and replaces steps 8 - 9 by running a Gibbs sampler on the model represented by the α_i values to generate a new population. This runs as specified in Algorithm 2.3.

The Gibbs sampler computes marginal probabilities for each variable using (2.4), where T is a temperature constant which can be used to vary the sampler's convergence. W_i is an energy function for the set of cliques which contain the variable x_i - we will return to discuss this in more detail in Chapter 3. Again the goal of the Gibbs sampler is to generate an individual which has a high probability of being low in energy, which is equivalent to

Algorithm 2.3 Gibbs Sampler

```
1: for each individual $x^o$ in the population do
2:   Set $g = 0$ and set initial value for temperature $T$
3:   repeat
4:     Set $x^{tmp} = x^o$
5:     Pick a variable $x_i^o$ (either at random or according to a specified scheme)
6:     Compute marginal probability distribution for $x_i^o$ according to (2.4)
7:     Sample distribution to obtain new value for $x_i^o$
8:     Increase $g$ by 1
9:   until $x^{tmp} = x^o$ or $g = limit$
10: end for
```

being high in fitness.

$$p(x_i = 1) = \frac{1}{1 + e^{2W_i/T}} \tag{2.4}$$

2.2.3 Learning Model Structure

2.2.3.1 General Approaches

In Section 2.2.1 we discussed the two aspects of probabilistic graphical models: structure and parameters. The univariate EDAs in the previous section, have a fixed structure with no interactions so only learn the model parameters. As we move on to EDAs which consider interactions between variables, a brief discussion of structure learning is required.

There are two approaches to finding the structure of a graphical model. The first, known as *search+score*, is similar to that used in the BOA (Pelikan, Goldberg & Cantú-Paz 1999). A scoring metric provides a means of evaluating the quality of the structure and two such metrics have been used in EDAs. Bayesian metrics (Cooper & Herskovits 1991, Heckerman, Geiger & Chickering 1995) compute a marginal likelihood of the structure with respect to the data (the set of selected solutions). Minimum Description Length (Rissanen 1978) metrics give a higher score to models which take a smaller space to represent while maximising the number of regularities in the data (selected solutions) which they include. In addition to the scoring metric, an algorithm is required to search through the space of possible structures. This problem is itself NP-hard (Chickering, Geiger & Heckerman 1994) and consequently many EDAs use a simple approach with low overhead such as a greedy algorithm to find a suitable Bayesian network structure.

Rather than testing complete structures, the second method of structure learning *independence tests* looks at individual relationships in turn. Pairs of variables are subjected to a statistical *independence test* to determine whether they are independent of each other. There are many methods which may be used to determine such independence; examples of this are the joint entropy of the pair as used in MIMIC (de Bonet et al. 1997) and the Chi-square independence test (Marascuilo & McSweeney 1977), as used in BMDA (Pelikan & Mühlenbein 1999) and MN-FDA (Santana 2003a). Each starts with a fully connected graphical model and removes edges where the independence score is below some threshold. We now describe these two independence tests as well as a related structure learning algorithm.

2.2.3.2 Information Entropy

$$H(X_i) = \sum_{x_i=1}^{n} p(x_i) log_b p(x_i) \tag{2.5}$$

$$H(X_i, X_j) = \sum_{x_i=1, x_j=1}^{n} p(x_{i,j}) log_b p(x_{i,j}) \tag{2.6}$$

Shannon's Information Entropy (Shannon 1948) is one measure used in structure learning, and was first used for EDAs by MIMIC (de Bonet et al. 1997), discussed further in Section 2.2.4.1. Entropy is essentially a measure of randomness; the marginal entropy $H(X)$ of a discrete random variable X_i of base b which can take one of n values is given in (2.5). The joint entropy of a pair of variables X_i and X_j is given in (2.6). The theory used here is that if a sample is taken of high fitness individuals, and within that sample one variable can be predicted accurately given another variable (e.g. in a binary string they are generally always equal or always opposite), there is some kind of relationship between the two. Thus if a pair of variables have a low joint entropy among the selected part of the population, there is likely to be an interaction between them. Marginal entropy can be used to choose a variable to be treated as independent. Further discussion of probability can be found in (Feller 1957). The above description of entropy should be sufficient for the summaries presented here; other means of structure building more specific to each algorithm are described in detail in the papers referenced with each algorithm's description.

2.2.3.3 Chi-square Test

The Chi-square (χ^2) test is a comparison between the joint distribution of a pair of variables and the product of their marginal distributions. If these are equal then the variables are said to be independent; though normally some threshold is used to reduce the effects of noise. A typical threshold would be 3.84, where the variables are said to be 95% independent (Boslaugh & Watters 2008). For each possible interaction, the test between a pair of variables X_i and X_j is calculated over all possible values x_i and x_j as in (2.7).

$$\chi^2_{i,j} = \sum_{x_i, x_j} \frac{(p(x_i, x_j) - p(x_i)p(x_j))^2}{p(x_i)p(x_j)} \tag{2.7}$$

Chi-square may also be extended to test higher order interactions by testing for conditional probabilities. In (Santana 2003a) the threshold for the Chi-square test was kept low to capture a large number of possible interactions. The resulting network was then refined by setting a maximum number of interactions involving each variable and where a variable exceeded this limit removing the interactions with the lowest Chi-square score.

2.2.3.4 Linkage Detection Algorithm

(Heckendorn & Wright 2004) describes a technique known as the Linkage Detection algorithm (LDA) which also fits into the category of independence tests. For bivariate interactions, the change in fitness of a chromosome is measured while flipping two bits separately and together. If the sum of the fitness changes when flipping separately is different to the change when flipping together, there deemed to be a relationship between the variables. This method can be expanded to higher levels of interaction but its complexity grows rapidly with the level of interaction. In fact, the number of possible interactions of order k in a set of n variables is given in (2.8).

$$\begin{pmatrix} n \\ k \end{pmatrix} = \frac{n!}{k!(n-k)!} \tag{2.8}$$

The learned network is likely to be noisy and potentially very dense as there is no threshold - a tiny difference between the fitness changes will register as an interaction. This can be mitigated by the introduction of such a threshold.

An alternative to directly learning high-order interactions from the data is to use an independence test algorithm to find all bivariate interactions, then use a deterministic clique-finding algorithm such as Bron and Kerbosch (Bron & Kerbosch 1973) which uses graph theory to locate maximal cliques.

2.2.4 EDAs using a Directed Model

2.2.4.1 Mutual Information Maximization for Input Clustering (MIMIC)

MIMIC (de Bonet et al. 1997) uses a chain model for the distribution structure where one variable is independent. Each remaining variable is then conditionally dependent on another according to an ordering learned by the algorithm. A greedy heuristic employing Shannon's Information Entropy (Shannon 1948) is used to build the chain for this ordering with the aim of minimising the Kullback-Leibler divergence (Kullback 1987) between the model and the true distribution; this is incorporated into the workflow of MIMIC given in Algorithm 2.4.

The chain model which MIMIC uses can be written as:

$$p(x) = p(x_{\pi_1}|x_{\pi_2})p(x_{\pi_2}|x_{\pi_3})\ldots p(x_{\pi_{n-2}}|x_{\pi_{n-1}})p(x_{\pi_n})$$

Where $\{\pi_1, \pi_2, \ldots, \pi_n\}$is a permutation of the numbers $\{1, 2, \ldots, n\}$, effectively a chain of the variables. Here:

1. x_{π_n} is independent

2. $x_{\pi_j}$is dependent on $x_{\pi_{j+1}}$ for $1 \leq j < n$

Algorithm 2.4 MIMIC

```
1:  Generate random initial population $p$
2:  while stopping criteria is not met do
3:    Select a subset $\sigma$ of $p$, all individual with a higher than median fitness
4:    Determine entropy of all variables using $\sigma$
5:    Add variable with lowest marginal entropy to start of chain (the independent variable)
6:    while chain does not contain all variables do
7:      Determine pairwise conditional entropy, given the last variable to be added to chain, for remaining variables
8:      Add the variable with the lowest conditional entropy to chain
9:    end while
10:   Generate new population to replace $p$:
11:   for all individuals do
12:     Sample marginal probability from $\sigma$ for independent variable
13:     Sample conditional probabilities of subsequent variables in the chain
14:   end for
15: end while
```

MIMIC was an important development in the field of EDAs, allowing interactions between variables to be considered for the first time. Unfortunately, while it performs well on test problems, real world problems rarely have the chain structure which MIMIC requires for optimal performance (Baluja, Davies 1997). However, this does not negate its importance in the broader context of EDAs; in particular, the approach to discovering and using these interactions has been extended and adapted in other algorithms such as COMIT, which we now move on to.

2.2.4.2 Combining Optimizers with Mutual Information Trees (COMIT)

COMIT (Baluja & Davies 1997a, Baluja & Davies 1997b) is a development of MIMIC using a tree-structured Baysian network rather than a chain. A maximum weight spanning tree (MWST) algorithm (Chow & Liu 1968) is used to find the tree, again using entropy values to find variable interactions. An interesting difference is that COMIT also uses a hillclimber to optimise individuals it generates. It runs as shown in Algorithm 2.5. COMIT has been applied successfully to a number of different problems such as scheduling and optimisation and its approach to combining two simpler algorithms is of particular relevance to this research.

Algorithm 2.5 COMIT

```
1: Generate random initial population $p$
2: while stopping criteria is not met do
3:    Select a subset $\sigma$ of $p$
4:    Build tree using MWST algorithm
5:    Compute conditional probabilities within tree
6:    Sample $M$ individuals from the distribution determined by tree
7:    Use best of the new individuals as start points for a fast search such as a hillclimber
8:    The best individuals obtained overall are substituted back into the population
9: end while
```

2.2.4.3 The Bivariate Marginal Distribution Algorithm (BMDA)

BMDA (Pelikan & Mühlenbein 1999) is a development of UMDA described previously. It uses a forest model – a set of mutually independent trees – and uses Chi-square statistics to find interactions. It runs as shown in Algorithm 2.6. The use of a structure which allows many independent variables ("root" nodes) means that BMDA can effectively model both univariate and bivariate problems; MIMIC and COMIT assume that only one of the variables is independent which can inhibit performance on problems with many independent variables. Of particular interest here is the method used for constructing the tree using a statistical dependency test; this will be revisited in Chapter 7.

Algorithm 2.6 BMDA

```
1: Generate random initial population $p$
2: while stopping criteria is not met do
3:    Select a subset $\sigma$ of $p$
4:    Create a set $S$ of all variables, and an empty graph $G$
5:    Remove a variable arbitrarily from $S$ and add to $G$ as a root node
6:    while $S$ still contains variables do
7:       while Dependencies over a previously defined threshold exist do
8:          Remove variable with the highest dependency between itself and the variables
            in $S$ add it to $G$
9:       end while
10:      Remove a variable arbitrarily from $S$ and add to $G$ as a further root node
11:   end while
12:   Use $\sigma$ to calculate marginal probabilities for root nodes variable, and conditional
      probabilities for variables dependent on them
13:   Sample these probabilities, generating individuals which are inserted back into the
      population
14: end while
```

2.2.4.4 The Bayesian Optimization Algorithm (BOA)

BOA (Pelikan, Goldberg & Cantú-Paz 1999) uses a Bayesian network to represent the problem structure. BOA employs the Bayesian-Dirichlet metric (though other metrics may be used) to measure the quality of the Bayesian network and a greedy algorithm to search the space of possible networks. At each generation, the marginal and conditional probabilities of a set of selected individuals are stored in the Bayesian network and used to generate new individuals. BOA is shown in Algorithm 2.7.

Algorithm 2.7 BOA

```
1: Generate random initial population $p$
2: while stopping criteria is not met do
3:     Select a subset $\sigma$ of $p$
4:     Estimate Bayesian Network from $\sigma$
5:     Sample Bayesian Network to generate new population and replace $p$
6: end while
```

BOA has also been extended in numerous ways; perhaps the most notable is Hierarchical BOA (Pelikan & Goldberg 2000), which has been shown to work well on a number of traditionally "hard" problems including MaxSAT and the Ising problem (Pelikan & Goldberg 2003). The Estimation of Bayesian Networks Algorithm or EBNA (Etxeberria & Larrañaga 1999) is another EDA which uses Bayesian networks.

2.2.5 EDAs using an Undirected Model

2.2.5.1 The Extended Compact Genetic Algorithm (EcGA)

EcGA (Harik 1999) employs a different approach to most other multivariate EDAs; it assumes that interdependent variables can be combined into groups which are independent of each other. BMDA also takes this approach but with a directed structure with multiple independent trees. In EcGA a greedy algorithm performs the grouping, starting with a group for each variable and pairing together groups if it will reduce the overall complexity of the model. Complexity in this case is defined in terms of entropy and minimum description length of the model; this is explained in more detail in Harik's paper and in (Larrañaga & Lozano 2002). Once arranged into groups, the algorithm treats each group as an independent variable and runs in a similar fashion to the original cGA.

A disadvantage of this approach is that, in most real-world problems, most if not all variables are part of a connected graph of interactions (ie not in distinct groups). The balance between model accuracy and computational complexity is an area of extensive research in itself. The benefit of the reduced complexity of the ECGA's structure building algorithm compared to that of other multivariate EDAs is likely to be highly problem-dependent.

2.2.5.2 The Factorised Distribution Algorithm (FDA)

FDA (Mühlenbein, Mahnig & Rodriguez 1999) estimates a Boltzmann distribution. The basic FDA requires that the graphical structure in the form of an additive decomposition of the problem function (ADF) be supplied – it does not learn the structure itself. This is a set of potential functions which define the cliques in the probabilistic model, a Markov network, closely related to the clique potential functions which we discuss in Chapter 3. The distribution specified by the Markov network is factorised into a junction tree; a product of marginal and conditional probabilities in which cliques are divided into residuals and separators. (Residual variables are dependent on separator variables) This junction tree is sampled to generate new individuals. The Markov network structure must satisfy what is called the running intersection property (Lauritzen 1996) to allow this factorisation. FDA is summarised in Algorithm 2.8.

Algorithm 2.8 FDA

```
1: Using given ADF calculate junction tree, comprising $b_i$ (residuals) and $C_i$ (separators)
2: Generate initial random population $p$ of size $M$
3: while stopping criteria is not met do
4:    Select subset $\sigma$ of $p$ using Boltzmann selection
5:    Estimate the conditional probabilities $p(x_{b_i}|x_{c_i})$ from $\sigma$
6:    Generate $M$ new individuals according to the distribution
7: end while
```

FDA has proved a versatile algorithm and a good foundation for many others building on its framework. The more recent Learning FDA (Mühlenbein et al. 1999) does not require the structure of the problem to be supplied to the algorithm. Two algorithms which remove the requirement for the graphical model to satisfy the running intersection property are the Mixture of Trees FDA (Santana, Ochoa & Soto 2001) and the Markov Network FDA (Santana 2003a) which we now move on to look at in more detail.

2.2.5.3 Markov Network Factorized Distribution Algorithm

(Santana 2003a) describes the Markov Network Factorized Distribution Algorithm. MN-FDA learns an independence graph from data using statistical independence tests. This graph is then refined to limit its density by removing edges that connect nodes having more than a specified number of incident edges. The Bron and Kerbosch algorithm (Bron & Kerbosch 1973) is used to find the maximal cliques on the dependency graph; each of these are given a weight related to the original dependency test values.

The maximal cliques and associated weights are passed to a deterministic algorithm which constructs an ordered junction graph. The junction graph is a graph where each node corresponds to a maximal clique in the dependency graph, and nodes are connected by edges if they share a common node in the dependency graph. Marginal probabilities for each clique in the junction graph are computed from the population and sampled to generate a population for the next generation.

Like Learning FDA, although the underlying probabilistic model is undirected (a Markov network) this is converted to a directed graph (in this case the junction graph) for parameter sampling. The key difference is that the junction graph in MN-FDA allows a larger range of structures to be represented than is the case for that used by Learning FDA, so the model can more closely fit the true probability distribution for the problem.

2.2.5.4 Markov Network Estimation of Distribution Algorithm

MN-EDA (Santana 2005) further extends the work on MN-FDA by replacing the junction graph with a Kikuchi approximation of the distribution. This is itself an approximation to a Gibbs random field and in this respect is more closely related to the work described in this thesis. The Hammersley-Clifford theorem describes the relationship between the Gibbs random field and the Markov network: if p is a Gibbs field defined on an independence graph G, p is Markovian with respect to the graph.

The Kikuchi approximation is taken from statistical mechanics where is was developed as a means to approximate the free energy in many-body systems. The energy is approximated as a function of a set of marginals; MN-EDA uses the set of maximal cliques on the dependency graph to determine these marginals. The Kikuchi approximation to the distribution satisfies the local Markov property which simplifies the sampling process. New individuals are generated from the distribution by use of a Gibbs sampler and reinserted into the population.

This approach allows for the representation of all interactions learned at the dependency test stage, in contrast with the earlier FDA variants which were unable to do this. (Santana 2005) describes this as allowing messy factorisations of the graph. Further work on MN-EDA was presented in (Santana, Larrañaga & Lozano 2006), where an algorithm for learning the Kikuchi approximation using the expectation-maximization approach is presented.

MN-FDA and MN-EDA follow the same overall workflow, outlined in Algorithm 2.9.

Algorithm 2.9 MN-FDA and MN-EDA

```
1:  Generate random initial population $p$
2:  while stopping criteria is not met do
3:    Select a subset $\sigma$ of $p$
4:    Learn an independence graph $G$ from the data in $\sigma$
5:    If necessary, refine the graph $G$
6:    Find the set of maximal cliques $C$ in $G$
7:    if MN-FDA then
8:      Derive a junction graph from $C$
9:      Calculate marginal probabilities for the JG
10:     Sample from JG to generate a new population and replace $p$
11:   else if MN-EDA then
12:     Construct a clique based composition of $G$
13:     Find marginal probabilities for the regions of the decomposition (this constitutes
        the Kikuchi approximation)
14:     Sample Kikuchi approximation to generate a new population and replace $p$
15:   end if
16: end while
```

2.2.5.5 Markovianity Optimisation Algorithm

MOA (Shakya & Santana 2008a, Shakya & Santana 2008b) exploits the local Markov property of the Markov network to give more efficient function optimisation than can be achieved by estimating and sampling the full distribution. The undirected structure can be estimated in a number of ways, including those used in MN-FDA and MN-EDA. In (Shakya & Santana 2008a, Shakya & Santana 2008b) a cross-entropy measure is used to test for dependency between variables. The local Markov property means that each variable is dependent solely on its immediate neighbours in the dependency graph, requiring computation of only the conditional probabilities. A Gibbs sampler is used to repeatedly sample these probabilities for each variable to generate an individual; this is run repeatedly to generate a new population.

The workflow of MOA is given in Algorithm 2.10.

Algorithm 2.10 MOA

```
1: Generate random initial population $p$
2: while stopping criteria is not met do
3:   Select a subset $\sigma$ of $p$
4:   Estimate structure of Markov network from $D$
5:   for all $X_i$ do
6:     Estimate local Markov conditional probabilities $p(x_i|N_i)$ as defined by the undirected structure
7:     Sample $p(x_i|N_i)$ to generate a new population and replace $p$
8:   end for
9: end while
```

2.2.5.6 Is-DEUM

An extension to the univariate DEUM called Is-DEUM (Shakya, McCall & Brown 2006, Shakya & McCall 2007) adds bivariate interactions to the Markov network used in the algorithm. Is-DEUM assumes a fixed Ising model and has been benchmarked on instances of the Ising spin glass problem (Kindermann & Snell 1980).

The DEUM framework has recently also been extended to incorporate higher order interactions (Brownlee, McCall & Brown 2007) and learning the relationships between variables (the structure of the Markov network) (Brownlee, McCall, Shakya & Zhang

2009, Shakya et al. 2009). These topics will be revisited in Chapters 6 and 7.

2.2.6 Graphical Summary of Discrete EDAs

It is helpful for comparison to show the characteristics of the different EDAs on a table, with a visual representation of their respective probabilistic graphical models. This is presented in Table 2.1.

2.2.7 Continuous EDAs

The EDAs described in the previous section all operate on discrete problems. Considerable research has also been done into EDAs operating in a continuous domain. It is logical to suppose that in the case of continuous variables something is lost when representing them by a relatively small number of bits; though there is a counter-argument that a larger range of problems may be represented by a string of bits. Genetic Algorithms using real and integer values have been shown to work well in many cases (Davis 1991) and the same is true for EDAs. Many of the elementary EDAs such as UMDA, PBIL and MIMIC have been adapted to encodings such as integers and real numbers; a good summary can be found in (Larrañaga & Lozano 2002). Multivariate EDAs for continous problems include real-coded hBOA (Ahn, Ramakrishna & Goldberg 2004), as well as a number based on Gaussian models including EMNA (Larrañaga & Lozano 2002), RECEDA (Paul & Iba 2003), EEDA (Wagner, Auger & Schoenauer 2004), ED-EDA (Dong & Yao 2008b) and BUMDA (Peña, Aguirre & Rionda 2008).

The Iterated Density Evolutionary Algorithm (IDEA) (Bosman & Thierens 2000, Bosman & Thierens 1999) is a general framework which has also been used in discrete problem domains, but is well known for implementations in continuous domains. IDEA has two main characteristics; in each generation only individuals better than the worst from the previous generation are selected, and only part of the population is replaced at a time. Several different distributions can be used within the framework including the Gaussian distribution. The histogram distribution is perhaps one of the simpler to explain. Here, rather than measuring probabilities of 0s or 1s, the possible range of values

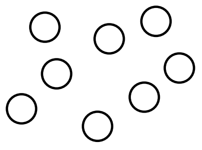
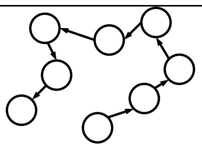
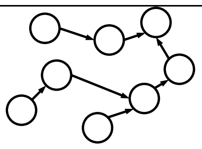
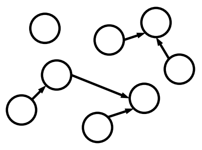
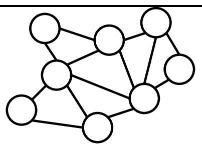
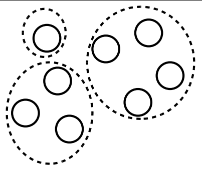
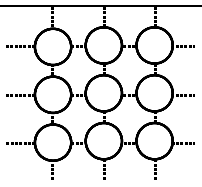
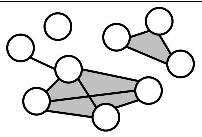
Algorithm (Originators)	Model Complexity and Notes	Graphical Representation of Structure
PBIL (Baluja, Caruana), cGA (Harik, Lobo, Goldberg), UMDA (Mühlenbein and Paass), DEUMd (Shakya, McCall, Brown)	Univariate - probability vector or marginal probabilities	
MIMIC (de Bonet, Davies)	Bivariate chain (directed) - conditional probabilities	
COMIT (de Bonet, Davies)	Bivariate trees (directed) - conditional probabilities	
BMDA (de Bonet, Davies), BOA (Pelikan, Sastry, Goldberg), EBNA (Etxeberria, Larrañaga)	Bivariate forest / Bayesian networks (directed) - conditional probabilities	
FDA (Mühlenbein, Mahnig)	Boltzmann Distribution with Triangulated Model (undirected, sampled as directed)	
EcGA (Harik)	Marginal Product Model (undirected) marginal probabilities of variable groups (undirected)	
Ising-DEUM (Shakya)	Markov network with lattice structure (undirected)	
MN-EDA (Santana), MOA (Santana and Shakya), General DEUM (Shakya)	Markov network (undirected)	

Table 2.1: Graphical Summary of Discrete EDAs

is divided into bins and a histogram used to count the number of individuals within each bin. New individuals are generated to match the distribution of values represented by each histogram.

2.2.8 Hybrid EDAs

A number of hybrid approaches which combine EDAs with other algorithms have also been researched (Zhang, Sun & Tsang 2007). The argument for such hybrids is that a less computationally expensive probabilistic model may be used, which will allow the algorithm to locate a highly fit individual but not necessarily reach the global optimum because of errors in the simplified model. A simple algorithm or heuristic is used to make the final step towards the global optimum. (Sun, Zhang, Li & Yao 2008, Zhang, Sun, Tsang & Ford 2004) uses a local search algorithm to improve the solutions sampled from a probabilistic model, using Lamarckian evolution rather like a memetic algorithm (Moscato 1989).

(Zhang & Sun 2006, Zhang et al. 2005, Zhang, Sun, Tsang & Ford 2003) present hybrids which work in the opposite direction. Rather than a simple hillclimber being used to improve the inviduals generated by an EDA, a probabilistic model is used to improve a simpler algorithm by guiding it - in this case by guiding mutation. This allows the mutation operator to select variables for mutation to values that are likely to improve fitness. Guided crossover and mutation operators in a modified version of eCGA are described in (Lima et al. 2005).

(Peña, Robles, Larrañaga, Herves, Rosales & Pérez 2004) proposes an algorithm which falls between these approaches. In GA-EDA, some individuals in the population are generated by sampling a probability distribution, and others are generated by traditional crossover and mutation operators.

2.3 Fitness Modelling

The ultimate aim of an evolutionary algorithm is optimisation; finding an optimal solution or set of solutions to a specific problem. This is achieved by building a model of the fitness function. This model can take a range of forms from being implicitly contained in a population to being explicitly defined as a response surface or similar. In general the algorithm only needs to model a part of the fitness function to efficiently optimise - modelling the complete fitness function represents a complex problem in itself.

The traditional evolutionary algorithm employs a coarsely grained approach to modelling the fitness function. In selecting a set of high fitness individuals and generating new individuals based on this set we are reclassifying the individuals as simply "high fitness". Selection of a larger proportion of high fitness individuals relative to low fitness ones and incorporation of information about low fitness individuals (negative selection) are approaches which help to mitigate this. However, once a set of individuals has been selected, their fitness relative to each other is still ignored by the algorithm. Maintaining a fine-grained distinction between individuals such as the raw fitness values or fitness ranks could allow the algorithm to make more intelligent choices when generating new individuals which we desire to have a high fitness.

Interest has been growing in algorithms which expand on this idea by attempting to build an explicit model of the fitness function. Aside from the potential to directly generate higher fitness individuals this can be used as a surrogate for an expensive fitness function (Lim et al. 2008, Sastry et al. 2006, Ong et al. 2004) particularly with applications where a human is performing the fitness evaluations such as evolutionary art (Romero & Machado 2007). It may also be used to guide genetic operators (Lima et al. 2005, Abboud & Schoenauer 2002, Jin & Sendhoff 2004, Rasheed & Hirsh 2000, Rasheed et al. 2002, Zhang & Sun 2006, Zhang et al. 2005) or to provide useful information about the fitness function.

A number of approaches to fitness modelling have been employed beyond treating the population or an EDA's probabilistic model as an implicit model of fitness. An EA may use fitness inheritance (Chen, Goldberg, S.-Y.Ho & K.Sastry 2002, Pelikan & Sastry 2004, Sastry, Goldberg & Pelikan 2001, Smith, Dike & Stegmann 1995) (passing of fitness values from parents to offspring) to reduce the number of fitness evaluations; though this still does not involve the construction of an explicit fitness model. More explicit models of fitness can be constructed by using techniques such as artificial neural networks (Jin & Sendhoff 2004) and machine learning techniques such as those used by LEM (discussed in Section 2.3.1). The algorithm presented in (Pošík & Franc 2007) models a contour line on the fitness landscape between high and low fitness individuals which the authors describe as a special case of the Learnable Evolution Model (LEM) (Michalski 2000). (Miquélez, Bengoetxea & Larrañaga 2004) describes an algorithm which groups individuals of similar fitness into classes which are then passed to Bayesian classifiers that can be sampled to generate individuals of high fitness. (Schmidt & Lipson 2008) uses coevolution as a means to generating fitness predictors efficiently. Further to these, polynomial regression or the fitting of a response surface has also been used to construct a model of fitness (Zhou, Ong, Nguyen & Lim 2005). The work described in this thesis bears some similarity to this; the Markov fitness model is in effect a response surface for the fitness function. A key difference is that this work concentrates on discrete fitness functions whereas (Zhou et al. 2005) concentrates on continuous fitness functions.

A comprehensive literature survey of fitness modelling for evolutionary computation was presented in (Jin 2005).

2.3.1 LEM

The Learnable Evolution Model or LEM (Michalski 2000) is a framework for optimisation. Like other evolutionary approaches it incorporates a population from which fit individuals are selected. LEM operates in two *modes* resulting in an interesting hybrid algorithm which can employ the strengths of traditional evolutionary computation and machine learning. In Darwinian Evolution mode it employs the biologically-inspired genetic reproduction operators. In Machine Learning mode LEM learns a set of inductive hypotheses that specify the features which make certain individuals fitter than others. This information is then used to generate new individuals. LEM may be argued to be a more general version of an EDA - the probabilistic model of an EDA forming the hypotheses of LEM. It is important to this work because by learning features which distinguish fit individuals from unfit ones it effectively builds a model of the fitness function.

LEM switches between the two modes dependent on a mode termination criterion which is preset by the user in much the same way as an evaluation count limit in a GA. The fitness function may be either discrete or continuous and the learning may also incorporate multiple populations from multiple generations. The workflow of LEM is given in Algorithm 2.11.

Algorithm 2.11 LEM

```
1: Generate random initial population $p$
2: while stopping criteria is not met do
3:    Alternate between modes starting step 4 and step 10
4:    Machine Learning mode:
5:    repeat
6:       Derive extrema: divide population into an H-group (high fitness) and an L-group
         (low fitness); conventional selection may be used to choose these groups and it is
         acceptable for an individual to be in both groups
7:       Create a hypothesis: use a machine learning method to create a set of hypotheses
         that distinguish the two groups
8:       Generate a new population using the hypotheses describing the H-group
9:    until some condition is met
10:   Darwinian Evolution mode:
11:   repeat
12:      Apply selection
13:      Apply mutation and crossover to generate a new population
14:   until some condition is met
15: end while
```

2.4 Benchmark Functions

A number of test problems are commonly used for benchmarking evolutionary algorithms. Like the algorithms themselves they may be classified according to the complexity of relationships between their variables. These problems are rather contrived in their nature but designed to allow prediction of an algorithm's performance on similar real-world problems. They are also designed to have specific properties such as known interactions between variables. It should be made clear that an algorithm which performs well on one problem will not necessarily perform well on another.

All functions described here assume a bit string representation and are consequently discrete optimisation problems. Further problems can be found in (Larrañaga & Lozano

2002, de Bonet et al. 1997, Harik et al. 1997) and throughout the literature. Those described here are chosen primarily for two reasons. Firstly, all functions have been used as benchmarks for other evolutionary algorithms and with the exception of the biocontrol and Huygens probe problems (section 2.4.8 and 2.4.9) all have been used for benchmarking EDAs. This is important as it allows comparisons to be made which have revelance to the wider community. Secondly, the functions have chosen to cover a wide range of underlying structures: univariate (onemax), directed chain (4 and 6 peaks), order-2 and 3 undirected interactions (checkerboard, Ising and MAXSAT), higher order undirected interactions (trap and royal road) and real-world functions (biocontrol and Huygens probe). This is important as we consider multivariate models of fitness and particularly as we look at the underlying features of fitness functions in Chapter 3 and structure learning in Chapter 7.

2.4.1 One Max

This is one of the simplest and most commonly used benchmarks; a good description of OneMax can be found in Mitchell's textbook on GAs (Mitchell 1998). In this problem the fitness is simply the number of bits with the value 1, expressed formally in (2.9).

$$f(x) = \sum_{i=1}^{n} x_i \tag{2.9}$$

This is a classic univariate problem as there are no interactions between variables – each one has an independent and equal contribution to the overall fitness. The fitness landscape is a smooth curve towards the global optimum with no local optima. The optimal solution is all $x_i = 1$, with $f(x) = n$.

2.4.2 4- and 6-peaks

The 4- and 6-peaks problems are described in Baluja's paper introducing PBIL (Baluja & Caruana 1995) and de Bonet's paper on MIMIC (de Bonet et al. 1997) respectively. They are functions designed to trap algorithms with a simple view of the problem's structure by giving a large fitness increase only when a complete chain of variables take on a specified value. Otherwise, a variant of onemax is used to award values which tend away from the true global optimum, deceiving an algorithm which can not find the chain. An algorithm which includes interactions between variables in solving the problem should be able to detect this feature and avoid the trap. The 4-peaks function is described in (2.10).

$$f(x) = MAX(o(x), z(x)) + REWARD \tag{2.10}$$

Where:

- $z(x)$ = Number of contiguous zeros ending in position 100
- $o(x)$ = Number of contiguous ones starting in position 1

$$REWARD = \begin{cases} 100 \quad if \quad o(x) > T \;\; and \;\; z(x) > T \\ 0 \quad otherwise \end{cases}$$

(The Threshold T is a parameter of the problem)

In 4-peaks, the global optima are a string of $T + 1$ 1s followed by all 0s or $T + 1$ 0s preceded by all 1s; local optima are strings of all 0s or all 1s. In this problem, an algorithm which does not heed relationships between variables will tend towards building a string of all 1s or 0s.

6-peaks is a modified version of 4-peaks, with an additional copies of the two global optima in 4-peaks that have 0s and 1s exchanged; $T+1$ 0s followed by all 1s and $T+1$ 0s preceded by all 1s.

2.4.3 Royal Road

This problem is described in (Mitchell 1998). Variables are arranged into groups which only contribute to the total fitness when all members of the group are 1. For example, using a group size of 4: 1100000011111011 would score a value of 4, as only the third set of variables are all 1. Originally designed to observe the behaviour of genetic algorithms, this problem introduces a simple relationship between groups of variables. It has been studied in the context of EDAs (Brown, Garmendia-Doval & McCall 2002). It can be modified using a permutation such that the logical groups of variables are distributed throughout the bitstring, rather than being composed of neighbouring bits. The instance of the function we will be using is expressed formally in (2.11).

$$f(x) = \sum_i c_i \delta_i(x) \quad where \quad \delta_i(x) = \begin{cases} 1 & if \quad x \in s_i \\ 0 & otherwise \end{cases} \tag{2.11}$$

Here, s_i is a member of a list of schemas defining the blocks, and c_i is a coefficient given with each schema. Essentially this is a weighting given to each block, which allows different blocks do be given different influences on fitness. In the example given in the text above, and through this thesis, this would be equal to the block length. This means that all blocks have the same weighting. To aid interpretation of results, through this thesis the problem is modified slightly so that the optimum for alternate groups are set to 1 and 0.

2.4.4 Checkerboard

The checkerboard problem (Baluja & Davies 1997b, Larrañaga & Lozano 2002) introduces bivariate interactions into a test problem. Chromosomes are a square number of bits in length; they represent the rows of a s x s grid concatenated into one string. The objective is to realise a grid with a checkerboard pattern with alternating 1s and 0s; thus each 1 should be surrounded by 0s and vice versa, not including corners. Formally this is written as in (2.12).

$$f(x) = 4(s-2)^2 - \sum_{i=2}^{s-1} \sum_{j=2}^{s-1} \{\delta(x_{ij}, x_{i-1j}) + \delta(x_{ij}, x_{i+1j}) + \delta(x_{ij}, x_{ij-1}) + \delta(x_{ij}, x_{ij+1})\} \tag{2.12}$$

Where δ is Kronecker's delta function,

$$\delta_{ij} = \begin{cases} 1 \quad if \quad i = j \\ 0 \quad if \quad i \neq j \end{cases} \tag{2.13}$$

In this thesis we will also be making use of a simplified version of this problem, 1D checkerboard. Rather than a grid, the variables are kept in the chain which makes up the bit string. Where a neighbouring pair of variables are opposite in value, 1 is added to the fitness. This is defined formally in (2.14). The optimum for this 1D Checkerboard is a string of alternating 1s and 0s.

$$f(x) = \sum_{i=1}^{n-1} \delta(x_i, x_x + 1) \tag{2.14}$$

2.4.5 Ising Spin Glasses

The general Ising spin glass problem (Kindermann & Snell 1980) is defined by an energy function over a set of spin variables $\sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_l\}$ and a set of coupling constants h and J as in (2.15).

$$H(\sigma) = -\sum_{i \in L} h_i \sigma_i - \sum_{i<j \in L} J_{ij} \sigma_i \sigma_j \tag{2.15}$$

Here L is a lattice of n sites, and each coupling constant h_i and J_{ij} relate to a single spin $\sigma_i$and pair of spins $\sigma_i$and σ_j respectively. Each spin variable can either be +1 or -1. In the context of binary EDAs, the coupling constants are also restricted to +1 and -1, although in the general case this restriction is removed. The objective of the problem is to find a configuration of coupling constants which minimises the energy H. The structure of the problem can be a 1-dimensional chain, 2-dimensional grid, or of a higher order; a property which adds to its usefulness in testing algorithm.

A good description of the Ising problem can be found in (Pelikan & Goldberg 2003), where it is used in testing the hierarchical BOA algorithm.

2.4.6 Trap-κ

The trap functions (Pelikan 2002) are designed to deceive evolutionary algorithms into converging on a local optimum. This is particularly a problem for algorithms which do not consider interactions between variables. The trap function of order k is defined in (2.16). It computes fitness by dividing the bitstring into blocks similar to Royal Road.

$$f(x) = \sum_{i=1}^{n/k} trap_k(x_{b_i,1} + ... + x_{b_i,k}) \tag{2.16}$$

Each block $(x_{b_i,1}, +... + x_{b_i,k})$ gives a fitness, calculated as in (2.17), where u is the number of 1s in the block of k bits. Trap-5, which is the specific instance used in this thesis, has $k = 5$, $f_{high} = 5$ and $f_{low} = 4$.

$$trap_k(u) = \begin{cases} f_{high} & \text{if} \quad u = k \\ f_{low} - u\frac{f_{low}}{k-1} & otherwise \end{cases} \tag{2.17}$$

The functions are designed to mislead or *trap* algorithms which do not account for interactions between variables. As u increases, fitness decreases, which leads the algorithm away from the global optimum. Algorithms which do find interactions should be able to determine that groups of k variables being switched to all 0 improves fitness. The functions are commonly referred to as Trap-k functions - in this thesis we use k to represent cliques so will refer to Trap-k as Trap-κ functions.

2.4.7 Maximum Satisfiability (MAXSAT)

The Maximum Satisfiability or MAXSAT Problems are described in (Johnson 1973). The problem is defined as the search for a set of values which maximise the number of satisfied clauses in a fixed predicate logic formula. Many real-world problems can be mapped on to MAXSAT, including the well-known graph colouring problem. It is known to be NP-complete in its general form. MAXSAT is particularly useful for experiments in modelling high order interactions as each instance of the problem uses a known predefined structure; each clause translates to a clique on the dependency graph. In the 3-CNF variant of the problem each clause consists of three variables, expressed in conjunctive normal form. 3-CNF MAXSAT has already been used to benchmark the EDA hierarchical Bayesian Optimisation Algorithm (hBOA) (Pelikan & Goldberg 2003) - that work made use of the SATLIB resource (Hoos & Stützle 2000) which provides a collection of a large number of sample MAXSAT problems.

Encoding MAXSAT for evolutionary algorithms is a straightforward task. The candidate solutions are bitstrings in which each bit encodes a predicate variable in the formula. An individual's fitness is equal to the number of satisfied clauses given the predicate values in it.

An example is helpful to illustrate this. A particular 5 variable, 3 clause instance 3-CNF MAXSAT problem has the set of predicates in (2.18).

$$(x_1 \vee x_2 \vee \bar{x_3}) \wedge (\bar{x_1} \vee x_3 \vee \bar{x_4}) \wedge (x_3 \vee \bar{x_4} \vee x_5) \tag{2.18}$$

The individual $x = \{11011\}$ would satisfy the first and third clauses, so would have a fitness of 2.

2.4.8 Bio-control in Mushroom Farming

When mushrooms are produced in commercial quantities, the quality and yield of the mushroom crop can be seriously damaged through infestation by sciarid flies. Sciarid fly larvae are known to feed on the mycelium in the casing layer of mushroom causing crop production to significantly decline. An important weapon in combatting sciarid fly is the use of the nematode worm, *Steinernema feltiae*, which feeds on sciarid larvae thus reducing the problem. Nematode worms are sold commercially for bio-control of sciarid flies on mushroom farms.

(Fenton, Gwynn, Gupta, R.Norman & J.P.Fairbairn 2002) gives a dynamic mathematical model which expresses the life cycle of Sciarid larvae in the presence of periodic dosing with nematode worms. A number of genetic algorithms have been applied to the problem of finding an optimal set of intervention points for this model in (Godley, Cairns & Cowie 2007a, Godley, Cairns & Cowie 2007b, Godley, Cowie & Cairns 2007). Each chromosome is a string of 50 bits, each bit representing a point at which a possible intervention of nematodes may occur. The function is a bang-bang control problem, in that interventions of worms are either applied or not applied at any given time slot. The global optimum is unknown but typical optima found by the genetic algorithms in the referenced papers have a large dose of nematode worms applied early on with short bursts later in the life cycle of the larvae.

2.4.9 Huygens Probe Function

The Huygens Probe Function (MacNish 2005) is a real valued function designed for benchmarking algorithms, and used as part of a competition at the 2006 Congress on Evolutionary Computation. The object is to find the lowest point on a simulated lunar landscape using only 1000 function evaluations and it is this restriction which is of interest here. Algorithms which use fitness modelling to reduce the number of function evaluations offer the potential to perform well on this problem.

The landscape is fractal generated; this means that zooming in on a small area of the landscape reveals similar levels of varation in fitness. According to the developers of the function this *scale invariance* continues to the limit of IEEE 64-bit floating point resolution so the function does not (effectively) “bottom out”. This means that the algorithm has to keep a highly delicate balance between exploration and exploitation. A typical landscape is illustrated in Figure 2.3 and the concept of scale invariance is illustrated in 2.4. The function takes an X and Y coordinate (both floats) and returns a float representing the height.

The natural representation for this problem in EA would be for an individual to comprise two floating point variables, one for each coordinate. In the experiments described in this thesis, the two values are encoded to a bitstring representation which is converted to a float just before evaluation. This fits the problem to the algorithm; it would be likely be more effective to adapt the algorithm to deal with floating point variables and this is discussed in Chapter 8.

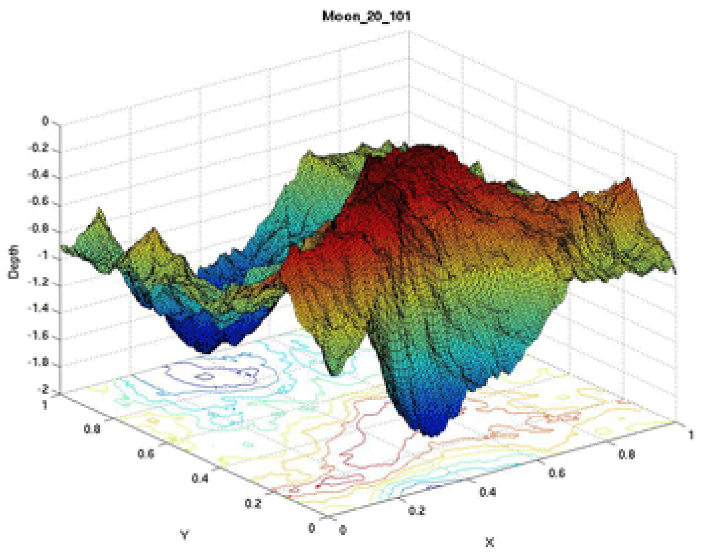


Figure 2.3: Visual representation of a Huygens Probe function lunar landscape

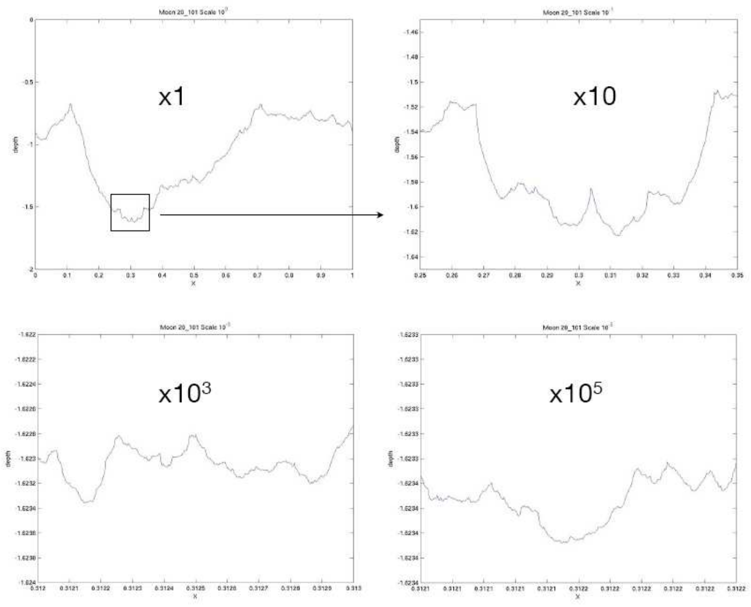


Figure 2.4: Scale invariance of Huygens Probe function

Chapter 3

Fitness Modelling with Markov Networks

In this chapter we look at how a Markov network may be used to model the distribution of fitness (expressed in terms of energy) over the variables of a problem, using what we will call the Markov Fitness Model (MFM). Optimisation is the end-goal for evolutionary algorithms in general; typically for optimisation we do not seek to model the whole fitness function but a subset of it which will allow the algorithm to locate regions of high fitness and the global optima. In Chapters 6 and 7 and in the previous work on optimisation using the MFM (Shakya 2006, Shakya et al. 2006, Shakya et al. 2005b, Shakya et al. 2005c) discussed in Chapter 2, the distribution modelled by the MFM is sampled to generate individuals which probably have a high fitness. The sampler performs a series of mutations on an individual which are biased according to a marginal probability calculated from the probabilistic model. To find the global optimum by direct sampling in this manner it is only necessary that the MFM models high fitness areas of the fitness function. In addition to direct sampling the information about fitness contained within the MFM can also be used to gain better understanding of the fitness function. This requires a model which approximates a much wider range of the fitness function. In order to achieve effective optimisation through either of these routes we need to understand the factors which affect how closely the MFM can approximate parts of the fitness function and develop techniques to measure the approximation. This chapter will provide the necessary terminology and measures which will allow us to go on to explore two of the major factors affecting the model (population size and selection) in Chapters 4 and 5.

The chapter is divided in to three sections. The first describes the MFM and provides some definitions which will be used through the rest of the thesis. The second section explores the fitness prediction capability of the MFM and defines a measure of this capability called the fitness prediction correlation. This is followed by a description of a series of experiments with results demonstrating the application of this measure to the MFM generated for the benchmark problems. The third section section moves on to compare the MFM parameters calculated to features of the benchmark problems considered.

3.1 From univariate to multivariate

Structure is the set of interactions or dependencies between variables in the probabilistic model used by an EDA. Previous work in EDAs using a Markov network was described in the literature review; that work used a univariate model structure within the DEUM framework (Shakya, McCall & Brown 2005a, Shakya et al. 2005c, Shakya et al. 2004b, Shakya et al. 2005b). This was extended to allow for bivariate interactions which occur in the 2D Ising problem (Shakya et al. 2006). This section uses Walsh analysis to provide the necessary expressive power to take the theoretical DEUM framework up to a full multivariate model. Specific examples are given for up to a trivariate model and experiments demonstrating optimisation with up to a trivariate model are presented in Chapter 6. This is followed by some definitions related to model structure which will be referred to through the rest of the thesis.

3.1.1 Walsh functions

The grounding for this approach lies in the Walsh analysis of fitness functions, originally presented in (Bethke 1980), further in (Goldberg 1989a, Goldberg 1989b) and (Thierens 1999a, Thierens 1999b). Walsh functions are a set of rectangular waveforms taking two amplitude values, +1 and -1. Similar to the use of Fourier transforms representing for analogue waveforms, Walsh functions may be combined linearly to represent any fitness function based on a bit string representation. Haar functions have been proposed (Khuri 1994) as a more efficient alternative to Walsh functions for representing fitness functions. Walsh functions have also been used in the theoretical analysis of epistasis (linkage) in evolutionary algorithms (Heckendorn & Whitley 1999). We now describe how Walsh functions can be used as the basis for the Markov Fitness Model.

For discrete problems, an individual x made up of random variables encoded as a bit string can be expressed as:

$$x = x_1, \ldots, x_n \; x_i \in \{0, 1\} \tag{3.1}$$

The variables x_i may be represented in a graphical model, in which a clique is a fully connected subset of $\{x_1 \ldots x_n\}$ which can be defined:

$$K \subset \{1, \ldots, n\} = S_n \tag{3.2}$$

We refer to the number of variables within a clique K as its *degree*, represented $|K|$. The set of Walsh functions $W_K(x)$ for increasing degrees of K are defined in (3.3) to (3.5).

$$K = \varnothing \qquad W_\varnothing(x) \equiv 1 \; \forall \; x \tag{3.3}$$

$$K = \{i\} \qquad W_i(x) = \begin{cases} 1 & x_i = 1 \\ -1 & x_i = 0 \end{cases} \tag{3.4}$$

$$\text{For } \; K \subseteq |1, \ldots, n|, \qquad |K| \geqslant 2, \qquad W_K(x) = \prod_{i \in K} W_i(x) \tag{3.5}$$

In 3.3 we define what we call the zero clique; that is, a term which is not associated with any of the variables. 3.4 defines the singleton cliques which represent each variable x_i. Finally, in 3.5 we are able to define any clique which consists of more than one variable. This gives us the vocabulary to describe the Markov Fitness Model precisely. The relationship between the energy distribution $U(x)$, fitness $f(x)$ and the problem variables for the general case is defined in (3.6).

$$U(x) = -\ln(f(x)) = \sum_K \alpha_K W_K(x) \tag{3.6}$$

where each α_K is a parameter associated with a clique on the Markov network. The set of α_K completely define the distribution. The model can incorporate all multivariate interactions, although in practice this will be limited by available space and processing power as the number of terms in the model for n variables will grow by order $O(2^n)$. We can exclude cliques by setting the corresponding α_K to zero. The α_K values are calulated by substituting the values for x_i and $f(x)$ from a population into (3.6) for each individual, forming a set of equations which can be solved to find the α_K. In the experiments through the rest of the thesis we use singular value decomposition (Press, Flannery, Teukolsky & Vetterling 1986) to solve the system of equations, as described in (Shakya 2006).

We will now use α_K to refer to the model coefficients in general; we can specify α_K for specific cliques by listing the variables in the clique in place of K. Thus the coefficient for a 2-clique $K = \{i, j\}$ is α_{ij}.

3.1.2 Chain Model

It is useful to explore this further with two specific models which are used by algorithms in later chapters. Firstly we look at the chain model, which incorporates explicit interactions between variables which neighbour each other within the bit string encoding. It could be argued that this model has some similarity to the way variable relationships are implicitly defined in a genetic algorithm which uses single or two point crossover. Such a GA maintains the ordering of neighbouring variables in the bitstring when applying values to them. A directed chain was used as the basis of the probabilistic model in one of the early EDAs, MIMIC (de Bonet et al. 1997).

Like the 2D Ising model structure, the chain is a bivariate structure as each interaction is limited to two variables. The structure is illustrated in Figure 3.1. We now describe the energy function for the chain structure and give an example of how it is applied to a specific individual in a population.

The chain structured model for a problem of n variables has the general energy function:

$$U(x) = \alpha_0 + \sum_{i=1}^{n} \alpha_i x_i + \sum_{i=1}^{n-1} \alpha_{ij} x_i x_j \tag{3.7}$$

where each α is a parameter associated with a clique on the Markov network, α_0 is a constant representing the zero-clique of background energy in the Markov Network, n is the number of variables in each individual and x_i represents the value of variable i in the solution x. As in the set of Walsh functions, $-1, 1$ are used as the values of $\mathrm{x_i}$ in place of $0, 1$ to ensure arithmetical symmetry between values. Redefining the energy function in terms of Walsh functions gives us:

$$U(x) = \sum_{k} \alpha_K W_K(x) \quad where \quad \alpha_K \neq 0 \ \forall \begin{cases} |K| = 0, 1 \\ K = i, i+1 (1 \leqslant i < n) \end{cases} \tag{3.8}$$

For the individual represented in Figure 3.1 the energy function would be as defined in (3.9). For the formula based on Walsh functions the cliques with non-zero α_K are $k = \varnothing$, $\{1\}$, $\{2\}$, $\{3\}$, $\{4\}$, $\{5\}$, $\{1, 2\}$, $\{2, 3\}$, $\{3, 4\}$, $\{4, 5\}$.

$$\begin{aligned} U(x) = \alpha_0 + \alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_3 + \alpha_4 x_4 + \alpha_5 x_5 \\ + \alpha_{12} x_1 x_2 + \alpha_{23} x_2 x_3 + \alpha_{34} x_3 x_4 + \alpha_{45} x_4 x_5 \end{aligned} \tag{3.9}$$

An individual $x = \{11001\}$ which for some fitness function has a fitness of 2 would have the energy function shown in (3.10).

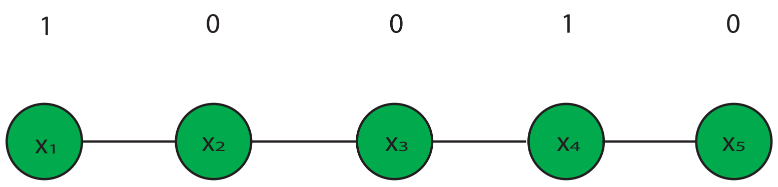


Figure 3.1: Relationships between variables in a chain structure

$$-\ln(2) = \alpha_0 + \alpha_1 + \alpha_2 - \alpha_3 - \alpha_4 + \alpha_5 + \alpha_{12} - \alpha_{23} + \alpha_{34} - \alpha_{45} \tag{3.10}$$

Techniques for learning the chain structure (essentially just an ordering of the variables) are discussed in Chapter 7. Otherwise, we use a fixed chain which uses the ordering of variables defined by the encoding of the fitness function.

3.1.3 3-CNF MaxSAT Model

The model structure for 3-CNF MAXSAT is more complex than than a chain or other bivariate structure, including terms for three-way (trivariate) interactions. A sample 3-CNF problem is given in (3.11); recall from Chapter 2 that a fitness function based on this has the fitness equal to the number of satisfied clauses. The negations may be ignored when considering the relationships between predicate variables giving us the undirected graphical structure shown in Figure 3.2. Here, the nodes (blue) are the variables, edges between them are 2-way or bivariate interactions and the coloured triangles are 3-way (which we call trivariate) interactions.

$$(x_1 \vee x_2 \vee \bar{x_3}) \wedge (\bar{x_1} \vee x_3 \vee \bar{x_4}) \wedge (x_3 \vee \bar{x_4} \vee x_5) \tag{3.11}$$

The sets of interactions are derived directly from the structure of the given MAXSAT problem instance. The general energy function for 3-CNF MaxSAT is defined in (3.12).

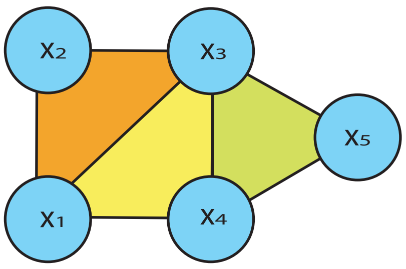


Figure 3.2: Relationships Between Predicate Variables

$$U(x) = \alpha_0 + \sum_{i=1}^{n} \alpha_i x_i + \sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_{ij} x_i x_j + \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} \alpha_{ijk} x_i x_j x_k \tag{3.12}$$

where as in section 3.1.2 each α is a parameter associated with a clique on the Markov network, α_0 is a constant representing the zero-clique of background energy in the Markov Network, n is the number of variables in each individual and x_i represents the value of variable i in the solution x. As before, $\{-1, 1\}$ are used as the values of x_i in place of $\{0, 1\}$. (3.12) can be represented in terms of Walsh functions as shown in (3.13).

$$U(x) = \sum_{K} \alpha_K W_K(x) \quad \textit{where} \quad \alpha_K \neq 0 \ \ \forall |K| \leq 3 \tag{3.13}$$

For a five bit individual using the structure in 3.2 the energy function is defined in 3.14. The cliques with non-zero α_K are $K = \varnothing$, $\{1\}$, $\{2\}$, $\{3\}$, $\{4\}$, $\{5\}$, $\{1,2\}$, $\{1,3\}$, $\{1,4\}$, $\{2,3\}$, $\{3,4\}$, $\{3,5\}$, $\{4,5\}$, $\{1,2,3\}$, $\{1,3,4\}$, $\{3,4,5\}$.

$$\begin{aligned} U(x) = {} & \alpha_0 + \alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_3 + \alpha_4 x_4 + \alpha_5 x_5 \\ & + \alpha_{12} x_1 x_2 + \alpha_{13} x_1 x_3 + \alpha_{14} x_1 x_4 + \alpha_{23} x_2 x_3 \\ & + \alpha_{34} x_3 x_4 + \alpha_{35} x_3 x_5 + \alpha_{45} x_4 x_5 \\ & + \alpha_{123} x_1 x_2 x_3 + \alpha_{134} x_1 x_3 x_4 + \alpha_{345} x_3 x_4 x_5 \end{aligned} \tag{3.14}$$

An individual $x = \{11011\}$ would have fitness 2, and consequently would have the energy function shown in (3.15).

$$\begin{aligned} -\ln(2) = c + \alpha_1 + \alpha_2 - \alpha_3 + \alpha_4 + \alpha_5 + \alpha_{12} - \alpha_{13} + \alpha_{14} - \alpha_{23} \\ - \alpha_{34} - \alpha_{35} + \alpha_{45} - \alpha_{123} - \alpha_{134} - \alpha_{345} \end{aligned} \tag{3.15}$$

3.1.4 Types of Model Structure

We can now make some definitions with regard to model structures which we will make use of through the rest of this thesis. We express the structure in terms of cliques which specify the Walsh functions for the MFM, and refer to the example 3-CNF formula from Section 3.1.3. When building the Markov network there are a number of scenarios which may occur:

1. *Full model structure* - the structure includes all possible interactions up to a specified level of complexity. That is, the α_K are non-zero for all K below a particular degree. The number of interactions present in the full model structure grows by 2^n in general and for the example 5-bit structure the full model will have 32 parameters including the univariate terms and the constant (the order-1 and order-0 cliques).

2. *Perfect model structure* - the structure includes exactly those cliques which are present in the underlying fitness function as well as any subcliques. Only those cliques have non-zero values for their associated α_K. This is the subset of the interactions present in the full structure which influence the absolute fitness value. In the 3-CNF MAXSAT example these are: x_1x_2, x_1x_3, x_1x_4, x_2x_3, x_3x_4, x_3x_5, x_4x_5, $x_1x_2x_3$, $x_1x_3x_4$, $x_3x_4x_5$. These interactions are required to perfectly fit the model to the fitness function. This is only possible for predefined test problems where the interactions are explicitly defined. In the case of onemax, there are no interactions (that is, the model has only non-zero α_K for $|K| \leq 1$). For the 2D Ising problem an interaction exists wherever there is a coupling between two spin variables. In the case of 3-CNF MAXSAT, each clause of three variables results in the addition of a 3-way interaction, and three pairwise interactions. In the example given in Section 3.11 the perfect model will have 16 parameters in total including the univariate and constant terms.

3. *Imperfect model structure* - the structure is missing some or all of the interactions present in the perfect model structure and may include some additional interactions which are not present in it. This is the likely situation for a structure inferred from a population of individuals. It is not always essential to construct a perfect model of the fitness function to optimise; often coefficients will be required in the model to reflect small changes in fitness that have no effect on the ranking of individuals by

fitness. In Chapter 7 we will use measures borrowed from the information retrieval community - precision, recall and the F-measure (Witten & Frank 2005) - to measure the differences between imperfect and perfect structures. This is similar to the concept of unnecessary interactions (Hauschild, Pelikan, Lima & Sastry 2007). This is not to be confused with the related idea of benign and malign interactions (Kallel, Naudts & Reeves 2000). These terms are used to describe whether the infuence of an interaction on fitness has a positive or negative correlation with the combined effect of its component parts. Spurious correlations (Mühlenbein & Mahnig 2000), (Santana, Larrañaga & Lozano 2007) are also related to this, but are false relationships in the model resulting from selection rather than interactions present in the fitness function but not required for optimisation. In the 3-CNF MAXSAT example, the model may have anything from 1 up to 31 parameters including the constant. It is useful to define two specific instances of imperfect structure:

3.1. *Decimated model structure* - we will use this term to refer to an imperfect structure which has been created by removing a fixed percentage of the cliques where $|K| > 1$ (that is, cliques representing interactions). The cliques are chosen for removal at random.

3.2. *Filtered model structure* - the structure is restricted to all interactions of a particular degree. A simplified structure will in turn also be an imperfect structure. We can relate common terms from the EDA community to structures specified in terms of cliques with non-zero α_K values:

  3.2.1. **Univariate** - $\alpha_K \neq 0 \ \forall \, |K| \leqslant 1$

  3.2.2. **Bivariate** - $\alpha_K \neq 0 \ \forall \, |K| \leqslant 2$

  3.2.3. **Trivariate** - $\alpha_K \neq 0 \ \forall \, |K| \leqslant 3$

  3.2.4. **Multivariate** - $\alpha_K \neq 0 \ \exists \, |K| > 2$

All structures will fall in to one of these categories. If a structure is not full it will either match the perfect structure precisely or if it does not it will be classed as an imperfect structure.

3.1.5 Structure sizes

(Shakya et al. 2006) makes some definitions related to the construction of the MFM which we will also make use of. If the number of parameters in the model is N (this is equal to $|K|$) and the number of individuals taken from the population to estimate the model parameters is M then there are three situations:

1. A *Under-specified* system, where $M < N$
2. A *Precisely-specified*, where $M = N$
3. A *Over-specified*, where $M > N$

We will make use of these terms through the rest of this thesis.

3.2 Fitness Prediction Correlation

As discussed in the introduction to this chapter and in greater detail in Chapter 6 one approach to optimisation using the MFM is to directly sample the distribution to generate individuals which have a high probability of being high in fitness. Sampling techniques such as the Gibbs sampler described in (Shakya 2006, Shakya et al. 2006) and Chapter 6 generate an individual by starting with a randomly generated individual and performing a series of mutations on it which are biased in the direction of high fitness (low energy) by the MFM. For this reason it is important to know how effectively the model is able to predict the change in fitness for an individual after mutation. It is interesting to look at both short and long distance mutation, which reflect the state of an individual at the beginning and end of a run of the sampler. In this section we develop two related measures of fitness prediction capability for the MFM which will be extensively used for analysis of fitness modelling in later chapters. These are the C_r and C_m values, which are both variants of what we call the fitness prediction correlation or FPC, which we now move on to describe.

We have already seen in (3.6) that the MFM relates fitness and raw variable values in the following expression:

$$-\ln(f(x)) = \sum_k \alpha_K W_K(x)$$

By estimating the parameters (α_K) values we have a model, the MFM, which approximates the fitness function based on a sample population. As has been described in Chapter 2 we can then optimise using this model by computing and sampling marginal probabilities for each variable (the DEUM framework). There are a range of other uses for the fitness modelling capability as discussed in Section 2.3 such as surrogate fitness functions (Sastry et al. 2006) or guided genetic operators (Abboud & Schoenauer 2002, Jin & Sendhoff 2004, Rasheed & Hirsh 2000, Rasheed et al. 2002, Zhang et al. 2005). Here, we use fitness prediction as the basis for a measure of how closely the MFM is modelling the fitness function. The premise is that if the model can accurately predict the fitness of an "unseen" individual it is a reasonably good fit to the fitness function. An alternative measure of model quality could be the least-square error output from SVD after computing the α_K but this is less desireable because it simply measures how closely the model reflects the fitnesses of the existing population, not unseen individuals as is the case for using fitness prediction.

Predicting the fitness of individuals is simply a reversal of the process used to estimate the α_K. The bitstring of a given individual is encoded as before so that for each x_i, 0 is coded as -1 and 1 coded as 1. These values are substituted into the energy function to give a predicted energy $U(x)$ for the individual. The predicted fitness can then be calculated as in (3.16).

$$f(x) = e^{-U(x)} \tag{3.16}$$

We now move on to discuss how this fitness prediction capability can be used in conjunction with a statistical correlation to measure fitness modelling capability.

3.2.1 Correlations

Initial studies on this (Brownlee et al. 2007, Brownlee, McCall, Zhang & Brown 2008) used the product moment correlation coefficient (Lucey 1984) to compare the predicted fitness values of a population with the true fitness values. This is calculated as shown in (3.17).

$$r = \frac{\sum_{i=0}^{m}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=0}^{m}(x_i - \overline{x})^2 \sum(y_i - \overline{y})^2}} \tag{3.17}$$

where x is an independent variable (in this case a true fitness value), y is a dependent variable (here a predicted fitness value), $\overline{x}$ and $\overline{y}$ are the means of x and y respectively and M is the number of data points (here the population size). This serves as a useful measure, but in reality an evolutionary algorithm's primary objective is optimisation. For discrete optimisation, it is only necessary to rank individuals in order of relative fitness. Thus we conclude that a rank correlation measure such as Spearman's rank correlation coefficient (Lucey 1984) is more appropriate. This is calculated as shown in (3.18).

$$R = 1 - \frac{6\sum_{i=0}^{m} d_i^2}{\sqrt{\sum_{i=0}^{m} M(M^2 - 1)}} \tag{3.18}$$

where d is the difference between ranks of corresponding values and M is the number of pairs of rankings.

Correlation coefficients range from -1 to +1. A value of +1 is said to be a perfect positive correlation which here would represent a model which perfectly fits the fitness function $f(x)$. A value of -1 is a perfect negative correlation; essentially the model is perfectly fitting $-f(x)$. A value of 0 represents no correlation; that is, the model has no correlation with the fitness function at all. Values between these extremes indicate strong and weak correlations with any absolute value greater than 0.7 being generally regarded as a strong statistical correlation (Rowntree 1981).

We use the rank correlation coefficient as a measure for the fitness prediction capability of the MFM; what we call the fitness prediction correlation or FPC. As discussed at the start of this section we are interested in two aspects of fitness modelling capability: how closely the model can predict change in fitness for individual over long and short mutations (that is, individuals which have respectively had many and a small number of bits mutated). The first translates into the ability of the model to predict the fitness of randomly generated individuals; it follows that if this is a strong correlation then the MFM is closely modelling the general fitness function. This is useful information for situations where the model may be used as a surrogate for the fitness function or directly sampled for the global optimum is in DEUM$_\mathrm{d}$ (Shakya et al. 2006, Shakya et al. 2005b). We will refer to the FPC which measures the correlation between predicted and true fitnesses for randomly generated individuals as C_r.

The second aspect of fitness modelling capability we are interested in is how the ability of the MFM to predict fitness of solutions "near to" those in the current population. This is important in the context of the proximate optimality principal (Glover & Laguna 1997) which assumes that good solutions have similar structure. Based on this idea, EAs will generally move to a population which closely resembles at least part of the current one. Related work on fitness of neighbouring solutions is described in (Collard, Verel & Clergue 2004). To measure this we develop a second FPC measure, C_m. This is the correlation between predicted fitness and true fitness for individuals which are a Hamming distance of 1 from the population use to estimate the MFM. In general we would expect that $C_m > C_r$ because it should be easier to model the fitness of individuals which are similar to those already seen than randomly generated individuals. In effect, C_m measures how well the MFM correlates with the area of the complete fitness function which includes the existing population. The two FPC measures may be calculated by the procedure in Algorithm 3.1.

The fitness prediction correlation is not to be confused with the fitness distance correlation (Jones & Forrest 1995) - this is a measure of the difficulty of problems for EAs and is completely unrelated.

Algorithm 3.1 Calculation of fitness prediction correlations

```
1: Generate random initial population $p$
2: Evaluate $p$
3: Select a subset $\sigma_1$ of $p$
4: Use $\sigma_1$ to build MFM
5: for all individuals in $\sigma_1$ do
6:     Mutate one bit in the individual
7:     Use MFM to predict fitness of individual
8:     Use fitness function to determine true fitness
9: end for
10: Calculate the correlation coefficient between the predicted and true fitnesses ($\mathrm{C_m}$)
11: Generate random population $\sigma_2$ equal in size to $\sigma_1$
12: for all individuals in $\sigma_2$ do
13:     Use MFM to predict fitnesses
14:     Use fitness function to determine true fitness
15:     Calculate the correlation coefficient between the predicted and true fitnesses ($\mathrm{C_r}$)
16: end for
```

3.2.2 Experiments

We now move on to a series of experiments following the procedure outlined in the previous section. These show the FPC values calculated using Spearman's rank correlation coefficients over a number of different fitness functions. This serves to illustrate the way in which we will discuss FPC values in later sections of the thesis. The benchmark functions and parameter settings used in these experiments are chosen using the knowledge gained in the experiments described in Chapters 4 and 5. They have been specifically chosen to demonstrate a range of conditions which give different FPC values; fitness functions with the different levels of structure, imperfect and perfect model structures, and under- and over-specified systems to estimate the model parameters. Table 3.1 lists the parameters used for each of the experiments, and the resulting $\mathrm{C_m}$ and $\mathrm{C_r}$ values. In this table, "Fit Fcn" is the benchmark problem used in each experiment: these comprised standard onemax, 2D Ising and 3-CNF MAXSAT. The bitstring length (problem size n) is given in brackets. "Structure" refers to the set of interactions used in the MFM - here the structures are fixed to *uni*variate, *bi*variate and *tri*variate with the specific variations of each as described in the descriptions of each fitness function in the literature review. "Pop Size" is the number of individuals in the starting population. "Selected" is the number of individuals selected using standard truncation selection. Finally the two different FPC values

(C_m and C_r) are given, using the two different correlation measures. Each experiment was repeated 100 times and the mean FPC value over those repeats is reported with standard deviation in brackets.

Fit Fcn	Structure	Pop Size	Selected	C_m	C_r
Onemax (100)	Uni	200	110	0.9939 (0.0012)	0.9956 (0.0028)
Onemax (1000)	Uni	200	110	0.1736 (0.1045)	0.2301 (0.0894)
2D Ising (100)	Uni + Bi	400	320	0.9277 (0.0231)	0.6615 (0.0822)
2D Ising (100)	Uni	400	320	0.4791 (0.0426)	0.0021 (0.0541)
MAXSAT (100)	Tri + Bi + Uni	2000	1700	0.9985 (0.0001)	0.9985 (0.0004)
MAXSAT (100)	Tri + Uni	600	550	0.6129 (0.0591)	0.1494 (0.0618)

Table 3.1: FPC Experiments

Firstly we can see that the FPC values vary considerably across the different experiments. We can see that in several of the experiments the two values are very close to 1; this indicates that in some circumstances the model of fitness fits extremely closely to the fitness function. We will investigate the factors which influence this in more detail over the following chapters. In all experiments bar one the C_r value is lower than the C_m value - this is because it is easier to predict fitnesses of neighbours to individuals already seen (those individuals used to build the model). The exception to this is the 1000 bit onemax with a population size of 200; the reason for the unexpected result is likely to be because this is a highly underspecified population size. We will discuss this issue in more depth in Chapter 4.

3.3 Features of the underlying fitness function

In this section we investigate in more depth the nature of the fitness model being constructed. In particular we will be looking at the coefficient values created for specific problems and the relationship these have with the underlying features of the problem. This relationship enables us to use the fitness model not only for optimisation but also to gain greater understanding of how the fitness function distributes fitness over the variables. We will look at eight problems: onemax, binval, 1D and 2D checkerboard, 4 peaks, royal road, trap-κ and bio-control in mushroom farming. Recall from Section 3.1.4 that when we refer to coefficients (*alpha* values) we can describe them as being univariate or bivariate: this is in reference to the number of variables in the clique on the MFM.

When sampling the energy distribution, minimising energy is equivalent to maximising fitness (Brown et al. 2002). For the univariate terms this means that a positive α_i will require a negative value for $V(x_i)$ to minimise the energy contribution from that term. This equates to x_i (the ith bit) being set to 0. Likewise, a negative α_i value indicates the ith bit should be set to 1 to minimise the contribution from that term. This picture is complicated by the addition of bivariate and multivariate terms to the model. In the same way as above, we can determine that a positive α_{ij} value indicates that the two bits x_i and x_j associated with it should be opposite in value, to minimise the contribution from the term involving $V(x_{ij})$. Similarly, a negative α_{ij} indicates that they should take the same value. Therefore univariate alpha coefficients directly influence the value a bit takes and bivariate alpha coefficients create a positive or negative binding interaction between the values of neighbouring bits. In general, this can be expressed as in (3.19).

$$\begin{aligned} \alpha_K > 0 \quad &needs \quad W_K(x) < 0 \\ \alpha_K < 0 \quad &needs \quad W_K(x) > 0 \end{aligned} \tag{3.19}$$

It becomes harder for a human to visualise the implications of a particular alpha value with increasing clique size above $|K| = 2$ so we will only discuss univariate and bivariate problems in this section.

In each of the experiments we follow the procedure in Algorithm 3.2.

For this study the constant will be omitted from the reported alpha values in each case because it simply provides a correction for the absolute fitness value of individuals. In this section we focus only on what the model can tell us about the dynamics of the

Algorithm 3.2 Experimental procedure

```
1: Generate random population $p$
2: Select a subset $\sigma$ of $p$
3: Use $\sigma$ to build MFM
4: Display values for each coefficient in the MFM
```

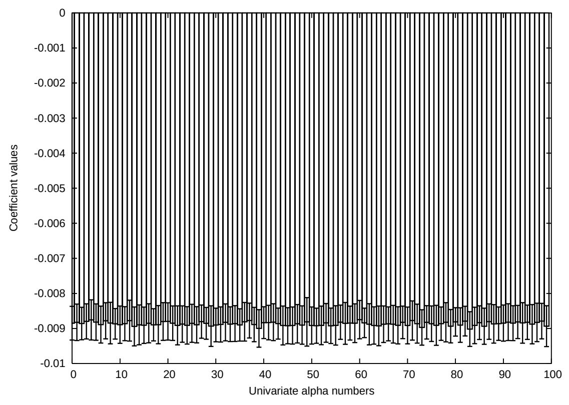


Figure 3.3: Coeffient values for the onemax problem

underlying fitness function; information about the ideal values and interactions between variables that affect the ranking of individuals not their absolute fitness.

3.3.1 Onemax

First we will look at the classic univariate problem onemax. The MFM for this problem has only univariate alphas, and we know that the optimum will be all variables set to 1. The population size used in this example was 200 with a selection size of 120. The resulting coefficient values are shown in Figure 3.3.

It can be seen that all the coefficients are given negative values of approximately the same magnitude, and all have a similar standard deviation. When sampling this model to generate new individuals the sampler has a choice of allocating either +1 or -1 (1 or 0 in the bitstring) to each variable x_i. In order to minimise energy it will give each variable a value of +1 making each term in the energy function will be negative. This fits with the dynamics of the problem, where each variable has the same influence on the overall fitness and the global optimum has all variables set to 1. This demonstrates a clear relationship between the MFM and optimisation as the global optimum can be directly related to the model parameters. This model is given in (3.20).

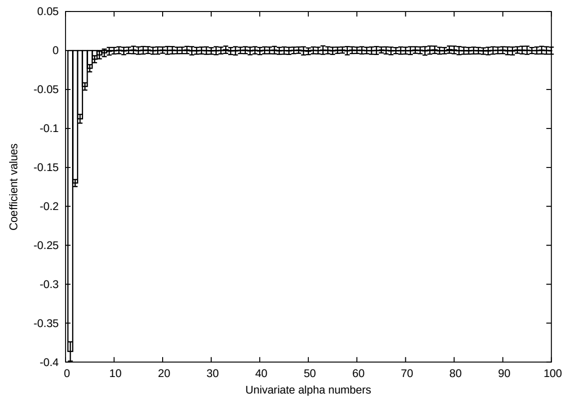


Figure 3.4: Coeffient values for the BinVal problem

$$
\begin{aligned}
Optimum : x_i &= 1 \quad \forall \quad i \\
\alpha_K &< 0 \quad \forall \quad |K| = 1 \\
\alpha_K &= 0 \quad \forall \quad |K| > 1
\end{aligned}
\tag{3.20}
$$

3.3.2 BinVal

A variant of onemax called BinVal (Droste, Jansen & Wegener 2002) can be used to illustrate the importance of the coefficient magnitude. In this problem, each bit has twice the contribution to fitness that the bit immediately to its right has - the bitstring is treated as a binary number. The previous experiment was repeated with exactly the same parameters, with only the fitness function changed.

We can see that now the magnitude of each coefficient corresponds with the impact on fitness of the associated variable. The coefficients are all still negative indicating that the optimum is still a string of all 1s. The model, given in (3.21), is as in the previous section.

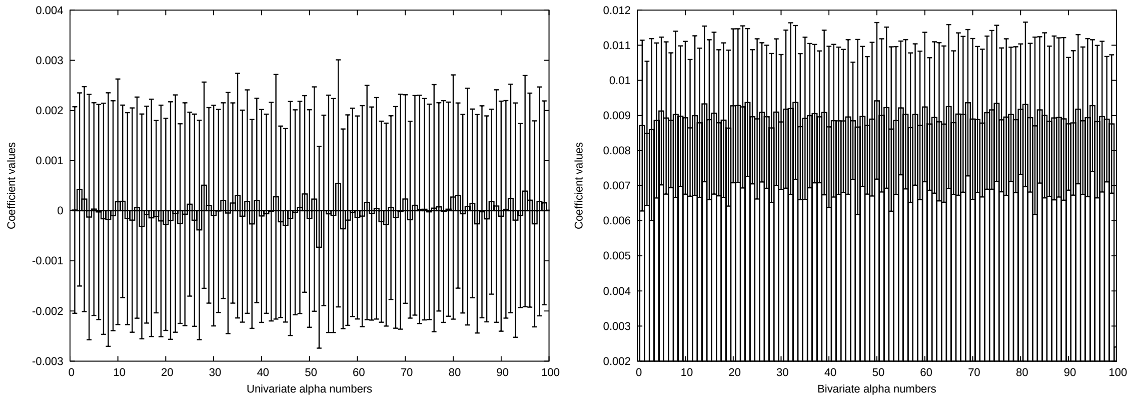
$$
\begin{aligned}
Optimum : x_i &= 1 \quad \forall \quad i \\
\alpha_K &< 0 \quad \forall \quad |K| = 1 \\
\alpha_K &= 0 \quad \forall \quad |K| > 1
\end{aligned}
\tag{3.21}
$$

These results demonstrate that the MFM cannot only tell us about the likely characteristics of the optimal solution, but also gives us additional information about the relative importance of each variable.

3.3.3 Checkerboard

We now move on to a problem which introduces a simple set of bivariate interactions to the model. 1D checkerboard suits a chain structure model and rather than being concerned with absolute values of variables the problem centres only on the values of variables relative to that of their immediate neighbours in the chain. Specifically, higher fitness is achieved when neighbouring variables are opposite in value. This experiment comprised a 100 bit 1D Checkerboard problem; the population size was 300 and the number selected was 220.

Figure 3.5 shows the univariate and bivariate alpha values for the 1D checkerboard problem. We can see that the univariate alpha values are all close to zero and the standard deviations are large with respect to their mean value indicating that they vary widely over each individual model. When allocating values to each variable the sampler will be free to set each to either +1 or -1; because the coefficients are close to zero these terms will have a negligable effect on the overall energy of the individual. This is exactly what would be expected because the 1D Checkerboard fitness function uses relative rather than absolute variable values in computing fitness. Thus the univariate alphas have varying



(a) Univariate alphas (b) Bivariate alphas

Figure 3.5: Coefficient values for the 1D Checkerboard problem

values clustered about zero indicating that each variable's absolute value is unimportant.

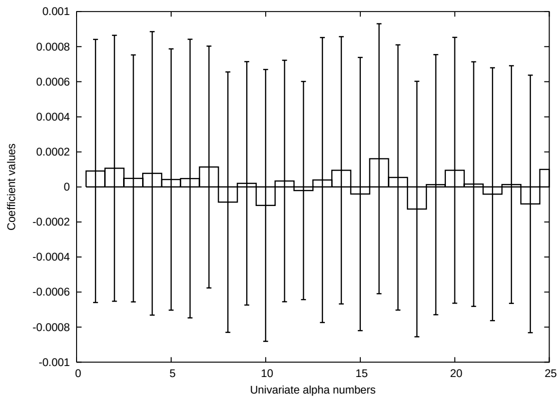
The bivariate alpha values are completely different. They are all positive and of approximately the same magnitude, and we can conclude that this is indicative of two points. Each bivariate alpha is associated with one interaction between a pair of variables. As for the onemax problem, the similar magnitude indicates that each of these interactions have a similar impact on the overall fitness. In addition to this, positive values for bivariate alphas indicate that the variables must be opposite in value (as we know from the definition of the problem). When running the sampler to find the optimum for the problem, we know that each x_i can only be set to either +1 or -1. Each bivariate term is of the form in (3.22).

$$\alpha_{ij} x_i x_j \tag{3.22}$$

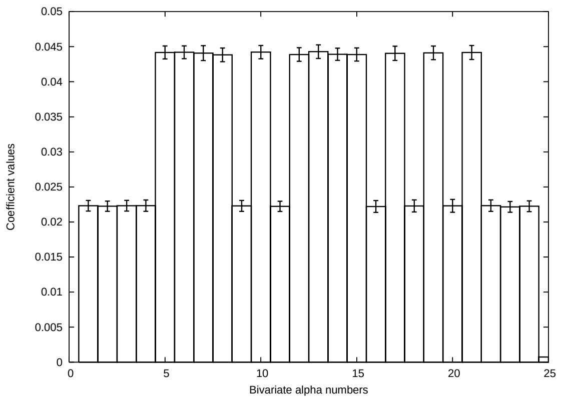
Clearly, if $x_i = x_j$ and $\alpha_{ij} > 0$ then the overall term will be positive. To minimise energy, the sampler will set x_i and x_j to be opposite in sign so that the overall term is negative. In this way we can see again that there is a clear relationship between the MFM and the global optimum.

2D Checkerboard as proposed in (Baluja & Davies 1997b) has a more complex structure, organised in a 2D lattice like a checkerboard. Interactions occur between neighbouring squares on the checkerboard without wrapping around at the edges. Again the population size used was 300 and selection size was 220. Here the problem size is 25 bits (a 5x5 lattice) to make it easier for the reader to see the effect which occurs.

As before we see that the univariate alphas are all close to zero, indicating that the absolute values of variables has no effect on fitness. Also similar to 1D checkerboard, the bivariate alphas are all positive indicating that neighbouring variables should be opposite in value; this again shows the relationship between the MFM and optimisation. In contrast, the magnitude of these is not approximately equal - half of the alpha values are approximately double the magnitude of the others. When we look closely at the particular alpha values which are high, we see that they are the ones in the middle of the lattice (that is, neither of the variables they are associated with is on the edge of the



(a) Univariate alphas



(b) Bivariate alphas

Figure 3.6: Coefficient values for the 2D Checkerboard problem

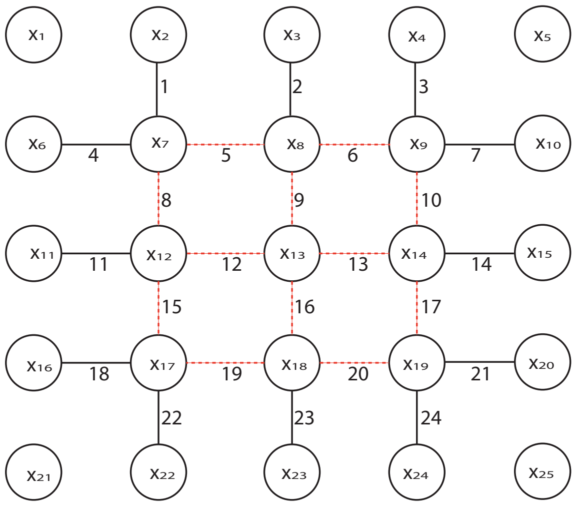
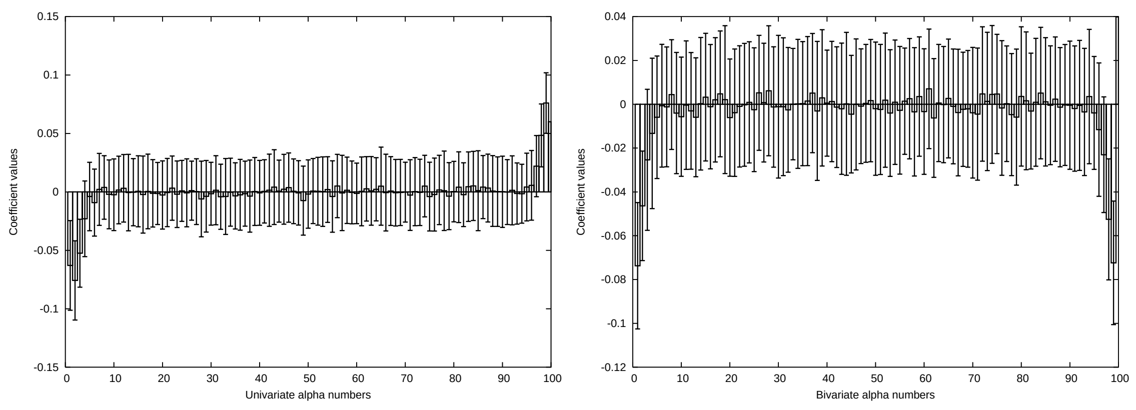


Figure 3.7: Model structure for 25 bit 2D Checkerboard Problem - numbers beside interactions correspond to those on the horizontal axis in Figure 3.6b

checkerboard). These are dashed and coloured red in Figure 3.7. We would expect these to have a greater influence on fitness than those near the edge because if they break the constraint of neighbours not matching, their neighbours will also be affected. Thus we can see that the model is placing greater importance on these alphas. This shows that the MFM provides us with more information about the fitness function than simply pointing us in the direction of the global optimum. In Chapter 6 we will demonstrate the sampling of the MFM for optimisation of these two problems.



(a) Univariate alphas (b) Bivariate alphas

Figure 3.8: Coefficient values for the 4-peaks problem

The general model for both 1D and 2D checkerboard is given in 3.23.

$$\begin{aligned} Optimum: x_i \neq x_j \quad &\forall \quad \{i,j\} \\ \alpha_K \approx 0 \quad &\forall \quad |K| = 1 \\ \alpha_K > 0 \quad &\forall \quad |K| = 2 \\ \alpha_K = 0 \quad &\forall \quad |K| > 2 \end{aligned} \tag{3.23}$$

3.3.4 4 Peaks

Four peaks was an early test problem for EDAs considering variable interactions and consequently is useful to observe the effect of such interactions on the model. This experiment was performed using a 100 bit 4-peaks problem with $T = 10$. The initial experiments on 4-peaks (de Bonet et al. 1997) were performed using the MIMIC algorithm which uses a chain structure - so the same structure was used here. Population size was 300 and selection size was 220.

We can see a clear trend for both sets of alpha values. Both sets have a length of "Don't care" alphas for the middle 80 or so variables and interactions. This is because the problem is specifically designed to mislead a genetic algorithm by encouraging the build up of a chain from either end. In the initial population, few individuals if any are likely to have a long chain of 0s or 1s to get the award. The higher fitness individuals will have a short run of 1s or 0s at the appropriate end of the chain; these will influence the α_K for $|K| = 1$. The deceptive nature of the problem is designed for algorithms that pick up these absolute values of variables but ignore interactions (that is, that the variables are in a chain).

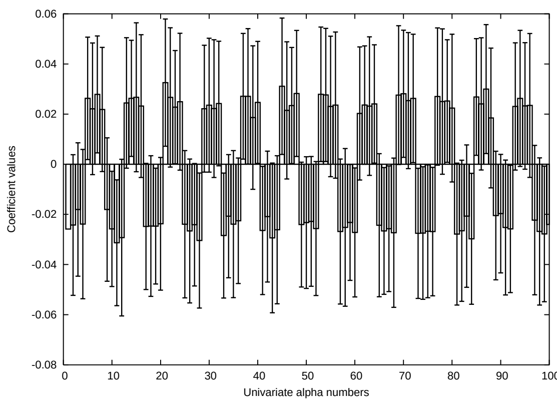
We can see that the bivariate alpha values are negative at each end of the chain which would indicate that a chain of equal-valued variables is preferred. When running the sampler to generate new individuals for optimisation, the x_i pairs being equal would make each term negative, minimising overall energy. The univariate alphas are negative for the first few variables and positive for the last few - this fits with the optimal solutions for the problem which comprise a chain of all 1s with a short chain of 0s at the end. The trend this follows is described in (3.24). Clearly for this problem, the MFM does not have enough information to point towards a global optimum as was the case with the preceeding fitness functions; this is a demonstration of the point made at the beginning of this chapter that we are only approximating part of the fitness function, not its entirety. There is however a clear indication that sampling the MFM will generate higher fitness individuals than the random starting population which would represent the first generation in an evolutionary algorithm.

$$\begin{aligned} Optimum: & x_i = \{-1, 1\} \quad \exists \quad i \\ & x_i \neq x_{\{}i+1\} \quad \forall \quad i \\ & \alpha_K \neq 0 \quad \forall \quad |K| = 1 \\ & \alpha_K < 0 \quad \forall \quad |K| = 2 \\ & \alpha_K = 0 \quad \forall \quad |K| > 2 \end{aligned} \tag{3.24}$$

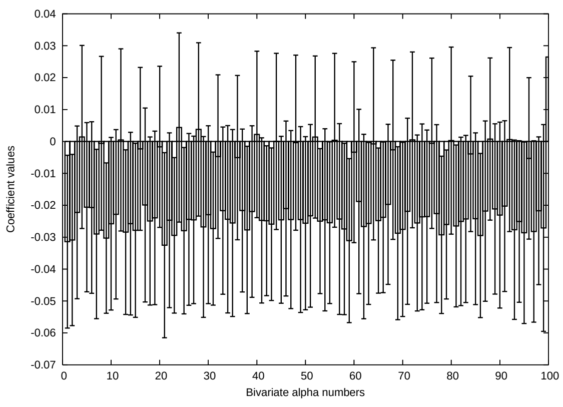
3.3.5 Royal Road

As described in Section 2.4.3, the fitness function here is slightly modified to make any observed effects clearer. The specific instance here has a block size of 4. Rather than the optimum of every group of four bits being 1111, groups alternate between 1111 and 0000.

We can see a clear pattern in both the univariate and bivariate alpha values, although it is set against a background of larger standard deviations than for the previous fitness function. The reason for this wider variation of values is likely to be a combination of the higher complexity of the problem coupled with a smaller likelihood that groups of four equal bits will appear in population. This is the reason that a larger population size is used for this problem. The univariate values reflect the alternating patterns of 0 and 1 groups, and the magnitude indicates that each bit is determined to have a similar influence on fitness. The bivariate alpha values are largely negative; this reflects the strings of bits which must be of the same value. As the sampler generates a new individual, it will set neighbouring x_i to be equal and when this is multiplied by the negative α_i the overall term will be negative reducing overall energy and consequently increasing fitness. Close observation reveals that every fourth bivariate alpha is close to zero; these fall on the boundaries between blocks. This shows the close relationship between the global optimum and the MFM. The model is described more precisely in (3.25).



(a) Univariate alphas



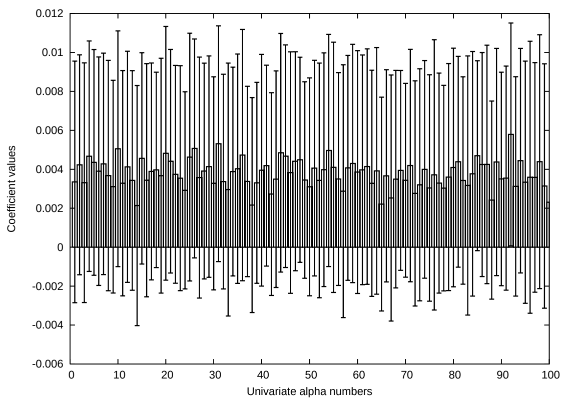
(b) Bivariate alphas

Figure 3.9: Coefficient values for the Royal Road problem

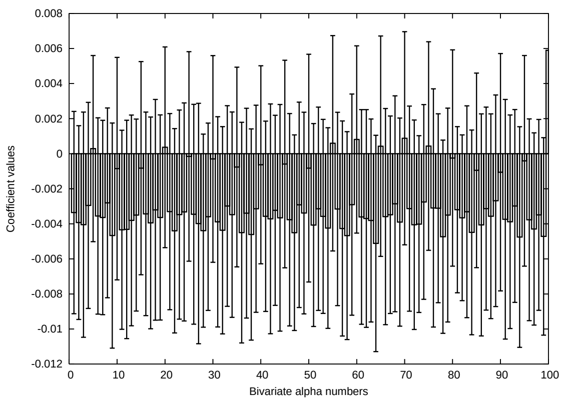
$$
\begin{aligned}
Optimum : x_i &= -1, 1 \quad \exists \quad i \\
x_i x_{i+1} &< 0 \quad \forall \quad \{i, i+1\}(i \neq n \bmod 4) \\
\alpha_K &\neq 0 \quad \forall \quad |K| = 1 \\
\alpha_K &\approx 0 \quad \forall \quad k = \{i, i+1\}(i = n \bmod 4) \\
\alpha_K &< 0 \quad \forall \quad k = \{i, i+1\}(i \neq n \bmod 4) \\
\alpha_K &= 0 \quad \forall \quad |K| > 2
\end{aligned}
\tag{3.25}
$$

3.3.6 Trap-κ

Recall that the trap functions are designed to deceive evolutionary algorithms that do not consider interactions between variables. Figure 3.10 shows the alpha values of a MFM for a 100 bit Trap-5 problem, again using the chain structure. Population size was 600 and number selected was 400.



(a) Univariate alphas



(b) Bivariate alphas

Figure 3.10: Coefficient values for the Trap-5 problem

The MFM here clearly shows the deceptive nature of the problem. The univariate alphas are largely positive, indicating the local optimum with all bits set to 0. The bivariate alphas show a similar pattern to that seen with the Royal Road problem - though largely negative (indicating chains of equal valued variables) the alpha representing the boundary between groups of five bits is close to zero. This shows that the model has learned the importance of the interactions between variables within each group of five, and is a substantial step toward overcoming the deceptive problem. Of particular note is that this is achieved using a model restricted to bivariate interactions, whereas intuition might suggest that five way interactions would also be required. The model learned here is summerised in (3.26).

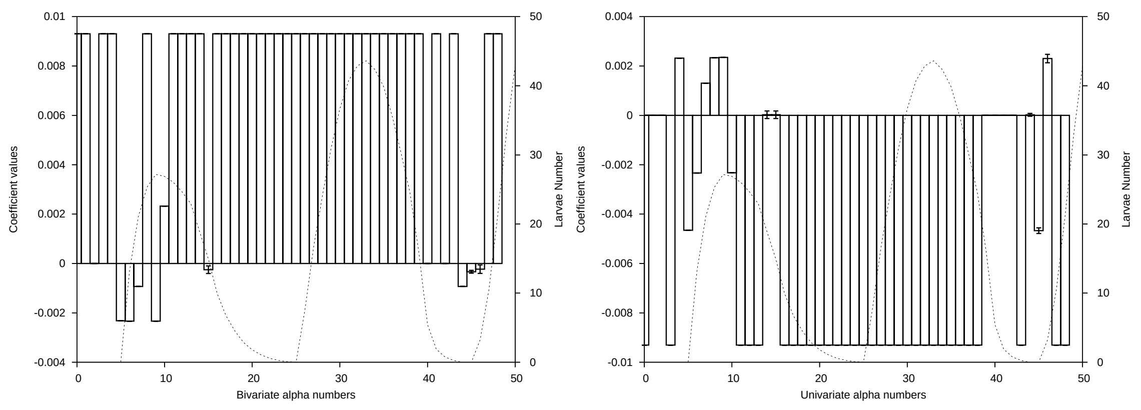
$$\begin{aligned} \mathit{Optimum} : x_i = 1 \quad \forall \quad i \\ x_i = x_j \quad \forall \quad \{i, j\}(i \neq n \bmod 5) \\ \alpha_K \neq 0 \quad \forall \quad |K| = 1 \\ \alpha_K \approx 0 \quad \forall \quad k = \{i, i+1\}(i = n \bmod 5) \\ \alpha_K < 0 \quad \forall \quad k = \{i, i+1\}(i \neq n \bmod 5) \\ \alpha_K = 0 \quad \forall \quad |K| > 2 \end{aligned} \tag{3.26}$$

3.3.7 Bio-control in Mushroom Farming

This problem differs from the others investigated in this section by being a real-world problem without a predefined set of variable interactions. The purpose of choosing this problem is to show the information about the fitness function that may be gained for a real-world problem. The application of DEUM to fitness modelling and optimisation of this problem is presented in (Brownlee, Wu, McCall, Godley, Cairns & Cowie 2008).

As we saw in Section 2.4.8, this problem features a 50-bit encoding where each bit represents a decision to apply or not apply bio-control on a particular day. As a bang-bang control problem where the representation relates to a time-series, the chain model is again an appropriate structure. The chain-based MFM used here has 50 univariate alphas and 49 bivariate alphas for the interactions between each day and the one following it. This is defined in (3.27).

$$\alpha_K \neq 0 \ \forall \begin{cases} |K| = 0, 1 \\ k = \{i, i+1\}(1 \leqslant i < 50) \end{cases} \tag{3.27}$$



(a) Univariate alphas (b) Bivariate alphas

Figure 3.11: Coefficient values for the Bio-control problem

Figure 3.11 illustrate the mean values of these coefficients after modelling the initial population over each of 100 runs of the algorithm. Error bars on each graph show one standard deviation. An addition to this graph over the previous ones in this section is the expected lifecycle of the sciarid larvae. Population size was 120 individuals - in this experiment no selection was used. A more detailed discussion of choice of selection operator is given in Chapter 5.

We note from Figure 3.11 that the coefficient values have very small standard deviations, so there is a very strong signal-to-noise ratio in the coefficients produced from randomly-generated initial populations. We now interpret the model coefficients in terms of the practical problem. We see that the alpha coefficients are predominantly positive, indicating that intervention should not take place most of the time. As with the previous examples, x_i will be set to -1 by the sampler to minimise energy, this represents setting the bit to 0. However from days 5 to 10 the univariate coefficients are predominantly negative indicating that intervention should take place in this period - x_i will be set to $+1$ by the sampler to minimise energy. We can also see that this matches the predicted point at which the larval population should significantly grow. Beta coefficients are predominantly negative indicating that, at most points in the treatment period, control should not switch. The relates x_i will be opposite in sign to minimise energy in the sampling process. As this coincides with alpha coefficients that indicate no intervention, it is clear that intervention should not take place on these days. Positive beta coefficients, indicating a switch of control, are mainly concentrated around the same point of the treatment period where intervention is indicated by the alpha coefficients, indicating that the control will switch on and off again during this period.

This analysis shows that essential information about the dynamics of the control system is detected by the model even from random initial populations. A mushroom farmer presented with this information alone could implement a dosing schedule that is close to optimal.

3.4 Summary

This chapter has given us the necessary terminology and theory to extend the existing DEUM framework to incorporate multivariate interactions of any order. Previous work had demonstrated the framework using an MFM with univariate structure and the 2D lattice of the Ising problem. A major contribution of the thesis is that here we have demonstrated how the model may related to Walsh functions which may be used to represent any bitstring encoded fitness function with interactions of any order. We have also been able to give two specific examples using structures not previously used for the MFM - the chain structure and the trivariate structure of 3-CNF MAXSAT.

We have developed a measure of fitness prediction capability termed the Fitness Prediction Correlation. When conducting optimisation, sampling the MFM to generate new individuals requires that the MFM is able to predict the change in fitness resulting from mutations. We will use this measure over the coming chapters to discuss the fitness prediction capability of the MFM in a number of contexts.

Finally, Section 3.3 has shown that given the right structure the parameters of the MFM show a clear link with the global optimum and other underlying features of a fitness function after relatively few fitness evaluations. This is an important addition to previous work using the MFM for optimisation within the DEUM algorithm as analysis of the parameters in this way has not been conducted for any fitness function. The only information supplied to the algorithm in each of the experiments was the model structure to be used - often a simple structure such as the chain model - and the fitness values for individuals by an essentially black-box function. The α_K parameters of the resulting model can be demonstrated to have a clear and direct relationship with the variables in the fitness function, proving that the MFM can be used to build an accurate model of a fitness function. As well as allowing us to better understand the MFM, this property can help us to see the characteristics of fitness functions. Potential applications for this include decision support tools and specialised genetic operators which are guided by the α_K values to target changes to specific variables that are known to have a large influence on fitness.

Chapter 4

Effect of population size on fitness model

There are many factors which affect the performance of an evolutionary algorithm such as population size, selection operator, reproduction operators and encoding. Similarly, there are many factors which affect how closely the probabilistic model relates to the fitness function. In turn this has an influence on the effectiveness of the fitness model as a tool for optimisation or analysis of the fitness function. In Chapter 3 we described the fitness prediction correlation (FPC) as a means of measuring the fitness modelling capability of the model. In this chapter we will use this measure to investigate the effect of population size on the quality of fitness model constructed as well as the effect of changing the interactions which are included in the model structure. The chapter will explore two general questions. Firstly, how the population size relates to the quality of the model; this is directly related to the effort required to learn a model which closely approximates the fitness function. Secondly, what the trade-off is between model complexity and model quality; essentially whether in reducing the model complexity we can still retain a good model of fitness by using more fitness evaluations to estimate the model parameters. Both of these are factors in determining the computational cost of building a model with a strong correlation with the fitness function.

Algorithm 4.1 Effect of Population Size Experiments

```
1: for each population size do
2:     Generate random initial population $p$
3:     Use $p$ to build MFM
4:     Calculate FPC values $C_m$ and $\mathrm{C_r}$
5: end for
```

4.1 Experiments

Each of the experiments described here follow the procedure in Algorithm 4.1. The range of population sizes are relative to the size of the MFM. Specifically, this is the number N of α_K (unknowns) present in the model which is equal to the number of Walsh functions. Every experiment looked at a range of population sizes. Recall from Chapter 3 the definitions of under-, over-, and perfectly- specified systems - the sizes used these experiments cover each of situations. These were chosen after preliminary results indicated that the largest change in FPC values occurred around the point at which population size exceeded N, the transition from under- to over-specified.

Below N, population sizes of N multiplied by 0.5, 0.8, 0.9, 0.95, 0.97, 0.98, 0.99 were used. Above N, population sizes of N multiplied by 1.01, 1.02, 1.03, 1.05, 1.1, 1.2, 1.5 and 2 were used. In addition to these, additional population sizes of $N \pm \{1, 2, 3\}$ were used to provide extra detail around the transition from under- to over-specified.

In assessing the computational cost to build the model we can also vary the number of model parameters. Recall from Chapter 3 that we have several classes of model structure - full, perfect and imperfect, as well as filtered and decimated which are special cases of imperfect structure. Full structures grow in size exponentially with the number of variables, which renders the resulting model extremely expensive to compute. Thus we will not consider full structure in this chapter. The experiments will look at the other structure types. The perfect structures are derived from the problem definition and the imperfect structures are created by removing some cliques from the perfect structures to produce either decimated or filtered structures. We will define each structure in terms of the Walsh functions which specify the set of cliques in the model.

One final thing to note is that these experiments do not use selection; the entire population is used to estimate the model parameters. Full discussion of this issue will take place in Chapter 5.

We now move on to the experiments, divided into the different benchmark functions used. In addition to this, for each benchmark function we looked at a range of different problem sizes n, problem size being the number of variables in the problem. To allow an appropriate balance between the space taken by figures and text the full set of results for all experiments are presented in Appendix B. For each experiment, a subset of the result graphs in the appendix for specific problem sizes are duplicated alongside the discussion on the results for ease of reading. As each run was repeated multiple times, we give mean figures with standard deviations. To ensure this was valid, the statistical package SPSS was used to perform the Shapiro-Wilk test for normality on the result sets. A selection of results intended to show a spread across benchmark functions, problem sizes, population sizes and structure types is shown in Table 4.1. Figures below 0.005 indicate a statisically significant probability that the data is not normal; all figures are above these so we can use the mean and standard deviation.

Experiment	C_m normality	C_r normality
Onemax 1000 bit, pop size 0.5N	0.587	0.086
Onemax 1000 bit, pop size 2N	0.992	0.974
2D Ising 100 bit, perfect structure, pop size 2N	0.109	0.107
2D Ising 100 bit, univariate structure, pop size 2N	0.204	0.600
Maxsat 100 bit, perfect structure, pop size 2N	0.309	0.006

Table 4.1: Shapiro-Wilks normality test results

4.2 Onemax

4.2.1 Structure

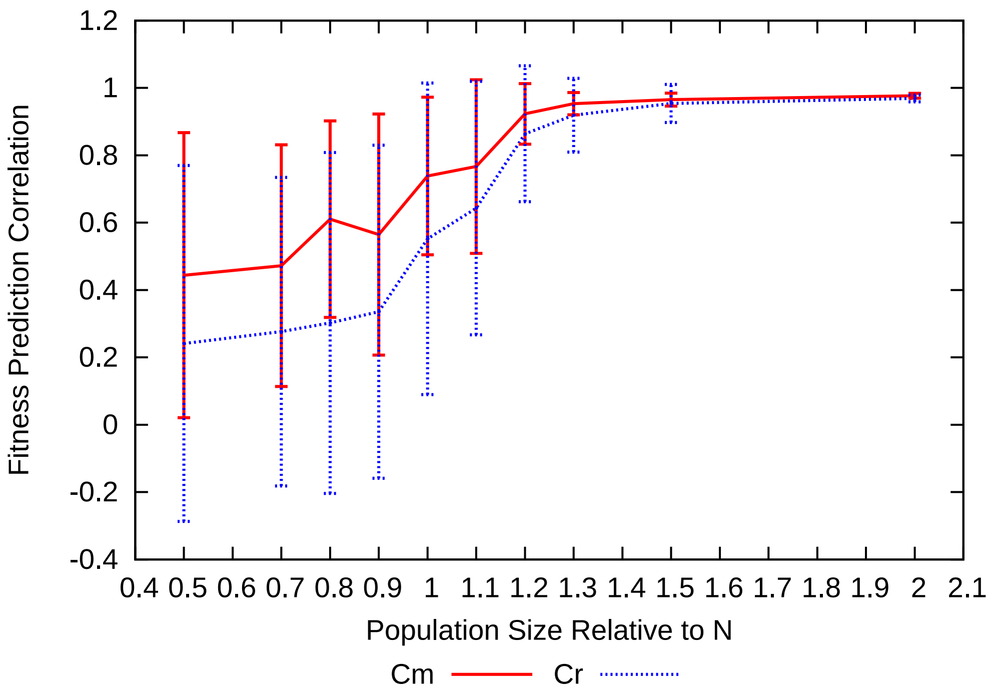
First we look at the onemax problem, which has a univariate structure. The perfect model structure for a univariate problem was given in Chapter 3 and repeated in (4.1). There is one term for the zero clique and one term for each variable x_i. This gives us $N = n + 1$, where N is the number of α_K in the model and n is the number of variable x_i in the problem.

$$U(x) = -\ln(x) = \sum_K \alpha_K W_K(x) \qquad |K| < 2 \tag{4.1}$$

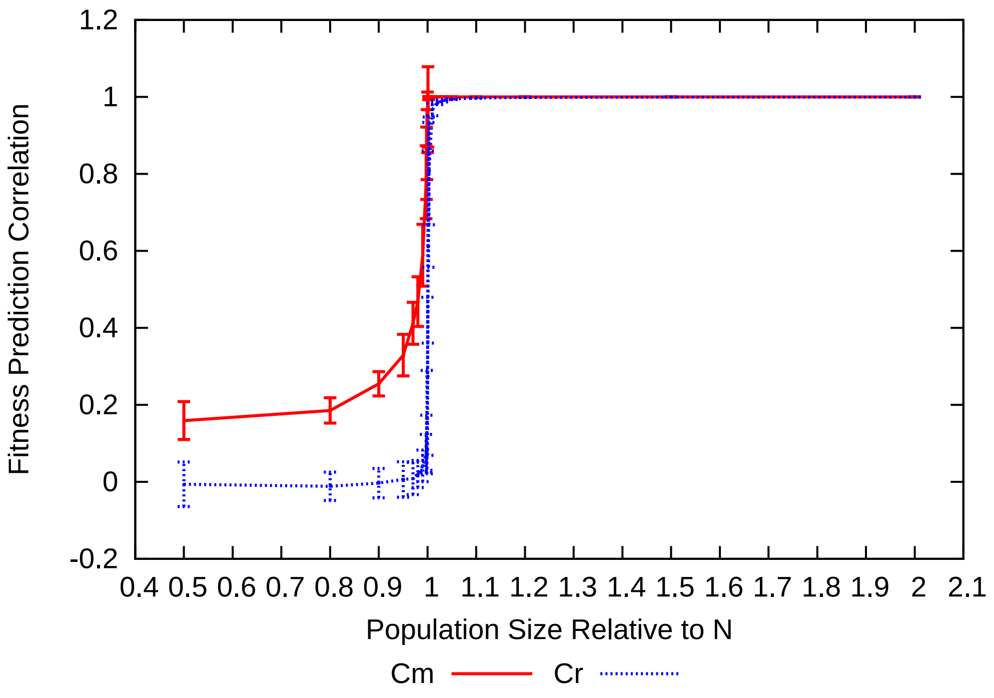
4.2.2 Results

We look at nine different sizes n of onemax ranging from 10 bits to 1000 bits; all are shown in Appendix B in Figures B.1 to B.9. Illustrated here in Figure 4.1 are the results for $n = 10$ bits and $n = 1000$ bits.

The graphs in Figure 4.1 shows the fitness prediction correlation values over the range of different population sizes. The C_m and C_r values are computed as described in Chapter 3. The values are averaged over 30 runs and error bars indicate one standard deviation. We would expect that the correlation between true and predicted fitness for mutated individuals (C_m) is higher than that for randomly generated individuals (C_r) because of their similarity to the individuals used to build the model. We can see that this is the case for each of these results. We can also see a clear pattern over all the instances of onemax; at the point at which the population size exceeds the number of α_K in the model there is a rapid increase in both FPC values. This is the point at which the number of knowns in the system of energy equations matches then exceeds the number of unknowns. Prior to this point both FPC values are between 0 and 0.6; this means there is only a very weak positive correlation between true fitnesses and those predicted by the model which indicated that the model is not closely fitted to the fitness function. Beyond this point both values rise to close to 1 (typically over 0.95, always over 0.6), a strong positive correlation which indicates that the model is very closely fitted to the fitness function. The increase in FPC values becomes sharper with increasing problem size. The high correlations between predicted fitnesses and true fitnesses are important because when we come to directly sampling the model for optimisation the MFM must be able to accurately predict the influence on fitness of mutations. High C_r values show that even after many mutations (equivalent to a randomly generated individual) the MFM can still accurately predict the fitness of an individual. An experiment illustrating this in practice is presented



(a) 10 bits



(b) 1000 bits

Figure 4.1: FPC against population size for onemax problems

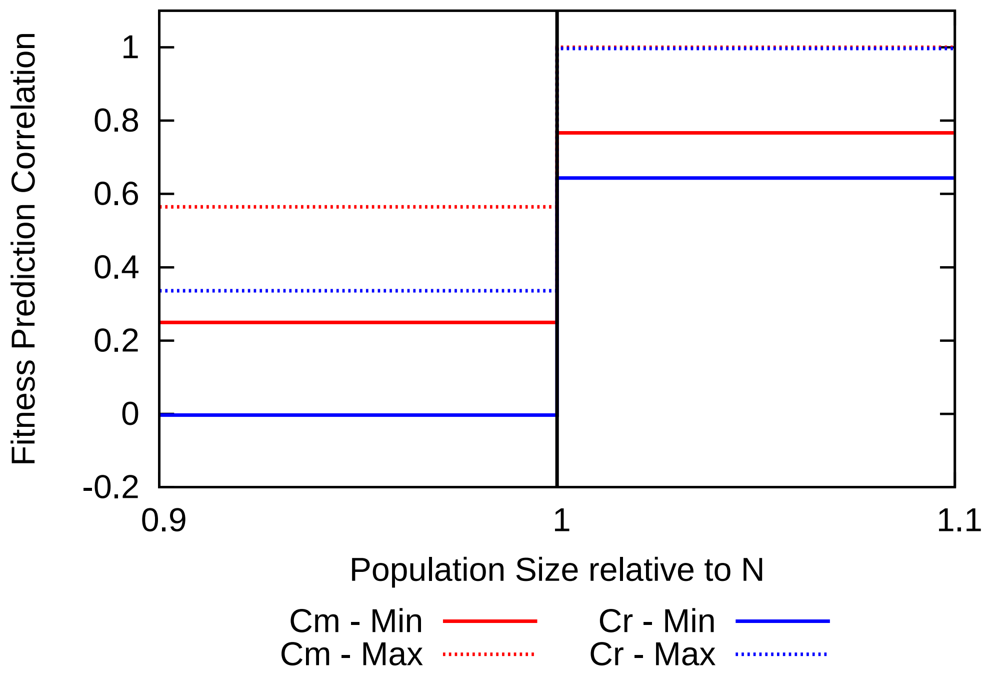


Figure 4.2: Summary of FPC against population size for onemax problems

in Section 6.2 alongside the discussion on directly sampling the MFM for optimisation.

We summarise the results for the series of experiments on onemax in Figure 4.2. This figure is intended to show the overall trend across the transition from under-specified to over-specified. We plot the minimum and maximum C_m and C_r across all problem sizes at population sizes $0.9N$ and $1.1N$. We can again see the sharp increase in C_m and C_r as population size exceeds N. This style of summary graph will be used to present the results in subsequent experiments; the individual graphs for all problem sizes in each experiment can be found in Appendix B.

4.3 1D Checkerboard

4.3.1 Structure

Figures 4.3 and B.10 to B.27 shows the results for the 1D checkerboard problem. The perfect structure for this problem is a chain; this was also given in Chapter 3 and is repeated in (4.2). The model includes both bivariate and univariate terms; this means that $N = 1 + n + (n-1) = 2n$.

$$U(x) = \alpha_0 + \sum_{i=1}^{n} \alpha_i x_i + \sum_{i=1}^{n-1} \alpha_{ij} x_i x_j \tag{4.2}$$

As this problem also has interactions between variables we can repeat the experiment with filtered model structures that have some of the cliques removed. This will allow us to compare the models to determine the influence that the interactions have on fitness modelling capability, as well as seeing if the relationship between FPC and population size is the same independent of model structure. For this 1D checkerboard we ran the experiments with two filtered structures; one with all singleton cliques ($|K| = 1$) removed and one with all 2-cliques removed ($|K| = 2$). We will refer to these as the bivariate model and univariate models respectively, indicating the cliques which remain in the structure.

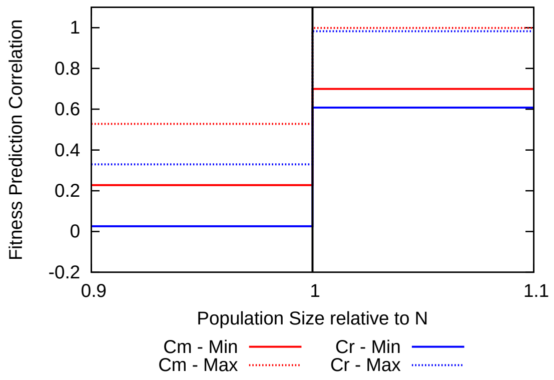
4.3.2 Results

In Figure 4.3 we give the summarised results for experiments on 1D checkerboard with each of the three model structures. The experiments on 1D checkerboard were run over the same set of problem sizes as for onemax, with the omission of $n = 1000$ which was omitted due to limitations on computational resource. We see a similar effect to that seen for onemax: C_m is higher than C_r for small populations but the mean values for both are still below 0.55 (and below 0.3 for larger instances) indicating a poor fitness model. As the population size increases to then exceeds N both values jump rapidly over 0.6. Again this increase becomes sharper with increasing problem size. The C_r of 0.6 achieved with a population size of 1.1N does not represent as strong a correlation between predicted and true fitnesses as that seen for onemax - it indicates that the model often incorrectly predicts the change in fitness caused by mutations. This is likely to have an adverse affect on the sampler which uses mutations for optimisation and we see in Section 6.3.1 that population size must be increased to 2N to achieve effective optimisation. When using a univariate model - one with all interactions removed - we can see that C_m values follow the same trend as displayed with the perfect model structure although with a greater variation in magnitude across problem sizes. C_r values remain below 0.15 irrespective of population size which indicates that the model is able to accurately predict fitness for neighbouring individuals but not randomly generated ones. This is because mutation of one bit will have an effect on fitness and the model can predict the likely change using the absolute variable values. In contrast, to predict the fitness of a randomly generated individual the model must account for the relative values of variables by using interactions between them. This is better explained by referring to the associated energy functions. In (4.3) we show the subset of the terms in the energy function which relate to a variable x_i, when using a perfect structure. We can see that the total energy (and hence fitness) is influenced by the value of x_i and its immediate neighbours in the chain. If we remove the terms for the neighbouring variables, we can predict the direction of change in overall energy if x_i changes as the other two terms will either cancel out or add to the value of the first term. However, if any of the neighbours also change this will no longer be the case.

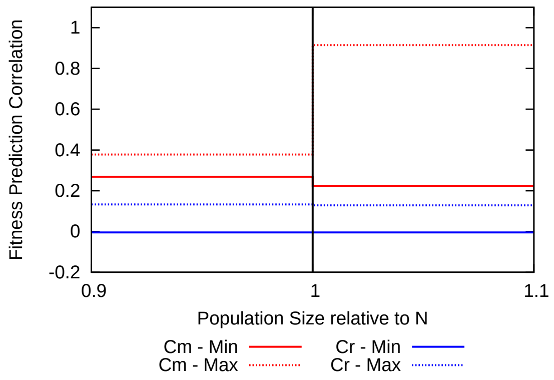
$$U(x_i) = \alpha_i x_i + \alpha_{i-1,i} x_{i-1} x_i + \alpha_{i,i+1} x_i x_{i+1} \tag{4.3}$$

Figure 4.3d shows the FPC values for increasing population size with a univariate structure with $n = 500$ bits. This shows an additional trend not visible in the summary; with the univariate structure the C_m values decrease as the population size moves towards 2N. This could be attributed to conflicting information being present in the population as to the effect of each variable. As the population size increases, the number of $x_i = 0$ and $x_i = 1$ will tend towards equality so the α_i will tend towards zero and the model will become unable even to make fitness predictions as described in the previous paragraph. (in Section 3.3.3 we saw that the model stores most of the information about fitness for 1D checkerboard in the bivariate terms; these having non-zero values in comparison to the near-zero values of the univariate terms)

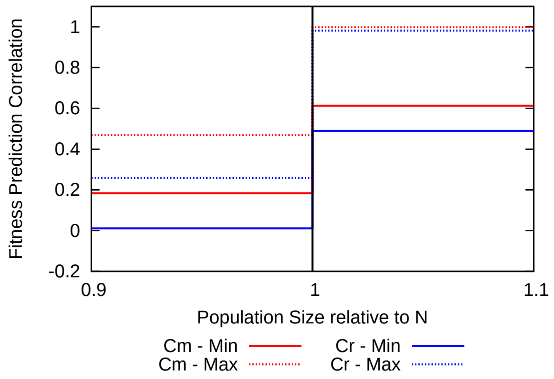
With only bivariate terms present in the model structure, both FPC values follow the same pattern as shown by the perfect model structure. In absolute terms the population is now smaller as N is lower (not having any univariate terms in the model). This result combined with that for the univariate model would indicate that although the univariate



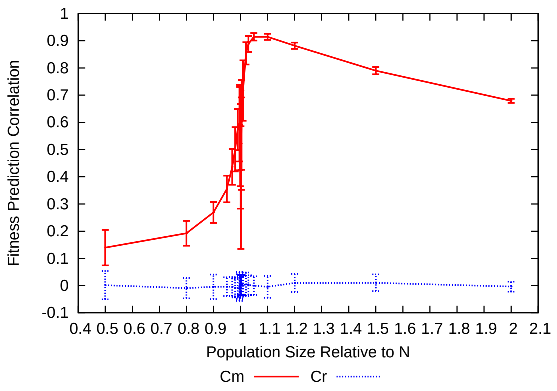
(a) Perfect model structure



(b) Univariate model structure



(c) Bivariate model structure



(d) Univariate structure on 500 bit problem

Figure 4.3: Summary of FPC against population size for 1D checkerboard problems

terms are helpful in building a good model of fitness for this problem, they are not needed to accurately predict the fitness of randomly generated individuals. This is an interesting result as the model with univariate terms filtered out is less expensive to build, of particular relevance for optimising real-world problems where a large number of interactions may exist but not need to be included in the model for efficient optimisation (a similar effect was seen comparing univariate and multivariate algorithms on a problem with a large number of interactions in (Brownlee, Pelikan, McCall & Petrovski 2008)).

Before moving on, it is also worth noting that for both benchmark functions we have considered so far the rapid increase in fitness prediction capability as M exceeds N becomes sharper with increasing problem size n. This is likely to be a result of the least-squares fitting used by SVD to compute the α_K values. For these experiments, as n increases, so do the absolute values of N and M and with a larger number of variables over which to blanace the error it becomes easier for SVD to stably fit the α_K.

4.4 2D Ising

4.4.1 Structure

The next set of experiments looks at the 2D Ising problem, which has a 2D lattice structure. The model for an l x l instance of the problem is given in (4.4) (note that the lattice structure requires us to use a slightly different notation for the variables, where a variable in row i and column j of the lattice is x_{ij}). The number of parameters in the perfect model is $N = 3n + 1$. As before each experiment was run 30 times at each problem size; each repeat used a randomly selected instance of Ising from the set we discussed in Chapter 2. The problem sizes n are different to those for the previous problems because the variables must be arranged into a square. As well as the perfect model structure, we look at a filtered structure with only univariate terms (that is cliques with $|K| > 1$ are removed). We also look at two decimated structures, with a random 10% and 50% of cliques with $|K| = 2$ removed.

$$U(x_i) = \alpha_0 + \sum_{i=1}^{l}\sum_{j=1}^{l}(\alpha_{ij}x_{ij} + \alpha_{ij,(i+1)j}jx_{ij}x_{(i+1)j} + \alpha_{ij,i(j+1)}jx_{ij}x_{i(j+1)}) \tag{4.4}$$

4.4.2 Results

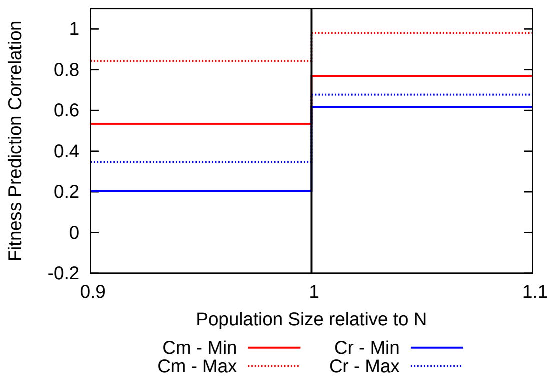
The results for these problems are shown in Figures 4.4 and B.28 to B.36. In common with the results for the other problems, with the perfect model structure as the population size increases above N both C_m and C_r increase towards 1. This means that the MFM can accurately predict the change in fitness caused by mutations and the sampler will be able to generate solutions of higher fitness during optimisation; this was demonstrated in (Shakya et al. 2006) and is shown alongside structure learning in Chapter 7. This is only possible because the perfect structure is supplied to the algorithm; at the point that the population size exceeds N the system of energy functions becomes fully specified and the model parameters can be estimated accurately. For populations smaller than N, we see C_m values between 0.5 and 0.8 (a moderate to strong positive correlation), in contrast with the low C_m values calculated for other fitness functions with population sizes less than N. Though the C_r values at 0.2-0.3 are also higher than for many of the other fitness functions studied, little conclusion can be drawn for C_r as these values represent a statisically insignificant correlation between predicted and true fitnesses. The result of this is that the Gibbs sampling approach used for optimisation in Chapter 6 and 7 does not work until the population size is increased beyond N individuals; it requires the MFM to be able to predict changes in fitness after many mutations. This is shown in Section 6.2. We see the same effect with a 10% decimated model structure; though C_r values are lower with the decimated model as it is less able to as accurately account for all interactions present in the problem.

When using a MFM with a 50% decimated structure or only univariate terms we see little if any increase as M exceeds N. C_m is above 0.4, and over 0.7 for larger problem sizes; the removal of interactions from the model results in the correlation for randomly generated individuals staying close to zero regardless of population size. The unusually high values for C_m can be attributed to a specific characteristic of the relationship between 2D Ising and the Markov network, which we now move on to explore.

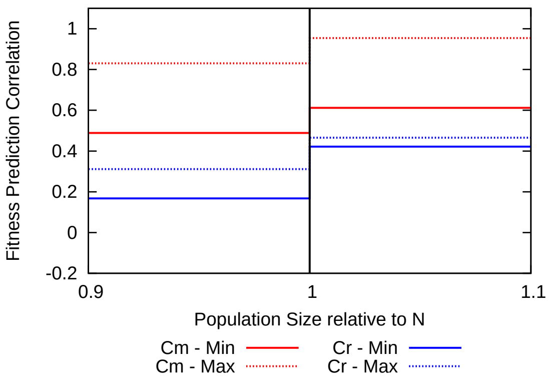
4.4.3 Behaviour around M=N

In addition to the summaries, the result for the experiment using a perfect structure on 400 bit 2D Ising problems is also given in Figure 4.5. Provided for comparison with the similarly structured 2D checkerboard problem, the full set of figures for which are B.64 to B.72. This is included to illustrate a trend only seen with 2D Ising within the benchmarks used for these experiments.

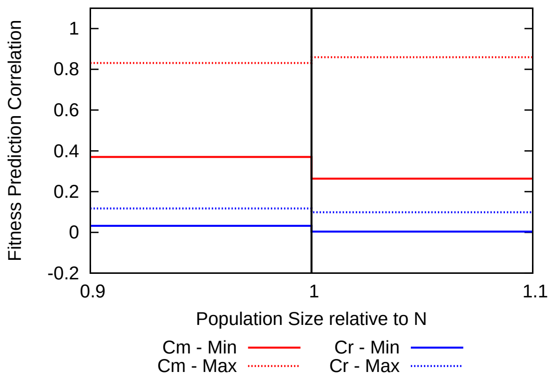
The correlations deteriorate very sharply as the population size nears the model size and then improve sharply again. This is not observed with the similarly-structured 2D checkerboard problem. The probable reason for this difference in behaviour lies in the



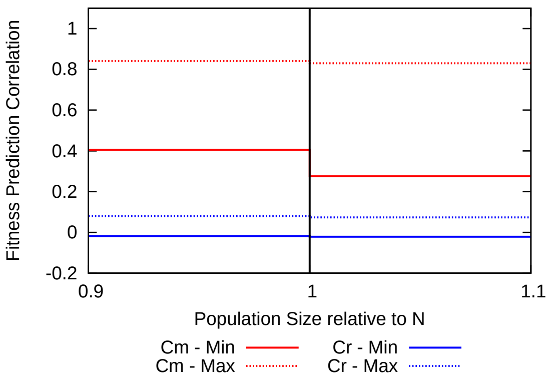
(a) Perfect model structure



(b) 10% decimated model structure

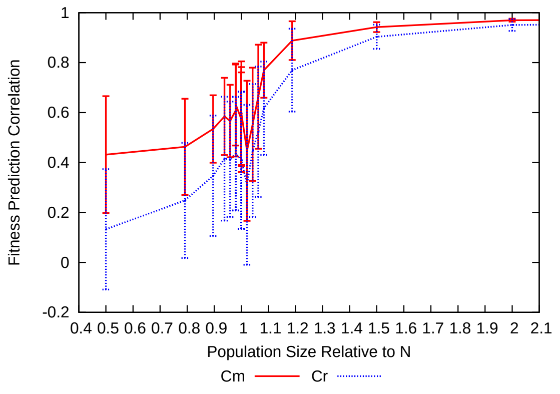


(c) 50% decimated model structure

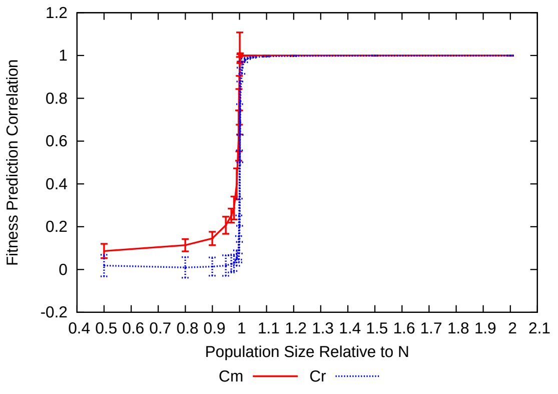


(d) Univariate model structure

Figure 4.4: Summary of FPC against population size for 2D Ising problems



(a) 2D Ising



(b) 2D Checkerboard

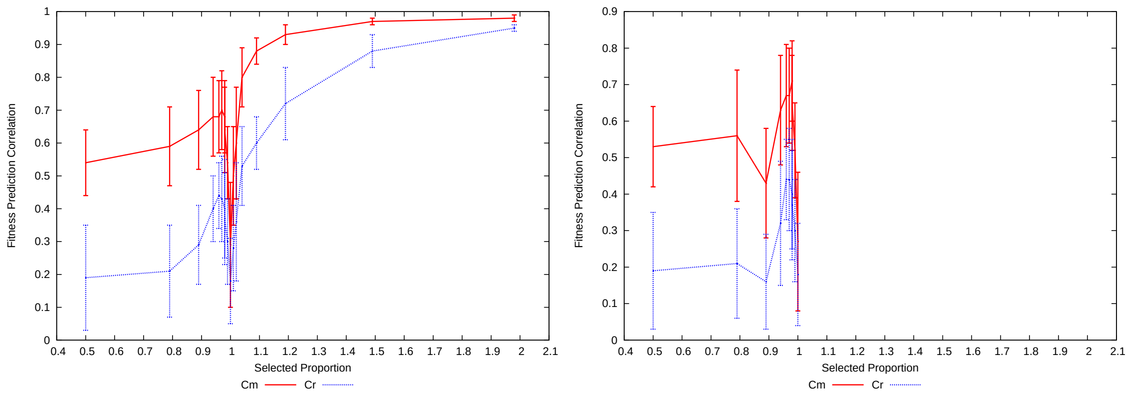
Figure 4.5: Comparison between 2D Ising and 2D checkerboard for 400 bit instances with perfect model structure

fact that the forms of the Ising problem fitness function and the general Markov network model are algebraically identical (the Walsh trasformations of the Ising problem match the structure of the Markov network). When solving an underspecified system, the SVD algorithm balances out variation in fitness not attributed to variation in the population uniformly across the population. In the case of individuals that are one-step mutations away from those solutions used to construct the model, it is unsurprising that there should still be a strong correlation with fitness, given the closeness in form of the model and fitness function. For the same reason however, when the number of solutions is very close to the model size, the model will be prone to overfitting to the particular population and so even small mutations may result in poor correlation with the change in resulting fitness. When solving an overspecified system, SVD performs a least squares fit of the model. Once beyond the overfitting point, this gives very good correlations.

4.4.4 SVD Type

A possible cause for the effect being observed with 2D Ising may be an instability in the SVD implementation. For this reason the experiments were repeated with two other SVD implementations; COLT (Hoschek 2004) and NETLIB SVDPACK (Berry, Do, O'Brien, Varadhan 1993). As fitting and error minimisation are performed by the SVD routine it is likely that this would be the source of any potential error. It would be unlikely that any problem would arise from the backsubstitution routine which substitutes the energy values into the singular value decomposition to find the α_K - common to all three algorithms. Figure 4.6 shows the results of the experiment detailed in section 4.4 carried out on the 64 bit 2D Ising problem using NETLIB SVDPACK and COLT respectively. NETLIB SVDPACK failed when the system became overspecified. However, the anomaly for Ising was in the underspecified part of the results so for comparison this is unimportant.

We can see that the results show the same pattern and conclude that the difference in behaviour observed for the Ising problems can be attributed to the strong similarity between model and fitness function, or at least is in some way specific to the problem rather than the implementation of SVD.



(a) COLT SVD (b) NETLIB SVD

Figure 4.6: FPC against population size for 64bit 2D Ising problems using a full model and different SVD implementations

4.5 MAXSAT

4.5.1 Structure

Figures 4.7 and B.73 to B.99 shows the results for a range of 3CNF MAXSAT problems. Again the experiment was repeated 30 times at each problem size. For this problem, each repeat used a different randomly selected instance of MAXSAT from the repository at SATLIB (Hoos & Stützle 2000). The problem sizes were thus restricted to those with corresponding instances available at SATLIB and for experiments involving a more dense model structure the larger instances had to be omitted due to available computational resources (for example - with the perfect structure the experiment stopped at $n = 100$). The perfect structure is defined in (4.5); the 3-cliques are derived from the 3 variable clauses in each instance of the problem. The number of 3-cliques is the same for each instance at a given problem size, but the number of 2-cliques can vary dependent on the number of pairs of variables which occur in more than one clause together. Thus the value of N and the corresponding absolute population sizes varied for experiments where the bivariate interactions were included in the structure. We ran the experiments on three imperfect model structures. Again a univariate structure is used (with cliques where $|K| > 1$ removed) to see the effect of removing all interactions. A structure with only trivariate and univariate terms (that is cliques where $|K| = 2$ removed) is tried to determine the effect of removing non-maximal cliques. Also, a structure with 50% decimation is included; in this half of the cliques where $|K| > 1$ are removed.

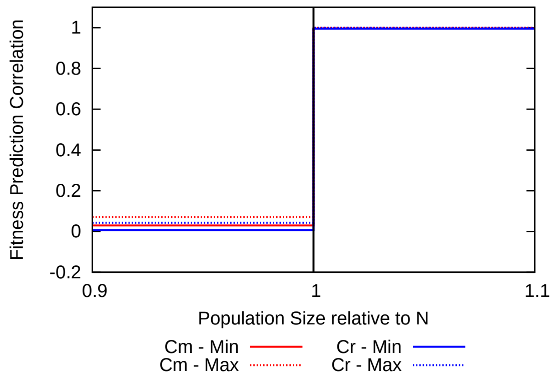
$$U(x) = \sum_K \alpha_K W_K(x) \quad where \quad \alpha_K \neq 0 \ \forall |K| \leq 3 \tag{4.5}$$

4.5.2 Results

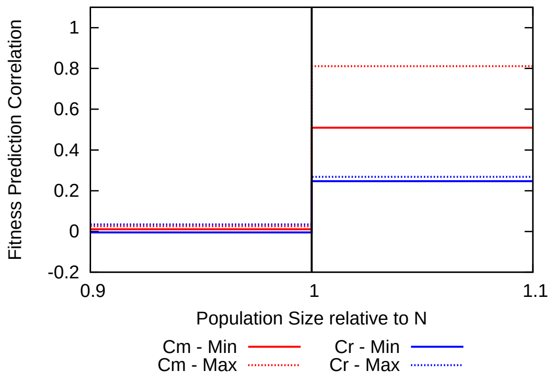
For the perfect model structure, we see a similar effect to that found with onemax and checkerboard, with an even sharper jump - for $M < N$ both C_m and C_r values are below 0.05 whereas for onemax C_m was noticeably positive (0.2-0.3). This is likely caused by the greatly increased complexity of the MAXSAT problem; it is harder to build a model which closely fits the fitness function. For $M > N$ both C_m and C_m are greater than 0.99 indicating a very strong positive correlation between the model and the fitness function. This means that the MFM is able to predict fitness changes for an individual after many mutations and this is employed for optimisation in Chapter 6.

When using a model which only includes univariate and trivariate terms we see the same effect for both FPC values, but neither reach the same absolute level. C_m rises to over 0.5 (and higher for larger problem sizes) but C_r does not rise over 0.3. This indicates that although the bivariate interactions effectived duplicate the trivariate interactions they are still important for building a model of fitness which strongly correlates to the fitness function. The patterns are very similar for the 50% decimated model.

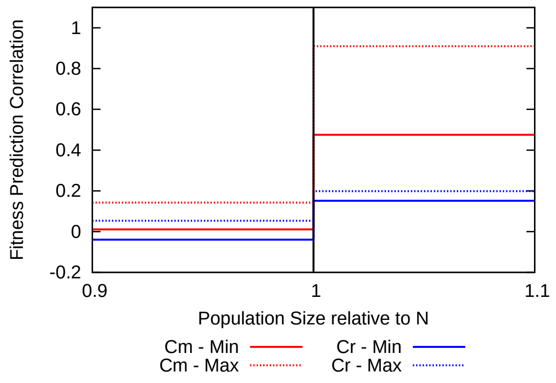
With the univariate model C_m values follow a similar trend to that seen for the univariate model on 1D checkerboard; they increase from under 0.2 to over 0.5. Again this is because the univariate model can be used to predict the fitness change for a single bit mutation with a reasonable accuracy. The C_r values follow a similar pattern but do not reach a particularly high value, especially when compared to the results for the perfect structure. This illustrates the importance of the higher order interactions in modelling the fitness of this more complex problem.



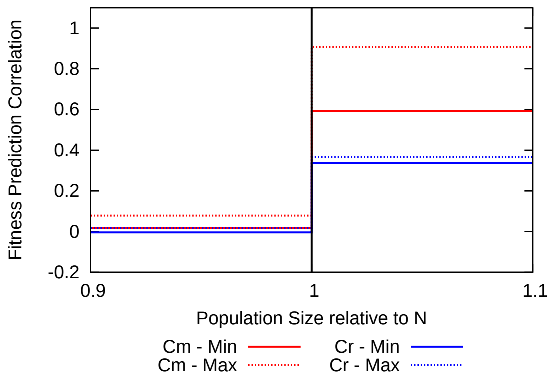
(a) Perfect model structure



(b) Trivariate + univariate model structure



(c) Univariate model structure



(d) 50% decimated model structure

Figure 4.7: Summary of FPC against population size for 3-CNF MAXSAT problems

4.6 Trap-κ

4.6.1 Structure

In Figures 4.8 and B to B.114 we see the results for a Trap-κ problem, in this instance with $\kappa = 5$. This problem is of interest at this point because the structure contains more complex interactions than the previous problems. Groups of κ variables must be set to 0; so the absolute value of a variable is important, as is its value relative to the others in its group. This means that the model has the potential to require up to κ-order interactions. For these experiments, it is difficult to identify a perfect structure. Including all possible interactions to order 5 would result in a model with a large number of α_K to find for even small problem sizes (remember from Chapter 3 that the number of interactions grows by $O(2^n)$). Thus we use a structure which is likely to be imperfect but is a best-guess at a close match to the perfect structure. We refer to this as an order-5,2,1 structure, which includes a term for each group of 5 bits as well as set of terms representing a chain structure and terms for the individuals x_i and contant. This is defined formally in (4.6), where b_l are the blocks of 5 x_i.

$$U(x) = \alpha_0 + \sum_{i=1}^{n} \alpha_i x_i + \sum_{i=1}^{n-1} \alpha_{i,i+1} x_i x_{i+1} + \sum_{l=1}^{n/k} \alpha_l b_l \tag{4.6}$$

We also ran the experiment on filtered versions of this structure. A bivariate + univariate structure in which cliques with $|k| = 5$ were removed; a bivariate structure in which cliques with $|K| = 5$ and $|K| = 1$ were removed and a univariate structure in which cliques with $|K| > 1$ were removed.

4.6.2 Results

We can see that the effect observed for other problems is still present in that there is an increase in both FPC values over the under-specified to over-specified transition. Figure 4.8e has been included to show that the C_m values begin to decrease gradually with the population size beyond 2N, an effect similar to that shown for the 1D Checkerboard problems. In addition to this the C_r value gradually increases beyond $M = 2N$, rather than reaching a plateau as seen for the other fitness functions. These experiments allowed the population size to increase to 7N to allow observation of this effect, which can be seen in the detailed figures in Appendix B. Another point of interest is that around $M = N$ there is a similar drop in FPC values as that seen for 2D Ising. This is not accompanied by the high C_m values with low populations seen with 2D Ising so is unlikely to be because of a high correlation between the fitness function and the Markov network as it the case for 2D Ising. It is presumably an artifact produced by the least squares fitting of SVD, but this would require much deeper investigation to determine.

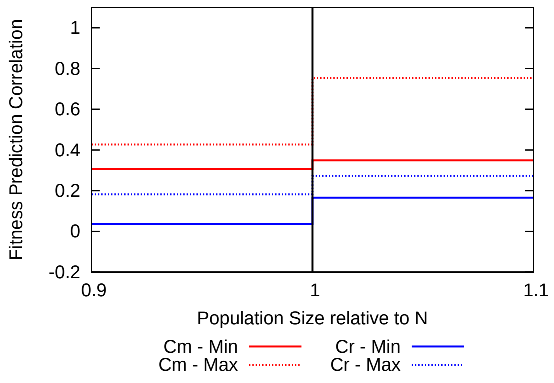
There appears to be little difference between the model using a chain structure and that with the extra terms for groups of 5 bits. It reveals that these extra cliques do little to improve the fitness modelling capability. This may be because this is still not a perfect model as there are no 3- and 4-cliques; as we saw with MAXSAT, removing subcliques results in a poorer model of fitness, although with the 1D checkerboard function the opposite was true: the model had higher FPC values when univariate terms were omitted. It could then be argued that the terms representing links between groups of 5 are superfluous and that the chain model have enough expressive power to represent the trap function.

It can be seen that the purely univariate model fits the fitness function more closely than the model with only bivariate terms, and reaches the highest C_m values obtained in the experiments. This is counter-intuitive for a deceptive problem with inter-variable interaction, especially having seen the results of the previous experiments fitting univariate models to multivariate fitness functions. This can be explained by remembering the nature of the trap-κ problem: it is essentially a variant of onemax, with fitness increasing linearly as groups contain more 1s, with the exception of all 0s in a group being given a high fitness. What can be seen here is that with a univariate model the MFM is able to sort most individuals by fitness well, with only those containing all 0 groups being incorrectly predicted. In contrast, the MFM including only interactions and not absolute variable values is able to identify that individuals containing all 0 groups should have higher fitness but is not able to distinguish between the suboptimal groups. This is an important observation: it reinforces the idea of the proximate optimality principal (Glover & Laguna 1997), that neighbouring solutions will have a similar fitness.

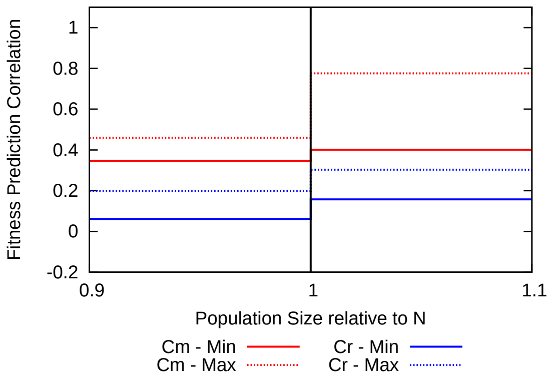
With none of the structures do we see C_r as close to 1.0 as we do with the other fitness functions studied. This means that the MFM is unable to as accurately predict changes in fitness over many mutations which means it is less likely to find a global optimum by running a Gibbs sampler on the MFM to convergence. The higher C_m values do indicate that the MFM can predict changes in fitness for single mutations; this could be useful for other approaches to optimisation such as a guided mutation operator.

4.7 Dropoff with Model Decimation

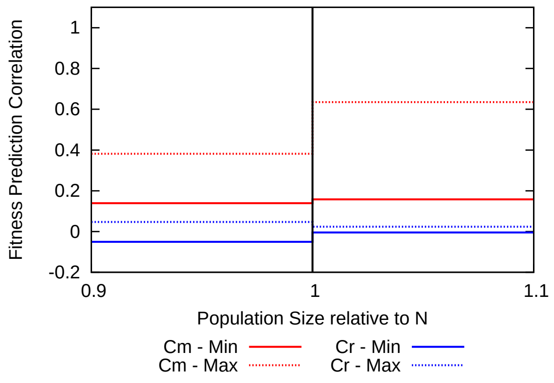
The preceeding experiments have outlined the influence of population size and model structure on the fitness modelling capability of the MFM. We have seen that the C_m, the



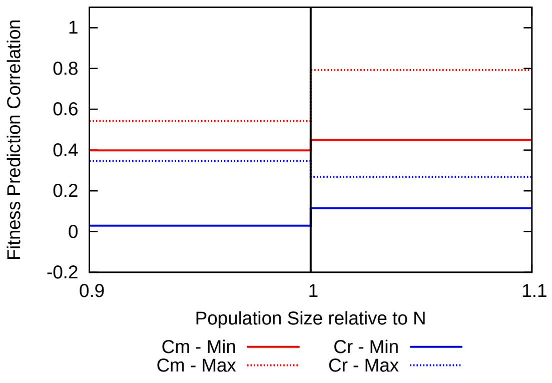
(a) 5,2,1 model structure



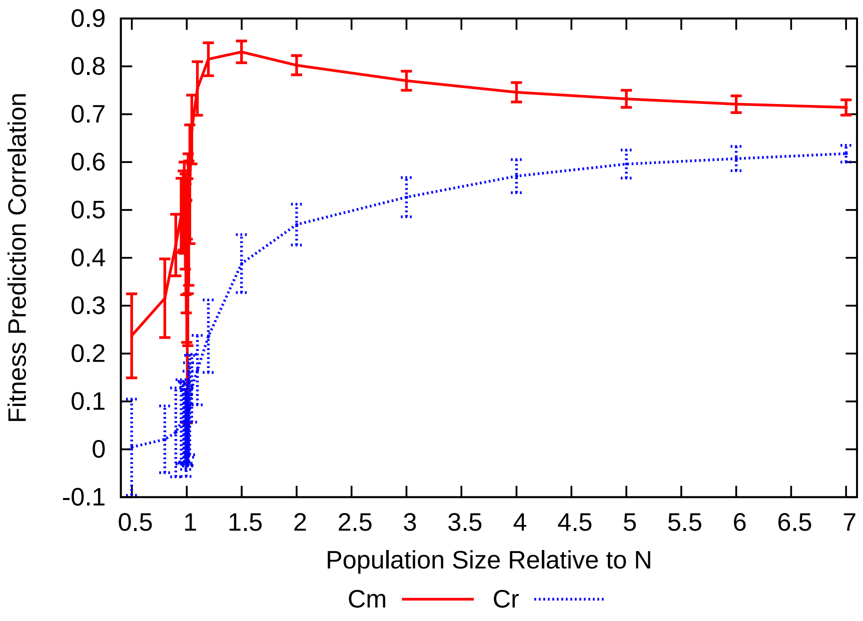
(b) Bivariate + univariate model structure



(c) Bivariate model structure



(d) Univariate model structure



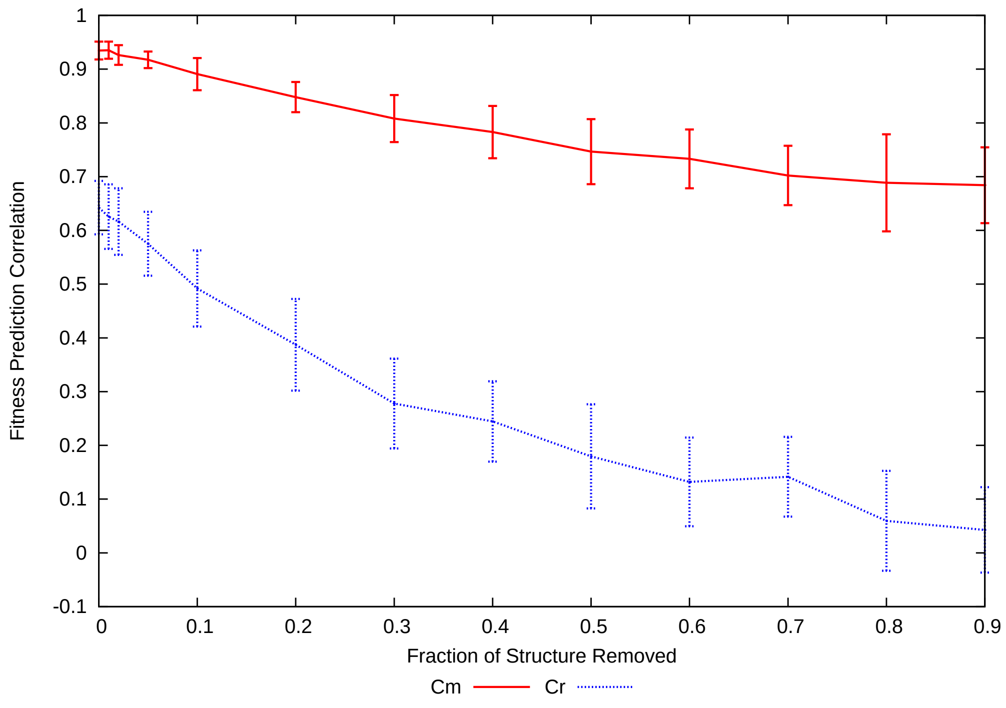
(e) 5,2,1 model structure on 100 bit Trap-5

Figure 4.8: Summary of FPC against population size for Trap-5 problems

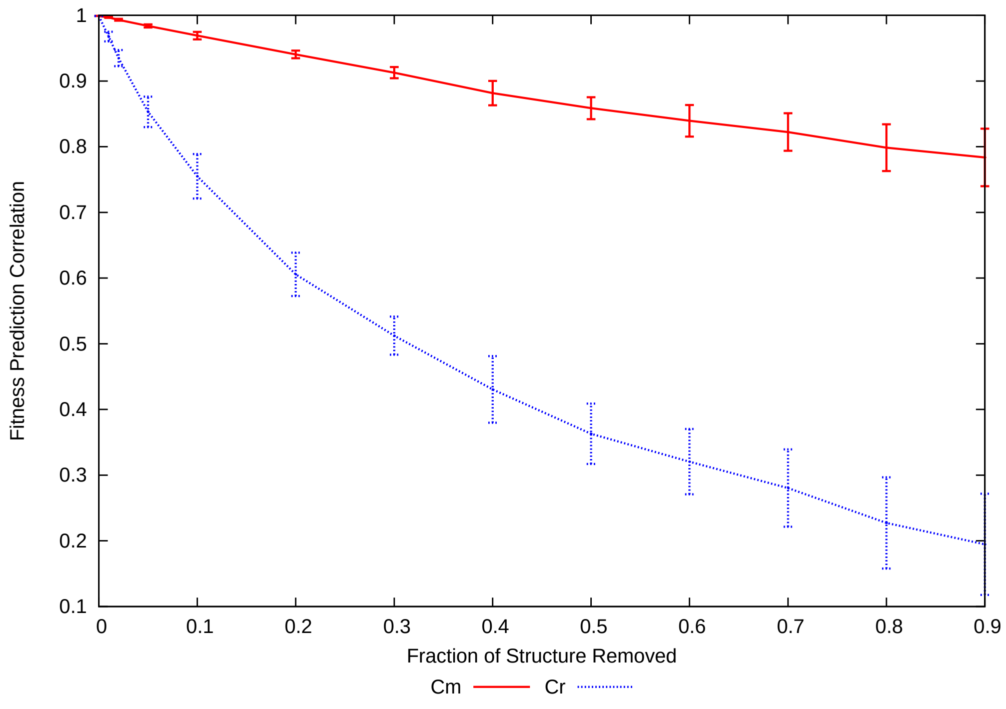
fitness prediction capability for mutated individuals, can be high with an imperfect model that has many interactions removed. C_r was typically much lower with imperfect than for perfect model structures. To explore this further, we repeated the experiments for 100 bit Ising and 100 bit MAXSAT as an increasing portion of the structure is removed from the model. This means that a proportion of the cliques with $K > 1$ were selected at random and had their α_K fixed to zero. The proportion was increased in each iteration of the experiment to show the dropoff in both FPC values. Figure 4.9 shows the fitness prediction correlations as an increasing portion of the structure is removed from the model. Both experiments were performed using a population size of 1.1N. We can see that fitness prediction for randomly generated individuals falls away quickly, and more slowly for mutated individuals. This indicates that the model can still have a close fit to the fitness function with a much smaller number of α_K, especially if we only want to model fitnesses of neighbouring solutions to the population. This will be useful in reducing the computational complexity for estimating the model parameters.

4.8 Dropoff with Mutation

C_m values are computed for individuals which are only a Hamming distance of 1 from those in the current population. Having seen that C_m values for univariate models are in general high it is also useful to know how much mutation will prevent a univariate model being able to accurately predict fitness. Figure 4.10 shows the fitness prediction capability of a univariate model on 100 bit Ising and 100 bit MAXSAT for increasing levels of mutation for each individual. Again both were performed using a population size of 1.1N. The FPC for a random individual is included and is low for both fitness functions - this is because there are no interactions on the model and it is unable to accurately predict fitness for randomly generated individuals. We can see that as we approach 50% of the bits being mutated the fitness predication ability falls to the level for randomly generated individuals. This is as we would expect; with a bit string representation any pair of individuals will on average differ by 50% of the bitstring, so compared to the current population any individual with half of the bits mutated could be considered “random”.



(a) 100 bit 2D Ising



(b) 100 bit MAXSAT

Figure 4.9: FPC against proportion of interactions removed from model

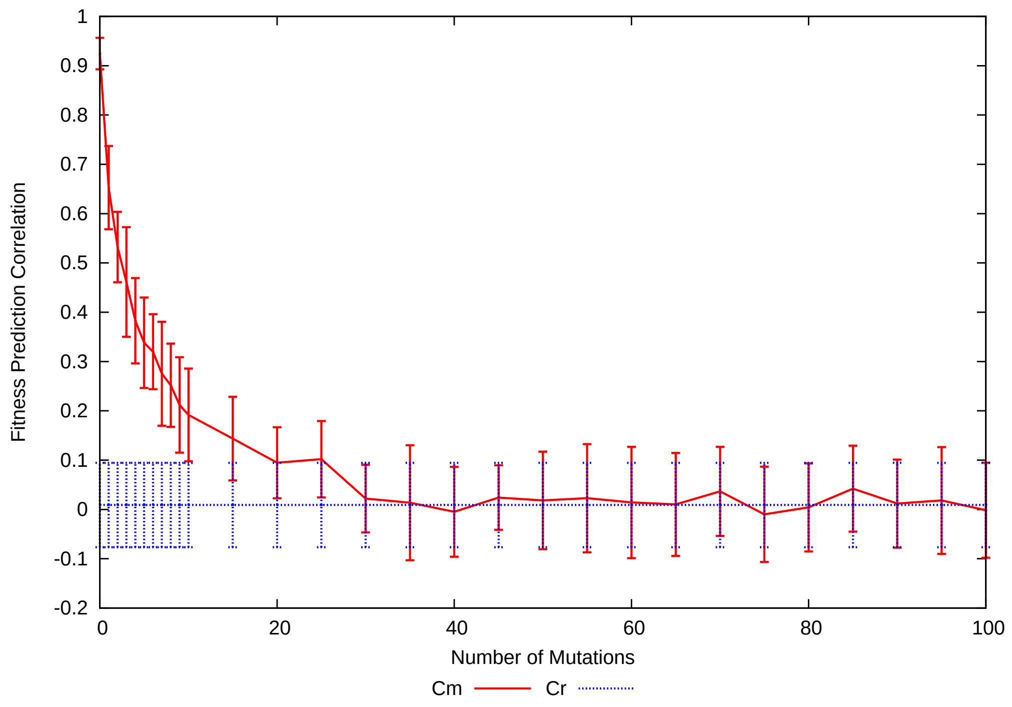
The model's fitness predictions correlate strongly with the true fitnesses of individuals up to a Hamming distance of around 2-3 and with a weaker correlation up to around a distance of 10 bits. These results show that even a model with many interactions missing from the structure can still model neighbouring solutions to those already seen.

4.9 Summary

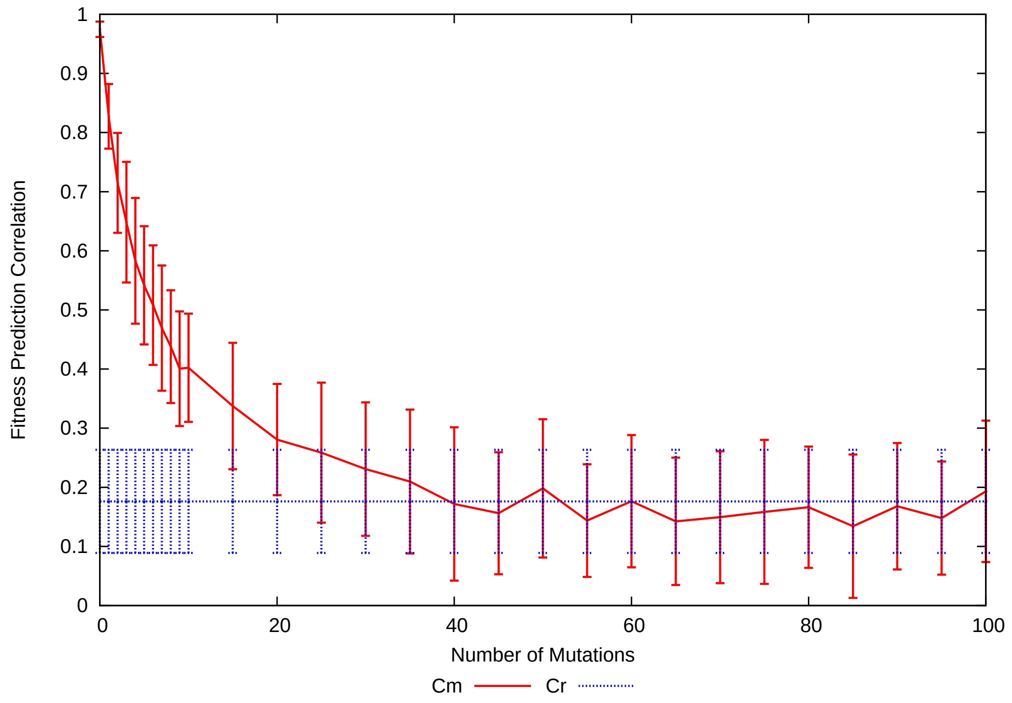
The results of the experiments described in this chapter give us an important insight into fitness modelling using the Markov network approach. We have made two major contributions through this chapter. Firstly we now know that, at least for the benchmark functions used here, there exists a strong and quantifiable relationship between the structure of the MFM and the effort required to estimate parameters for a model with strong correlation to fitness. Secondly, again in the context of the benchmark functions used here, we have observed that the ability of the MFM to predict changes in fitness for an individual is highly dependent on the MFM's structure.

Where we are able to supply the perfect stucture for the MFM, the fitness prediction capability remains low until the number of individuals in the population used for estimating model parameters exceeds the number of those parameters. Where we use an imperfect model we can generally build an MFM which can accurately predict the fitness change resulting from single mutations (a high C_m value) but in general it is unable to accurately predict fitness over many mutations (low C_r value). C_r has been shown to increase for trap-5 by using a large population of 7N individuals. In contrast to this, by increasing the population size beyond N for some benchmark functions (1D checkerboard, 2D Ising and trap-5) with imperfect structures we see a fall off in C_m value. This is possibly caused by conflicting information about fitness being present in the larger population which the model cannot incorporate because of the imperfect structure. This is related to recent work (Ashlock, Bryden & Corns 2008) in the genetic programming community which indicates that a small population is better at solving some problems.

The fitness prediction capability has a direct impact of the usefulness of the MFM for optimisation (explored further in Chapters 6 and 7). When sampling the MFM, we



(a) 100 bit 2D Ising



(b) 100 bit MAXSAT

Figure 4.10: FPC against number of mutations for univariate models

need to be able to accurately predict changes in fitness caused by mutations to be able to generate individuals nearer to the global optimum. These findings allow us to now say with some certainty that there is a minimum population size required to build a useful model which is directly related to the model complexity. In terms of fitness evaluations, this effort is low compared with that typically expended by model building algorithms on these problems. However, the model building costs are high, principally because of the computation required for SVD (Press et al. 1986), and in the cases demonstrated here the structure is supplied to the algorithm. This is important in considering the overhead for any algorithm using the MFM approach and particularly when deciding the complexity of the probabilistic model to use. We have also seen the importance of the model structure in building a model which correlates closely to the fitness function. Model structures, which are missing a considerable number of the interactions that would be present in a perfect structure, can still give a model which shows a reasonable ability to predict fitness over a small number of mutations. This will be important later as we move on to look at using the model for optimisation with a structure learned from data, which is unlikely to match the perfect model.

Chapter 5

Effect of Selection on Fitness Modelling

In this chapter we consider another factor which affects the quality of fitness model; the selection operator. In Chapter 3, we saw that one of the features which distinguishes EDAs from other evolutionary methods is that an explicit probabilistic model of the problem is created which can be analysed to reveal underlying information about the problem itself. This chapter makes further use of the Fitness Prediction Correlation (FPC) described in Chapter 3 as a measure of model quality to explore the effect that the selection operator has on the construction of the probabilistic model. From this we may infer clues as to the nature of the operator itself; what it does to the information about fitness contained in the population and what is lost or gained by the use of selection.

Most evolutionary algorithms apply a distinct selection operator so that a subset of the population is chosen for recombination or model building. The Markov fitness model is a probabilistic distribution which is monotonically related to the fitness function and because of this selective pressure is directly built in to the model. This means that construction of the MFM does not require an explicit selection operator, although one may still be used. Previous work on DEUM (Brownlee et al. 2007, Shakya 2006, Shakya et al. 2006, Shakya, McCall & Brown 2004a) varied in the use or absence of selection prior to building the model - in each case it was determined empirically whether selection was beneficial. This

Algorithm 5.1 Effect of Selection Experiments

```
1: for each selection operator, and each selection proportion do
2:     Generate random initial population $p$
3:     Select a subset $\sigma$ of $p$
4:     Use $\sigma$ to build MFM
5:     Calculate FPC values $C_m$ and $C_r$
6: end for
```

chapter will examine this issue systematically.

5.1 Selection Operators

The selection operator being examined here is truncation selection (Mühlenbein & Schlierkamp-Voosen 1993), frequently used in EDAs (Larrañaga & Lozano 2002). It selects the fittest $\varphi * M$ solutions from the population, where $0 \leq \varphi \leq 1$ and M is the population size. Truncation selection selects a solution only once (unless it is duplicated in the population), and there is zero probability of poor fitness individuals being selected, in contrast to operators such as tournament selection (Goldberg & Deb 1991).

Here we use four related variants of truncation selection. We call the standard truncation selection *top selection*. *Bottom selection* selects the least fit $\varphi * M$ and *top & bottom selection* selects the fittest $(\varphi/2) * M$ and least fit $(\varphi/2) * M$. The fourth operator is created implicitly; the number of selected solutions may be varied to change the selective pressure and when the number selected equals the population size, we effectively have *no selection*.

5.2 Experiments

The experiments all follow the procedure given in Algorithm 5.1. Considering the discussion on specified and underspecified models in Chapter 4, it is important that the number of individuals used to build a model remains constant. This is achieved by choosing the proportion of the population that will be selected, then generating a population of suitable size (1.1N) to result in the chosen number being selected. This allows us to observe the effects of selection on the model building without matters being confused by the under/over-specification issue. In all experiments, the proportions p of the population selected were 0.01, 0.02, 0.05, 0.1, 0.2, 0.28, 0.5, 0.67, 0.83 and 1 (that is, no selection). To illustrate: the model for 100-bit 2D Ising has 301 parameters - 100 univeriate terms, 200 bivariate interactions and the constant. To keep the system over-specified, 1.1N (331) individuals are selected to build the model in all experiments. The population size is altered so that 331 individuals are always selected - which means that to achieve a selected proportion of 0.01 the starting population must be 33100 individuals, and to achieve a selected proportion of 1.0 (no selection) the starting population will be 331 individuals.

Each experiment was repeated 30 times and the mean and standard deviation for each FPC calculated. Each experiment presented here looks at a range of instances of a number of different fitness functions. Similar to Chapter 4, the results are presented in Appendix C with a subset presented here for ease of reading.

5.3 Perfect Model Structures

5.3.1 Outline

Recall from Section 3.1.4 the different types of structure which may be used to build the model. We will look at both perfect and imperfect structures in this chapter, with the series of experiments in this section using a perfect model structure. We will look at imperfect model structures in section 5.4.

In Chapter 4 we saw that population size is an important factor in building the model and defined the concepts of fully specified and underspecified models. That is, where the number of individuals used to build the model is greater than and less than N, the number of terms in the MFM. In this chapter we will look at both situations.

5.3.2 Results Using Fully Specified Models

Firstly we look at the univariate problem, onemax, over a range of different problem sizes n from 10 bits to 1000 bits. There are no interactions between variables and the MFM includes only univariate terms. Recall from Chapter 4 that this means the perfect model has a total of $N = n + 1$ parameters including the constant. The results are shown in Figures 5.1 and C.1 to C.9.

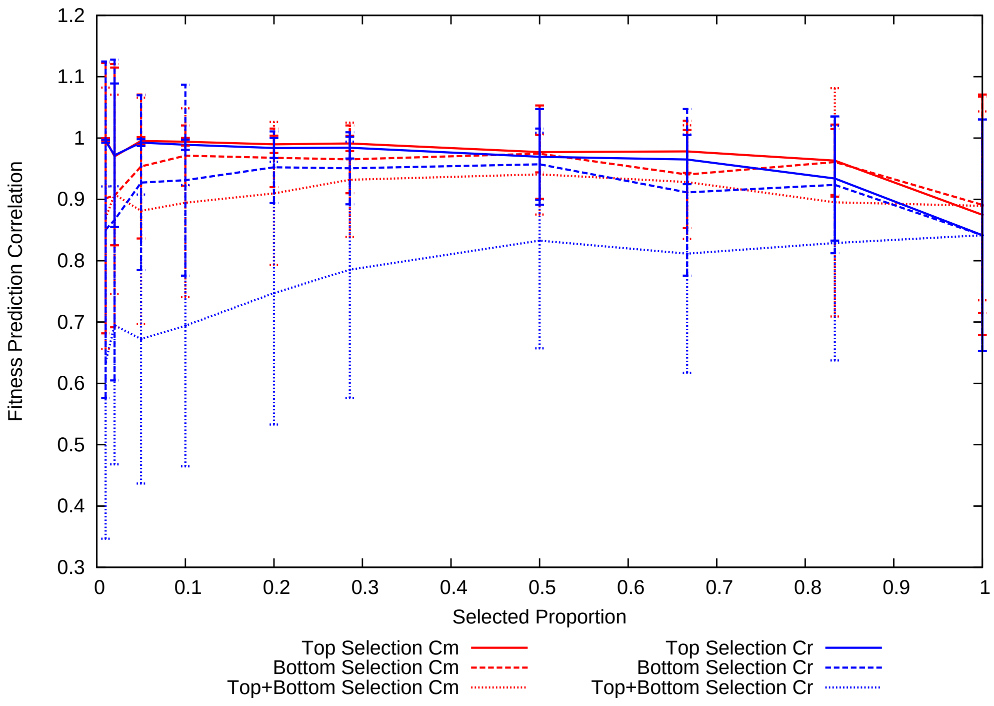
A pattern is visible across the 9 problem sizes, though it is less clear for the two smallest sizes where the standard deviations exceed the gap between most of the plots on the graphs. Indeed, in this and most of the other experiments in this chapter the FPC values for small problem sizes are often close to zero and have large standard deviations in comparison to the results for larger problem sizes. This is likely caused by a lack of diversity among the selected individuals caused by the small number of variables and high selective pressure.

We can see that the MFM has a strong positive correlation with the fitness function (note the values on the y-axis); C_r is always higher than 0.8 and comes close to 1. Top selection yields the best model quality, which falls off as the proportion of the population selected increases. Bottom selection shows a similar trend, starting with a slightly lower C_r. In contrast, top & bottom selection results in the poorest model which improves as a larger proportion of the population is included. C_m is consistently close to 1 in all cases.

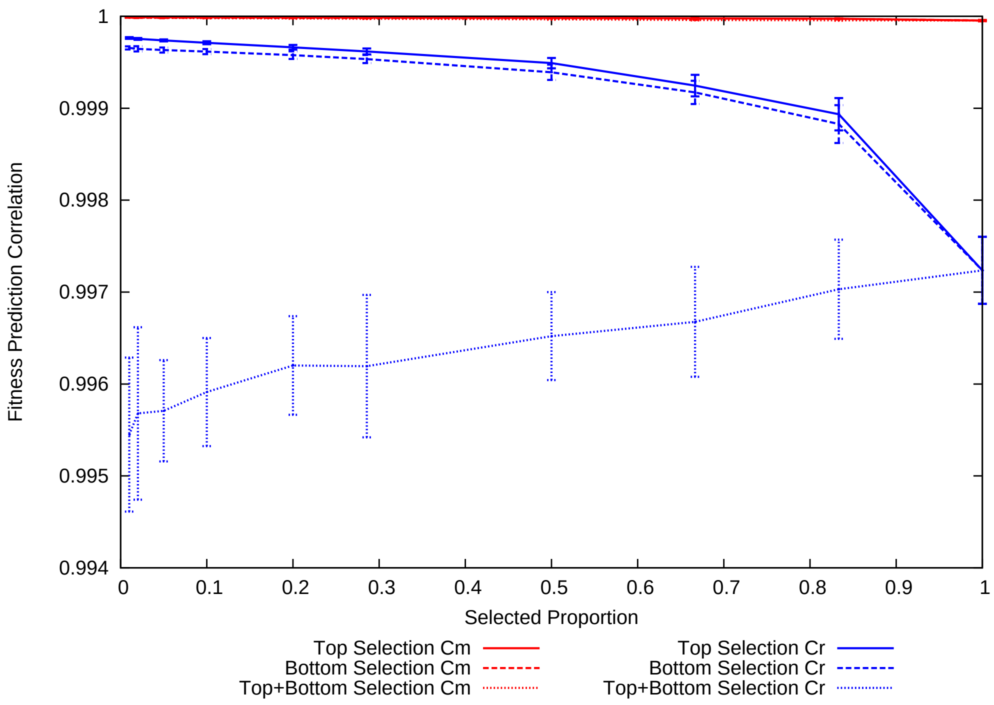
Figures 5.2 and C.10 to C.18 show the results for a 2D Ising Spin Glass problem. Here problem instances have interactions between pairs of variables, giving a model size of N=3*ProblemSize+1 (a term for each variable, a term for each interaction and a constant term). As we saw in Chapter 4, this problem exhibits unique properties in the context of fitness modelling with Markov networks.

We see the same trends appearing as with onemax; top selection gives values of over 0.95 for both C_m and C_r. These fall toward 0.95 and 0.5-0.7 respectively as the proportion of solutions selected approaches 1. Bottom selection gives values over 0.9 and 0.8 for C_m and C_r and follows the same gradual decrease. Top & bottom selection gives the lowest values - 0.95 for C_m and around 0.5 for C_r - with C_m remaining level and C_r gradually increasing toward 0.5-0.07 as more of the population is selected. Again we see the effect becoming more pronounced as the problem size increases.

Results for the 3-CNF MAXSAT problem are presented in Figures 5.3 and C.19 to C.25. Recall from Chapter 4 that N varies because of differing numbers of bivariate

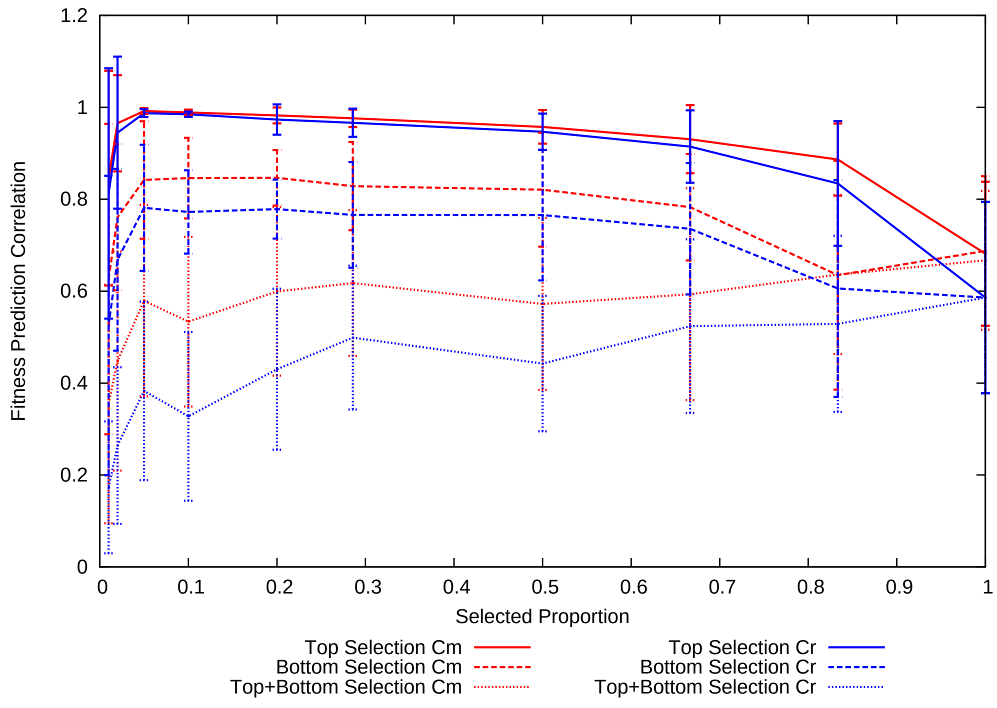


(a) 20 bits

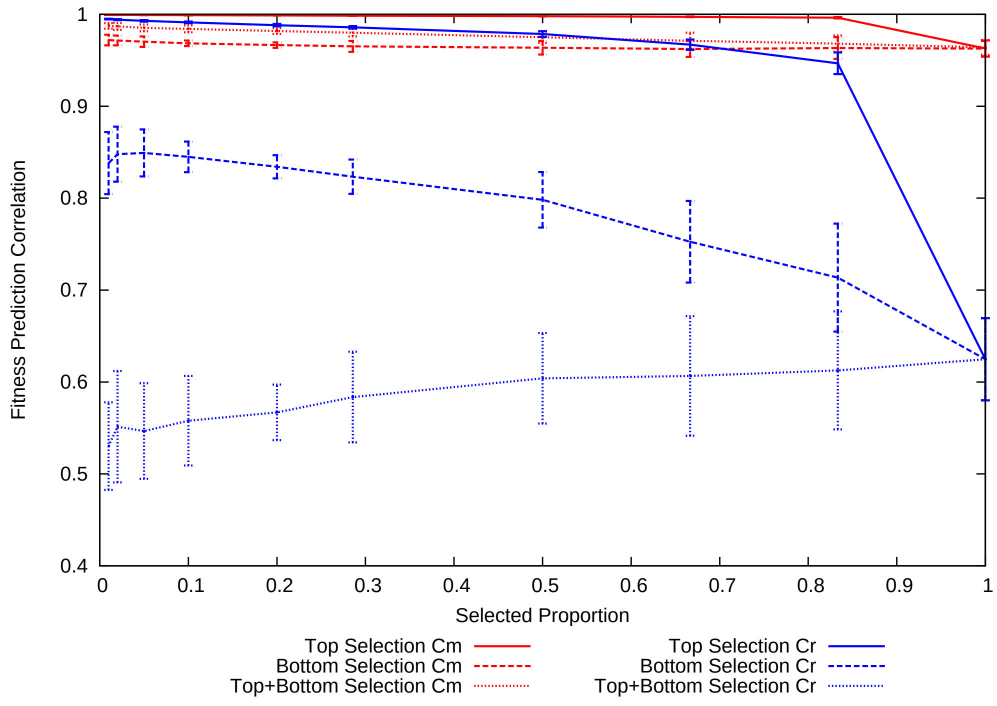


(b) 1000 bits

Figure 5.1: FPC against selection proportion for fully specified OneMax problems



(a) 16 bits



(b) 400 bits

Figure 5.2: FPC against selection proportion for fully specified 2D Ising problems

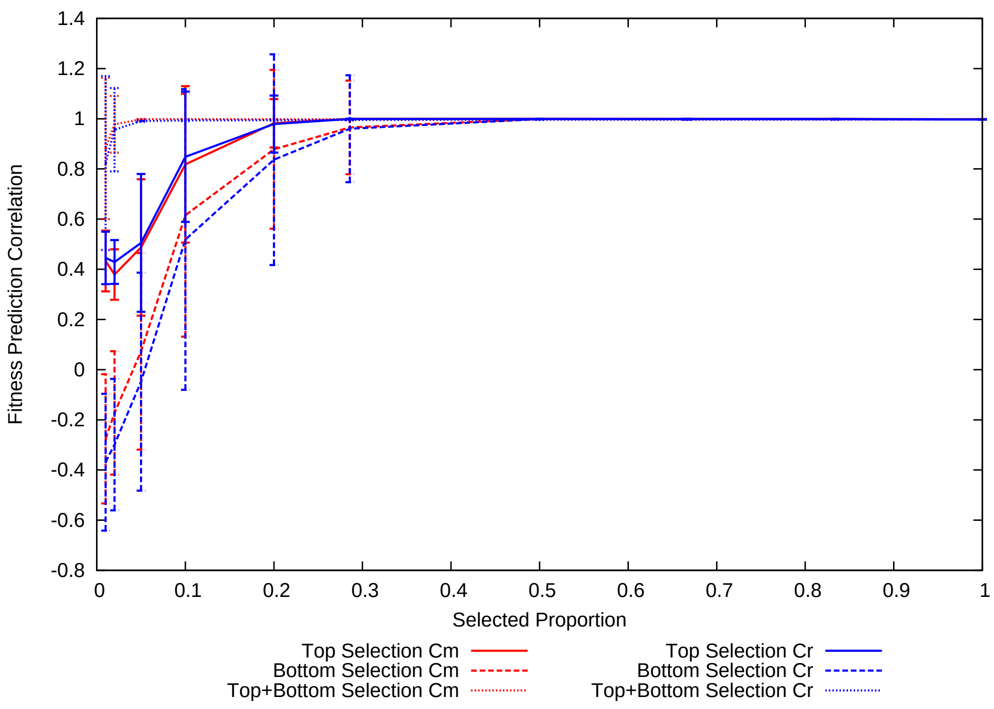
interactions in each instance, so the population size and number of individuals selected were varied accordingly.

We see similar results to those for the onemax problem, though with all correlation values considerably closer to 1. Again top selection gives the best C_r values, while bottom selection follows a similar trend, slightly lower. Top & bottom selection gives the lowest values for C_r. Also like onemax, C_m values are consistently close to 1 for all selection types, and for the smallest problem sizes the FPC values are low with a small selected proportion because of a lack of diversity in the population.

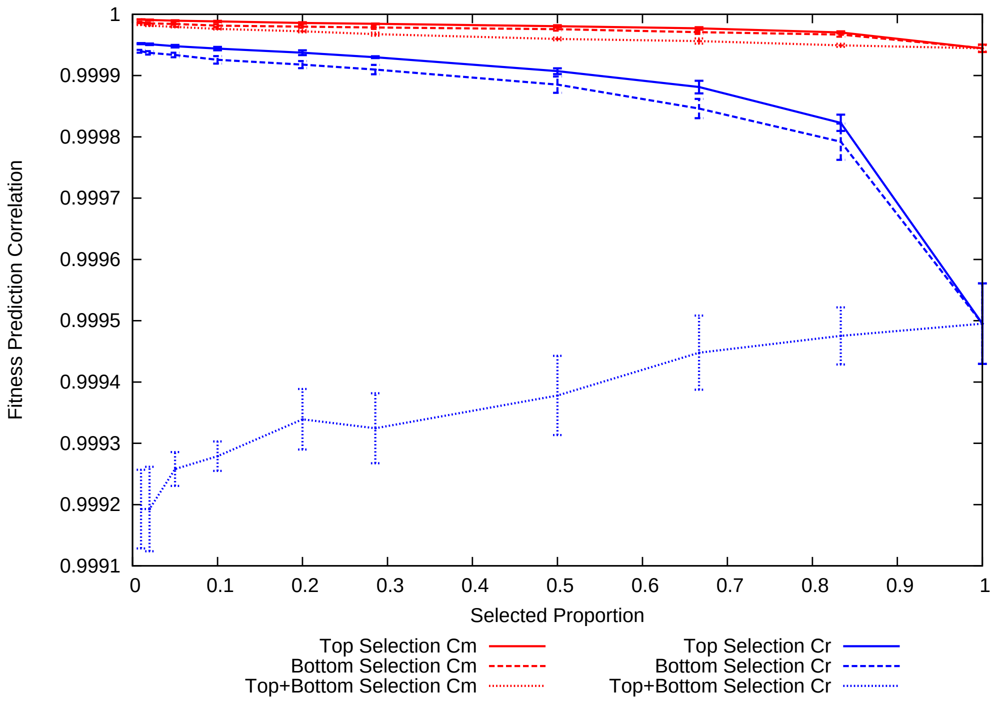
5.3.3 Analysis

In this section we have seen that - for the benchmark functions used - with a perfect model structure and a selected population large enough to give a fully specified system, use of the traditional top selection operator will produce a model very closely fitted to the fitness function. More interestingly, the similar C_m and C_r values would indicate that a model with almost the same fitness modelling capability was obtained by selecting the poorest part of the population. A model with high fitness prediction capability was also obtained by selecting the entire population, equivalent to not having any explicit selection operator. It is also worth observing the strong correlations seen between predicted and true fitness for random individuals - particularly for random individuals - showing the strength of the MFM approach to modelling fitness.

Conventional wisdom from the EA context would have us expect that top selection would give a better model. However, when building a model of the fitness function and attempting to predict fitness of randomly generated individuals we might speculate that there would be more information in the full population, resulting in higher C_m and C_r values without selection. From the results this is not the case and the model improves considerably by removing even a small proportion of the population (selection of 80% rather than 100%). It is possible that there is a trade-off - with the full population there is too much information and SVD is unable to correctly minimise the error; this being reduced by discarding a small number of solutions (selection of a high proportion). Another reason



(a) 20 bits



(b) 175 bits

Figure 5.3: FPC against selection proportion for fully specified MaxSAT problems

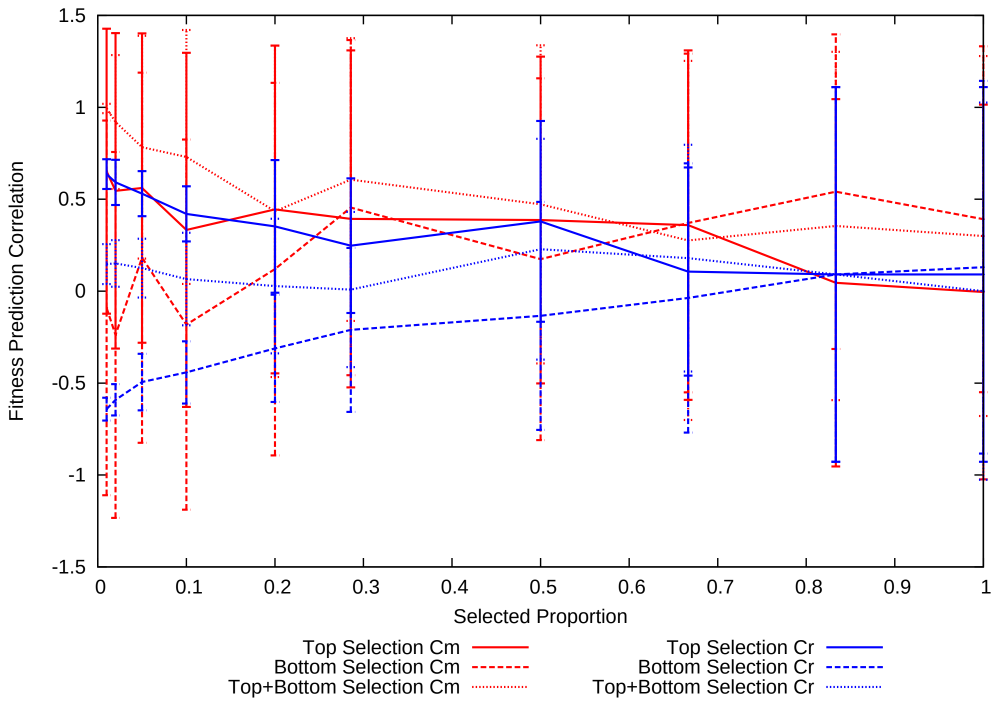
could be that because the population is randomly generated, there is some duplication and redundancy, meaning that the true number of individuals needed to achieve a full specified system is larger than N or even $1.1N$ as used in the experiments. By introducing a small selective pressure, the design of this experiment requires a larger population to be generated which would avoid this problem. This factor could be counterbalanced in future work by the use of a larger population, or by using experimental design in generating the population.

5.3.4 Results Using Underspecified Models

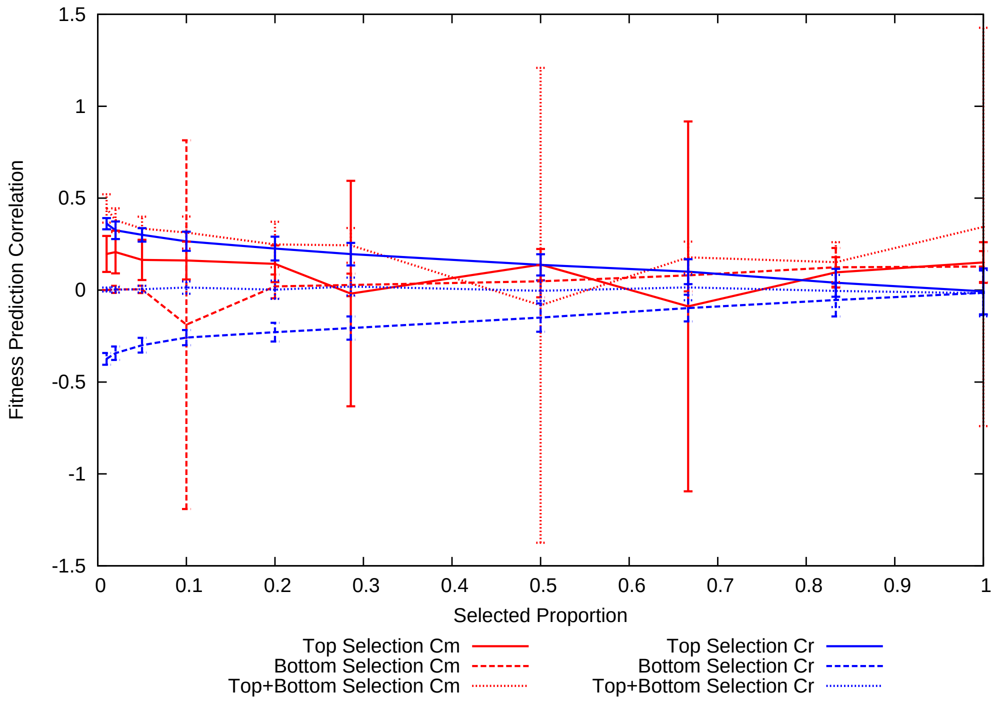
As illustrated in Chapter 4, an underspecified MFM does not generally model the fitness function well. However, some of the earlier experiments with DEUM such as those outlined in (Shakya et al. 2005c) have produced good results with what we have now determined to be too small a population for obtaining a fitness model with a high FPC. One possibility here is that selection can be used to focus the model on important parts of a population - reinforcing the limited information content of a small population.

The first experiment in this section looks at results for the onemax problem, shown in Figures 5.4 and C.26 to C.33. Here, the number of solutions selected equals 0.1N; underspecified. Recall from Section 5.2 that this is not to be confused with the proportion of the population selected - we are always selecting 0.1N solutions but change the population size so this represents selecting different proportions of the population. 0.1N is chosen as a population size that is an order of magnitude smaller than the point at which the system becomes fully specified and the results from Chapter 4 would indicated that such a small population will produce very low FPC values. Indeed, for the two smallest problem sizes (10 and 20 bits) the starting population was so small (1 and 2 individuals respectively) that it frequently resulted in SVD failure and a useless model. Where this occurred the run was terminated and does not influence the mean value shown on the graph; for the size 10 problem this means that there were no results to report and it is omitted from the figure.

One point of particular note is that when using top selection, the fitness prediction



(a) 20 bits



(b) 1000 bits

Figure 5.4: FPC against selection proportion for 0.1N under specified onemax problems

capability for random individuals at some points appears better than that for mutated individuals similar to those in the selected set, though set against a background of large standard deviations. The explanation for this is likely to lie in the *best-fit* behaviour of SVD, used to construct the MFM, when supplied with such a heavily underspecified system. Looking at the C_r values separately, we see that C_r appears to be highest when using top selection, top & bottom selection gives a correlation of approximately zero between true and predicted fitnesses, and bottom selection shows an inverse correlation with true fitness. This is perhaps a result in keeping with intuition - that selecting the fittest individuals results in the best model of fitness.

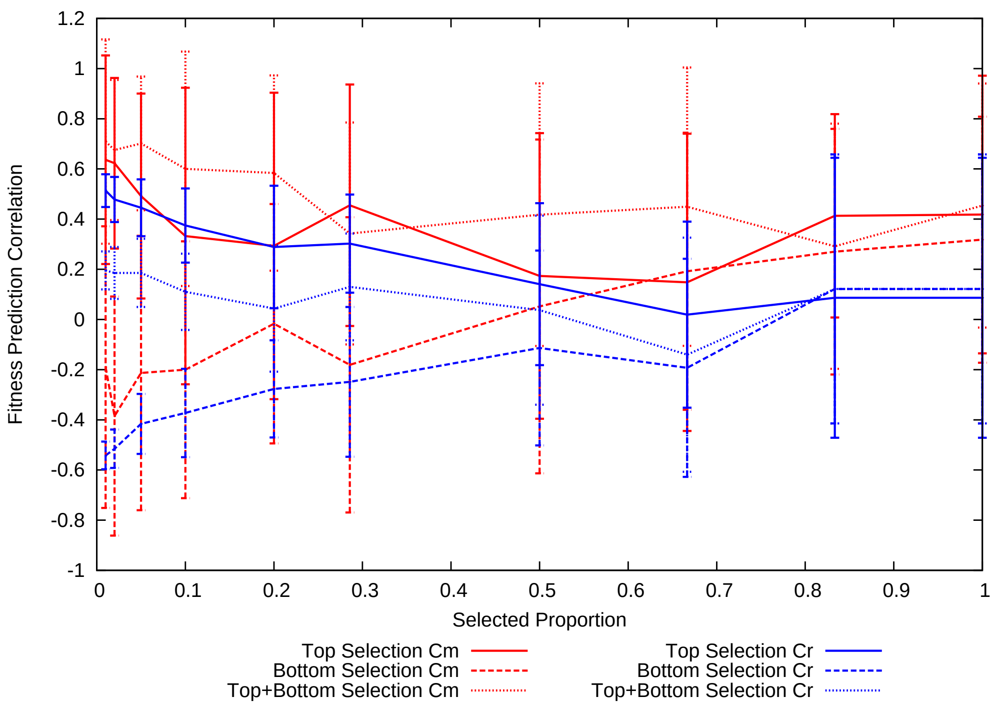
Perhaps most interesting in this context is that when selecting a small proportion of the population, top & bottom selection produces a model with a higher fitness modelling capability than top selection and bottom selection. Given the large standard deviations we cannot read anything further into this.

Figures 5.5 and C.34 to C.42 give results for the 2D Ising problem. Again the number of solutions selected equals 0.1N.

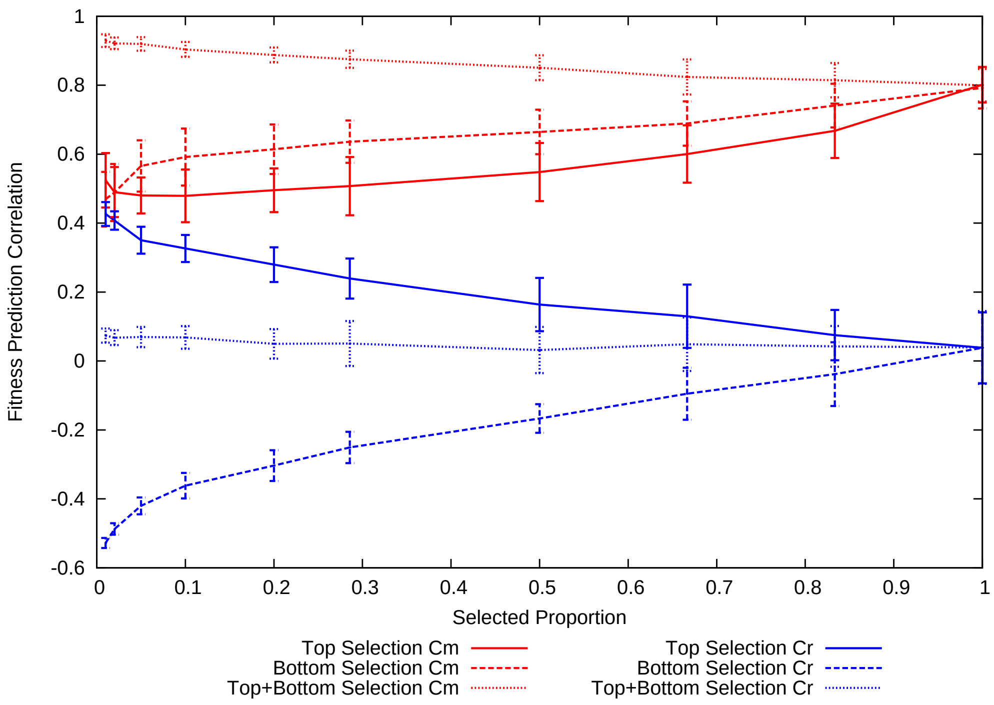
Here we see a similar trend to that seen for onemax but with less noise. Top selection gives a higher C_r (around 0.4 and falling) than the other operators, which gave $C_r \approx 0$ for top & bottom selection and no selection, and $C_r \approx -0.4$ for bottom selection. Results for C_m show that top & bottom selection produces the best model of fitness out of the four operators - for larger instances of the problem C_m starts at around 0.9 and falling to 0.8 as the proportion selected increases. Top selection and bottom selection give similar results to each other - C_m at around 0.5 which increases as a larger proportion of the population is selected.

Figures 5.6 and C.43 to C.49 show the results for 3-CNF MAXSAT with a selected proportion of 0.1N.

Here we can see selection exerting a strong influence on the quality fitness model produced. Top selection gives values for both C_r and C_m over 0.4, whereas top & bottom selection and bottom selection give close to zero and -0.4 correlations respectively. The strong influence of selection here is likely to be a reflection of the problem's complexity -

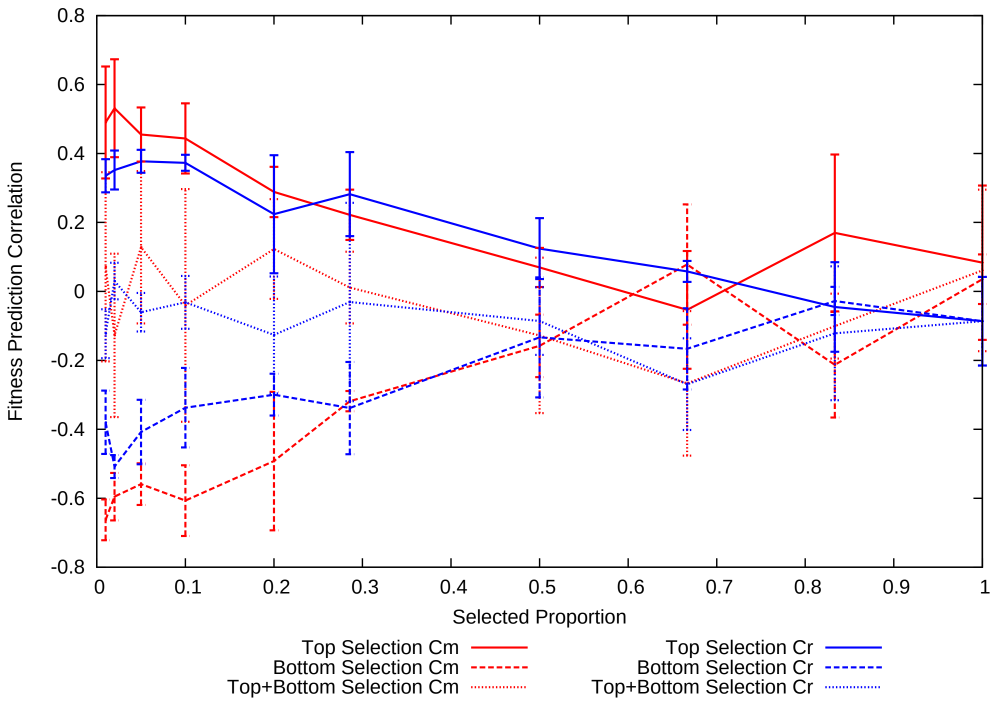


(a) 16 bits

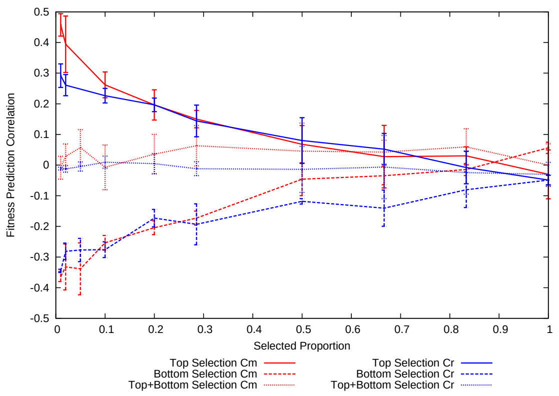


(b) 400 bits

Figure 5.5: FPC against selection proportion for 0.1N under specified 2D Ising problems



(a) 20 bits



(b) 175 bits

Figure 5.6: FPC against selection proportion for 0.1N under specified MaxSAT problems

with so little information in the problem, emphasis of strong solutions is required to build a good model of fitness.

5.3.5 Analysis

In contrast to the overspecified MFMs in the previous section, with an underspecified MFM the selection operators produce result more in keeping with intuition. Top selection is important in building the general fitness model - C_r values only approach strong positive correlation when using this operator. In this context, this means that the fitness model is able to accurately predict fitness of a randomly generated individual (not necessarily high in fitness), as opposed to only the change in fitness over a few mutations. In contrast, no selection and top & bottom selection produce models with C_r close to zero. Bottom selection produces a model which appears to be inversely correlated with the fitness function - this latter result does still show that there is useful information about fitness in the fitness function. When observing the model's ability to predict fitness of neighbouring solutions, the results would indicate that any selection operator is better than having no selection which gives C_m values of near zero. Top & bottom selection gives the highest C_m values - this would indicate that this operator is able to sharpen the information already present in the population without completely discarding either poor or fit solutions.

5.4 Imperfect Model Structures

5.4.1 Outline

We now move on to experiments using imperfect model structures. Here the MFM does not include all the terms required to perfectly model the fitness function and as in Chapter 4 we specify the structure in terms of the cliques retained in or removed from the model. Such structures are important because they are comparable with structures built by learning algorithms such as independence tests, which will inevitably miss some interactions.

5.4.2 Experimental Results

In these experiments we cannot investigate onemax because it has no variable interactions to omit. Thus we begin by revisiting the 2D Ising problem.

The results shown in Figures 5.7 and C.50 to C.58 are produced using the univariate model applied to the Ising problem (that is, only retaining cliques where $|K| < 2$). In parallel with the findings in Chapter 4 we can see that ignoring interactions between variables results in a poor model compared to those in Section 5.3. C_r values regardless of selection type are all close to 0 indicating that the model is unable to predict fitness of random solutions. For the smaller instances of the problem the C_m values are also low - rarely exceeding 0.4 indicating that the model has a weak positive correlation with the fitness function. However, for the larger problems (256 bit upward) all three selection operators we have C_m values over 0.6. This increases to between 0.7 and 0.8 for top selection and top & bottom selection of approximately half of the population. This indicates that modelling of nearby solutions is good but prone to error. The three operators give similar results, with top selection giving the highest C_m values, followed by top & bottom selection. Bottom selection gives a lower C_m value than no selection. Again the results for small instances of the problem are much less clear - the small population and small number of variables resulting in low diversity and a poor model.

In Figures 5.8 and C.59 to C.76 we show results for decimated models applied to the Ising problem. This included two experiments; one with 10% of the cliques with $|K| = 2$ removed from the model structure and one with 50% of the cliques with $|K| = 2$ removed. The same trends are visible for both, though with the 50% decimated model the FPC values are lower overall that for the 10% decimated model.

Top selection gives a model with strong positive C_m and C_r values (0.98-0.99). Bottom selection also gives a good model of fitness, though with lower C_m and C_r values, beginning at around 0.95 and 0.8 respectively. Top & bottom selection gives a model with a C_m of 0.4-0.8 and C_r of 0.4-0.5. The values for these three operators converge to those for no selection with C_m and C_r of 0.6-0.95 and 0.6 respectively.

In Figures 5.9 and C.77 to C.85 we see a univariate model applied to the 3-CNF

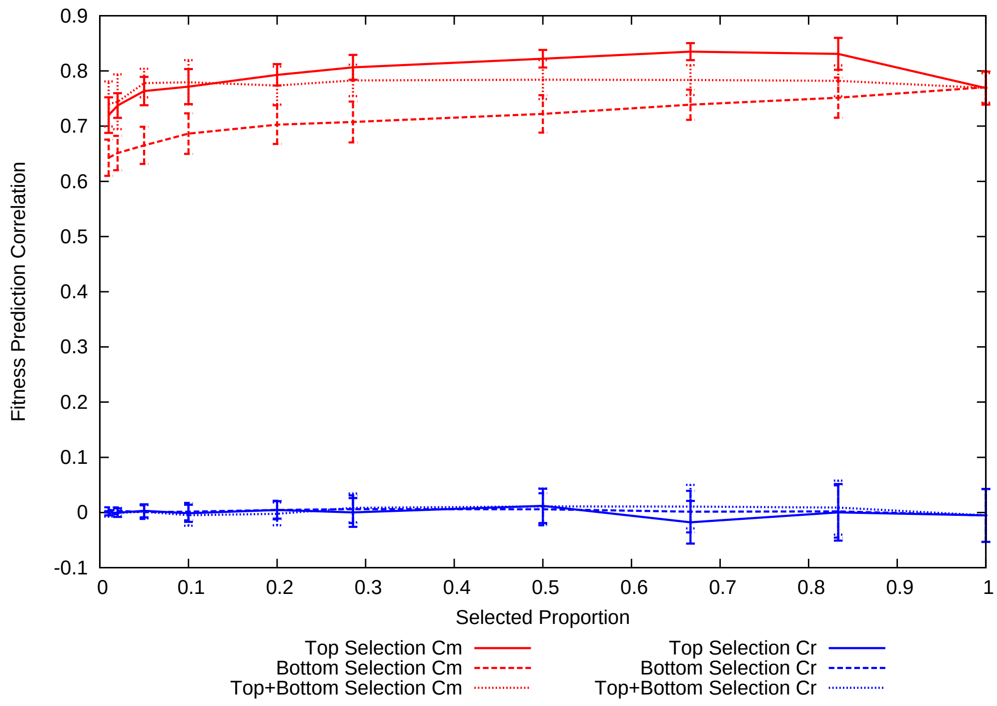
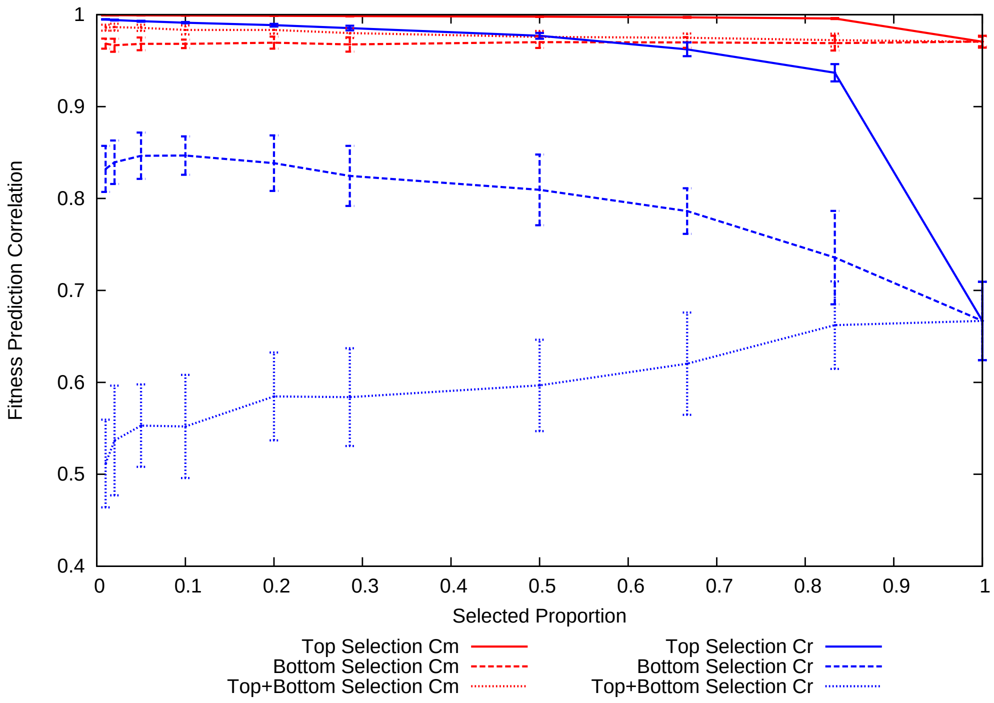


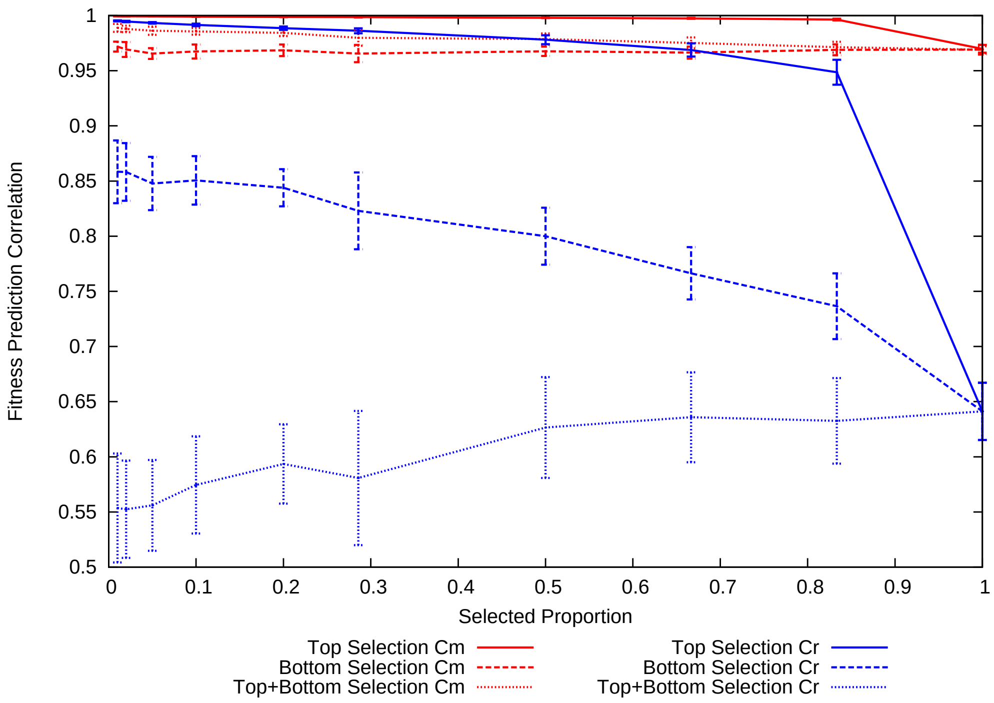
Figure 5.7: FPC against selection proportion for a fully specified 400 bit 2D Ising problems with univariate model structure

MAXSAT problem (that is, a model with $|K| < 2$). With all of the interactions missing from the model and a problem which has bivariate and trivariate interactions, we would expect the C_r value to be poor for all forms of selection. This is indeed the case, with C_r never exceeding 0.3. In contrast with the results for 2D Ising the highest C_r was with top & bottom selection, with less than half of the population selected - the other selection operators mostly give C_r around 0.2. More interesting is the result for fitness prediction on mutated individuals. A similar amount of information about fitness is gained using either top selection or bottom selection, both showing C_m values well within one standard deviation of each other; the best results are found when using top & bottom selection, particularly with a small proportion of the population where for most of the problem instances the C_m was over 0.9.

In Figures 5.10 and C.86 to C.94 we see the same problem with all univariate and trivariate interactions included in the model (that is, all cliques with $|K| = 2$ have been omitted). Again we see that top & bottom selection yields a more accurate fitness model



(a) 10% decimated model structure



(b) 50% decimated model structure

Figure 5.8: FPC against selection proportion for fully specified 400 bit 2D Ising problems with decimated model structures

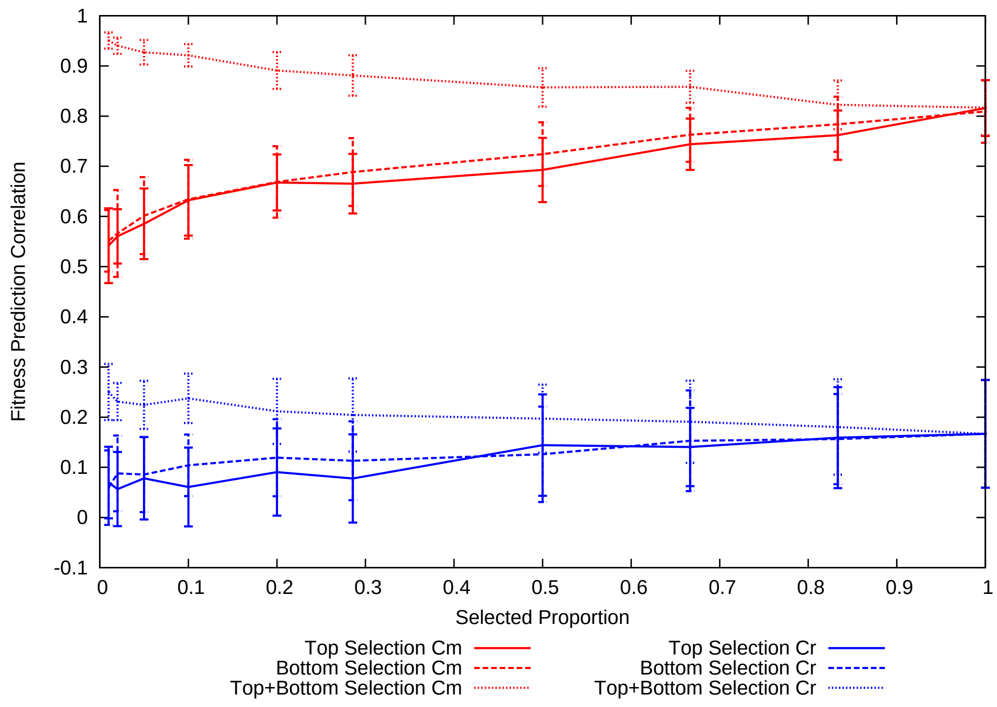


Figure 5.9: FPC against selection proportion for fully specified 150 bit MaxSAT problem with univariate model structure

of neighbouring solutions than the other two methods, while C_r values are low (0.4 and lower) for all selection operators.

5.4.3 Analysis

As with underspecified systems with perfect model structures, when we use an imperfect structure with a fully specified system we also see an improvement in model quality by using some form of selection. For the Ising problem, top selection gives the best model; for MAXSAT, top & bottom selection gives the best results. As might be expected, the models are unable to predict fitness of random individuals; however with appropriate choice of selection operator the model shows a strong correlation between predicted and true fitness values of solutions near the existing population. This effect was also observed to a limited extent for underspecified systems. The ability to predict the change in fitness of individuals following a mutation is potentially useful for incorporating the MFM into algorithms employing short-distance mutation for optimisation. We will make use of this

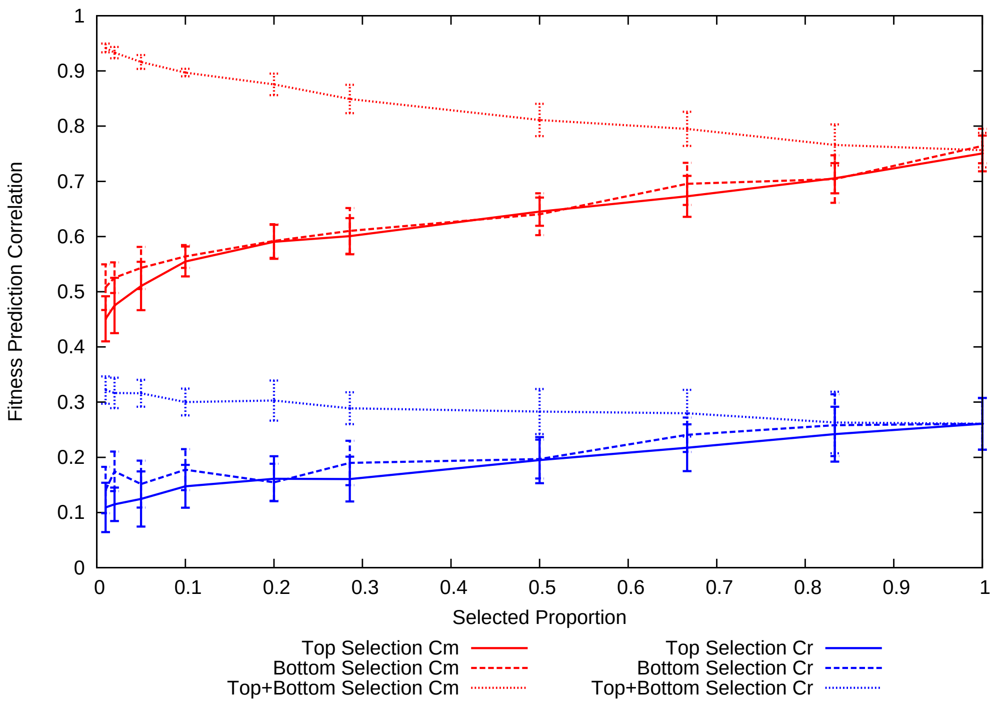


Figure 5.10: FPC against selection proportion for fully specified 150 bit MaxSAT problem with trivariate and univariate model structure

in an evolutionary version of DEUM applied to Ising problems in Chapter 7.

In the wider context of EAs we believe that this complements the other work which shows that algorithms which include information from the high and low fitness areas of the population can yield better performance. The algorithm presented in (Pošík & Franc 2007) models a contour line on the fitness landscape between high and low fitness individuals which the authors describe as a special case of the Learnable Evolution Model (LEM) (Michalski 2000). LEM uses machine learning techniques to determine the features which distinguish high and low fitness individuals. The key difference between those works and this is that following selection the MFM incorporates the fitness values directly into the model, allowing it to be used with more diverse operators such as selection of the whole population. (Miquélez et al. 2004) describes an algorithm which groups individuals of similar fitness into classes which are then passed to Bayesian classifiers that can be sampled to generate individuals of high fitness. That paper also raises two other possibilities for making use of individuals of all fitnesses; weighting the individuals by fitness when building the model and adding fitness as an extra variable to be passed to the probabilistic model. This latter approach is discussed in detail in (Miquélez, Bengoetxea, Mendiburu & Larrañaga 2007) and bears some similarity to the MFM concept but continues to make use of a Bayesian classifier (directed network) rather than an undirected network. That work also concentrates on continuous problems in contrast to the discrete problems looked at in this thesis.

The results presented here show that the MFM also benefits from being supplied with individuals from high and low fitness parts of of the population. Indeed, the results which show a strong correlation between the model and the fitness function for bottom selection and top & bottom selection would indicate that there is considerable information about fitness in the poorer parts of a population. This information could be beneficial to reproduction operators (whether they be probabilistic models or more traditional genetic operators) if they were designed to incorporate this information about fitness.

The results also indicate that selection plays an important role in gathering fitness information from the population. This is particularly so for the experiments which show top & bottom selection to give a model more strongly correlated with the fitness function than the other selection operators. This indicates that selection is able to sharpen to fitness information within the population by only supplying the high and low (rather than mid) fitness individuals for building the model. The MFM approach is able to make use of this by incorporating the fitnesses directly into the model; other algorithms which estimate a distribution from selected individuals would need to be modified to make use of top & bottom selection.

As the model is unable to perfectly or closely fit the fitness function it is not likely that sampling it as a surrogate for the fitness function will reach a global optimum of the fitness function. The main impact of this is that a different approach to optimisation using the MFM concept would need to be employed. Rather than building the model once then sampling it to find the optimum as in previous work (Brownlee et al. 2007, Shakya et al. 2006, Shakya et al. 2005b) we can use the ability to model neighbouring populations to push the population gradually towards the optimum. In summary, reverting to a more typical evolutionary approach. In this case, population size becomes more important as the number of function evaluations to create a population is multiplied by the number of generations - this means an underspecified system is more likely to be the norm. To conteract the effects of a small population and imperfect model, selection is more likely to play a bigger role and the results here will be useful in matching a selection operator to a given situation.

5.5 Summary

This chapter has described a series of experiments that investigate the effect of different approaches to selection on the ability of the MFM to model fitness for certain benchmark functions. We have seen that while the traditional truncation selection often results in a good fitness model, in the MFM approach comparable fitness models can also be produced when selecting different parts of the population (or indeed all of it). From the results presented here, it is clear that selection is particularly important when the fitness model is unlikely to perfectly match the fitness function. This is the case when variable interactions have been learned from data rather than given as part of the problem, which particularly relevant for later work in this thesis looking at optimisation of problems without the structure being supplied to the algorithm. An explanation for this behaviour may be that dependent on the fitness function, higher fitness individuals could be more similar to each other than lower fitness individuals. Thus the individuals in the mutated population will be similar to more of the individuals in the selected population than would be the case for a poorer selected set.

In situations where the model structure is imperfect, selection improves the ability of the algorithm to accurately model neighbouring areas of the search space - aiding the movement towards a global optimum. This shows that a well-tuned computationally cheap selection operator can be used to improve the accuracy of a model with a simpler structure. The results also show that selection can improve the fitness modelling capability of the MFM with an underspecified system, which results from using a smaller population. An optimisation algorithm employing the Markov fitness model could use a well-chosen selection operator to improve the fitness modelling capability of a more computationally efficient model structure with an underspecified population size, in place of the large computational expense required to build a model with a perfect structure and overspecified population. This information will be useful in developing more efficient optimisation.

We have also seen that in some circumstances the pure "top selection" operator can result in a model less closely fitted to the fitness function than that resulting from a "top & bottom selection" which selects and equal number of high and poor fitness solutions. "Bottom selection" and even selection of the whole population also result in models that show a strong positive correlation with the fitness function. This is in line with other work on selection - from the introduction of operators like tournament selection to more recent work on EDAs. It reinforces the idea that a considerable improvement in the performance (in this case the fitness modelling capability of the model) may be obtained by correct choice of selection operator.

In summary, the main contributions we can make based on the results from this chapter are as follows. For the benchmark functions tested, if the MFM is supplied with a perfect structure and a large enough population to be fully specified, then any of the selection operators tried here will result in a model which can predict fitness well. When the population is smaller (underspecified) then the standard top selection gives the best model, with the other operators resulting in models with no or a negative correlation to fitness. With an imperfect structure, again the top selection operator produces a good model in most cases, but for MAXSAT the selection of both top and bottom parts of the population gives the best model.

It will be interesting to extend this work to cover a much larger range of problems, to determine more precisely when the different approaches to selection should be used. It will also be useful to look at the effects of other selection operators such as fitness proportionate selection and tournament selection.

Chapter 6

Optimisation using Fixed Structure Markov Networks

We have seen over the previous chapters that a Markov network may be used to build a Markov Fitness Model (MFM) with a close correlation to the fitness function. Function optimisation has been one of the driving purposes for research in evolutionary computation and so this represents an important area to look at in more detail. The DEUM framework (Shakya et al. 2006, McCall, Petrovski & Shakya 2008, Shakya et al. 2005b) uses the Markov fitness model for optimisation. It has been successfully applied to the optimisation of a range of objective functions when provided with the structure of known variable interactions. Two problems used for benchmarking DEUM have been onemax (Shakya et al. 2005b) and the Ising spin glass problem (Shakya et al. 2006). In this chapter we first describe an existing approach to sampling the MFM to find solutions of probable high fitness, taken from those papers. Our contributions are to incorporate different structures within the algorithm to those used previously and changes to the operation of the Gibbs sampler including its cooling scheme. We then discuss the impact of the fitness model quality as defined by the C_r measure on optimisation; this gives us a new way of theoretically analysing the algorithm. In the remainder of the chapter we apply the DEUM framework to a number of benchmark functions it has not been previously applied to. In the experiments described in this chapter, problem-specific information is supplied to the algorithm in the form of a fixed structure which specifies the interactions present between variables. This is similar to the technique originally used by the Factorized Distribution Algorithm (Mühlenbein et al. 1999) which requires the additively decomposable function for a problem to be supplied to the algorithm. Chapter 7 will go on to look at optimisation without this requirement where the structure is instead learned by analysis of the population.

6.1 Sampling the MFM

6.1.1 Probability Vector

To generate a new population the DEUM framework can employ a number of sampling techniques. Earlier versions of DEUM used a PBIL-style probability vector; subsequently this was replaced by direct sampling of the Markov network. The PBIL-style probability vector as used by DEUM_{pv} (Shakya et al. 2004b) can be discounted for use with a multivariate model. This is because it implicitly factorises the model to a univariate structure by only storing a marginal probability distribution for each variable. Even if the marginal distributions are updated using a multivariate model, when they are sampled to generate new individuals the interactions have already been lost. A matrix of conditional probabilities can be used in place of the probability vector to allow interactions to be stored. This would rapidly grow in space complexity and instead we now discuss how the probability vector can be completely replaced by direct sampling from the Markov network with an appropriate cooling scheme.

6.1.2 Gibbs Sampler

In (Shakya et al. 2005b) a zero-temperature Metropolis method was used to directly sample the Markov network. In (Shakya et al. 2006) it was found that the Gibbs sampler performed better than the zero-temperature Metropolis sampler on larger Ising problems. The Gibbs sampler can be fine-tuned for different problems using a cooling rate coefficient and because of this is the technique used in all of the experiments described in this chapter. The Gibbs sampler repeatedly samples marginal probabilities for individual variables with the aim of reducing the energy of the individual and thus increasing its fitness. This continues until no further reduction in energy is possible or until a maximum number of iterations is reached. For each variable x_i the marginal probability of that variable taking the value 1 is given by:

$$p(x_i = 1) = \frac{1}{1 + e^{2\omega_i / T}} \tag{6.1}$$

where T is a temperature constant and ω_i is an energy function for all the cliques which contain x_i, defined in terms of Walsh functions $W_K(x)$:

$$\omega_i = \sum_{K \supset \{i\}} \alpha_K W_K(x) \tag{6.2}$$

The temperature T falls over the run of the Gibbs sampler according to a cooling scheme. The temperature at iteration g is defined in (6.3). τ is the *cooling rate* parameter which allows us to control the convergence rate of the sampler.

$$T_g = \frac{1}{\tau g} \tag{6.3}$$

In the work described here, we have made methodological changes to the implementation of the algorithm used in (Shakya et al. 2006). These were found empirically to yield better results. Firstly, in some of the experiments we now adopt the exponential cooling scheme proposed by Kirkpatrick in (Kirkpatrick, Gerlatt & Vecchi 1983) in place of (6.3). The scheme starts with an initial temperature T_o, and at iteration g the temperature is given by:

$$T_g = \lambda T_{g-1} \tag{6.4}$$

where λ is a constant in the range $0 < \lambda < 1$. This scheme was found to give better results than the cooling scheme described in (Shakya et al. 2006) for some problems; the description of each experiment makes clear which cooling scheme was used.

Secondly, the Bitwise Gibbs Sampler used in (Shakya et al. 2006) is modified so that bits are now sampled at random rather than in a raster scan. The sampler runs until no further improvement to the current individual or a problem-dependent maximum number of iterations have completed. This change was made to remove any implicit bias towards certain optima caused by always sampling the bits in the same order. We call the resulting sampler the Random Walk Gibbs Sampler, and its workflow is given in Algorithm 6.1. This is used in all of the optimisation experiments presented in this chapter.

Algorithm 6.1 Random Walk Gibbs Sampler

```
1: for all individuals $x^o$ in the previous population do
2:    Set $g = 0$ and set initial value for $T$
3:    repeat
4:       Set $x^{tmp} = x^o$
5:       Pick a variable $x_i^o$ at random
6:       Compute marginal probability distribution for $x_i^o$ according to (6.1)
7:       Sample distribution to obtain new value for $x_i^o$
8:       Increase $g$ by 1
9:    until $x^{tmp} = x^o$ or $g = 10000$
10:   Terminate with answer $x^o$
11: end for
```

6.1.3 DEUM with Gibbs Sampler

Algorithm 6.2 DEUM with Gibbs sampler and fixed structure

```
1: Set structure of Markov network depending on problem
2: Generate an initial population, $p$, of size $M$ with uniform distribution.
3: while stopping criteria not met do
4:    Select a subset $\sigma$, the fittest $|\sigma|$ members of $p$
5:    Calculate the Markov Network parameters by making a maximum likelihood esti-
      mation from the selected population
6:    Calculate $C_m$ and $C_r$
7:    repeat
8:       Run Gibbs sampler to sample a new individual from the Markov network
9:    until $|\sigma|$ individuals or an optimal individual is generated
10:   if single-generation DEUM then
11:      Terminate with the fittest solution found in step 7
12:   else
13:      Replace poorest $|\sigma|$ in $p$ with population generated in Step 7
14:   end if
15: end while
16: Terminate with the fittest solution found over complete run
```

Incorporating the Gibbs sampler into DEUM gives us Algorithm 6.2. At Step 4, selection is not used if it was found to result in a poorer fitness model. Step 6 is not a functional part of the algorithm but in the experimental results where the FPC values are reported this is the point in the algorithm's run at which they were calculated. This procedure is adapted throughout this chapter depending on the problem. Step 10 represents a parameter chosen before the algorithm runs: in many of the experiments, once the MFM is built using a large enough population, repeatedly sampling the model will yield the global optimum. In this case there is only a single generation to the algorithm (which we refer to here as *single-generation DEUM*). Single-generation DEUM may be likened to simulated annealing (Kirkpatrick et al. 1983), although rather than repeatedly calling the fitness function we make repeated use of the fitness model instead.

6.2 Fitness modelling and optimisation

Chapters 3 to 5 have given us a theoretical and experimental background to the factors involved in building a model of fitness which closely correlates to the fitness function. In this chapter we wish to use the fitness model for optimisation and it is important to realise the link between fitness prediction capability and optimisation. This section describes an experiment that demonstrates a clear link between the fitness prediction capability and the optimisation capability of the algorithm.

The approach for this experiment follows the same template as the population size experiments in Chapter 4. We run the DEUM optimisation described in Section 6.1.3 for multiple population sizes which cover both underspecified and overspecified systems. Alongside the fitness prediction correlation we also consider the proportion of 30 runs which successfully found a global optimum (that is, the success rate of the optimisation algorithm). The algorithm runs for a single generation, finishing with repeated runs of the Gibbs sampler. The temperature used for the Gibbs sampler was computed using (6.3). Parameters for the algorithm on both problems are the same with the exception of the cooling rate parameter which is set to 0.005 for onemax and 0.0005 for Ising. The rate for Ising is the same as that used in (Shakya et al. 2006) and the rate for onemax was determined by increasing the rate (while keeping all other parameters fixed) until a 100%

success rate was achieved.

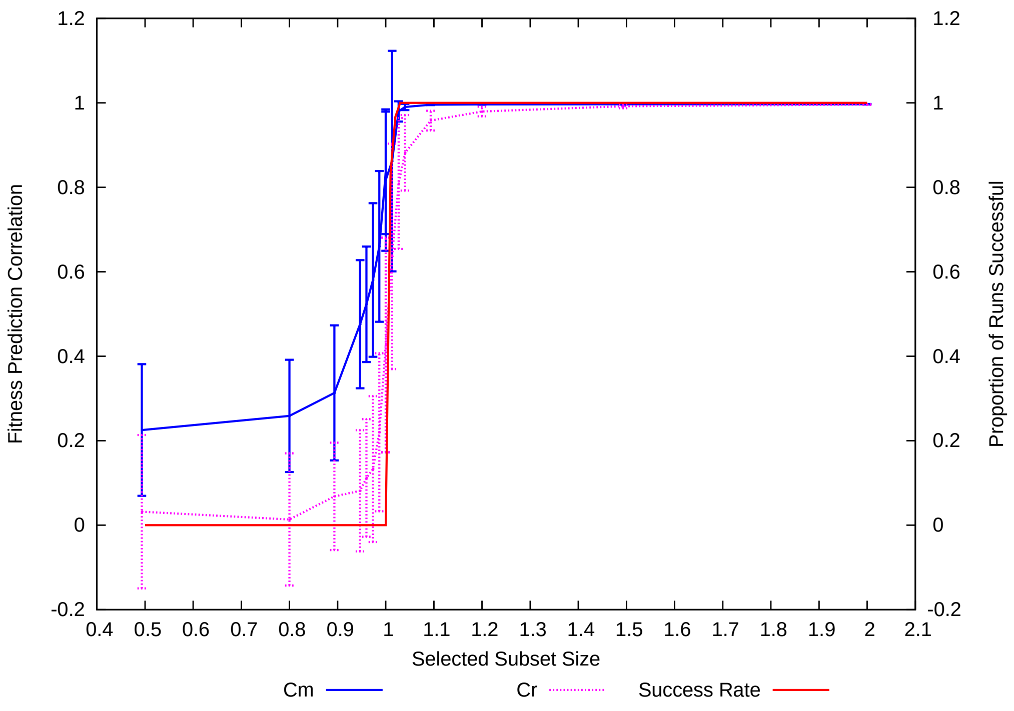
This experiment differs slightly from those in Chapter 4 in that truncation selection is used to properly reproduce the results given in (Shakya et al. 2006). Thus the horizontal axis of these graphs is the size of the selected set and not the starting population size. To maintain the same selective pressure in all experiments, the desired size of σ was determined and the initial population was generated to be four times this size.

6.2.1 Results

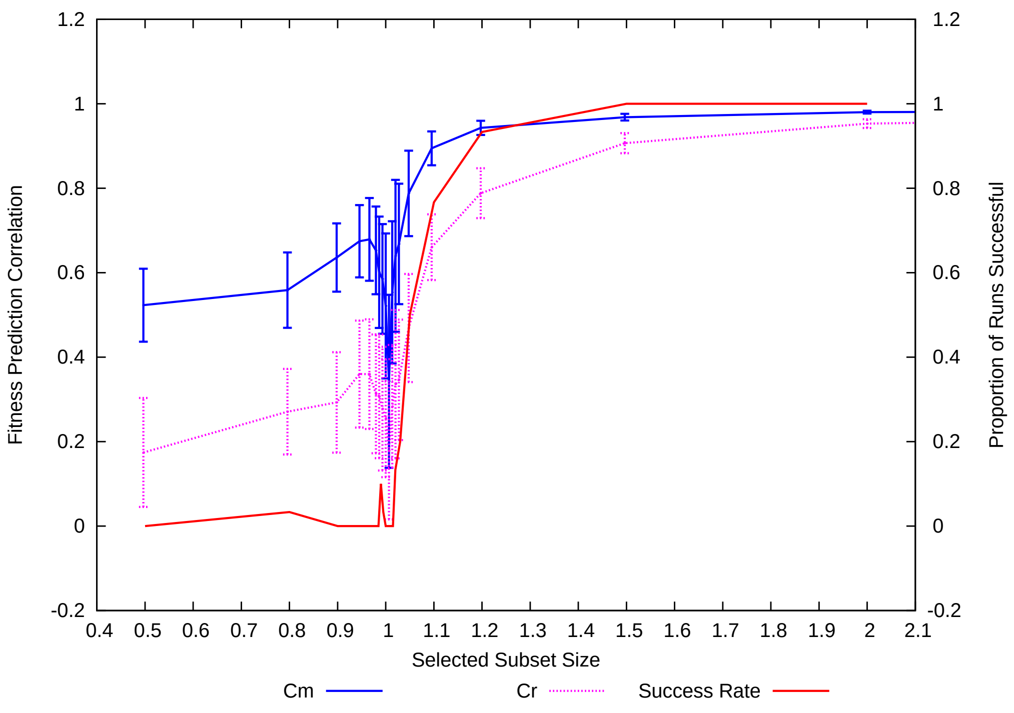
First we look at the results for a 100 bit onemax problem in Figure 6.1a. Here, the C_m and C_r are plotted on the left vertical axis against increasing number of individuals selected $|\sigma|$. As we saw in Chapter 4 both values show a rapid increase as $|\sigma|$ exceeds N. Plotted on the right vertical axis is the success rate of the optimisation stage of the algorithm. We can see that while the population remains underspecified there are no runs which find the global optimum. At the point where $|\sigma|$ exceeds N, alongside the jump in fitness prediction capability we see a similar jump in problem solving capability which remains high as the selected set grows even larger. The results for the Ising problem in Figure 6.1b show a similar leap in problem solving capability, although there is one instance where the global solution is found by the algorithm with an underspecified system. This may be a random occurence or could be attributed to the unique properties of the Ising problem in the context of fitness modelling, discussed in Chapter 4.

Figure 6.2 shows the dropoff in fitness prediction capability caused by removing parts of the model structure at random, together with the corresponding dropoff in optimisation capability. The success rate drops very quickly, indicating that where we have an imperfect model structure we are also unlikely to be able to use the single-step version of DEUM.

These results demonstrate a strong link between the fitness modelling capability discussed in previous chapters and optimisation capability. They highlight an important drawback to the single-step DEUM approach; the system of equations must be fully specified using a near-perfect structure to produce a model which can be directly sampled to find the global optimum. Typically where the algorithm fails in both cases the Gibbs sam-



(a) 100 bit onemax



(b) 100 bit Ising

Figure 6.1: Effect of population size on FPC and optimisation capability

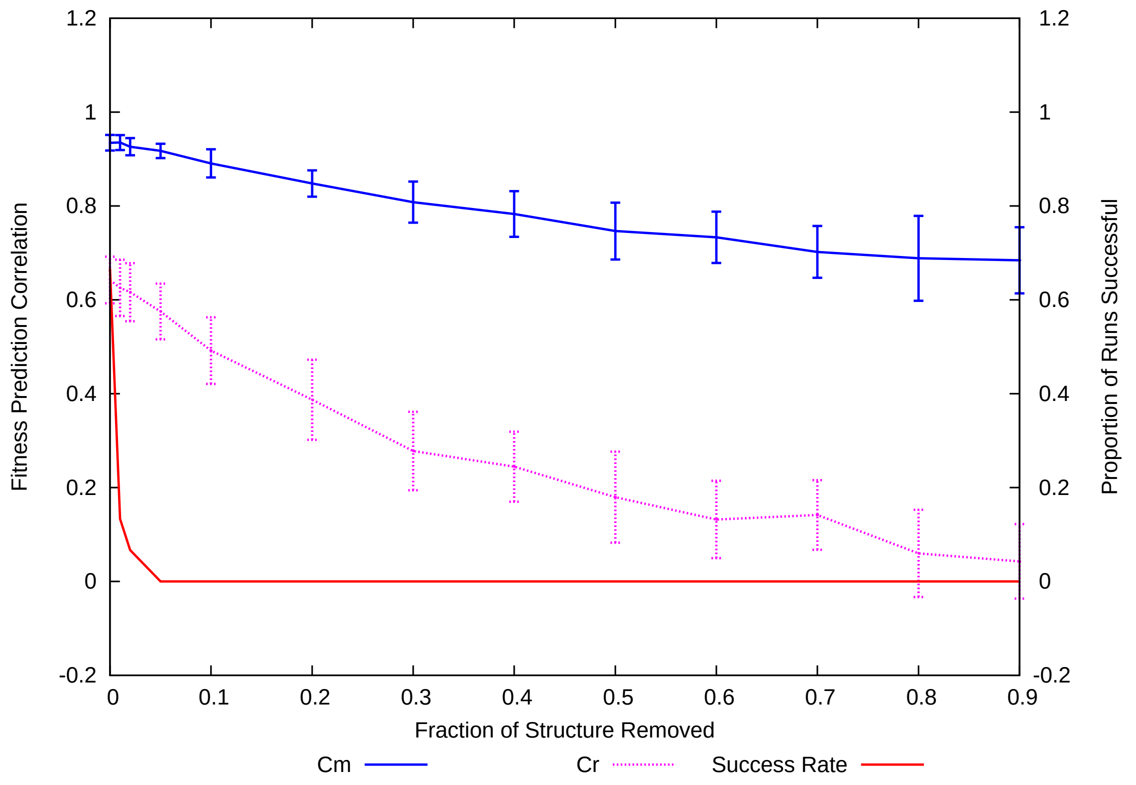


Figure 6.2: Effect of structure decimation on FPC and optimisation capability

pler is producing locally optimal individuals very close in fitness to the global optimum. This confirms the previous observation that the model is still providing useful information about the fitness function. The logical progression of this is that to develop a general purpose optimisation algorithm employing Markov network fitness modelling we will need to consider additional techniques to enhance the approach by making use of these locally optimal solutions. The foremost of these would be the use of multiple generations (making a true evolutionary algorithm within the DEUM framework).

In the rest of this chapter we see a mixture of both approaches. The benchmark problems with a known structure employ the single step version of the algorithm; the biocontrol problem in Section 6.3.4 is a real-world black-box problem which needs the additional power of the multi-generation algorithm.

6.3 Optimisation Experiments

In this section, we run a number of experiments applying DEUM to different problems and comparing its performance to that of other algorithms. The object of this series of experiments was to demonstrate the application of DEUM to optimisation problems of an increasing level of complexity. The novelty in this work is the presence of a number of different bivariate and multivariate models, whereas previous work used a univariate model or in one instance a 2D lattice (Shakya et al. 2006).

Each sub section describes an application of the algorithm to a particular fitness function. Detailed descriptions of the fitness functions can be found in Chapter 2 - here the summary of each experiment begins with a short description of the aspects relevant to optimisation. The algorithms which DEUM are compared with are then described with the reasons for their selection. Then follow the results with a table showing a detailed breakdown of the performance of DEUM and a graph comparing it with the other algorithms. The detailed results for DEUM are arranged in a table which depicts:

- PS - The problem size (number of variables).
- FE - The number of fitness evaluations required by DEUM to find an optimum, including evaluation of the initial population and the population produced by the Gibbs sampler, averaged over the 30 runs.
- FE-SD - The corresponding standard deviation for the number of fitness evaluations.
- IT - The average number of iterations of the Gibbs sampler - each iteration contains one marginal probability calculation.
- IT-SD - The standard deviation of Gibbs sampler runs
- SR - The success rate; the percentage of runs which found the global optimum.
- C_r - The mean FPC of the model for a population of randomly generated individuals given for comparison.
- C_r-SD - The corresponding standard deviation of the FPC.

In the graphical comparisons of algorithms the error bars represent one standard deviation in the set of results. In both the FE and IT figures the means only include successful runs to avoid being skewed by the unsuccessful runs in which the Gibbs sampler was restarted 10 x population size times. This also applies to the graphical comparison of the different algorithms. Each sub section then ends with an analysis of the results. The last experiment (Bio-control in Mushroom Farming) goes into some more detailed analysis of the finishing population and the effects of seeding the population to provide some additional comparisons with other algorithms applied to the problem.

6.3.1 1D Checkerboard

First we start with the chain structured problem 1D checkerboard. This is a simplification of the 2D Checkerboard problem used for benchmarking in (Baluja & Davies 1997b), designed to be easier to solve because it has half as many interactions. The model has one term for each variable and a coupling term between neigbouring variables. We saw in Chapter 4 that with a large enough population and bivariate structure a good model of fitness can be obtained for this problem. In Chapter 3 we saw that the parameters estimated for the chain model can be analysed manually to determine the global optimum for the problem.

6.3.1.1 Algorithms used for Comparison

Here we compare against a run using UMDA, MIMIC and hBOA. UMDA was chosen to demonstrate the performance of an EDA using a univariate model on this problem. It was chosen in preference to other univariate EDAs because the implementation of hBOA was used includes UMDA and this allowed for a fairer comparison, using exactly the same code base for those algorithms. MIMIC uses a chain structure which should closely match the structure of the problem - consequently, MIMIC should perform well on this problem. To mitigate the effect of premature convergence, a mutation operator was added to MIMIC which flips bits in newly generated individuals. hBOA is chosen as an algorithm with sophisticated structure learning which has been demonstrated to perform well on higher order problems. We do not compare with univariate DEUM$_\mathrm{d}$ because as we saw in the previous section, even slight decimation of the model (removing a few interactions) greatly reduces the optimisation capability. Univariate DEUM includes no interactions and cannot be expected to perform as well as implementations which do include some structure.

6.3.1.2 Method

We start by defining the model structure used by DEUM to solve this problem; we used the chain structure as defined in Chapter 3. For reference, this is repeated in (6.5).

$$U(x) = \alpha_0 + \sum_{i=1}^{n} \alpha_i x_i + \sum_{i=1}^{n-1} \alpha_{ij} x_i x_j \tag{6.5}$$

This experiment is repeated over a number of different problem sizes, from 10 bits up to 200 bits. We use the Algorithm 6.2 with no explicit selection operator and populations of $1.1N$ and $2N$ individuals (where N is the number of terms in the energy function). The single-generation version of the algorithm is used; the model is repeatedly sampled using a Gibbs sampler until an optimal individual is found. The Gibbs sampler temperature is calculated using (6.3) with a cooling rate parameter τ of 0.01. This was found by repeatedly running the algorithm with cooling rates from 1 to 0.0001, then choosing the setting at which most runs ended successully finding the global optimum. To determine the optimal parameters for the other algorithms each was run on the 100 bit version of the problem with a range of different values for selection pressure for both tournament and truncation selection. The bisection technique (Pelikan 2005) was used to determine optimal population size. The bit-flip probability for the mutation operator added to MIMIC was determined by repeatedly running the algorithm with values from 0.0001 up to 0.5 and choosing the value which gave algorithm the highest success rate.

6.3.1.3 Results

During the parameter setting phase of the experiments it was clear that regardless of selection type and population size (populations of up to 10^6 individuals were tried), UMDA was unable to find the optimum for this problem.

Table 6.1 shows the experimental results of the performance of DEUM on differently sized instances of the 1D Checkerboard problem using a population size of 1.1N to estimate the model parameters. The number of function evaluations required to find an optimum is also represented graphically alongside that of MIMIC and hBOA in Figure 6.3. MIMIC shows very large variations in run time; the lack of error bars on the 100 bit point for MIMIC is because there was only one successful run so standard deviation was zero. At 200 bits none of the MIMIC runs found the optimum.

PS	FE	FE-SD	IT	IT-SD	SR	C_r	C_r-SD
10	98	24	962	1326	57	0.615	0.145
20	205	43	5363	6157	47	0.733	0.100
50	512	80	15772	15771	67	0.883	0.045
100	1109	288	53730	65202	43	0.941	0.013
200	2617	389	267834	120644	43	0.967	0.010

Table 6.1: Performance of DEUM on 1D Checkerboard Problem

These results are poor even on the small instances of the problem: although the number of function evaluations required are low for the successful runs, the success rate is lower than 50% for most sizes of the problem that were tried. The C_r values are also lower than those seen for other problems (see later sections in this chapter), and based on the results in Chapter 4, a higher value for the fitness prediction capability for this problem could be obtained by increasing the population size to $2N$. Thus the experiment was repeated with the increased population size and the results are given in Table 6.2. We can see that with the increased population size, the algorithm was able to successfully find the global optimum for all instanced of the problem.

PS	FE	FE-SD	IT	IT-SD	SR	C_r	C_r-SD
10	156	3	343	224	100	0.870	0.063
20	316	3	660	522	100	0.937	0.016
50	801	10	1733	1819	100	0.975	0.006
100	1620	27	6715	6309	100	0.986	0.003
200	3408	172	70762	56158	100	0.993	0.002

Table 6.2: Performance of DEUM on 1D Checkerboard Problem with Population Size 2N

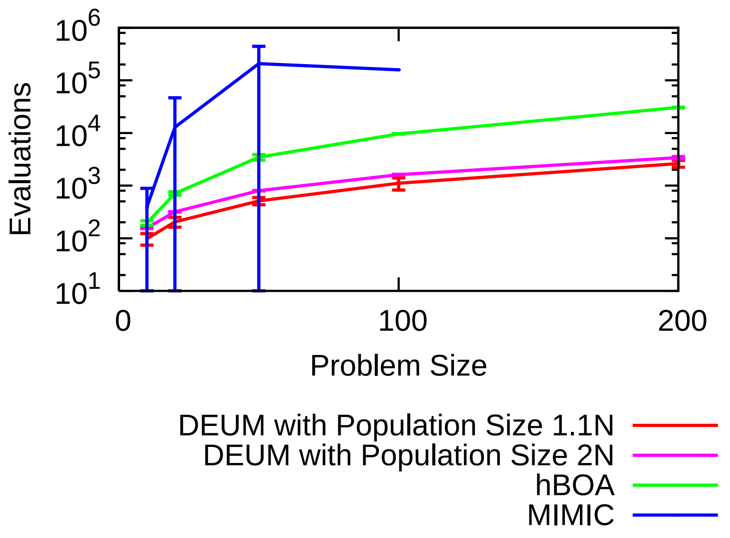


Figure 6.3: Performance of DEUM on 1D Checkerboard Problem

PS	SR
10	100
20	79
50	2
100	1
200	0

Table 6.3: Success Rate for MIMIC on 1D Checkerboard Problem

6.3.1.4 Analysis

The poor performance of UMDA is the expected behaviour as the univariate probability distribution does not account for the crucial interactions between neighbouring variables in the bitstring. MIMIC uses a chain structure and might be expected to perform well on this problem; however despite also running the algorithm with very large populations and a range of other parameter variations this was not the case. The success rate after 100 independent runs for MIMIC is shown in Table 6.3. For the successful runs, MIMIC was able to use relatively few function evaluations. In contrast hBOA was able to solve all of the problems given a large enough population although this came at the cost of using a large number of function evaluations.

The important result of this experiment for us is that DEUM was able to successfully optimise the problem with a large enough population. We saw in Chapter 4 that a population size of $1.1N$ was enough to build a model which closely fitted to the probability distribution, although for the 1D Checkerboard problem the fitness prediction capability of the model continued to rise with population size up to and over $2N$. There is clearly an additional dynamic to the problem - perhaps that has no univariate element at all - which affects the construction of the model from a random population. This is borne out in the optimisation success rate increasing to 100% when the population reaches this size.

It is also important to note the substantial overhead of DEUM using an overspecified system with perfect structure - the reason the experiments did not include larger instances of the problem. This is because the memory requirement and model build time for a fully specified system at best grows by $O(N^3)$ with the number of model parameters (which in this case grows linearly with the problem size) and sufficient resources were not available to accomodate the algorithm. For illustration, on the 2 GHz machine available, to build a single model for a 100 bit instance of the problem (201 model parameters) took around 10 seconds. The 200 bit instances took around 100 seconds each; further doubling the problem size resulted in a similar tenfold increase in time, and this was multiplied up by the number of repeated runs of the algorithm for parameter setting and statistically significant results. In contrast, hBOA was able to solve the problem up to size 1000 bits - albeit with a very large population (11200 individuals) - taking minutes in contrast to hours.

6.3.2 2D Checkerboard

2D Checkerboard is very similar in structure to the 2D Ising problem; the structure is also a 2D lattice, although it does not wrap around at the edges. The optimum is always a grid of alternating 1s and 0s rather than being dependent on the specific instance of the problem.

6.3.2.1 Algorithms used for comparison

We compare to the results found for runs of hBOA and UMDA. UMDA is again chosen as an algorithm with small overhead and with the simplest probabilistic model (univariate) and hBOA is chosen as an algorithm with a multivariate model which has already been shown to work well for a similar problem (2D Ising).

6.3.2.2 Method

Again we start by defining the model structure used for this problem. In the general energy function for the 2D Checkerboard problem we have a constant, a term for each of the variables x_i, and a term for the bivariate interactions. This is repeated in (6.6) for reference.

$$U(x_i) = \alpha_0 + \sum_{i=1}^{l}\sum_{j=1}^{l}(\alpha_{ij}x_{ij} + \alpha_{ij,(i+1)j}x_{ij}x_{(i+1)j} + \alpha_{ij,i(j+1)}x_{ij}x_{i(j+1)}) \tag{6.6}$$

Nine sizes for the problem were used from 16 bits up 400 bits. The single generation version of Algorithm 6.2 was run on each problem 30 times with different random starting populations. The model was built by using standard truncation selection to select the top 25% of the population, and the population size was chosen so that the number of individuals selected was always $1.1N$ (giving an overspecified system). The selection proportion of 25% represents a balance between having a high selective pressure and not needing to have a huge population in order to still select $1.1N$ individuals. It was chosen in light of the results in Chapter 5 which show that (at least for the benchmark functions tried there) with a perfect structure and fully specified system a high selective pressure will result in an MFM with a high fitness prediction capability. The Gibbs sampler temperature coefficient was computed using (6.3) with $\tau = 0.005$. The sampler was limited to 2000 iterations.

6.3.2.3 Results

Table 6.4 shows the experimental results of the performance of DEUM on differently sized instances of the 2D Checkerboard problem. The number of function evaluations required to find an optimum is also represented graphically in Figure 6.4.

PS	FE	FE-SD	IT	IT-SD	SR	C_r	C_r-SD
16	121.05	0.23	447.26	196.31	63	0.912784	0.115555
25	213	0	558.18	161.63	73	0.989496	0.006504
36	333	0	623.64	190.82	83	0.993608	0.006333
49	477	0	813.63	307.81	80	0.9966	0.002237
64	649	0	805.48	100.7	90	0.996742	0.002844
100	1073	0	1019.26	253.59	90	0.998135	0.00116
256	2973	0	1305.63	285.12	100	0.999357	0.000225
324	3817	0	1734.83	602.3	77	0.999411	0.000178
400	4769	0	1560.27	440.76	87	0.999436	0.000235

Table 6.4: Performance of DEUM on selected instances of the 2D Checkerboard Problem

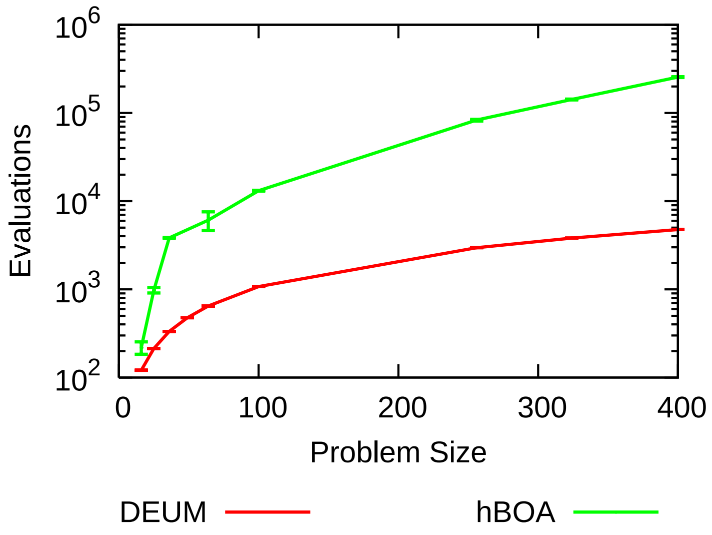


Figure 6.4: Fitness evaluations required by DEUM and hBOA to solve instances of the 2D Checkerboard Problem

6.3.2.4 Analysis

As in the experiments on 1D checkerboard, this has show that DEUM is able to solve a bivariate problem with a smaller number of function evaluations than hBOA, albeit with a higher overhead.

One striking aspect of the results are that most of the FE values have a standard deviation of zero. This is because the runs in which the algorithm found the global optimum almost always did so with ths first run of the Gibbs sampler. This used one function evaluation to compute the fitness of the newly generated individual. The structure was the same size N for all runs in a particular instance of the problem, meaning that the number of selected individuals was always $1.1N$ and thus the population was the same size $4.4N$. This meant that the number of evaluations ϵ required to find the global optimum was constant and as defined in (6.7).

$$\epsilon = 4(1.1N + 1) \tag{6.7}$$

6.3.3 3-CNF MaxSAT

We now move on from bivariate problems to a problem with interactions of a higher order: 3-CNF MAXSAT. We saw in Chapter 2 that this problem also shows interactions between groups of three variables, and in Chapter 4 we saw that the three-way "trivariate" interactions are also important for building a good model of fitness.

6.3.3.1 Approaches used for comparison

Pelikan's work in (Pelikan & Goldberg 2003) combined a deterministic hillclimber GSAT (Selman, Levesque & Mitchell 1992) with the multivariate EDA hBOA and compared the results to those for plain GSAT and for WalkSAT. GSAT starts with a randomly generated solution and runs in a steepest descent search, flipping the one bit the improves the solution most in each iteration. If it fails to find an optimum it is restarted with a new random solution. WalkSAT extends GSAT by adding random mutations - each iteration it either performs the greedy step of GSAT or chooses at random one bit from those present in unsatisfied clauses and mutates it. The probability p of choosing the greedy step or random mutation is a parameter of the algorithm which may be varied to improve performance on a particular instance of MAXSAT.

It was shown that WalkSAT outperformed hBOA+GSAT on the randomly generated 3-CNF MAXSAT problems that we will be looking at here; though both easily outperformed GSAT alone. Graph colouring problems translated into MAXSAT have more complex structure and less randomness; on these hBOA+GSAT did much better than both GSAT and WalkSAT. We do not look at the translated graph colouring problems here as the increased size of the test problems (500 variables) increases the model parameter learning time of DEUM considerably when using as fixed complete model structure as these experiments do. These will be worth returning to if a means of reducing the model building time is found.

The results for hBOA and WalkSAT presented in (Pelikan & Goldberg 2003) are used here for comparison with DEUM.

6.3.3.2 Method

Recall from Chapter 3 that the energy function for 3-CNF MaxSAT for each individual can be expressed as:

$$U(x) = \sum_{K} \alpha_K W_K(x) \quad where \quad \alpha_K \neq 0 \ \ \forall |K| \leq 3 \tag{6.8}$$

The single generation version of Algorithm 6.2 was run on the set of 3-CNF benchmark problems obtained from SATLIB (Hoos & Stützle 2000). The problem sizes were 20, 50, 75, 100, 125 and 150 (as used in previous chapters and (Pelikan & Goldberg 2003)) On each size of problem, the algorithm was repeated on 20 different instances selected at random from the set held by SATLIB. Each instance tested belongs to the phase transition region, the point at which the problems tip from generally solvable instances to generally unsolvable. At this point the number of clauses is equal to the number of predicates multiplied by 4.3. Each instance tested is from the set of those proven to be solvable.

The temperature for Gibbs sampler was computed using (6.4). The cooling rate parameter λ was 0.995 for problems up to size 100 and 0.999 for the larger problems. The sampler ran for a maximum of 10000 iterations. The population size used in each case was $1.1N$, and no explicit selection operator was used.

6.3.3.3 Results

As in previous sections, Table 6.5 shows the experimental results of the performance of DEUM on differently sized instances of the MAXSAT problem. Like before, the number of function evaluations required to find an optimum is also represented graphically in Figure

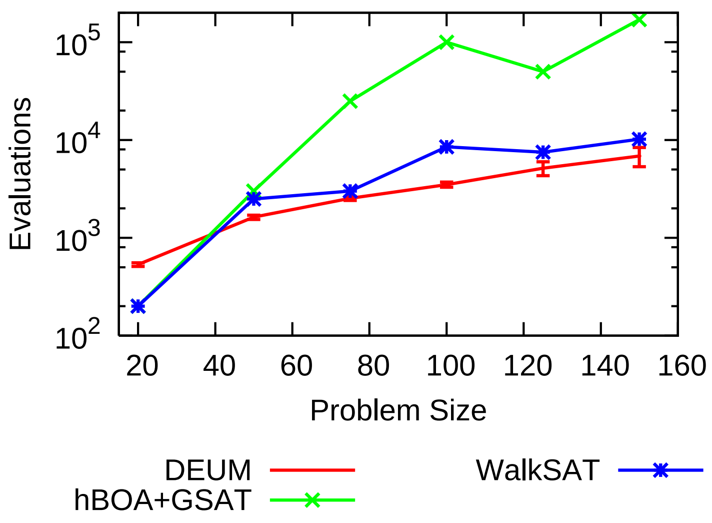


Figure 6.5: Performance of DEUM on MAXSAT Problem

6.5. This is the mean number of evaluations required to estimate model parameters plus the number required to confirm the true fitness of individuals generated by repeated runs of the Gibbs sampler.

PS	FE	FE-SD	IT	IT-SD	SR	C_r	C_r-SD
20	533	22	5.27x10^6	6.87 x10^4	100	0.9970	0.0011
50	1626	82	1.58 x10^7	1.87 x10^5	100	0.9984	0.0004
75	2980	608	2.49 x10^7	1.85 x10^5	100	0.9988	0.0002
100	3667	507	3.40 x10^7	1.17 x10^5	100	0.9992	0.0001
125	5151	826	4.32 x10^7	1.99 x10^5	65	0.9993	0.0001
150	6853	1510	5.25 x10^7	1.51 x10^5	70	0.9994	0.0001

Table 6.5: Performance of DEUM on MAXSAT Problem

6.3.3.4 Analysis

Previously hBOA had been tested on the same set of problems and had been reported to perform comparably with the MAXSAT-specific solver WalkSAT, when hBOA was used in a hybrid with the deterministic hillclimber GSAT. From Table 6.5 we can see that DEUM without GSAT requires significantly fewer fitness evaluations than reported for the hBOA+GSAT hybrid. For example, to solve instances of the problem at size 100 bits, DEUM requires an average of 3667 evaluations. This compares to 10^5 evaluations for hBOA + GSAT, although it must be noted that hBOA also learned the structure whereas it was supplied to DEUM in these experiments. DEUM also compares favourably with WalkSAT, which was reported to require 10^4 evaluations. The decreasing success rate with larger problems is most likely caused by the Gibbs sampler, which is partly dependent on the random start and is highly dependent on an optimal cooling rate and run time. In (Pelikan & Goldberg 2003) WalkSAT and hBOA were reported to have 100% rate on these instances of the problem. In addition, DEUM's overhead with the larger instances (for example, a 150 bit MAXSAT instance has around 2500 parameters in the MFM) meant that in contrast to hBOA's run time which could be measured in minutes, DEUM required several hours to build a single model.

6.3.4 Bio-Control in Mushroom Farming

Recall from Chapter 2 that the biocontol problem is a bang-bang control problem; the string of variables is a time series of points at which an intervention may or may not occur. It is a black box problem in that the structure of variable interactions is not as clearly defined as in the other fitness functions looked at in this chapter. In addition, the optimal fitness value is unknown, in contrast with well-known benchmark functions. Consequently we take a modified approach for this problem, returning to the use of evolution rather than the single step algorithm.

6.3.4.1 Method

DEUM has been applied to this problem using a univariate model (Wu, McCall, Godley, Brownlee & Cairns 2008); there it demonstrated comparable performance with the genetic algorithm approaches.

In this section this is extended to incorporate bivariate interactions, using a fixed chain model as described in Chapter 3. The motivation for this is the encoding used by the problem. As the bit string is a time series it is conceivable that neighbouring bits - which represent consecutive interventions - will have an interdependency. We incorporate these interactions in the model by adding α terms for each neighbouring pair of bits - the same chain model as we used for the 1D checkerboard problem. As a reminder, the chain model has the energy function shown in (6.9).

$$U(x) = \alpha_0 + \sum_{i=1}^{n} \alpha_i x_i + \sum_{i=1}^{n-1} \alpha_{ij} x_i x_j \tag{6.9}$$

The algorithm adopts the multi-genetaion version of the workflow in Algorithm 6.2. No explicit selection operator is used. Note here the difference to the workflow when applying DEUM to the other problems in this chapter - here a true evolutionary approach is used with multiple generations. With the other benchmark problems, the optimal fitness is known and as the algorithm was able to find individuals with this fitness in one generation - there was no justification in running for further generations. Here, the optimal fitness is unknown and it is worth running the algorithm for longer to keep improving fitness.

Further to this, the structure of the problem (that is, the relationships between variables) is also unknown and although the chain model follows the time series structure of the encoding used by the fitness function, there are likely to be further interactions and the model is unlikely to be a perfect fit. This is borne out by results from the chapter on fitness prediction correlation - for this problem after one generation the model has a C_r value of 0.8 rather than the values extremely close to 1.0 which we see for MaxSAT. This means that the model has some mismatches with the fitness function and the Gibbs sampler is unlikely to find the global optimum. This can be mitigated by allowing the algorithm to develop the model as the population focuses on areas of high fitness over several generations.

To compare the effectiveness of different approaches as closely as possible, we set up parameters to be identical wherever possible, using the original source code for TinSSel from the authors. One key difference is the population size: for DEUM this was set to 120 ($1.2N$). We take parameters for TinSSel from (Godley, Cairns & Cowie 2007a) noting in particular that a population size of 50 allows TinSSel fewer fitness evaluations over 200 generations than we allow for DEUM with a population of 120. However, both algorithms had converged by 200 generations on each experimental run. All parameters used are detailed in Table 6.6. The genetic algorithm using TInSSel adopts the same two-stage mutation operator as detailed in (Godley, Cairns & Cowie 2007a). DEUM replaces these operators with modelling and sampling.

Parameter	TInSSel	DEUM
Population size	50	120
Length of individual	50	50
Maximum generations	200	200
Selection operator	Truncation	Whole population
Crossover probability	1	N/A
Mutation probability	0.05	N/A
Cooling rate	N/A	0.95
Intervention Penalty P	50	50

Table 6.6: Experimental parameters

6.3.4.2 Results: Comparison by fitness

First we report the performance of each algorithm in terms of the quality of solutions reached after 200 generations. Running beyond this point yielded no further significant observations. Each algorithm was run with randomly generated initial populations operating with different values of initialisation control μ, from 4 to 50 possible interventions. (Below 4 both algorithms perform poorly as there is not enough information for either to detect good intervention points) We ran each algorithm 100 times for each value of μ.

In (Godley, Cairns & Cowie 2007a) population seeding was described as a means of improving the performance of genetic algorithms applied to this problem. In Figure 6.6 we see the best fitness found by each algorithm for different starting conditions. Remembering that this is a minimisation problem (that is, lower fitnesses are preferable), we can see that for populations constrained to have a low number of intervention points, TInSSel outperforms DEUM. TinSSel reaches a minimum fitness of just below 2000, comparable with that of 1983 in (Godley, Cairns & Cowie 2007a). However, as the number, μ, of interventions allowed in the starting population approaches the maximum possible (closer to the real world scenario) DEUM gives a marginally better result. It is clear that, once enough interventions are allowed in the starting population, both algorithms are finding very good solutions that may well be near optimal. However, while TinSSel appears

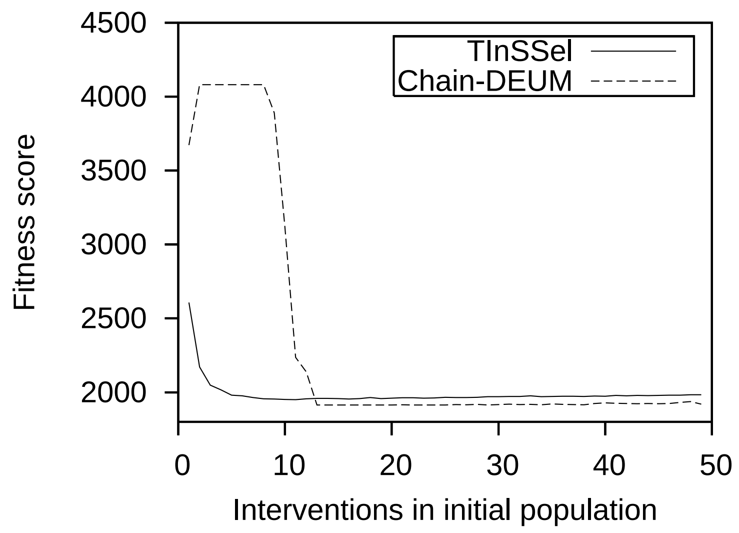


Figure 6.6: Fitness of best solution after 200 generations for various values of initialisation control limit μ

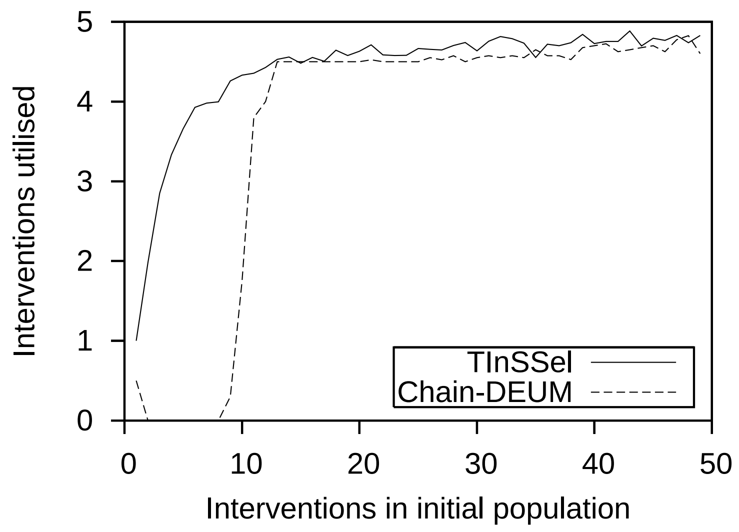


Figure 6.7: Intervention usage of best solution after 200 generations for various values of initialisation control limit μ

indifferent to the initial intervention control, DEUM is hindered by it, only achieving the best solutions when the control is removed altogether. It is worth also noting the rapid change in the best fitness found by DEUM as the number of interventions allowed in the initial population. We believe that this occurs at the point where the diversity in the population becomes high enough to build a useful model of fitness, with a clear tipping point similar to that seen with increasing population size in Chapter 4.

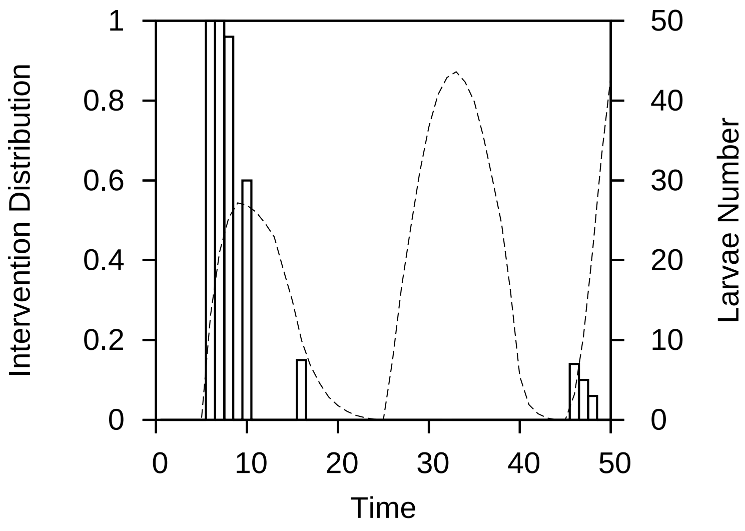


Figure 6.8: Probability of an intervention at each day (DEUM)

In Figure 6.7 we see the number of interventions present in the final solutions found by each algorithm. The fitness function penalises solutions with larger number of interventions but at least some are required to achieve the goal of larvae reduction. We see a similar effect to that shown for the raw fitness value - as the intervention control is relaxed, DEUM is more consistently finding solutions with a slightly smaller number of interventions.

6.3.4.3 Results: Intervention points in solutions

Now we explore in more detail the solutions found by DEUM when no initialisation control operated, i.e. $\mu = 50$. After each of 100 runs, we recorded the days on which interventions occurred in the best solution found. Figure 6.9 contains a histogram showing the distribution over all runs of an intervention occurring at each day. We have superimposed this on the larvae population graph obtained by running the model without interventions, i.e. the natural growth cycle of the larvae (this was originally derived by Fenton et. al. in (Fenton et al. 2002).

We can see that in the large majority of runs the interventions coincide with the early parts of the larval growth cycle, in particular, the first cycle occurring in the treatment period.

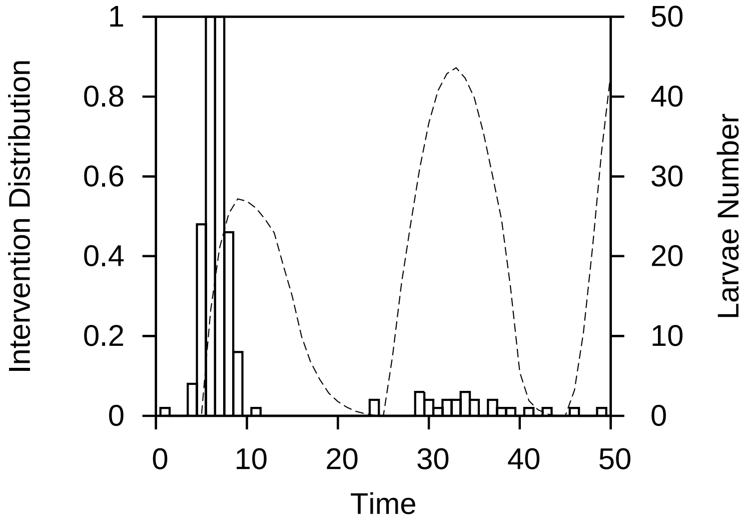


Figure 6.9: Probability of an intervention at each day (TInSSel)

The reason that few interventions occur over the second and third larval population cycles is that the destruction of the population by the first treatment would mean there would be few larvae remaining to breed and repopulate and so the expense of treatment would outweigh the benefit to be gained by treating. Occasionally an additional intervention is required later in the treatment period to catch any remaining larvae as they begin to re-populate.[1]

Figure 6.9 shows the same information produced from runs of TinSSel. Here we can see that, although the algorithm has found solutions which place interventions predominantly in the first cycle of larval population growth, the distribution of interventions is more diffuse and less obviously bound to the underlying system dynamics. So DEUM consistently evolves schedules that sharply identify critical intervention points that can be understood in terms of the system dynamics. In order to achieve this by evolution, a substantial number of fitness evaluations (typically around 24 000) needed to be made. However, as we have seen in Chapter 3, it is possible to build a good model of fitness using a far smaller number of fitness evaluations (around 120).

[1]It is a feature of the mathematical model that the period of the sciarid larva growth cycle is not delayed or advanced by nematode predation.

6.4 Summary

This study has shown that there is a link between the fitness prediction capability of the Markov network fitness model and optimisation performance for the benchmark functions studied. This allows us to take the conclusions already reached for factors affecting the fitness modelling capability and relate them to optimisation. This makes the important step of demonstrating the wider applicability of the findings presented so far.

This chapter has also demonstrated how the DEUM fitness modelling approach can be extended to more complex problems than the bivariate and univariate problems previously investigated. This allows us a take a more general approach to problem-solving with DEUM. A Markov network can be built and sampled to produce an optimal individual and the experiments here demonstrate that this can be achieved with a small number of function evaluations relative to other algorithms. However, this is achieved at a high cost in terms of algorithm overhead, and requires a structure of variable interactions present within the problem to be supplied. In earlier experiments, DEUM gained a significant advantage over other algorithms through its use of fitness modelling. With increasing problem complexity the number of terms in the energy function increases which leads to an $O(n^3)$ increase in model build time. The time required by the sampler also increases considerably as problem complexity grows.

In addition this section has shown that the DEUM approach can be applied to a real-world problem (that is, not a synthetic benchmark problem). This problem does not have explicitly defined interactions between variables, yet an approximation using the chain model structure yields successful results. A useful aside from this work is the knowledge that while initialisation control can be helpful or at worst neutral for a genetic algorithm, it actually hinders the EDA due to biasing of the model. There is simply not enough information in the controlled population to construct a good model of fitness. In this case this is because a reduced number of interventions in the population make it difficult for the EDA to easily find critical regions for intervention. Solutions with large numbers of initial interventions, whilst of poor fitness, can still provide useful information to the modelling process about where interventions should occur.

The bio-control experiment also demonstrates the DEUM framework being applied in a more traditional evolutionary manner rather than the single step approach described in the previous section. This approach and combination of fitness modelling with other techniques are likely to be important in overcoming the obstacles presented by algorithm overhead and imperfect structures inferred from data. The results here demonstrate how this may be achieved, moving us closer to developing DEUM as a practical tool for levering the MFM for optimisation of real-world black-box problems.

Chapter 7

The Impact of Structure Learning on Fitness Modelling and Optimisation

The previous chapters have discussed a number of factors impacting upon the fitness model as well as means by which it may be employed for solving optimisation problems. The approaches described so far have all used a fixed structure for the Markov network which must be supplied to the algorithm. This has either been derived from the definition of the problem or has been a "best-guess" at a likely structure. We now move on to look at learning the structure from the population; a critical step towards a general purpose fitness modelling and optimisation algorithm which can be applied to black-box and real-world problems.

In this chapter we make use of a number of techniques for structure learning which were discussed in Chapter 2; the MIMIC-style entropy based chain builder, the linkage detection algorithm and a number of approaches based around the Chi-squared (χ^2) independence test. We begin by exploring concepts related to structure learning and measurement of structure quality; we then move on to look at factors affecting the performance of a structure learning algorithm using the χ^2 independence test. Then follows a series of experiments applying a number of variants of DEUM with structure learning to two benchmark problems. For each algorithm on each problem we examine both the fitness modelling capability and the optimisation capability. As the proposed algorithms incorporate structure learning - in contrast to previous versions of DEUM which each used a pre-supplied known structure of the problem - these algorithms may be applied to optimisation of problems with a wider range of structures. They are limited by only learning particular structure types, which means they will likely be suitable for optimisation of only problems with either a chain structure (for the two DEUM-Chain variants) or bivariate interactions (for DEUM-LDA and the latter two DEUM-χ^2 variants). Consequently, in Section 7.3 we describe DEUM-Chain and DEUM-Chain-χ^2 and apply them to the 1D Checkerboard problem which has a chain structure. In Section 7.4 we describe DEUM-LDA, DEUM-χ^2 and evDEUM-χ^2 and apply those algorithms to the 2D Ising problem.

In (Santana 2003a, Santana 2003b, Santana 2005) the variable interaction (independence) graph is learned from data using a statistical independence test. The structure is then refined to reduce its density and maximal cliques are found. Finally, a junction graph is learned in the case of MN-FDA (Santana 2003a, Santana 2003b) or a Kikuchi Approximation is learned by MN-EDA (Santana 2005) to approximate the distribution. In (Wright & Pulavarty 2005) an algorithm is proposed which uses the Linkage Detection Algorithm (Heckendorn & Wright 2004) to discover interactions when building a Boltzmann distribution of the fitness function. These approaches all result in an undirected structure which can also be used by the DEUM framework. The fitness modelling approach of DEUM allows us to make observations as to the quality of the structure learned and its effect on the fitness modelling capability of the resulting model which should be of interest to others using undirected graphical models and the wider EDA community.

7.1 How good is the structure?

Before moving on to discussion of the different approaches to structure learning, it is useful to consider how we might evaluate a given structure. Some work has been done on how to analyse structures learned in EDAs. (Mühlenbein & Höns 2005) included an analysis of EDA structure learning with a study of the structure learning capability

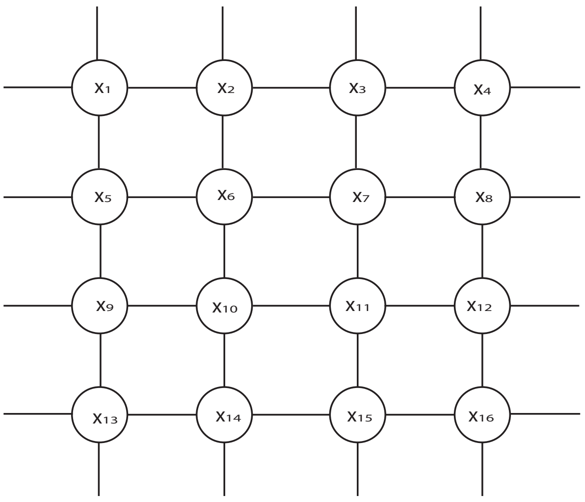


Figure 7.1: 16bit 2D Ising lattice

of Learning Factorized Distribution Algorithm (LFDA) and (Zhang 2004) discussed the importance of higher-order interactions in EDAs. It is known that not all interactions which are present in a problem will necessarily be required in the model for the algorithm to rank individuals by fitness and find a global optimum. This observation is related to the concept of unneccessary interactions (Hauschild et al. 2007). Also related is the idea of benign and malign interactions (Kallel et al. 2000), essentially that some interactions result in a deceptive influence on fitness. This is in addition to the idea of spurious correlations (Mühlenbein & Mahnig 2000, Santana et al. 2007) which are false relationships in the model resulting from selection.

Here we build on these concepts by comparing the structure learned by the algorithm to what we call the *perfect model structure*, which was defined in Chapter 3. Recall that the perfect structure includes exactly those interactions which are present in the underlying fitness function, which we will be referring to as *true* interactions. The perfect structure does not include all possible interactions but does include those which influence the absolute fitness value. In contrast, we also have *false* interactions, which are any interactions present in the model that are not present in the perfect structure. In the example Ising problem shown in Figure 7.1 the true interactions which make up perfect structure in addition to univariate and constant terms are:

x_1x_2, x_2x_3, x_3x_4, x_1x_4, x_5x_6, x_6x_7, x_7x_8, x_5x_8, x_9x_{10}, $x_{10}x_{11}$, $x_{11}x_{12}$, x_9x_{12}, $x_{13}x_{14}$,

$x_{14}x_{15}$, $x_{15}x_{16}$, $x_{13}x_{16}$, x_1x_5, x_2x_6, x_3x_7, x_4x_8, x_5x_9, x_6x_{10}, x_7x_{11}, x_8x_{12}, x_9x_{13}, $x_{10}x_{14}$, $x_{11}x_{15}$, $x_{12}x_{16}$, x_1x_{13}, x_2x_{14}, x_3x_{15}, x_4x_{16}.

This is more precisely defined by the MFM energy function, which we repeat here for reference. In general for an l x l 2D Ising problem this is:

$$U(x_i) = \alpha_0 + \sum_{i=1}^{l}\sum_{j=1}^{l}(\alpha_{ij}x_{ij} + \alpha_{ij,(i+1)(mod\ l)j}x_{ij}x_{(i+1)(mod\ l)j} + \alpha_{ij,i(j+1)(mod\ l)}x_{ij}x_{i(j+1)(mod\ l)}) \tag{7.1}$$

The interactions in the perfect structure are required to perfectly fit the model to the fitness function. Identification of the perfect structure is only practical for predefined test problems where the interactions are explicit. In the case of onemax there are no interactions. For the 2D Ising problem, an interaction exists wherever there is a coupling between two spin variables. In the example above, for a 16 bit 2D Ising problem the model will have 49 parameters in total including the univariate parameters and the constant.

In assessing the structures learned by an algorithm we use two measures from the information retrieval community: Precision (p) and Recall (r) (Witten & Frank 2005). These are defined in (7.2) and (7.3).

$$Precision = \frac{True\ interactions\ found}{Total\ interactions\ found} \tag{7.2}$$

$$Recall = \frac{True\ interactions\ found}{Total\ true\ interactions\ present} \tag{7.3}$$

That is, p measures how much of the learned structure comprises correctly identified interactions and r measures how many of the interactions present in the problem have been found. Both are proportions ranging from 0 to 1, with 1 being the best (if both measures are 1 then the learned structure perfectly matches the true structure). These can be combined into the single figure the F-measure (7.4) (Witten & Frank 2005). [1]

[1]This particular definition is known the F_1 measure in which p and r are equally weighted.

$$F = \frac{2 \cdot p \cdot r}{p + r} \tag{7.4}$$

As with p and r values, the F-measure ranges from 0 to 1, with $F = 1$ representing p and r of 1.

7.2 Chi-Square Structure Learning

We now move on to discuss structure learning using the Chi-squared (χ^2) independence test. The structure learner has a large number of parameters which can be tuned. In this section we look at the impact of a selection operator and a refinement algorithm, using the precision and recall measures discussed in Section 7.1 to measure the distance between the learned structures and the perfect structure for a number of benchmark problems.

7.2.1 Effect of selection

In Chapter 5 we saw that there is a clear influence of selection on fitness modelling when building the model. In that chapter and in Chapters 3 and 4 that the model structure strongly influences the fitness modelling capability. It is then logical to infer that as selection forms part of the χ^2 structure learning algorithm that here too it will influence the resulting fitness modelling capability.

The results for onemax show absolute numbers of false bivariates found (in this case as there are zero true interactions to find, precision would always be zero). Results for 2D Ising show the precision and recall for the bivariate interactions. These experiments followed the pattern of Algorithm 7.1. The threshold Γ for the χ^2 tests was set to 3.84, as noted in Section 2.2.3.3 this represents a 95% independence between variables. The population size was set to $100n$ where n is the number of variables in the problem (the problem size). This was the lowest size found that produced a precision and recall of over 0.95 with any selection operator and proportion on the 2D Ising problem. Section 7.2.2 shows results for additional poulation sizes.

As in Chapter 5, the experiments explored three operators: selecting the best $\varphi * M$

Algorithm 7.1 Experimental Procedure - Effect of Selection on Structure Learning

```
1: Generate random initial population $p$
2: Select a subset $\sigma$ of $p$
3: Use Chi-square statistics on $\sigma$ to determine interactions between variables; retain
   dependencies where $\chi^2 > 3.84$
4: Count number of true/false interactions found, and calculate $p$ and $r$
```

individuals (the top), selecting the worst $\varphi * M$ individuals (the bottom) and selecting the best $(\varphi/2) * M$ and worst $(\varphi/2) * M$ individuals (the top and bottom). Again, the proportion φ selected varied from 0.01 to 1.0 (equivalent to no selection).

When selecting the top and bottom of the population, the two groups had a separate set of independence tests run, which were then combined to produce the full structure, by simply taking the union of the two dependency sets. An example will explain the reasoning behind this approach.

Say a dependency exists between two variables X_1 and X_2. Assume that fitness increases if they are equal; thus in the "top" selected set, they will generally be equal and in the "bottom" selected set they will be opposite in value. If the two sets were added together prior to running the independence tests, the resulting set would see approximately half the individuals with $X_1 = X_2$ and half with $X_1 \neq X_2$. This is no better than a completely random set and no interaction will be detected. However, if the two sets are kept separate, it will be determined that in highly fit individuals they are equal and in low fitness individuals they are different. Thus the interaction will be detected and incorporated into the model.

7.2.1.1 Experimental Results

In this series of experiments there is no perfect / imperfect model structure distinction in the results because the structure is learned and is unlikely to perfectly match the underlying structure of the problem. Each graph shows the FPC values as in the previous section. Here, each graph also shows the number of interactions found, split in to true and false interactions. The graphs are scaled so the number of bivariate interactions on the full model is at the top of the right vertical axis; thus we can see the number of interactions found as a proportion of the total number present. In keeping with the practice of earlier chapters, a sample of the results are given here, with the full series of figures presented in Appendix D.

In Figures 7.2 and D.1 to D.9 we see the results from a series of fully specified onemax problems; again we look at a number of problem sizes. In the case of onemax there are no interactions so the perfect model has only univariate terms. As was the case with the experiments in Chapters 4 and 5 with the smallest instances of the problem (10 and 20 bits in this case) the results vary considerably (note the standard deviations) and are inconclusive. This is likely to be because the small problem size results in a correspondingly small population and the selection operator causes a high lack of diversity in the population so the statistical tests produce greatly varying results from run to run.

For the larger instances of the problem a trend starts to emerge. We can see that selection does indeed influence the number of interactions being found; top+bottom selection finds twice as many false interactions at the other operators. This is likely to be because it adds together dependencies detected at each end of the population. One factor which is important is that with more interactions present on the model the time required to compute the model paremeters will be larger.

In Figures 7.3 and D.10 to D.18 we see the results from a series of fully specified Ising problems. Here we have a definite set of bivariate interactions to be found. Selection has a large effect on the precision and recall for the structure learned - again the trend is much clearer for larger instances of the problem (25 bits and larger). With a high selective pressure the structure has both high precision and recall. As selective pressure decreases (selecting a higher proportion of the population) recall falls off rapidly, falling below 0.1 once 20% of the population is selected with the top selection and bottom selection operators. This means that 10% of the true structure is being identified at this point. Precision falls more slowly as selective pressure is decreased - remaining over 0.8 until over 40% of the population is being selected, with any of the selection operators. This means that 80% of the interactions detected are true interactions. An interesting result here is that the three selection operators detect structure with similar precision and recall; this reinforces the findings presented in Chapter 5 that useful information is contained

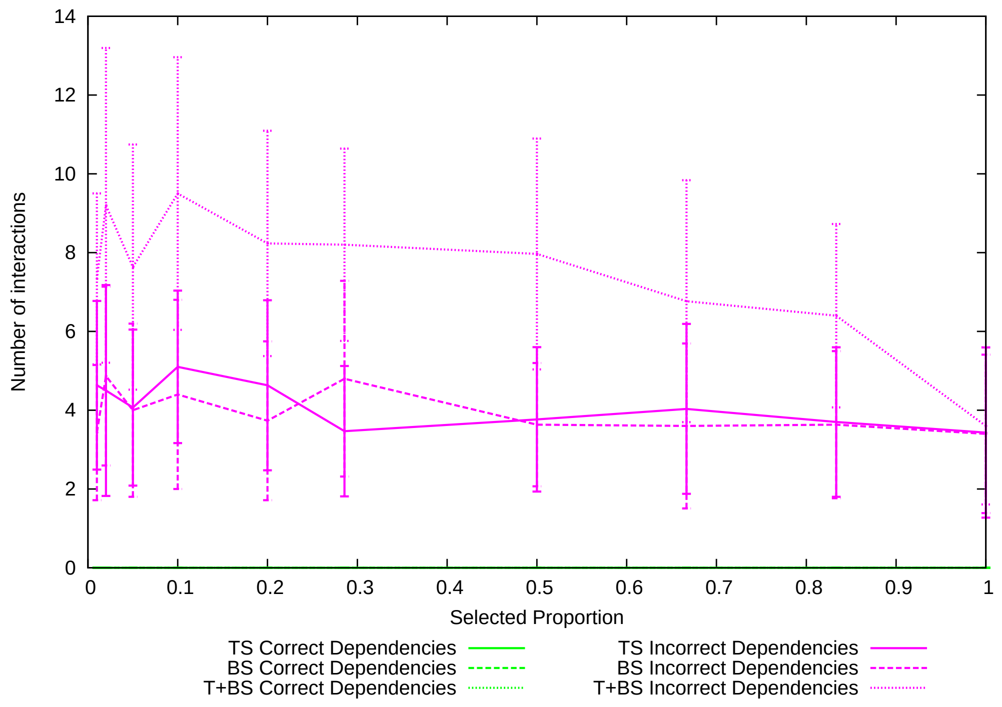


Figure 7.2: Number of interactions found against selection proportion for 100 bit fully specified OneMax

throughout the population. On this figure we also show the results of running a Gibbs sampler using the same approach and parameters as those in (Shakya et al. 2006), with the exception that instead of supplying the lattice structure to the algorithm, the structure discovered by the indepence tests is used. The success rate starts at 100% but falls off very quickly, for more detail on this figure we show the maximum fitness found by the algorithm. We can see that the optimisation capability falls off following the same trend as the precision.

We can see that selection operator and selection pressure both have an impact upon the structure learned. We know from Chapters 4 and 5 that this in turn has an influence on the fitness modelling capability and this is why we observe an effect on the optimisation capability.

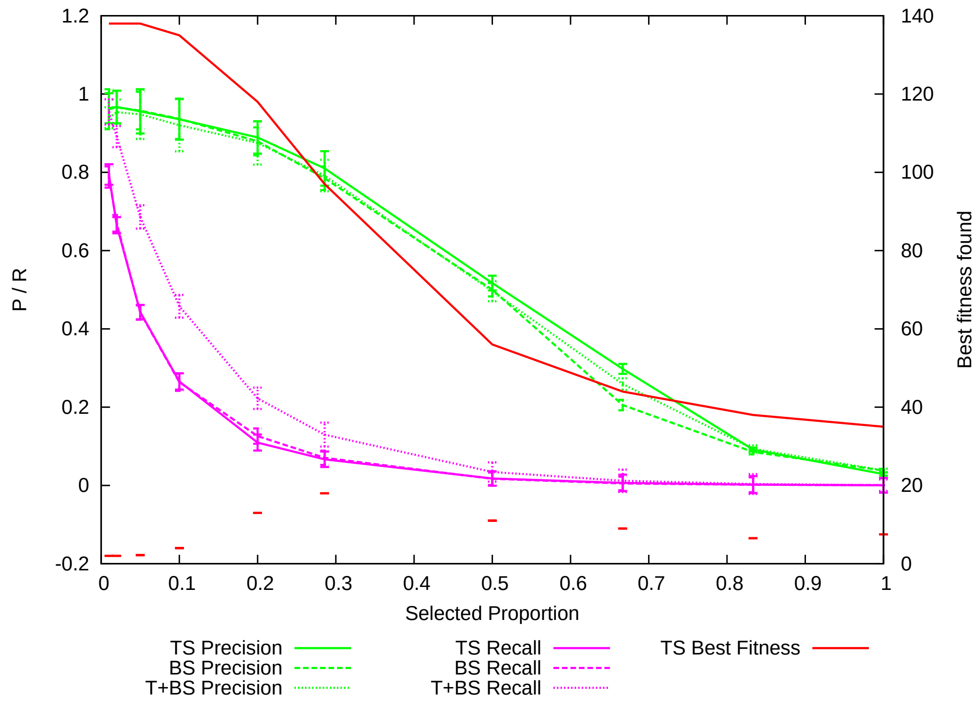


Figure 7.3: P and R against selection proportion for fully specified 100 bit 2D Ising

7.2.2 Effect of Threshold and Refinement Algorithm

A further important parameter to vary is the threshold Γ at which the algorithm deems there to be a statistically significant relationship between two variables. In the previous section this was set to 3.84; in (Santana 2003a) it was set to 1.0 to capture a larger number of dependencies. In this section we will investigate setting Γ to different levels.

One consequence of lowering Γ is that a larger number of dependencies will be present in the resulting structure. This makes the parameter estimation stage of the algorithm more expensive. In (Santana 2003a), this problem was mitigated by introducing a refinement algorithm which finds nodes in the graph with a large number of incident edges and removes the edges with low χ^2 values until a preset number is reached. Figures 7.4 and D.19 to D.24 show the results of finding model structure with and without refinement, with a number of different threshold levels across several different selective pressures.

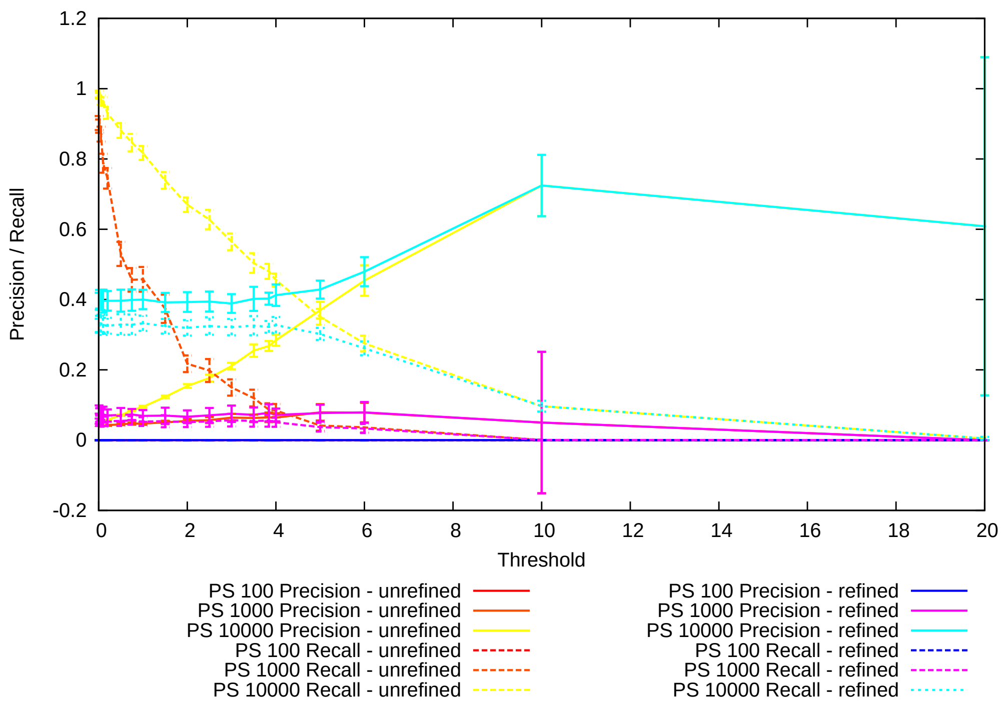
Each seperate graph shows a set of results using a different selection pressure. Within each graph, we have the precision and recall values for three different population sizes (PS) and structures before and after refinement. Given that the 2D Ising model has four edges incident to each node, refinement limited the learned structure to the same. The results are the means from 30 independent runs each using an instance of 2D Ising chosen at random from a set of four.

A number of conclusions may be drawn from these results. Firstly, it is clear that population size exerts a strong influence on both precision and recall, with the large population generally resulting in the highest values for both. Secondly, the lower selective pressures appear to produce better results at all population sizes.

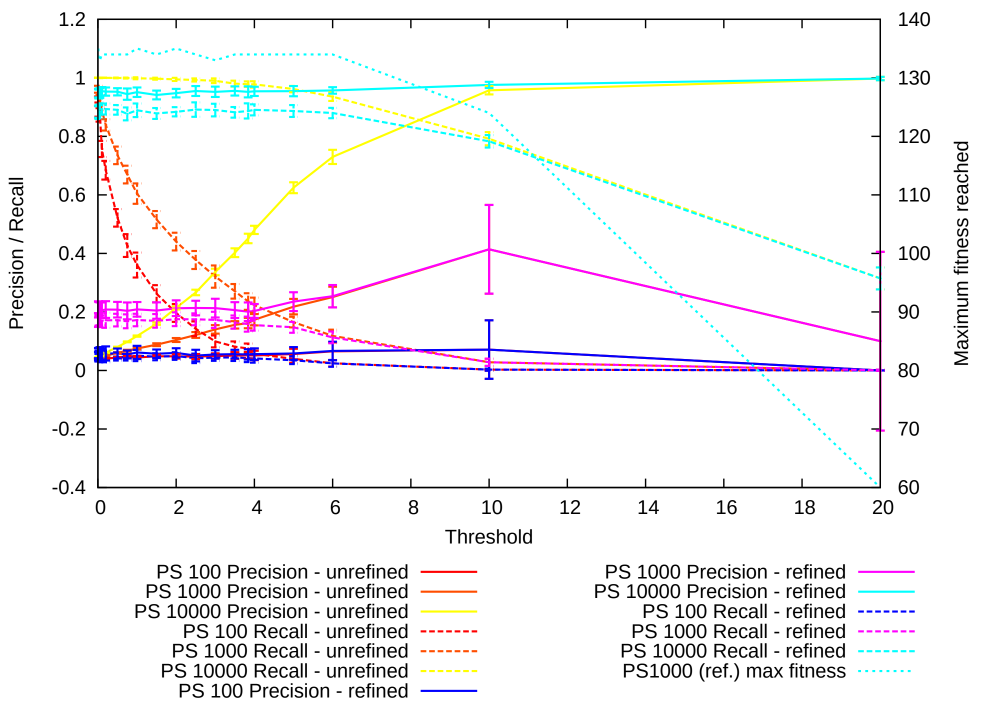
Increasing the threshold results in an increase in precision and a decrease in recall. This is because with a higher threshold, fewer interactions are present in the learned structure - resulting in fewer erroneous interactions being present but at the expense of losing some valid interactions. To capture as many of the true interactions as possible, it would appear then to be necessary to have a low threshold (an observation also made in (Santana 2003a)). This has the negative consequence of having a large number of incorrect interactions in the model; however this effect is considerably mitigated by the refinement stage. Particularly when selecting over 5% of the population (Figures D.21 to D.24) we can see that precision for an unrefined structure starts near zero and increase to near one as threshold increase, but precision for the refined structure remains high for all threshold values. Refinement does have the negative effect of reducing the recall but this difference is slight.

Again on this figure (Figure 7.4b) we show the maximum fitness found by running the optimisation algorithm using the structure learned. In this case, for simplicity we only show the optimisation for a population size of 10000 with selection 50% as this gave the best precision and recall values overall. We can see that as recall falls off (while precision remains high) the optimisation capability also falls.

This presents us with a problem: we can either have a dense structure which has a maximal number of true interactions present but which will have a large number of parameters (Chapter 4 in particular looks at the problems with this in more detail), or we can have a much simpler structure which is missing some interactions. This is likely to be highly dependent on the problem. In (Santana 2003a) a preference for having a larger

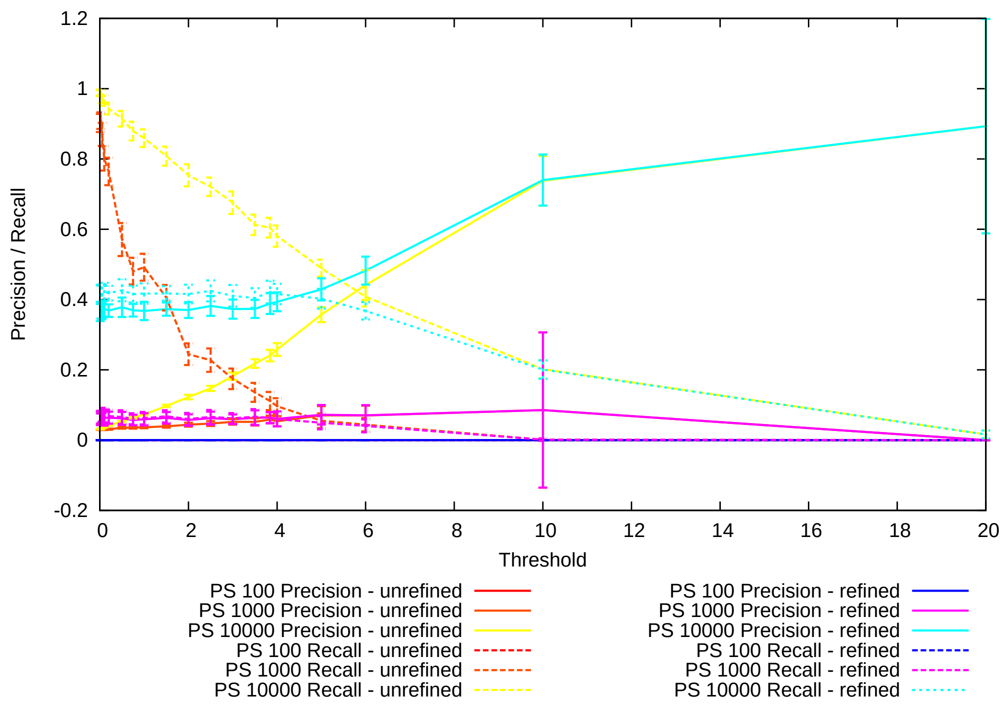


(a) Select top 1%

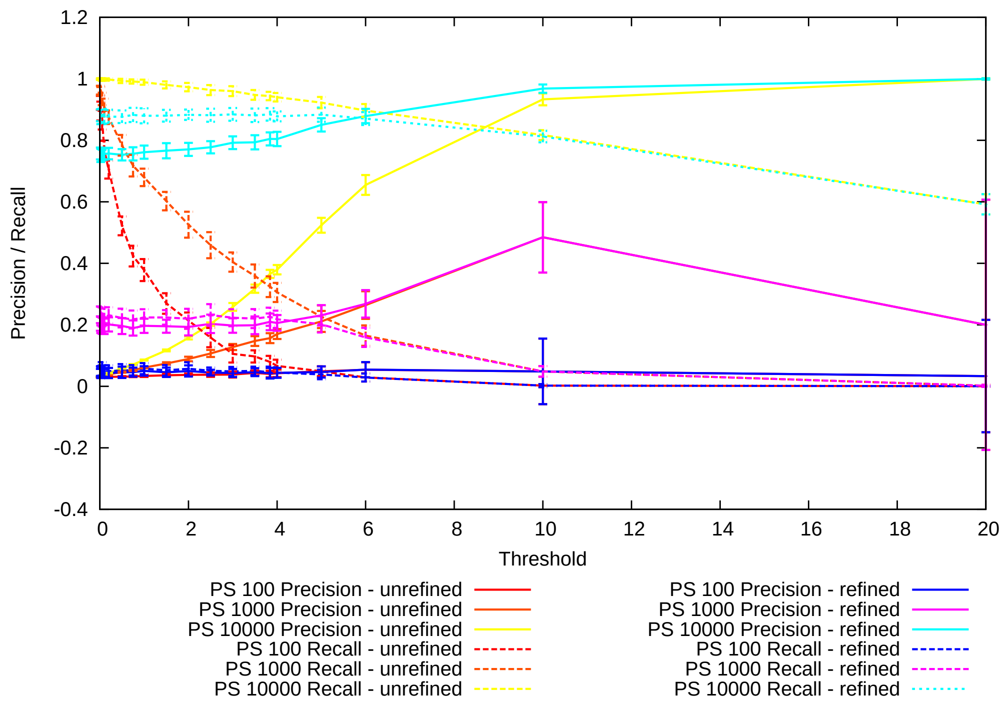


(b) Select top 50%

Figure 7.4: Precision and recall of interactions learned for 100 bit Ising problem

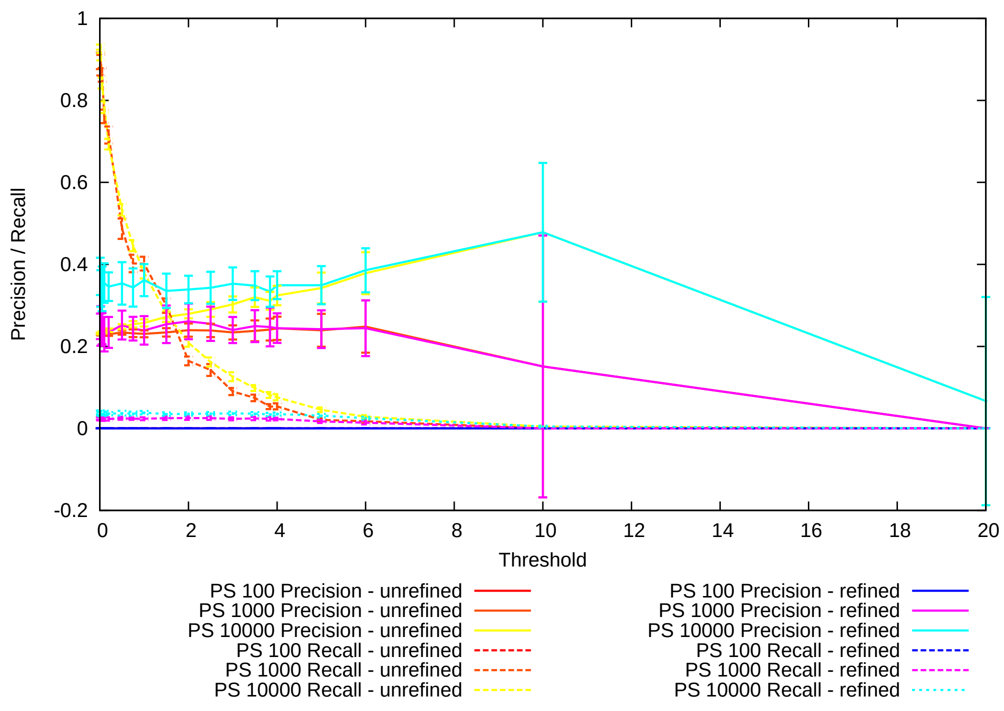


(a) Select top 1%

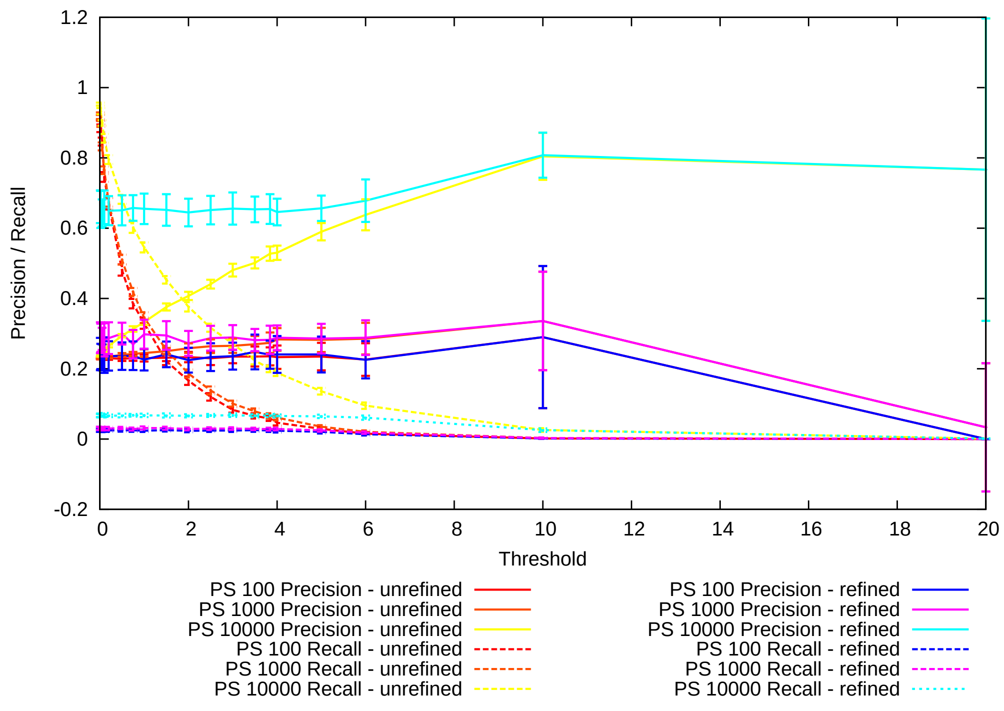


(b) Select top 50%

Figure 7.5: Precision and recall of interactions learned for 100 bit Checkerboard problem



(a) Select top 1%



(b) Select top 50%

Figure 7.6: Precision and recall of interactions learned for 100 bit MaxSAT problem

number of interactions was stated, and based on the poor fitness modelling capability for imperfect structures this would be the preference for the MFM approach.

We can see results from a repeat of this experiment for 2D Checkerboard and 3-CNF MAXSAT shown in Figures 7.5, 7.6 and D.25 to D.36. Results for 2D Checkerboard are very close to those seen for 2D Ising, which is unsurprising as both have the same lattice structure. For 3-CNF MAXSAT, the algorithm was still finding biviariate interactions. The problem itself has trivariate interactions (groups of three variables) - for calculating precision and recall in this experiment the "true" set of bivariate interactions was determined by assuming that each pair of variables which shared a clique in the trivariate structure also share a bivariate interaction. The results again show similar trends to those described for 2D Ising and Checkerboard.

7.2.3 Higher order interactions

For problems with higher order interactions than bivariate, some more sophisticated structure learning will be required. Either higher order dependency tests or LDA with higher clique size can be run, or a clique finding algorithm such as Bron-Kerbosch (Bron & Kerbosch 1973, Shakya et al. 2009) could be run on the bivariate structure found by the lower order dependency tests.

7.3 Optimisation of 1D Checkboard

As in the previous chapter we start by looking at the simple chain based problem 1D checkerboard. We try two approaches to structure learning for this problem: the MIMIC style chain learning algorithm and the Chi-square independence test structure builder.

7.3.1 DEUM-Chain

7.3.1.1 The Algorithm

This algorithm is based on the information entropy (Shannon 1948) approach taken by MIMIC (de Bonet et al. 1997), with a slight modification to the selection operator. To estimate the structure, the top 50% of individuals are selected rather than all individuals fitter than the median fitness. In addition to this the model paramaters are calculated using a differently selected set of individuals - the top $2N$ of the population - rather than the exact same set as those used to estimate structure. $2N$ was chosen because it was shown to give good performance with the fixed structure algorithm presented in Chapter 6. Elitism was also used differently to MIMIC, in this case preserving the best 90% of individuals from one generation to the next - essentially a steady-state approach to prevent premature convergence (in the workflow below, $R = 0.9$). The permutation as used by MIMIC is interpreted as a chain of bivariate dependencies which become the structure of the Markov network. This gives us the workflow in Algorithm 7.2.

Algorithm 7.2 DEUM-Chain

```
1: Generate random initial population $p$
2: while Stopping criteria not met do
3:    Select a subset $\sigma_1$, the top 50% of $\pi$
4:    Run the MIMIC chain builder:
5:    Choose the variable with the lowest information entropy within $\sigma_1$ and denote it $X_i$
6:    repeat
7:       Choose the variable with the lowest conditional entropy given $X_i$ not in the chain
         and denote it $X_j$
8:       Add interaction $K_{ij}$ to the structure of the Markov network
9:       Relabel $X_j$ as $X_i$
10:   until All variables have been added to chain
11:   Refine structure
12:   Select a subset $\sigma_2$, the top 2N of $p$
13:   Use $\sigma_2$ to compute MFM parameters
14:   Calculate $C_r$
15:   Sample $m$ new individuals from MFM using random walk Gibbs sampler and replace
      poorest $m$ individuals in the old population with these
16: end while
```

Calculation of C_r is not part of the algorithm as such but is shown here to indicate where the operation takes place. For the experiments, population size was set to $8(2n-1)$ where n is the number of variables. This figure was chosen to ensure that selecting the top 25% of the population would select $2N$ individuals (we know that N will be $2n-1$ because the chain structure will have a fixed number of terms). With this algorithm there is an assumption that in early generations the model will not be a close fit to the fitness function. This means that areas of high probability within the model will not necessarily be areas of high fitness, and running the Gibbs sampler slowly to convergence is likely to result in an individual of inferior fitness to the global optimum. The algorithm was initially run with a fixed value for cooling rate and maximum number of iterations. It was found that performance was improved by varying these parameters over the course of the evolution - reducing the cooling rate and increasing the maximum number of iterations with each generation. This allowed the algorithm to be balanced towards exploration in early generations and exploitation in later ones as the model fits more closely to the fitness function. For each generation g, the cooling rate r was calculated according to (7.5) and the maximum number of iterations of the Gibbs sampler I was calculated according to (7.6); the parameters for these were determined empirically. The algorithm was stopped at 100 generations if the optimum had not been found.

$$r = 10 + 5g \tag{7.5}$$

$$I = 0.01/(1 + g) \tag{7.6}$$

7.3.1.2 Fitness Model

The evolutionary approach means that multiple models are being created during each run, with corresponding multiple figures for p, r, F and C_r. For ease of interpretation, the values over the course of evolution for three instances of the problem (50 bit, 100 bit and 200 bit) are represented graphically in Figures 7.7, 7.8 and 7.9. The four measures can be shown relative to the same vertical axis as they all have a range of zero to one (Strictly speaking C_r has a range of -1 to +1 but in the examples here it is always positive). The smaller problem sizes are not shown because the algorithm often stopped in a single generation so the chance to observe an effect over many generations was reduced. This is because the reduced problem size makes it probable that an optimal individual will be in the initial population.

Precision and recall values (and hence F values) are the same because the chain structure fixes the number of possible interaction in the model. For each true interaction omitted from the model, a false one will be in its place. These never rise above 0.1 (10% of the chain correctly identified). This in turn means that the model's correlations with

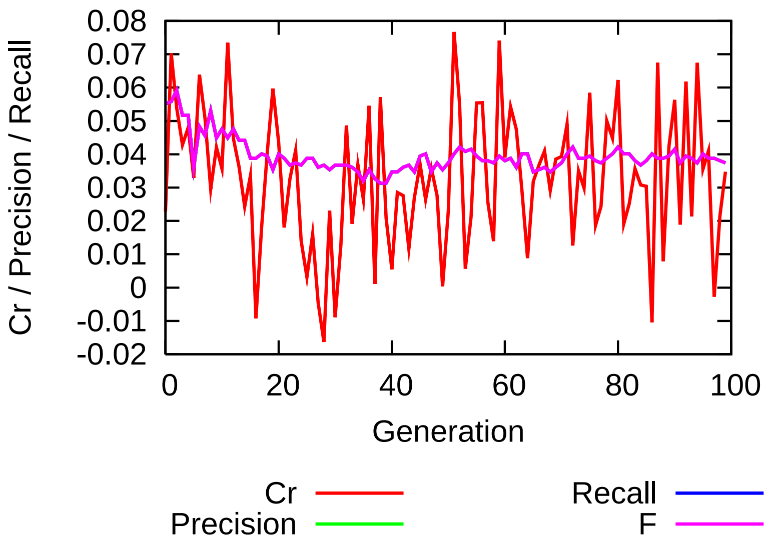


Figure 7.7: Fitness Model Statistics for DEUM-Chain on 50 bit 1D checkerboard over 30 runs

the fitness function C_r also remains low, also never rising above 0.1. It must be highlighted that poor structure learning capability is not because the chain builder was designed for building a directed model rather than the undirected one used by the MFM. The entropy based chain builder simply generates an ordering of the variables and the precision and recall measures indicate that the chains (orderings) learned bear no resemblance to the correct chain (ordering). It may be possible to improve on this by increasing the size of the population.

7.3.1.3 Optimisation Results

Now we run the full optimisation algorithm incorporating the MIMIC chain learner and report the results in Table 7.1. The mean number of function evaluations (FE) and internal iterations of the Gibbs sampler (IT) are given along with their standard deviations (FE-SD and IT-SD). All of these values only include the successful runs; the success rate (SR) is given as a percentage over 30 independent runs.

The results show that the algorithm was unable to finmd the global solution to this problem, even in the 10 bit instance. The only runs which registered as successfully completed found the optimum by chance in the initial starting population, which is why

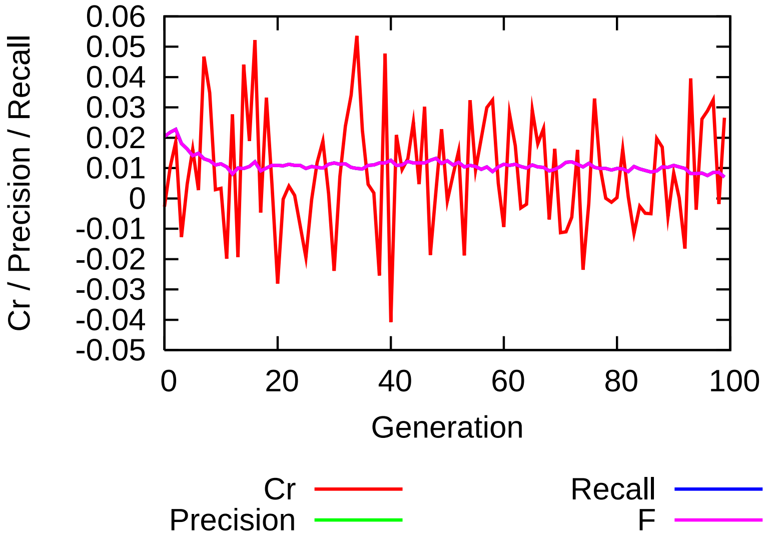


Figure 7.8: Fitness Model Statistics for DEUM-Chain on 100 bit 1D checkerboard over 30 runs

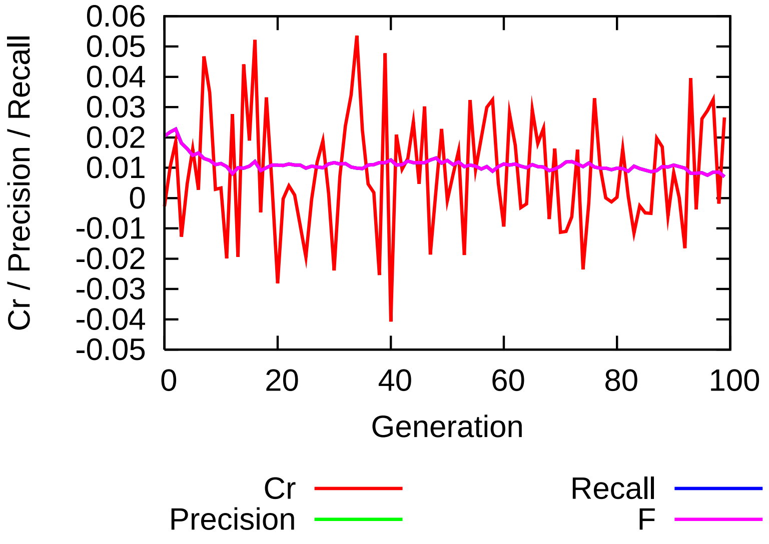


Figure 7.9: Fitness Model Statistics for DEUM-Chain on 200 bit 1D checkerboard over 30 runs

there is no variation in the number of function evaluations and no Gibbs sampler iteration needed to find the optimum in the 10 bit case. This is a reflection of the poor model seen in the previous section. The MFM has little correlation with the fitness function in any generation during the run, so sampling it does not yield a globally optimal individual at any point.

PS	FE	FE-SD	IT	IT-SD	SR
10	152	0	0	0	10
20	-	-	-	-	0
50	-	-	-	-	0
75	-	-	-	-	0
100	-	-	-	-	0
200	-	-	-	-	0

Table 7.1: DEUM-Chain Optimisation Statistics over 30 runs

7.3.2 DEUM-Chain-χ^2

7.3.2.1 The Algorithm

We now move on to a single-step algorithm similar to that used for optimisation of 1D Checkerboard in Chapter 6. A single step version of the algorithm incorporating the MIMIC chain builder was attempted but found to be unable to find a useful chain in a single generation. Instead, Algorithm 7.3 uses the χ^2 independence test to build the chain; a variable is chosen at random to act as one end of the chain and then the chain is formed by following a path through the strongest dependencies.

Algorithm 7.3 DEUM-Chain-χ^2

```
1: Generate random initial population $p$
2: Select a subset $\sigma_1$, the top 25% of $\pi$
3: Run the $\chi^2$ chain builder:
4: Choose a variable at random and denote it $i$
5: repeat
6:    Choose the variable with the highest $\chi^2$ value given $i$ not already in the chain and
      denote it $j$
7:    Add interaction $ij$ to the structure of the Markov network
8:    Relabel $j$ as $i$
9: until All variables have been added to chain
10: Refine structure
11: Select a subset $\sigma_2$, the top $2N$ of $p$
12: Use $\sigma_2$ to compute MFM parameters
13: Calculate $C_r$
14: Sample new population from MFM using random walk Gibbs sampler
```

Earlier in this chapter was saw that a large population size was required to find a useful structure with the χ^2 algorithm. The population size in this case was set to 100 multiplied by the number of variables, which although large is reasonable given that there is only one

PS	SFE	P	P-SD	R	R-SD	F	FE	FE-SD	C_r	C_r-SD
10	1000	0.874	0.038	0.874	0.038	0.874	157	8	0.841	0.102
20	2000	0.951	0.013	0.951	0.013	0.951	316	3	0.940	0.015
50	5000	0.980	0.004	0.980	0.004	0.980	801	9	0.975	0.005
100	10000	0.990	0.000	0.990	0.000	0.990	1624	26	0.987	0.003
200	20000	0.994	0.004	0.994	0.004	0.994	3470	284	0.991	0.005

Table 7.2: DEUM-Chain-χ^2 Fitness Model Statistics over 30 runs

generation. There is no need to set a threshold for the χ^2 tests in this algorithm because we do not keep all interactions above a certain level; instead we keep those interactions with the highest score relative to others incident to the same node. The Gibbs sampler used was the same as that used for the fixed structure in Chapter 6, with a cooling rate parameter of 0.01 and an iteration cap of 2000.

7.3.2.2 Fitness Model

In this case only one model is built for each run of the algorithm so we can present the mean and standard deviation for p, r and F alongside those for C_r.

As was the case with DEUM-Chain, precision and recall values (and consequently F values) are the same because of the chain structure. We can see that the precision and recall values are around 0.9 rather than near zero as was the case with the MIMIC chain builder. This results in models with a strong C_r values, all over 0.8. (recall that values over 0.7 represent a strong positive correlation)

7.3.2.3 Optimisation Results

Table 7.3 shows the results for optimising using the algorithm at each problem size (PS). The C_r value is given to show the fitness prediction power of the model in the context of optimisation. The mean number of function evaluations (SFE) required by the structure learning component of the algorithm is given. Then the mean number of additional function evaluations (FE) and internal iterations of the Gibbs sampler (IT) are given along with their standard deviations (FE-SD and IT-SD). FE includes the evaluations needed to estimate model parameters and the evaluations needed to confirm the fitness of individuals generated by the Gobbs sampler. All of these value only include the successful runs; the

PS	SFE	FE	FE-SD	IT	IT-SD	SR	C_r	C_r-SD
10	1000	157	8	472	630	100	0.841	0.102
20	2000	316	3	605	423	100	0.940	0.015
50	5000	801	9	1710	1632	100	0.975	0.005
100	10000	1624	26	7501	6107	100	0.987	0.003
200	20000	3470	284	91023	93248	100	0.991	0.005

Table 7.3: DEUM-Chain-χ^2 Optimisation Statistics over 30 runs

success rate (SR) is given as a percentage of times the algorithm found a known global optimum over 30 independent runs.

7.3.2.4 Analysis

In contrast to the poor results for DEUM-Chain, the algorithm is able to successfully find the global optimum in all instances. This can be attributed to the increased fitness modelling capability of the MFM. In turn this can be attributed to the greater success of the χ^2 chain learning algorithm over the entropy-based MIMIC chain builder at finding a structure which closely matches the perfect structure.

7.4 Optimisation of 2D Ising

The 2D Ising problem has a two dimensional lattice structure similar to the checkerboard problem featured in Chapter 6. This has previously been used as a benchmark problem for EDAs (Pelikan & Goldberg 2003, Pelikan 2002, Pelikan, Ocenasek, Trebst, Troyer & Alet 2004, Santana 2003a, Santana 2003b, Santana 2005, Shakya et al. 2006) due to interesting properties such as symmetry and a large number of plateaux. The problem exhibits an undirected network of interactions between variables and consequently EDAs using Markov networks are naturally suited to it and should perform well. Given its lattice structure the chain learning approach used in the preceeding section is unlikely to achieve a satisfactory match to the problem structure in this case. (Shakya et al. 2006) reported the application of a Markov network approach to optimisation of the 2D Ising problem. That paper described an algorithm incorporating several different sampling techniques but using the same underlying model-building algorithm which used a fixed-structure that was inferred from the definition of the problem (a 2D lattice). This set of experiments explores different ways that a structure learning step may be added to the existing algorithm.

Here we investigate the impact of three structure learning approaches on the fitness modelling and optimisation capability of the underlying algorithm.

7.4.1 DEUM-LDA

This approach adds the Linkage Detection Algorithm (Heckendorn & Wright 2004) to the DEUM framework, giving us the workflow in Algorithm 7.4.

Algorithm 7.4 DEUM-LDA

```
1: Run LDA on the fitness function
2: Generate random initial population $p$, of size $4.4N$
3: Select a subset $\sigma$, the top $1.1N$ of $p$
4: Use $\sigma$ to build MFM
5: Calculate $C_r$
6: Sample new population from MFM using random walk Gibbs sampler
```

Step 5 is not an integral part of the algorithm. It is only included in the above workflow to show the point at which we calculate the value for C_r.

The proportion of the population selected and population size used were determined by earlier experiments described in (Brownlee, McCall, Zhang & Brown 2008). There we found that fitness modelling capability of the MFM increases greatly when the number of individuals selected is more than the number of parameters in the model (N). Once the structure is known, we then set the number to be selected to 1.1N to ensure that this is the case (we call this an over-specified system). The population size is then set to be large enough to achieve the desired selection pressure. The results in (Brownlee, McCall, Zhang & Brown 2008) revealed that fitness modelling capability improves as the selection pressure is increased. In this case we set the proportion of the population selected to 0.25 - that is, the population is four times the number of individuals required to build the MFM. This represents a good tradeoff between a useful selective pressure and an excessively large population.

The cooling rate parameter for the Gibbs sampler was 0.0005, taken from (Shakya et al. 2006). The iteration cap of the sampler was increased from 500 to 2000 - this

PS	P	P-SD	R	R-SD	F	C_r	C_r-SD
16	1.00	0.00	1.00	0.00	1.00	0.887	0.091
25	1.00	0.00	1.00	0.00	1.00	0.924	0.049
36	1.00	0.00	1.00	0.00	1.00	0.932	0.034
49	1.00	0.00	1.00	0.00	1.00	0.939	0.037
64	1.00	0.00	1.00	0.00	1.00	0.949	0.023
100	1.00	0.00	1.00	0.00	1.00	0.942	0.027
256	1.00	0.00	1.00	0.00	1.00	0.945	0.018
324	1.00	0.00	1.00	0.00	1.00	0.943	0.015
400	1.00	0.00	1.00	0.00	1.00	0.940	0.022

Table 7.4: DEUM-LDA Fitness Model Statistics over 30 runs

improved the success rate of the algorithm and decreased the total number of iterations required to find the global optimum.

7.4.1.1 Fitness Model

First we will look at the model generated by the algorithm for each problem size. In Table 7.4 we see the mean precision p and recall r for the structures found over 30 runs, with corresponding standard deviations (p-SD and r-SD). This is followed by the F-measure F combining the mean precision and recall. This is shown alongside the mean fitness prediction correlation C_r for each size of the problem with its corresponding standard deviation C_r-SD.

With a precision and recall (and hence F) of 1.0, the structure discovered matches the perfect structure for the problem. This means we would expect similar optimisation results to those seen in (Shakya et al. 2006).

7.4.1.2 Optimisation

Table 7.5 shows the results for optimising using the algorithm. The C_r value is given to show the fitness prediction power of the model in the context of optimisation. The number of fitness evaluations needed by LDA (FE-LDA) for each problem size is given next. Then the mean number of additional function evaluations (FE) and internal iterations of the Gibbs sampler (IT) are given along with their standard deviations (FE-SD and IT-SD). FE includes the evaluations needed to estimate model paramters and the evaluations needed

PS	C_r	LDA-FE	FE	FE-SD	IT	IT-SD	SR
16	0.887	480	210	4	163	310	100
25	0.920	1200	330	1	989	532	100
36	0.932	2520	483	25	821	1995	100
49	0.905	4704	647	3	1675	1234	100
64	0.939	8064	846	1	902	688	100
100	0.944	19800	1446	221	11989	21296	87
256	0.950	130560	4342	862	130995	118419	67
324	-	-	-	-	-	-	0
400	-	-	-	-	-	-	0

Table 7.5: DEUM-LDA Optimisation Statistics over 30 runs

PS	C_r	FE-LDA	FE	FE-SD	IT	IT-SD	SR
100	0.993	19800	2409	7	8015	6825	100
256	0.993	130560	6166	48	37705	82985	100
324	0.993	209304	7786	7	18947	11962	80
400	0.992	319200	9688	86	160237	154439	100

Table 7.6: DEUM-LDA Optimisation Statistics with increased population size over 30 runs

to confirm the fitness of individuals generated by the Gibbs sampler. All of these value only include the successful runs; the success rate (SR) is given as a percentage of times the algorithm found a known global optimum over 30 independent runs.

The results were unexpectedly poor given the perfect structure learned by LDA. We increased the number of individuals selected from the population to estimate the model parameters to $2N$ (that is, twice the number of parameters in the model). Part of our work on the fitness information content in a population (Brownlee, McCall, Zhang & Brown 2008) indicated that Ising requires a larger number of individuals than other fitness functions we have looked at to obtain a good model of fitness. The results for the rerun are given in Table 7.6. It can be seen that this resulted in a marked improvement of the optimisation capability, even for the largest instances of the problem that we used.

One problem with this approach is that the linkage detection algorithm requires a large number of fitness evaluations to learn the structure. This could be mitigated by recycling the solutions generated by the LDA run for estimating the model parameters but this would not make a large difference. The number of possible interactions for a particular problem size n is given in (7.7); given that LDA requires four evaluations for each possible interaction, the number of evaluations required by LDA at a particular problem size n is given in (7.8). This can be reduced by caching individuals but not by a large amount and this comes with a rapidly growing space complexity.

$$Possible = n \times (n-1)/2 \tag{7.7}$$

$$Evals = 2 \times (n \times (n-1)) \tag{7.8}$$

In addition to this, without some threshold it will include any interactions which are not useful for optimisation and this will lead to a large and overly complex model which will result in the algorithm being overloaded and potentially resulting in poor performance. An effect similar to this was seen with the noisy chemotherapy problem used for benchmarking hBOA in (Brownlee, Pelikan, McCall & Petrovski 2008). The 2D Ising problem is in comparison very "clean" - because of the nature of the problem any interactions which LDA discovers are important for optimisation with the result that the structure found is perfect. This allows DEUM to build a model which closely matches the fitness function. For other problems this may not be possible which provides motivation for the independence test approach.

7.4.2 DEUM-χ^2

7.4.2.1 The Algorithm

The workflow for this algorithm is very similar to that for DEUM incorporating LDA and is given in Algorithm 7.5. The only additions are the extra selection step to choose individuals for the Chi-Square structure learning algorithm and the structure refinement step. The number of individuals for the structure learning selection was set to the top 25%; the selection step for estimating the MFM parameters selected the top 1.1N individuals. The structure refinement step is the same as that used in (Santana 2003a, Santana 2003b, Santana 2005): a limit is imposed on the number of edges (interactions) incident to a node (variable) on the graph. For any node exceeding this limit, the edges with the lowest Chi-Square scores are removed until the limit is reached. For 2D Ising we set this limit to

PS	p	p-SD	r	r-SD	F	C_r	C_r-SD
16	0.785	0.183	0.979	0.028	0.988	0.997	0.01
25	0.749	0.213	0.968	0.029	0.98	0.992	0.015
36	0.73	0.166	0.97	0.022	0.982	0.994	0.008
49	0.754	0.101	0.963	0.019	0.976	0.99	0.012
64	0.741	0.107	0.958	0.018	0.974	0.99	0.008
100	0.695	0.116	0.948	0.018	0.966	0.985	0.01
256	0.589	0.092	0.914	0.009	0.94	0.968	0.007
324	0.584	0.083	0.908	0.011	0.935	0.964	0.008
400	0.529	0.085	0.894	0.011	0.924	0.956	0.007

Table 7.7: DEUM-χ^2 Fitness Model Statistics over 30 runs

4 - that is, each variable can have only four neighbours. The clique finding step from those papers is not applied here as we know that the 2DIsing problem has a bivariate structure and we wish to keep the structure at this level of complexity. The population size was set to be the problem size multiplied by 100. As in the previous section, the cooling rate for the Gibbs sampler was 0.0005 and the iteration cap was set to 2000.

Algorithm 7.5 DEUM-χ^2

```
1: Generate random initial population $p$
2: Select a subset $\sigma_1$, the top 25% of $p$
3: Run Chi-Square edge detection algorithm to search for statistical dependencies appar-
   ent in $\sigma_1$
4: Refine structure
5: Select a subset $\sigma_2$, the top 1.1N of $p$
6: Use $\sigma_2$ to build MFM
7: Calculate $C_r$
8: Sample new population from MFM using random walk Gibbs sampler
```

7.4.2.2 Fitness Model

Again, before looking at optimisation results we will look at the model generated by the algorithm for each problem size. In Table 7.7 we see the mean p and r for the structures found over 30 runs, with corresponding standard deviations (p-SD and r-SD). The next column is the F-measure combining these figures. This is shown alongside the mean fitness prediction correlation C_r for each size of the problem with its corresponding standard deviation C_r-SD.

We can see that increasing problem size results in a decrease in both p and r, reflected

PS	p	p-SD	r	r-SD	F	C_r	C_r-SD
16	0.8	0.037	0.703	0.052	0.854	0.573	0.163
25	0.896	0.051	0.816	0.053	0.921	0.733	0.095
36	0.954	0.02	0.89	0.027	0.976	0.869	0.05
49	0.992	0.011	0.961	0.023	0.989	0.941	0.047
64	0.995	0.007	0.984	0.009	1	0.974	0.014
100	1	0	1	0.002	1	0.992	0.004
256	1	0	1	0	1	0.993	0.002
324	1	0	1	0	1	0.993	0.002
400	1	0	1	0	1	0.993	0.001

Table 7.8: DEUM-χ^2 Fitness Model Statistics with LDA-equivalent population size over 30 runs

in a steadily decreasing F. This indicates that with increasing problem size the algorithm finds it more difficult to correctly identify all interactions and also begins to match some false positives. This also results in a corresponding decrease in the fitness prediction power of the model, revealed in the decreasing C_r values. We can see that these fall off very quickly with a comparatively small decrease in r; this highlights the importance of finding a good model structure. The experiment was rerun with population sizes equal to the number of fitness evaluations required by LDA at each step; the results for this are given in Table 7.8. We can see that the structures are considerably better at the larger sizes; for the smaller sizes the number of evaluations required by LDA was actually less than 100 times problem size as used in the previous experiment so the resulting structures are poorer. The increased C_r values for this algorithm on large problem sizes are a by-product of this algorithm's workflow. In DEUM-LDA the parameter estimation step selected individuals from a new population of size 4.4N rather than recycling that produced in the course of the LDA run. DEUM-χ^2 uses the structure learning population, which is very large. The high selective pressure results in a slightly better model of fitness with the same model structure, in line with the results reported in (Brownlee, McCall, Zhang & Brown 2008).

7.4.2.3 Optimisation Results

We know from the results in section 7.4.1 that the algorithm can find the global optimum when the learned structure has F equal to 1.0 relative to the true structure so the optimi-

PS	C_r	FE	FE-SD	IT	IT-SD	SR
16	0.814	216	17	646	1074	90
25	0.831	340	32	1278	2828	77
36	0.830	479	21	887	1621	60
49	0.807	722	88	6863	6948	50
64	0.816	1005	256	17435	26404	43
100	-	-	-	-	-	0
256	-	-	-	-	-	0
324	-	-	-	-	-	0
400	-	-	-	-	-	0

Table 7.9: DEUM-χ^2 Optimisation Statistics over 30 runs

sation experiment was instead run on the structures learned by the Chi-Square algorithm with the smaller population - statistics for which are shown in Table 7.7. The motivation for this is that if the global optimum can be found using these imperfect structures then we have a significant saving in function evaluations over the LDA based algorithm.

Now we run the full optimisation algorithm incorporating the Chi-Square structure learner and report the results in Table 7.9. As before, the C_r value is given to show the fitness prediction power of the model in the context of optimisation. The mean number of function evaluations (FE) and internal iterations of the Gibbs sampler (IT) are given along with their standard deviations (FE-SD and IT-SD). All of these values only relate to successful runs; the success rate (SR) is given as a percentage over 30 independent runs.

We can see that with the decrease in the fitness prediction capability of the model there is a marked decrease in the optimisation capability of the algorithm. Indeed, it is clear that the C_r values for the runs which proved successful (Table 7.9) were on average higher than those found across all runs 7.7. To improve on these results, the population size was again increased to 2N for the larger instances of the problem and the experiment rerun.

In contrast to the previous section, we see that the results are still poor. This can be attributed to the imperfections in the structure learned by the independence test method. The recall values of around 0.9 indicate that up to 10% of the interactions present in the perfect structure are missing; precision values of around 0.96 indicate that up to 4% of the interactions which were added to the model are not present in the perfect structure.

PS	C_r	FE	FE-SD	IT	IT-SD	SR
16	0.951	381	5	598	230	100
25	0.971	599	5	2122	2142	100
36	0.951	852	9	1425	543	100
49	0.940	1166	13	8192	14735	100
64	0.936	1505	16	1875	1941	100
100	0.939	2799	744	316472	525816	70
256	-	-	-	-	-	0
324	-	-	-	-	-	0
400	-	-	-	-	-	0

Table 7.10: DEUM-χ^2 Optimisation Statistics with increase population size over 30 runs

7.4.3 EvDEUM-χ^2

Based on the poor results for the single-step algorithm it is worth determining whether an evolutionary approach would be able to overcome the issues with poor structure. This is given in Algorithm 7.6.

One important change was a reduction in the run time for the Gibbs sampler. As for DEUM-Chain, with this algorithm there is no longer an assumption that a model with a close fit to the fitness function will be found in the first generation. Again, it was found that performance was improved by varying these parameters over the course of the evolution - reducing the cooling rate and increasing the maximum number of iterations with each generation. For each generation g, the cooling rate r was calculated according to (7.9) and the maximum number of iterations of the Gibbs sampler I was calculated according to (7.10).

$$r = 1 + (g^2/10) \tag{7.9}$$

$$I = 0.1/2g \tag{7.10}$$

We also adopted a steady-state approach for the algorithm, replacing 5% of the population each generation. This allows us to use a large population for the structure learning component and maintain diversity for as long as possible.

Algorithm 7.6 EvDEUM-χ^2

```
1: Generate random initial population $p$
2: while Stopping criteria not met do
3:     Select a subset $\sigma_1$, the top 25% of $p$
4:     Run Chi-Square edge detection algorithm to search for statistical dependencies apparent in $\sigma_1$
5:     Refine structure
6:     Select a subset $\sigma_2$, the top 1.1N of $p$
7:     Use $\sigma_2$ to build MFM
8:     Calculate $C_r$
9:     Sample $r$ new individuals from MFM using random walk Gibbs sampler and replace poorest $r$ individuals in $p$ with these
10: end while
```

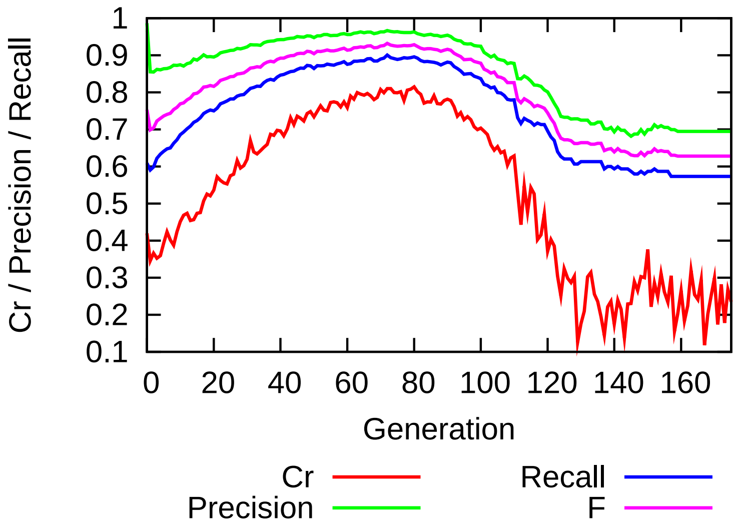


Figure 7.10: Fitness Model Statistics for EvDEUM-χ^2 on 25bit 2D Ising lattice over 30 runs

7.4.3.1 Fitness Model

As with DEUM-Chain, there are now multiple models being created with corresponding multiple figures for p, r, F and C_r. For ease of interpretation, the values over the course of evolution for three instances of the problem (25 bit, 100 bit and 256 bit) are represented graphically in Figures 7.10, 7.11 and 7.12. 25 bits was chosen for the first problem to look at rather than 16 bits because the algorithm was often able to solve the 16 bit instances of the problem in a single or very small number of generations so the chance to observe an effect over many generations was reduced.

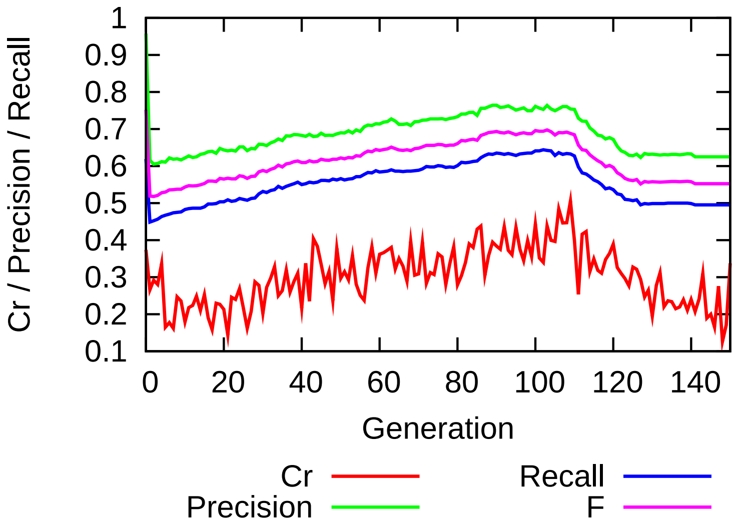


Figure 7.11: Fitness Model Statistics for EvDEUM-χ^2 on 100bit 2D Ising lattice over 30 runs

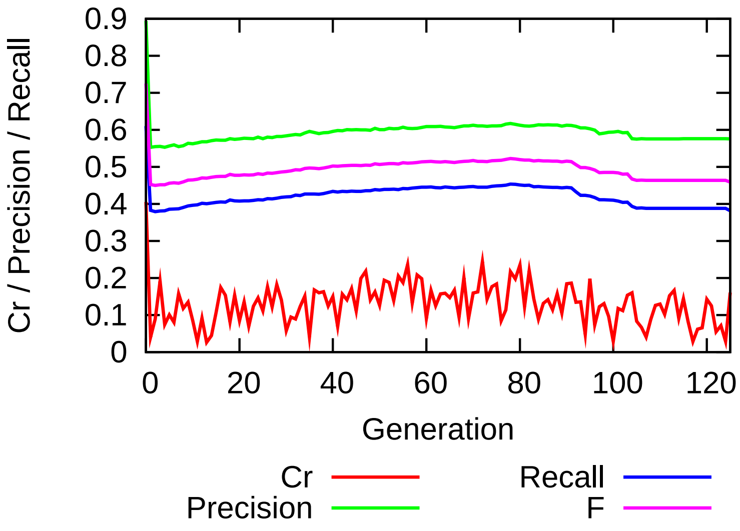


Figure 7.12: Fitness Model Statistics for EvDEUM-χ^2 on 256bit 2D Ising lattice over 30 runs

We can see that the structure quality and consequently the fitness modelling capability of the model rise to begin with and then fall off as the population converges and diversity decreases. Further work is needed to determine the factors which affect the point at which this occurs. With increasing problem size the maximum values reached for structure quality and fitness prediction capability becomes lower, never exceeding an F of 0.5 or a C_r of 0.3 as evolution proceeds. This means the model is unlikely to be good enough to allow the algorithm to find the global optimum. It is also notable that in the first generation the precision and recall of the structure and C_r for the model are all relatively high, this then drops off immediately in the second generation.

7.4.3.2 Optimisation Results

For the optimisation capability of the algorithm, we can present the results in the same way as for the previous algorithms in Table 7.11. As before, the mean number of function evaluations (FE) and internal iterations of the Gibbs sampler (IT) are given along with their standard deviations (FE-SD and IT-SD). All of these values only include the successful runs; the success rate (SR) is given as a percentage over 30 independent runs. No C_r values are present this time because that value varied over the course of the evolution.

The results in this section reflect the poor quality of the models learned. We can see that even for small instances of the problem, the success rate is below 100% and the total number of function evaluations used by the algorithm is higher than for the single step algorithms. As the problem size increases, the perfect structure is never found and the algorithm is unable to find the global optimum. In each generation, the model build time is reduced because the structure learned has fewer interactions, means fewer terms in the model. The sampling times are greatly reduced over the single step approach as we are deliberately lowering the iteration cap on the sampler to encourage diversity in the population. Both of these benefits are lost when repeated over many generations. While

PS	FE	FE-SD	IT	IT-SD	SR
16	2465	3254	4292	10233	100
25	21135	2861	2564001	2010019	83
36	25913	911	3226402	1154681	100
49	33321	1826	8193559	3778421	67
64	-	-	-	-	0
100	-	-	-	-	0
256	-	-	-	-	0
324	-	-	-	-	0
400	-	-	-	-	0

Table 7.11: EvDEUM-χ^2 Optimisation Statistics over 30 runs

the number of function evaluations is reduced in each generation (compared to the number required by the single-step structure learning algorithms), the evolutionary approach does not allow the structure to be found with reduced data.

7.4.3.3 Analysis

In each of the experiments we can see a rather different picture. The onemax problem has no interactions to be found and consequently a high selection pressure on the structure learning algorithm results in a number of useless interactions being added to the model. The 2D Ising problem naturally lends itself to a bivariate structure learning algorithm, itself being constructed round an undirected bivariate graph.

Thus for future applications of the DEUM framework to new optimisation problems we can propose a standard approach to be taken; attempt to construct a MFM using the univariate structure first. If the FPC values for this are poor, increase the maximum clique size on the model to 2 and run the structure learning algorithm. This could be extended further to repeat for higher order interactions.

7.5 Summary

In this chapter we have looked at structure learning in the context of Markov networks and our main contribution is the incorporation of structure learning into the existing DEUM framework, which previously required the structure to be supplied to it. Further to this, in our analysis of the different structure learning approaches we have introduced measures which are new to the EDA community: precision, recall and the F-measure. We believe that these are helpful terms when comparing structure learning algorithms on benchmark problems with a clearly defined underlying structure such as 1D Checkerboard and Ising. They differ from existing terms which describe aspects of structure such as benign/malign and unnecessary interactions - those terms describe the influence an interaction has on fitness and whether an interaction is required by the model for optimisation. Precision, recall and the F-measure refer specifically to the number of interactions that the structure learning algorithm has found relative to the known structure and allow a precise measurement of the effectiveness of a structure learning algorithm.

We have seen that the algorithm is able to optimise both of the benchmark functions it was applied to but the overhead (model build and sampling time) is expensive. The MFM requires a near-perfect structure to be supplied and a large population for optimisation to be successful. This means that in its current form it is unlikely to be competitive with other EDAs, except perhaps where the MFM could be used as a surrogate fitness model for a very expensive fitness function. An incremental approach to the model building step like that of iBOA (Pelikan, Sastry & Goldberg 2008) offers a potential improvement and offers potential for future work. Additionally, other ways of making use of the fitness model may give better results than the Gibbs sampler. These include guided operators in a hybrid algorithm incorporating guided operators (Zhang et al. 2005) and surrogate fitness models (Pelikan & Sastry 2004, Lima et al. 2006, Sastry et al. 2006, Orriols-Puig et al. 2007).

An interesting observation to come out of this work is the dropoff in fitness modelling capability as evolution proceeds. We attribute this to loss of diversity in the population. The problem was mitigated by the use of a steady-state approach but did still prove a hindrance over time. This relates to other work on diversity in EAs (Ochoa & Soto 2006, Handa 2005, Mahnig & Mühlenbein 2001, Branke, Lode & Shapiro 2007, Pošík 2008, Higo & Takadama 2008) and another avenue for future work is mutation of the probabilistic model, niching (Dong & Yao 2008a) or other technique to reduce the effect of diversity loss and improve optimisation performance.

It could be argued that some problem-specific knowledge is still being supplied to the algorithm; for 1D Checkerboard both structure learning approaches will produce a chain structure and for 2D Ising we limit the approaches to only finding bivariate interactions. With the simple addition of a maximal clique finding algorithm it will also be possible to apply the independence test approach to problems with higher order interactions such as SAT. Further to this it would be worth looking at the effectiveness of other independence tests in the context of building an MFM, as well as alternative means of learning permutations for the chain model such as Ant Colony Optimisation (Dorigo & Gambardella 1997).

ACO has been shown to perform well on permutation based problems such as the travelling salesman problem and could be combined in a hybrid EDA with DEUM-Chain. It will be interesting to see if the effects described here hold true for these other problems and structure learning techniques.

Chapter 8

Future Work

This chapter explores some of the possibilities for future research based on this work. This can be grouped into four sections:

1. Exploration of higher cost and continuous fitness functions
2. Improving the efficiency of model building
3. Further work on effects of genetic operators
4. Hybrid algorithms exploiting MFM

8.1 Different fitness functions

The work presented in Chapters 6 and 7 using DEUM for function optimisation has made use of a number of benchmark functions which are inexpensive to run. It would be interesting to conduct further experiments using more expensive fitness functions. In particular, fitness functions which incorporate human feedback like evolutionary art (Romero & Machado 2007) or which run over a network such as the Huygens probe function discussed in Section 8.4. In these contexts the high algorithm overhead could be more easily justified by the reduction in fitness evaluations required in searching for a global optimum.

Further to this, it would be interesting to adapt DEUM to use discrete integer or continuous variables in place of a bitstring. This would initially appear to be a simple change; the energy functions in the Markov network use variables which can take on any real value and we currently encode bitstrings into +1 and -1 before the system of equations is built. Additionally the least squares fitting of SVD has no requirement for the variables to take only two values. This would make it appear that it would be simple to adapt the algorithm to work directly with values other than +/-1. In practice it is unlikely to be so straightforward. We would have to ensure that variables having a value of zero are processed in some way, so that terms are not cancelled out from the energy function (as would be the case when trying to use the bitstring encoding directly). More importantly it is not clear what the effect would be of removing the mathematical symmetry inherent in the +/-1 encoding. The α_k coefficients would have to represent both the influence on fitness of a clique k and the energy distribution across absolute values of a variable. It is possible that some information would not be encoded by the model because of this.

Further, the algorithm would no longer be building a set of energy functions with direct parity with the Walsh functions for the problem. Considerable work would need to be done to explore the implications of this on the underlying model being built by DEUM. The ability to use integer or real variables would be very useful as this is a more natural encoding for many real-world problems.

8.2 Improving model build efficiency

The results presented in Chapters 4 and 5 illustrate that building a model with a strong positive correlation with the fitness function requires a large population and perfect structure (typically with a large number of interactions). This renders the model building process highly expensive using a deterministic algorithm such as SVD. A number of possibilities exist for reducing the cost of model building and we discuss some of these in this section.

8.2.1 Experimental design

Experimental design is a technique used to structure a set of experiments to systematically observe the effect of a number of interacting variables on a dependent variable.

It has been used for tuning the highly interdependent parameters of genetic algorithms (Petrovski, Brownlee & McCall 2005). In (Zhang 2001) experimental design is used to generate the starting population for the algorithm in place of random generation. (Reeves & Wright 1995) explores the relationships between experimental design, genetic algorithms and Walsh transformations.

Experimental design ensures improves the efficiency of the population with no duplication due to chance repeated patterns in the randomly generated individuals. This enables the total size of the population to be reduced. The technique also enables an algorithm to ensure that likely interaction points can be tested in detail by increasing the variation of variables around these points. This is of particular interest in the context of building an MFM where we may have strong coefficients associated with particular cliques found in one generation which would be useful to study more closely in future generations.

This approach could also lead to a reversal of the way in which the algorithm parameters are determined. Instead of determining the number of parameters we can estimate in the MFM based on population size, we determine how much computation can be afforded on model building and use this to determine the population size. This may be because for a given problem it is desirable to find only a good solution in a fixed number of function evaluations (similar to Section 8.4). Experimental design could be used to build a population which fits with the size requirement but still allows points of interest in the set of variables to be studied closely.

8.2.2 Incremental model

In subsection 8.2.1 we discussed ways of reducing parameter estimation time by reducing the population, in turn reducing the matrix size so that SVD has a smaller system of equations to solve. An alternative to this is to build the model incrementally, similar to the approach recently proposed for the Incremental Bayesian Optimisation Algorithm (iBOA) (Pelikan et al. 2008). This would fit more naturally into an evolutionary algorithm where the information about fitness from new individuals in each generation could be incorporated into the existing model. (Brand 2002) proposes a technique for updating a singular value decomposition by rotating the component matrices of the decomposition. That paper also provides a summary of previous work in the area.

An alternative method would be to use the fitness prediction correlation as the basis for a fitness function which measures the error between predicted and true fitness of new individuals in the population. A simple hillclimber would alter the parameters to reduce this error. Subsection 8.2.3 extends this idea further.

8.2.3 Parameter and structural evolution

A radically different means of reducing the computation time required would be to use an evolutionary algorithm such as PSO or a real-valued GA to evolve the model parameters. This is inspired by existing hybrid algorithms such as meta-GAs (Back, Fogel & Michalewicz 1997); (Schmidt & Lipson 2008) demonstrates coevolution as an efficient means of obtaining fitness predictors for a range of fitness functions.

An obstacle to this has previously been the selection of an appropriate fitness function. The fitness prediction correlation presented in Chapter 3 could be used for such a purpose. The objective would be to evolve the set of parameters which give the model with a FPC closest to +1 for a given randomly generated population.

8.2.4 Model partitioning

A further means of reducing the parameter estimation time would be to partition the model into parts. At a high level this approach is similar to the operation of eCGA (Harik 1999). That algorithm divides the problem variables into independent groups and computes marginal probability distributions for each group. In a similar fashion the Markov network could be partitioned into smaller independent networks which have their parameters computed separately. Given the $O(n^3)$ complexity of the SVD algorithm, a small number of divisions would result in a considerable reduction in computational cost.

Two major issues arise with this: the first is the manner in which the network would be partitioned. A greedy algorithm could be used for this purpose, removing interactions from the model which have a small corresponding energy (that is, the magnitude of the associated alpha value is small, indicating the interaction has a small influence on fitness relative to other interactions). Alternatively, the model could be partitioned by removing low-order maximal cliques which join higher-order cliques. The second issue is how to attribute changes in fitness to the different partitions. This could be solved by fixing values of some variables to determine the influence of others. In practice, cross-group interactions are likely occur and any partitioning of the model in this manner will result in a tradeoff in fitness modelling capability. The balance between this and acceptable computing cost will need to be considered carefully as is the case with other methods of reducing computational complexity.

The Kikuchi approximation used by MN-EDA (Santana 2005) is based on a similar region-based decomposition of the Markov network. Another approach would be to exploit the Markovianity property of the model to avoid needing to compute the full Markov network. An algorithm which does this is MOA (Shakya & Santana 2008b).

One further approach could be to reduce the model complexity by including in the MFM only the cliques which have large corresponding Walsh coefficients. In (Thierens 1999a, Thierens 1999b) it was proposed that the KM Algorithm (Kushilevitz & Mansour 1993) could be used for computing Walsh coefficients for such a purpose.

8.3 Further work on effects of genetic operators

Chapter 5 described a series of experiments exploring the effect of selection on the fitness model. An obvious extension to this work would be further experiments using different selection operators such as tournament or fitness proportionate selection and well as different fitness functions. It would be useful to the wider community to explore the question of which selection operator is best suited to what problem and how best to use selection to efficiently extract information about fitness from a population.

It would also be interesting to use the fitness model as a means of exploring the distribution of fitness information within the population as an EA evolves. This was briefly described in Chapter 7. The EA would run as normal but at each generation the population would be used to generate a MFM for which an FPC value could be calculated. This would complement other work on fitness variance and distribution (Ochoa & Soto 2006, Handa 2005, Mahnig & Mühlenbein 2001, Branke et al. 2007, Pošík 2008, Higo & Takadama 2008). Further to this, (Brown et al. 2002) described some initial work exploring the fitness models produced from populations which had crossover applied to them. This concept could be taken further by investigating the quality of fitness model obtained after crossover and mutation have been applied. From this it may be possible to make inferences into the effect of crossover and mutation on the fitness information held by a population. This would help us to have a better understanding of which genetic operators suit what classes of problems, as well as when and how often they should be applied. It would also have the potential to help us design better crossover and moutation operators.

The study of C_m values also reaches into the issue of locality - and what makes an individual "close" to another. There is also much potential for study in this area.

8.4 Hybrid approach - Huygens probe

The Huygens probe problem was presented as part of a competition at the 2006 Congress on Evolutionary Computation, used to benchmark competing algorithms. The objective of the problem is to find the lowest point on each of a series of 20 "moons" - fractal landscapes (generated by sequences of meteor impacts) that are wrapped in both x and y dimensions. For each moon an algorithm is restricted to 1000 probes (fitness evalutations). The problem used a SOAP interface so that competing algorithms could be compared using a central server. More on this and the fitness function can be found in (MacNish 2005).

The heavy restriction on function evaluations meant that fitness modelling had strong potential. This section will explain the approach which we employed and present results with comparisons to the other algorithms tried in the competition, as the basis for future development of hybrid algorithms using the MFM.

8.4.1 MFM and Encoding

As we have seen, to build a useful structure requires a large number of fitness evaluations and so the model was restricted to a univariate structure. This would also reduce the complexity of the model, reducing the size of population required to build a good model of fitness. Initial experiments revealed that the best encoding for the problem using a bitstring would be to have the bits represent a binary encoding of the coordinate values. An alternative using a direct mapping of bits onto regions of the landscape was tried but did not offer a high enough resolution to efficiently find the global optimum.

8.4.2 Guided Hillclimber

The guided hillclimber was developed for optimisation of the Huygens Probe problem. It makes use of the fitness prediction capability of the MFM. The general approach is given in Algorithm 8.1.

Algorithm 8.1 Guided Hillclimber

```
1: Generate random initial population $p$ of size $M$
2: while chosen proportion of the available fitness evaluations not completely used do
3:   Build univariate MFM modelling $p$
4:   Trunction selection: select a subset $\sigma$, the fittest $s$ individuals in $p$
5:   for all individuals in $\sigma$ do
6:     Generate $m$ neighbours $\mu$:
7:     Convert bitstring into real valued coordinates
8:     Mutate values by up to a fixed amount (which decreases with each generaton)
9:     Convert numbers back to bitstring
10:  end for
11:  Use MFM to predict fitnesses of $\mu$
12:  Select predicted best $l$ individuals $\varsigma$ from $\mu$, calculate true fitnesses
13:  Take best $M$ from combined pool of $p$ and $\varsigma$ and replace $p$
14: end while
```

The method for generating neighbours was adopted because of the representation used for the problem and could be simplified to a simple bit-flip mutation if a Gray encoding or similar were used. The algorithm described allows the model to be rebuilt around progressively smaller areas of the lunar surface which is suited to the fractal nature of the landscape (equal levels of detail at different zoom levels). Initial experiments for the guided hillclimber used a univariate model; given that it is looking at neighbouring solutions to those used to build the model the results in Chapter 4 indicate that this should be enough to provide a reasonable fitness prediction capability.

Once the guided hillclimber terminated with a single best solution x found the remainder of the 1000 evaluations were used up by an exhaustive search of the neighbouring solutions to x. The proportion of the function evaluations allocated to each stage of the algorithm was a preset parameter; in the first instance the guided hillclimber was given 2/3 of the total, in the second it was given 3/4.

Results for this algorithm were compared with a number of others taking part in a competition as part of the IEEE Congress on Evolutionary Computation 2006, and are shown in Table 8.1. To produce these results the algorithm was run 100 times, with each run being perfomed on a different randomly generated moon. Each algorithm was run on the same set of 100 moons, with a central server providing fitness evaluations and perfoming comparisons between algorithms. It can be seen that the algorithm performed comparably with a number of well-known problem solvers such as evolutionary strategies (ES), memetic algorithms (MA) and simulated annealing (SA). Unfortunately no more data is available on the specific implementation of these algorithms for this problem.

In this thesis we have seen that the MFM with an imperfect model can predict fitness accurately for neighbouring solutions, even with a greatly simplified structure. We have also seen that building a perfect model and running the Gibbs sampler is highly computationally expensive. Considering these factors, the work presented in this section has great promise. It combines an imperfect MFM which is cheap to construct with an existing and well-understood local search technique with low overhead to achieve competitive optimisation performance. This presents an interesting alternative application for the MFM which lies somewhere between guided genetic operators such as (Sun et al. 2008, Zhang & Sun 2006, Zhang et al. 2005, Peña et al. 2004) and surrogate fitness models (Lim et al. 2008, Sastry et al. 2006, Ong et al. 2004). A more extensive study using different fitness functions and a multivariate MFM could yield some interesting results. It would also be interesting to see if a real-valued DEUM would be more efficient at optimising this problem.

Rank	Algorithm	Description
1	WF3	ES+NS
2	Voronoi	16,16 Method A
3	Voronoi	10,10 Method A
4	WF2	ES
5	WF1	Particle Swarm
6	Voronoi search	10,10 Method B
7	Memetic Algorithm	MA-5
8	Memetic Algorithm	MA-4
9	MFM Guider Hillclimber	2/3 guided hillclimber
10	Memetic Algorithm	MA-3
11	MFM Guider Hillclimber	3/4 guided hillclimber
12	Simulated Annealing	SA-5
13	Recursive Sampling Search	Recursively search from uniform samples
14	Simulated Annealing	SA-5
15	Simulated Annealing	L1
16	Recursive Sampling Search	Recursively search from uniform samples

Table 8.1: Competition Results

Chapter 9

Conclusion

This chapter outlines the contributions made by the research and concludes the thesis.

9.1 Important contributions

The following are the most important contributions made over the course of the research.

1. **Review of EDA and fitness modelling:** A review of algorithms incorporating probabilistic distributions and fitness models has been presented.

2. **Extension of MFM approach:** The existing Markov Fitness Model of DEUM has been extended to use a multivariate graphical structure, specified in terms of Walsh functions.

3. **Fitness Prediction Correlation:** The FPC has been proposed and demonstrated as a measure of fitness modelling capability for Markov fitness models.

4. **Analysis of relationship between MFM and fitness functions:** A study of the coefficient values in the MFM for a range of benchmark functions has been conducted. This helps us to better understand the relationship between the MFM and fitness.

5. **Analysis of effects of selection and population size on fitness modelling**: an extensive study of the effects of two key EA parameters (selection and popula-

tion size) on fitness modelling has been performed, with experimental results for a selection of benchmark problems assuming that other EA parameters are constant.

6. **Introduction of new fixed structure DEUM algorithms:** Variations of the DEUM framework using different structures have been proposed and applied to several benchmark functions which it had not been previously. It has been applied to the chain structured 1D Checkerboard and biocontrol problems. It was also applied to 3CNF MAXSAT, which required an MFM with trivariate interactions; higher order interactions than DEUM had previously incorporated. A new variation of the Gibbs sampler was also introduced into the framework.

7. **Introduction of structure learning DEUM algorithms** DEUM-Chain, DEUM-Chain-χ^2, DEUM-LDA, DEUM-χ^2 and evDEUM-χ^2 have been proposed as algorithms which incorporate structure learning into the DEUM framework, which previously required a known fixed structure to be supplied before it could run. These algorithms have been applied to different benchmark functions to demonstrate their operation.

8. **Applications:** DEUM has been applied to optimisation of a number of benchmark problems

9. **Precision and Recall:** Precision and Recall have been introduced as quality measures for structure learning algorithms.

9.2 General Conclusion

In this thesis, we have presented the Markov fitness model (MFM) as a tool for evolutionary computation. We have explored a number of themes related to the construction and application of the fitness model which are of interest.

We have developed a language to describe the MFM in terms of Walsh functions; this helps us relate the MFM approach to fitness functions in terms already accepted the wider evolutionary computation community. It also allows us to theoretically extend the MFM to incorporate higher order multivariate interactions than the bivariate structure of the existing Ising-DEUM algorithm. We have been able to gain a deeper understanding of the relationship between the MFM and the fitness function by showing that the model parameters have a direct relationship with the underlying dynamics of a selection of benchmark fitness functions. We have developed a new measure of the fitness modelling capability of the MFM which we have been able to demonstrate is closely related to the usefulness of the model for optimisation.

We have explored several of the many factors which affect how closely the MFM correlates with a fitness function. We can now describe with some certainty the impact which different model structures have for modelling different fitness functions. We have shown that there is a clear relationship between the complexity of a problem's structure and effort in fitness evaluations required to learn a useful model of fitness. We can now say that, at least for the benchmark functions studied, the MFM requires a near-perfect structure and over-specified population to be supplied to it in order for it to predict the fitness changes resulting from a large number of mutations. In many cases studied, with an imperfect structure or underspecified population, the MFM can be used to predict the change in fitness resulting from a single mutation. Further to this we have studied the impact on the fitness prediction capability of the MFM of four variations of truncation selection. These results reveal useful information about how the selection operator sharpens information about fitness in a population which is used to build the MFM. This is of relevance to recent works in the EC community studying diversity, fitness distribution and convergence in populations.

We have also been able to present extensions to existing works directly sampling the MFM for optimisation of problems, using different benchmark functions with different structures to those demonstrated previously. This is useful to demonstrate that the MFM is useful in real world - that models of fitness have a direct link to optimisation. We have also extended the DEUM framework to include multivariate interactions and a structure learning component so it no longer must be supplied with the known fixed structure of a problem. As part of this we have introduced precision and recall for describing the structure of the Markov network and in particular for measuring the quality of learned structures which will be of use to the wider EDA and EA community.

In general, we have been able to develop the MFM as a useful tool to improve the performance of evolutionary computation. We propose that it presents great potential for greater understanding of existing evolutionary algorithms and ultimately improving function optimisation and problem solving.

Bibliography

Abboud, K. & Schoenauer, M. (2002). Surrogate Deterministic Mutation: Preliminary Results, *Selected Papers from the 5th European Conference on Artificial Evolution*, Springer-Verlag, London, UK, pp. 104–116.

Ahn, C. W., Ramakrishna, R. S. & Goldberg, D. E. (2004). Real-coded Bayesian Optimization Algorithm: Bringing the Strength of BOA into the Continuous World, *in* K. Deb (ed.), *Proceedings of the Genetic and Evolutionary Computation COnference (GECCO 2004)*, Springer-Verlag, Berlin, pp. 840–851.

Ashlock, D. A., Bryden, K. M. & Corns, S. (2008). Small population effects and hybridization, *in* J. Wang. (ed.), *Proceedings of the IEEE World Congress on Computational Intelligence (CEC 2008)*, IEEE Computational Intelligence Society, IEEE Press, Hong Kong.

Back, T., Fogel, D. B. & Michalewicz, Z. (eds) (1997). *Handbook of Evolutionary Computation*, IOP Publishing Ltd., Bristol, UK, UK.

Back, T., Hoffmeister, F. & Schwefel, H. (1991). A survey of evolution strategies, *Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufmann, pp. 2–9.

Baluja, S. & Caruana, R. (1995). Removing the genetics from the standard genetic algorithm, *Proceedings of the International Conference on Machine Learning*, Morgan Kaufmann Publishers, pp. 38–46.

Baluja, S. & Davies, S. (1997a). Combining multiple optimization runs with optimal dependency trees, *Technical Report CMU-CS-97-157*, School of Computer Science, Carnegie Mellon University.

Baluja, S. & Davies, S. (1997b). Using optimal dependency-trees for combinational optimization, *ICML '97: Proceedings of the Fourteenth International Conference on Machine Learning*, Morgan Kaufmann Publishers Inc, pp. 30–38.

Berry, M., Do, T., O'Brien, G., (1993). SVDPACKC (Version 1.0) User's Guide, *Technical report*, University of Tennessee, Knoxville, TN, USA.

Bethke, A. (1980). *Genetic Algorithms as Function Optimizers*, PhD thesis, University of Mitchigan.

Boslaugh, S. & Watters, P. A. (2008). *Statistics: A Desktop Quick Reference*, O'Reilly.

Bosman, P. A. N. & Thierens, D. (1999). Linkage information processing in distribution estimation algorithms, *in* W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela & R. E. Smith (eds), *Proceedings of the Genetic and Evolutionary Computation COnference (GECCO 1999)*, Morgan Kaufmann, pp. 60–67.

Bosman, P. A. N. & Thierens, D. (2000). Continuous iterated density estimation evolutionary algorithms within the IDEA framework, *in* H. Mühlenbein, M. Pelikan & A. O. Rodriguez (eds), *Proceedings of the Genetic and Evolutionary Computation COnference (GECCO 2000)*, Morgan Kauffmann, pp. 197–200.

Brand, M. (2002). Incremental singular value decomposition of uncertain data with missing values, *ECCV '02: Proceedings of the 7th European Conference on Computer Vision-Part I*, Springer-Verlag, London, UK, pp. 707–720.

Branke, J., Lode, C. & Shapiro, J. L. (2007). Addressing sampling errors and diversity loss in UMDA, *Proceedings of the Genetic and Evolutionary Computation COnference (GECCO 2007)*, ACM Press, New York, NY, USA, pp. 508–515.

Bron, C. & Kerbosch, J. (1973). Algorithm 457: finding all cliques of an undirected graph, *Communications of the ACM* **16**(9): 575–577.

Brown, D. F., Garmendia-Doval, A. B. & McCall, J. A. W. (2002). Markov random field modelling of royal road genetic algorithms, *Selected Papers from the 5th European Conference on Artificial Evolution*, Springer-Verlag, pp. 65–76.

Brownlee, A. E. I., McCall, J. A. W. & Brown, D. F. (2007). Solving the MAXSAT problem using a multivariate EDA based on Markov networks, *Proceedings of the Genetic and Evolutionary Computation COnference (GECCO 2007) (Late Breaking Papers)*, ACM Press, New York, NY, USA, pp. 2423–2428.

Brownlee, A. E. I., McCall, J. A. W., Shakya, S. K. & Zhang, Q. (2009). Structure Learning and Optimisation in a Markov-network based Estimation of Distribution Algorithm, *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2009)*, IEEE Press, Trondheim, Norway.

Brownlee, A. E. I., McCall, J. A. W., Zhang, Q. & Brown, D. (2008). Approaches to Selection and their effect on Fitness Modeling in an Estimation of Distribution Algorithm, *Proceedings of the IEEE World Congress on Computational Intelligence (CEC 2008)*, IEEE Press, Hong Kong, China.

Brownlee, A. E. I., Pelikan, M., McCall, J. A. W. & Petrovski, A. (2008). An application of a multivariate estimation of distribution algorithm to cancer chemotherapy, *Proceedings of the Genetic and Evolutionary Computation COnference (GECCO 2008)*, ACM Press, New York, NY, USA, pp. 463–464.

Brownlee, A. E. I., Wu, Y., McCall, J. A. W., Godley, P. M., Cairns, D. E. & Cowie, J. (2008). Optimisation and fitness modelling of bio-control in mushroom farming using a Markov network EDA, *Proceedings of the Genetic and Evolutionary Computation COnference (GECCO 2008)*, pp. 465–466. Atlanta, Georgia, USA, ACM.

Chen, J.-H., Goldberg, D., S.-Y.Ho & K.Sastry (2002). Fitness inheritance in multi-objective optimization, *Proceedings of the Genetic and Evolutionary Computation COnference (GECCO 2002)*, ACM Press, pp. 319–326.

Chickering, D. M., Geiger, D. & Heckerman, D. (1994). Learning Bayesian networks is

NP-Hard, *Technical Report MSR-TR-94-17*, Microsoft.
URL: *"citeseer.ist.psu.edu/chickering94learning.html"*

Chow, C. K. & Liu, C. N. (1968). Approximating discrete probability distributions with dependence trees, *IEEE Transactions on Information Theory* **14**(3): 462–467.

Collard, P., Verel, S. & Clergue, M. (2004). Local search heuristics: Fitness cloud versus fitness landscape, *in* R. L. de Mántaras & L. Saitta (eds), *Poster at the 2004 European Conference on Artificial Intelligence (ECAI04)*, IOS Press, Valencia, Spain, pp. 973–974.

Cooper, G. F. & Herskovits, E. (1991). A bayesian method for the induction of probabilistic networks from data, *Technical Report KSL-91-02*, Knowledge Systems, AI Laboratory, Stanford, CA, USA.

Dasgupta, D. (ed.) (1998). *Artificial Immune Systems and Their Applications*, Springer-Verlag.

Davis, L. (1991). *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York.

de Bonet, J. S., Isbell Jr., C. L. & Viola, P. (1997). MIMIC: Finding optima by estimating probability densities, *in* M. C. Mozer, M. I. Jordan & T. Petsche (eds), *Advances in Neural Information Processing Systems*.

De Castro, L. N. & Timmis, J. (2002). *Artificial Immune Systems: A New Computational Intelligence Approach*, Springer.

Deb, K. (2001). *Multi-Objective Optimization using Evolutionary Algorithms*, Wiley-Interscience Series in Systems and Optimization, John Wiley & Sons, Chichester.

Dong, W. & Yao, X. (2008a). NichingEDA: Utilizing the diversity inside a population of EDAs for continuous optimization, *Proceedings of the IEEE World Congress on Computational Intelligence (CEC 2008)*, pp. 1260–1267.

Dong, W. & Yao, X. (2008b). Unified eigen analysis on multivariate gaussian based estimation of distribution algorithms, *Information Science* **178**(15): 3000–3023.

Dorigo, M. & Gambardella, L. M. (1997). Ant colonies for the travelling salesman problem, *Biosystems* **43**(2): 73–81.

Droste, S., Jansen, T. & Wegener, I. (2002). On the analysis of the (1+1) evolutionary algorithm, *Theor. Comput. Sci.* **276**(1-2): 51–81.

Eberhart, R. C. & Kennedy, J. (1995). A new optimizer using particle swarm theory, *Proceedings of the Sixth International Symposium on Micromachine and Human Science*, pp. 39–43.

Etxeberria, R. & Larrañaga, P. (1999). Global optimization with Bayesian networks, *II Symposium on Artificial Intelligence, CIMAF99*.

Feller, W. (1957). *An introduction to probability theory and its applications (2 vols)*, John Wiley & Sons.

Fenton, A., Gwynn, R., Gupta, A., R.Norman & J.P.Fairbairn (2002). Optimal application strategies for entomopathogenic nematodes: integrating theoretical and empirical approaches, *Journal of Applied Ecology* **39**: 481–492.

Fogel, L. J. (1962). Autonomous automata, *Industrial Research* **4**: 14–19.

Fogel, L. J. (1964). *On the Organization of Intellect*, PhD thesis, University of California, Los Angeles, CA.

Glover, F. & Laguna, F. (1997). *Tabu Search*, Kluwer Academic Publishers, Norwell, MA, USA.

Godley, P. M., Cairns, D. E. & Cowie, J. (2007a). Directed intervention crossover applied to bio-control scheduling, *Proceedings of the 2007 IEEE Congress on Evolutionary Computation (CEC 2007)*, pp. 638–645.

Godley, P. M., Cairns, D. E. & Cowie, J. (2007b). Maximising the efficiency of bio-control application utilising genetic algorithms, *EFITA / WCCA 2007: Proceedings of the 6th Biennial Conference of European Federation of IT in Agriculture*, Glasgow Caledonian University, Glasgow, Scotland, UK.

Godley, P. M., Cowie, J. & Cairns, D. E. (2007). Novel genetic algorithm approaches for time-series problems, *Doctoral Symposium on Engineering Stochastic Local Search Algorithms*, IRIDIA, pp. 47–51. ISSN: 1781-3794.

Goldberg, D. (1989a). Genetic Algorithms and Walsh Functions: Part I, A Gentle Introduction, *Complex Systems* **3**: 129–152.

Goldberg, D. (1989b). Genetic Algorithms and Walsh Functions: Part II, Deception and its Analysis, *Complex Systems* **3**: 153–171.

Goldberg, D. & Deb, K. (1991). A comparative analysis of selection schemes used in genetic algorithms, *in* G. J. E. Rawlins (ed.), *Foundations of Genetic Algorithms*, Morgan Kaufmann, pp. 69–93.

Goldberg, D. E. (1989c). *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, MA.

Handa, H. (2005). Estimation of distribution algorithms with mutation, *Evolutionary Computation in Combinatorial Optimization*, Vol. 3448 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 112–121.

Harik, G. (1999). Linkage learning via probabilistic modeling in the ECGA, *Technical Report IlliGAL Report No. 99010*, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign.

Harik, G. R., Lobo, F. G. & Goldberg, D. E. (1997). The compact genetic algorithm, *Technical Report IlliGAL Report No. 97006*, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign.

Hauschild, M., Pelikan, M., Lima, C. F. & Sastry, K. (2007). Analyzing probabilistic models in hierarchical BOA on traps and spin glasses, *Proceedings of the Genetic and Evolutionary Computation COnference (GECCO 2007)*, ACM Press, New York, NY, USA, pp. 523–530.

Heckendorn, R. B. & Whitley, D. (1999). Predicting epistasis directly from mathematical models, *Evolutionary Computation* **7**(1): 69–101.

Heckendorn, R. B. & Wright, A. H. (2004). Efficient linkage discovery by limited probing, *Evolutionary Computation* **12**(4): 517–545.

Heckerman, D., Geiger, D. & Chickering, D. M. (1995). Learning Bayesian networks: The combination of knowledge and statistical data, *Machine Learning* **20**(3): 197–243.

Higo, T. & Takadama, K. (2008). Maintaining multiple populations with different diversities for evolutionary optimization based on probability models, *IPSJ Digital Courier* **4**: 268–280.

Holland, J. H. (1975). *Adaptation in natural and artificial systems : an introductory analysis with applications to biology, control, and artificial intelligence*, University of Michigan Press, Ann Arbor. by John H. Holland.; Includes index.; Bibliography: p. 175-177.

Hoos, H. H. & Stützle, T. (2000). *SATLIB: An Online Resource for Research on SAT*, SAT 2000, IOS Press, pp. 283–292.

Hoschek, W. (2004). Colt.
URL: *http://dsd.lbl.gov/˜hoschek/colt/*

Jin, Y. (2005). A comprehensive survey of fitness approximation in evolutionary computation, *Soft Computing* **9**(1): 3–12.

Jin, Y. & Sendhoff, B. (2004). Reducing fitness evaluations using clustering techniques and neural network ensembles, *Proceedings of the Genetic and Evolutionary Computation COnference (GECCO 2004)*, Springer, Seattle, WA, pp. 688–699.

Johnson, D. S. (1973). Approximation algorithms for combinatorial problems, *STOC '73: Proceedings of the fifth annual ACM symposium on Theory of computing*, ACM Press, New York, NY, USA, pp. 38–49.

Jones, T. & Forrest, S. (1995). Fitness distance correlation as a measure of problem difficulty for genetic algorithms, *Proceedings of the Sixth International Conference on Genetic Algorithms*, Morgan Kaufmann, pp. 184–192.

Kallel, L., Naudts, B. & Reeves, R. (2000). Properties of fitness functions and search landscapes, *in* L. Kallel, B. Naudts & A. Rogers (eds), *Theoretical Aspects of Evolutionary Computing*, Springer Verlag, pp. 177–208.

Khuri, S. (1994). Walsh and Haar functions in genetic algorithms, *SAC '94: Proceedings of the 1994 ACM symposium on Applied computing*, ACM Press, New York, NY, USA, pp. 201–205.

Kindermann, R. & Snell, J. L. (1980). *Markov Random Fields and their Applications*, American Mathematical Society, Providence, RI.

Kirkpatrick, S., Gerlatt, C. D. & Vecchi, M. P. (1983). Optimization by simulated annealing, *Science* **220**: 671–680.

Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, Cambridge, MA.

Kullback, S. (1987). The kullback-leibler distance, *The American Statistician* **41**: 340–341.

Kushilevitz, E. & Mansour, Y. (1993). Learning decision trees using the Fourier spectrum, *SIAM J. Comput.* **22**(6): 1331–1348.

Larrañaga, P., Etxeberria, R., Lozano, J. A. & Penã, J. (1999). Optimization by learning and simulation of Bayesian and Gaussian networks, *Technical Report EHU-KZAAIK-4/99*, University of the Basque Country.

Larrañaga, P. & Lozano, J. A. (2002). *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*, Kluwer Academic Publishers, Boston.

Lauritzen, S. L. (1996). *Graphical models*, Vol. 17, Clarendon Press; Oxford University Press, New York; Oxford.

Li, S. Z. (1995). *Markov random field modeling in computer vision*, Springer-Verlag.

Lim, D., Jin, Y., Ong, Y.-S. & Sendhoff, B. (2008). Generalizing surrogate-assisted evolutionary computation, *IEEE Transactions on Evolutionary Computation* .

Lima, C. F., Pelikan, M., Sastry, K., Butz, M. V., Goldberg, D. E. & Lobo, F. G. (2006). Substructural neighborhoods for local search in the Bayesian optimization algorithm, *PPSN IX: Proceedings of the 9th International Conference on Parallel Problem Solving from Nature*, Vol. 4193 of *Lecture Notes in Computer Science (LNCS)*, Springer-Verlag, Reykjavik, Iceland, pp. 232–241.

Lima, C. F., Sastry, K., Goldberg, D. E. & Lobo, F. G. (2005). Combining competent crossover and mutation operators: a probabilistic model building approach, *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, ACM, New York, NY, USA, pp. 735–742.

Lucey, T. (1984). *Quantatitive Techniques: An Instructional Manual*, D. P. Publications, Eastleigh, Hampshire, UK.

MacNish, C. (2005). Benchmarking Evolutionary Algorithms: The Huygens Suite, *Proceedings of the Genetic and Evolutionary Computation COnference (GECCO 2005) (Late Breaking Papers)*, ACM Press, New York, NY, USA, pp. 2423–2428.

Mahnig, T. & Mühlenbein, H. (2001). Optimal mutation rate using Bayesian priors for estimation of distribution algorithms, *SAGA '01: Proceedings of the International Symposium on Stochastic Algorithms*, Springer-Verlag, London, UK, pp. 33–48.

Marascuilo, L. A. & McSweeney, M. (1977). *Nonparametric and Distribution-Free Methods for Social Sciences*, Brooks / Cole Publishing Company, California.

McCall, J., Petrovski, A. & Shakya, S. (2008). *Evolutionary Algorithms for Cancer Chemotherapy Optimization*, Computational Intelligence in Bioinformatics, Wiley, chapter 12, pp. 265–296.

Michalski, R. S. (2000). Learnable evolution model: Evolutionary processes guided by machine learning, *Machine Learning* **38**(1-2): 9–40.

Miquélez, T., Bengoetxea, E. & Larrañaga, P. (2004). Evolutionary computation based on Bayesian classifiers, *International Journal of Applied Mathematics and Computer Science* **14**(3): 101–115.

Miquélez, T., Bengoetxea, E., Mendiburu, A. & Larrañaga, P. (2007). Combining Bayesian classifiers and estimation of distribution algorithms for optimization in continuous domains, *Connection Science* .

Mitchell, M. (1998). An introduction to genetic algorithms, Paperback.

Moscato, P. (1989). On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms, *Technical Report C3P 826*, California Institute of Technology.

Mühlenbein, H. & Höns, R. (2005). The estimation of distributions and the minimum relative entropy principle, *Evolutionary Compututation* **13**(1): 1–27.

Mühlenbein, H. & Mahnig, T. (2000). Evolutionary optimization using graphical models, *New Gen. Comput.* **18**(2): 157–166.

Mühlenbein, H., Mahnig, T. & Rodriguez, A. O. (1999). Schemata, distributions and graphical models in evolutionary optimization, *Journal of Heuristics* **5**(2): 215–247.
URL: *http://dx.doi.org/10.1023/A:1009689913453*

Mühlenbein, H. & Paaß, G. (1996). From recombination of genes to the estimation of distributions I. binary parameters, *PPSN IV: Proceedings of the 4th International Conference on Parallel Problem Solving from Nature*, Springer-Verlag, pp. 178–187.

Mühlenbein, H. & Schlierkamp-Voosen, D. (1993). Predictive models for the breeder genetic algorithm I. continuous parameter optimization, *Evolutionary Computuation* **1**(1): 25–49.

Ochoa, A. & Soto, M. R. (2006). Linking entropy to estimation of distribution algorithms, *in* J. A. Lozano, P. L. naga, I. Inza & E. Bengoetxea (eds), *Towards a New Evolutionary Computation: Advances on Estimation of Distribution Algorithms*, Springer-Verlag, pp. 1–38.

Ong, Y. S., Nair, P. B., Keane, A. J. & Wong, K. W. (2004). Surrogate-assisted evolutionary optimization frameworks for high-fidelity engineering design problems, *In Knowledge Incorporation in Evolutionary Computation*, Springer Verlag, pp. 307–332.

Orriols-Puig, A., Bernadó-Mansilla, E., Sastry, K. & Goldberg, D. E. (2007). Substructural surrogates for learning decomposable classification problems: implementation and first results, *Proceedings of the Genetic and Evolutionary Computation COnference (GECCO 2007)*, ACM Press, New York, NY, USA, pp. 2875–2882.

Paul, T. & Iba, H. (2003). Real-coded estimation of distribution algorithm,, *Proceedings of the 5th Metaheuristics International Conference*, pp. 61–66.

Peña, J., Robles, V., Larrañaga, P., Herves, V., Rosales, F. & Pérez, M. (2004). GA-EDA: Hybrid evolutionary algorithm using genetic and estimation of distribution algorithms, *IEA/AIE'2004: Proceedings of the 17th international conference on Innovations in applied artificial intelligence*, Lecture Notes In Computer Science, Springer-Verlag, pp. 361–371.
URL: *http://dx.doi.org/10.1007/b97304*

Pelikan, M. (2002). *Bayesian optimization algorithm: from single level to hierarchy*, PhD thesis, University of Illinois at Urbana-Champaign, Urbana, IL.

Pelikan, M. (2005). *Hierarchical Bayesian Optimization Algorithms.*, Springer Verlag.

Pelikan, M. & Goldberg, D. (2003). Hierarchical BOA solves Ising spin glasses and MAXSAT, *Proceedings of the Genetic and Evolutionary Computation COnference (GECCO 2003)*, Springer-Verlag, pp. 1271–1282.

Pelikan, M. & Goldberg, D. E. (2000). Hierarchical problem solving by the Bayesian optimization algorithm, *in* D. Whitley, D. Goldberg, E. Cantú-Paz, L. Spector, I. Parmee

& H.-G. Beyer (eds), *Proceedings of the Genetic and Evolutionary Computation COnference (GECCO 2000)*, Morgan Kaufmann, pp. 267–274.

Pelikan, M., Goldberg, D. E. & Cantú-Paz, E. (1999). BOA: The Bayesian optimization algorithm, *in* W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela & R. E. Smith (eds), *Proceedings of the Genetic and Evolutionary Computation COnference (GECCO 1999)*, Morgan Kaufmann Publishers, San Francisco, CA, p. 532.

Pelikan, M., Goldberg, D. E. & Lobo, F. (1999). A survey of optimization by building and using probabilistic models, *Technical Report 99018*, Illinois Genetic Algorithms Laboratory (IlliGAL).

Pelikan, M. & Mühlenbein, H. (1999). The bivariate marginal distribution algorithm, *in* R. Roy, T. Furuhashi & P. K. Chawdhry (eds), *Advances in Soft Computing - Engineering Design and Manufacturing*, pp. 521–535.

Pelikan, M., Ocenasek, J., Trebst, S., Troyer, M. & Alet, F. (2004). Computational complexity and simulation of rare events of Ising spin glasses, *Proceedings of the Genetic and Evolutionary Computation COnference (GECCO 2004)*, pp. 36–47.

Pelikan, M. & Sastry, K. (2004). Fitness inheritance in the Bayesian optimization algorithm, *Proceedings of the Genetic and Evolutionary Computation COnference (GECCO 2004)*, pp. 48–59.

Pelikan, M., Sastry, K. & Goldberg, D. E. (2008). iBOA: The incremental Bayesian optimization algorithm, *Proceedings of the Genetic and Evolutionary Computation COnference (GECCO 2008)*, ACM Press, New York, NY, USA, pp. 455–462.

Peña, S. I. V., Aguirre, A. H. & Rionda, S. B. (2008). Designing EDAs by using the elitist convergent EDA concept and the Boltzmann distribution, *Proceedings of the Genetic and Evolutionary Computation COnference (GECCO 2008)*, pp. 477–478.

Petrovski, A., Brownlee, A. & McCall, J. (2005). Statistical optimisation and tuning of

GA factors, *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2005)*, Vol. 1, IEEE Press, p. 758.

Pošík, P. (2008). Preventing premature convergence in a simple EDA via global step size setting, *in* G. R. et al. (ed.), *PPSN X: Proceedings of the 10th International Conference on Parallel Problem Solving from Nature*, Vol. 5199 of *LNCS*, Springer-Verlag Berlin Heidelberg, pp. 549–558.

Pošík, P. & Franc, V. (2007). Estimation of fitness landscape contours in EAs, *in* D. Thierens (ed.), *Proceedings of the Genetic and Evolutionary Computation COnference (GECCO 2007)*, Vol. 1, ACM Press, New York, NY, USA, pp. 562–569. note: ISBN 978-1-59593-697-4.

Press, W. H., Flannery, B. P., Teukolsky, S. A. & Vetterling, W. T. (1986). *Numerical Recipes: The Art of Scientific Computing*, Cambridge University Press, Cambridge, UK.

Price, K. V., Storn, R. M. & Lampinen, J. A. (2005). *Differential Evolution - A Practical Approach to Global Optimization*, Natural Computing Series, Springer.

Rasheed, K. & Hirsh, H. (2000). Informed operators: Speeding up genetic-algorithm-based design optimization using reduced models, *Proceedings of the Genetic and Evolutionary Computation COnference (GECCO 2000)*, Morgan Kaufmann, pp. 628–635.

Rasheed, K., Vattam, S. & Ni, X. (2002). Comparison of methods for using reduced models to speed up design optimization, *Proceedings of the Genetic and Evolutionary Computation COnference (GECCO 2002)*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 1180–1187.

Rechenberg, I. (1973). *Evolutionsstrategie – Optimierung technisher Systeme nach Prinzipien der biologischen Evolution*, Frommann-Holzboog, Stuttgart, GER.

Reeves, C. & Wright, C. (1995). An experimental design perspective on genetic algorithms, *Foundations of Genetic Algorithms 3*, Morgan Kaufmann, pp. 7–22.

Rissanen, J. (1978). Modeling by shortest data description, *Automatica* **14**: 465–471.

Romero, J. & Machado, P. (eds) (2007). *The Art of Artificial Evolution: A Handbook on Evolutionary Art and Music*, Natural Computing Series, Springer Berlin Heidelberg.

Rowntree, D. (1981). *Statistics without tears : a primer for non-mathematicians*, Penguin, Harmondsworth, UK.

Santana, R. (2003a). A Markov network based factorized distribution algorithm for optimization, *Proceedings of the 14th European Conference on Machine Learning (ECML-PKDD 2003); Lecture Notes in Artificial Intelligence*, Vol. 2837, Springer-Verlag, Berlin, pp. 337–348.

Santana, R. (2003b). *Probabilistic modeling based on undirected graphs in Estimation Distribution Algorithms*, PhD thesis, Institute of Cybernetics,Mathematics and Physics, Havana, Cuba.

Santana, R. (2005). Estimation of distribution algorithms with Kikuchi approximations, *Evolutionary Computation* **13**(1): 67–97.

Santana, R., Larrañaga, P. & Lozano, J. A. (2006). Mixtures of Kikuchi approximations, *ECML*, pp. 365–376.

Santana, R., Larrañaga, P. & Lozano, J. A. (2007). Challenges and open problems in discrete EDAs, *Technical Report EHU-KZAA-IK-1/07*, Department of Computer Science and Artificial Intelligence, University of the Basque Country.
URL: *http://www.sc.ehu.es/ccwbayes/technical.htm*

Santana, R., Ochoa, A. & Soto, M. R. (2001). The mixture of trees factorized distribution algorithm, *in* L. Spector, E. Goodman, A. Wu, W. B. Langdon, H. M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. Garzon & E. Burke (eds), *Proceedings of the Genetic and Evolutionary Computation COnference (GECCO 2001)*, Morgan Kaufmann Publishers, pp. 543–550.

Sastry, K., Goldberg, D. E. & Pelikan, M. (2001). Doná evaluate, inherit, *Proceedings of the Genetic and Evolutionary Computation COnference (GECCO 2001). (Also IlliGAL*, Morgan Kaufmann, pp. 551–558.

Sastry, K., Lima, C. & Goldberg, D. E. (2006). Evaluation relaxation using substructural information and linear estimation, *Proceedings of the Genetic and Evolutionary Computation COnference (GECCO 2006)*, ACM Press, New York, NY, USA, pp. 419–426.

Schmidt, M. D. & Lipson, H. (2008). Coevolution of fitness predictors, *IEEE Transactions on Evolutionary Computation* **12**(6): 736–749.

Schwefel, H. P. (1981). *Numerical optimization of computer models*, Wiley & Sons, Chichester.

Selman, B., Levesque, H. J. & Mitchell, D. (1992). A new method for solving hard satisfiability problems, *in* P. Rosenbloom & P. Szolovits (eds), *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, AAAI Press, Menlo Park, California, pp. 440–446.

Shakya, S. K. (2006). *DEUM: A framework for an Estimation of Distribution Algorithm based on Markov Random Fields*, PhD thesis, The Robert Gordon University, Aberdeen, UK.

Shakya, S. K., Brownlee, A. E. I., McCall, J. A. W., Fournier, F. & Owusu, G. (2009). A fully multivariate DEUM algorithm, *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2009)*, IEEE Press.

Shakya, S. K. & McCall, J. A. W. (2007). Optimization by estimation of distribution with DEUM framework based on Markov random fields, *International Journal of Automation and Computing* **4**(3): 262–272.

Shakya, S. K., McCall, J. A. W. & Brown, D. F. (2004a). Preliminary results on evolution without selection, *PREP 2004*.

Shakya, S. K., McCall, J. A. W. & Brown, D. F. (2004b). Updating the probability vector using MRF technique for a univariate EDA, *Proceedings of STAIRS 2004*, IOS Press, pp. 15–25.

Shakya, S. K., McCall, J. A. W. & Brown, D. F. (2005a). Estimating the distribution in an EDA, *Proceedings of the International Conference on Adaptive and Natural computiNG Algorithms (ICANNGA 2005)*, Springer-Verlag, pp. 202–205.

Shakya, S. K., McCall, J. A. W. & Brown, D. F. (2005b). Incorporating a Metropolis method in a distribution estimation using Markov random field algorithm, *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2005)*, IEEE Press, pp. 2576–2583.

Shakya, S. K., McCall, J. A. W. & Brown, D. F. (2005c). Using a Markov network model in a univariate EDA: an empirical cost-benefit analysis, *Proceedings of the Genetic and Evolutionary Computation COnference (GECCO 2005)*, ACM Press, pp. 727–734.

Shakya, S. K., McCall, J. A. W. & Brown, D. F. (2006). Solving the Ising spin glass problem using a bivariate EDA based on Markov random fields, *Proceedings of the IEEE World Congress on Computational Intelligence (CEC 2006)*, IEEE Press.

Shakya, S. & Santana, R. (2008a). An EDA based on local Markov property and Gibbs sampling, *Proceedings of the Genetic and Evolutionary Computation COnference (GECCO 2008)*, ACM Press, pp. 475–476.

Shakya, S. & Santana, R. (2008b). A Markovianity based optimisation algorithm, *Technical Report EHU-KZAA-IK-3/08*, Department of Computer Science and Artificial Intelligence, University of the Basque Country.

Shannon, C. E. (1948). A mathematical theory of communication, *Bell Syst. Tech. J.* **27**(3): 379–423.

Smith, R. E., Dike, B. A. & Stegmann, S. A. (1995). Fitness inheritance in genetic algorithms, *SAC '95: Proceedings of the 1995 ACM symposium on Applied computing*, ACM Press, New York, NY, USA, pp. 345–350.

Sun, J., Zhang, Q., Li, J. & Yao, X. (2008). A hybrid estimation of distribution algorithm for cdma cellular system design, *International Journal of Computational Intelligence and Applications* **7**(2): 187–200.

Thierens, D. (1999a). Estimating the significant non-linearities in the genome problem-coding, *Technical Report UU-CS-1999-47*, Department of Information and Computing Sciences, Utrecht University.
URL: *http://www.cs.uu.nl/research/techreps/UU-CS-1999-47.html*

Thierens, D. (1999b). Estimating the significant non-linearities in the genome problem coding, *Proceedings of the Genetic and Evolutionary Computation COnference (GECCO 1999)*, Morgan Kaufmann, San Fransisco, pp. 643–648.

Vose, M. D. (1999). *The simple genetic algorithm: foundations and theory*, MIT Press, Cambridge, MA.

Wagner, M. R., Auger, A. & Schoenauer, M. (2004). Eeda: A new robust estimation of distribution algorithm, *Technical Report RR-5190*, INRIA.

Witten, I. H. & Frank, E. (2005). *Data Mining: Practical Machine Learning Tools and Techniques*, Morgan Kaufmann Series in Data Management Sys, second edn, Morgan Kaufmann.

Wright, A. H. & Pulavarty, S. (2005). On the convergence of an estimation of distribution algorithm based on linkage discovery and factorization, *Proceedings of the Genetic and Evolutionary Computation COnference (GECCO 2005)*, ACM Press, pp. 695–702.

Wu, Y., McCall, J., Godley, P., Brownlee, A. & Cairns, D. (2008). Bio-control in mushroom farming using a Markov network EDA, *Proceedings of the IEEE World Congress on Computational Intelligence (CEC 2008)*, pp. 2996–3001.

Zhang, Q. (2001). *Advances in Fuzzy Systems and Evolutionary Computation*, World Sci. and Eng. Press.

Zhang, Q. (2004). On stability of fixed points of limit models of univariate marginal distribution algorithm and factorized distribution algorithm, *IEEE Transactions on Evolutionary Computation* **8**(1): 80–93.

Zhang, Q. & Sun, J. (2006). Iterated local search with guided mutation, *Proceedings of the IEEE World Congress on Computational Intelligence (CEC 2006)*, IEEE Press, pp. 924 – 929.

Zhang, Q., Sun, J. & Tsang, E. (2005). An evolutionary algorithm with guided mutation for the maximum clique problem, *IEEE Transactions on Evolutionary Computation* **9**(2): 192–200.

Zhang, Q., Sun, J. & Tsang, E. (2007). Combinations of estimation of distribution algorithms and other techniques, *International Journal of Automation & Computing* pp. 273–280.

Zhang, Q., Sun, J., Tsang, E. & Ford, J. (2004). Hybrid estimation of distribution algorithm for global optimization, *Engineering Computations* **21**(1): 91.

Zhang, Q., Sun, J., Tsang, E. P. K. & Ford, J. (2003). Combination of guided local search and estimation of distribution algorithm for solving quadratic assignment problem, *Proceedings of the Genetic and Evolutionary Computation COnference (GECCO 2003)*, ACM Press, New York, NY, USA, pp. 42–48.

Zhou, Z., Ong, Y. S., Nguyen, M. H. & Lim, D. (2005). A study on polynomial regression and Gaussian process global surrogate model in hierarchical surrogate-assisted evolutionary algorithm, *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2005)*, Vol. 3, pp. 2832–2839 Vol. 3.

Appendix A

Terminology and Notation

Provided here for ease of reference are a few definitions which are found within the thesis.

A.1 Notation for Individuals and Fitness

Chromosome / individual A single member $x = x_1, x_2 ... x_n$ of the population in an evolutionary algorithm, representing a set of values for the n variables in the problem

Evolutionary Algorithm An algorithm which solves a problem using a population of solutions that evolve over time

Fitness function A means by which an evolutionary algorithm can evaluate individuals within a population; applies a fitness $f(x)$ to an individual x

Search space The set of all possible individual solutions for a given fitness function

Stopping Criteria An algorithm can terminate in a number of ways and thus those described simply refer to "stopping criteria". This may be when a solution with a fitness value equal to the known optimum is found, the population has converged on a particular point, a certain number of function evaluations has been performed or some other threshold has been reached.

Variable A component part of an individual which can take on different values - represented in general by X_i and in a specific case by x_i

A.2 Probability and Information Theory

Chi square test An measure of independence between two variables. For two variables X_i and X_j, calculated over all possible values x_i and x_j: $X^2_{i,j} = \sum_{x_i,X_j} \frac{(p(x_i,x_j)-p(x_i)p(x_j))^2}{p(x_i)p(x_j)}$. A typical threshold would be 3.84, where the variables are said to be 95% independent. See: (Marascuilo & McSweeney 1977)

Conditional Probability $p(x_i|x_j)$ The probability of x_i given that the value of x_j is known; a separate probability must be stored for each different values which x_j may take.

Kullback Liebler Divergence Also known as the relative entropy; a measure of the distance between an estimated distribution and the true distribution. See: (Kullback 1987)

Marginal entropy of variable x_i $H(x) = \sum_{x=1}^{n} p(x_i) log_b p(x_i)$

Marginal Probability The probability of variable x_i taking the value v is written as $p(x_i = v)$

A.3 Terms specific to MFM or experiments in this thesis

Bottom selection A variant of top selection, which selectes the least fit m individuals in a population (See section 5.1)

C_m FPC for a population of mutated individuals (See Section 3.2)

C_r FPC for a population of randomly generateted individuals (See Section 3.2)

Decimated model structure The perfect structure with a fixed proportion of cliques chosen atrandom and removed (See Section 3.1.4)

F-measure A combination of precision and recall (See Section 7.1)

Filtered model structure The perfect structure with all cliques of a specific order removed (See Section 3.1.4)

FPC Fitness Prediction Correlation; the correlation between a population's true fitnesses and fitnesses predicted by a MFM (See Section 3.2)

Full model structure The set of all possible cliques and subcliques up to certain dimension (See Section 3.1.4)

Imperfect model structure The perfect structure with some cliques added or removed (See Section 3.1.4)

Over-specified The situation where the number of individuals used to estimate model parameters is greater than the number of parameters 3.1.5

Perfect structure The set of all cliques and subcliques known to be present in the fitness function (See Section 3.1.4)

Precisely-specified The situation where the number of individuals used to estimate model parameters is equal to the number of parameters 3.1.5

Precision The proportion of interactions found which are true interactions (See Section 7.1)

Recall The proportion of true interactions which have been learned (See Section 7.1)

Top & Bottom selection A combination of top selection and bottom selection, which selects the fittest $m/2$ and least fit $m/2$ individuals in a population (See section 5.1)

Top selection The standard truncation selection operator, which selectes the fittest m individuals in a population (See section 5.1)

Under-specified The situation where the number of individuals used to estimate model parameters is less than the number of parameters 3.1.5

A.4 Algorithm Abbreviations

The Chapter in which to find a detailed described is given after the algorithm's name.

BOA Bayesian Optimisation Algorithm (Chapter 2)

cGA compact Genetic Algorithm (Chapter 2)

COMIT Combining Optimizers with Mutual Information Trees (Chapter 2)

DEUM Distribution Estimation Using Markov Random Fields / Markov Networks (Chapter 2 and (Chapter 6))

DEUM-χ^2 Distribution Estimation Using Markov Networks with χ^2 Structure Learning (Chapter 7)

DEUM-χ^2-Chain Distribution Estimation Using Markov Networks with χ^2 Chain Learning (Chapter 7)

DEUM-Chain Distribution Estimation Using Markov Networks with Chain Structure (Chapter 7)

DEUM-LDA Evolutionary Distribution Estimation Using Markov Networks with Linkage Detection Algorithm (Chapter 7)

evDEUM-χ^2 Evolutionary Distribution Estimation Using Markov Networks with χ^2 Structure Learning (Chapter 7)

FDA Factorised Distribution Algorithm (Chapter 2)

GA Genetic algorithm (Chapter 2)

hBOA hierarchical Bayesian Optimisation Algorithm (Chapter 2)

iBOA incremental Bayesian Optimisation Algorithm (Chapter 2)

Ising-DEUM Ising Distribution Estimation Using Markov Random Fields / Markov Networks (Chapter 2)

LDA Linkage Detection Algorithm

LFDA Learning Factorised Distribution Algorithm (Chapter 2)

MIMIC Mutual Information Maximisation by Input Clustering (Chapter 2)

MN-EDA Markov Network Estimation of Distribution Algorithm (Chapter 2)

MN-FDA Markov Network Factorised Distribution Algorithm (Chapter 2)

MOA Markovianity Optimisation Algorithm (Chapter 2)

PBIL Population Based Incremental Learning (Chapter 2)

UMDA Univariate Marginal Distribution Algorithm (Chapter 2)

Appendix B

Figures - Population Size Experiments

The figures in this chapter relate to the experiments discussed in Chapter 4.

Figure B.1: FPC vs population size for 10 bit onemax

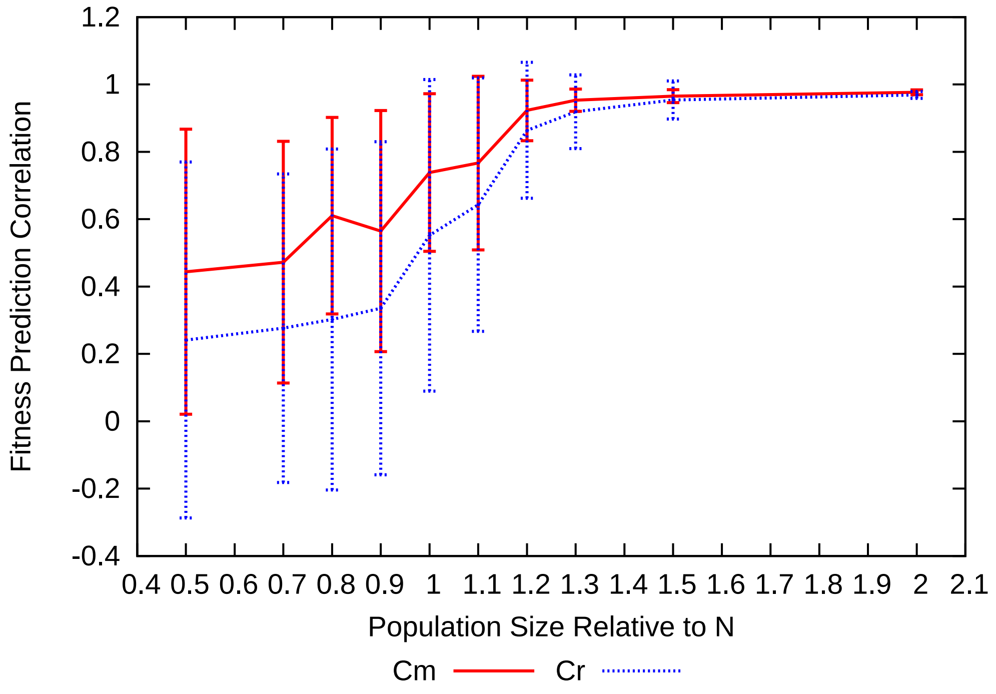
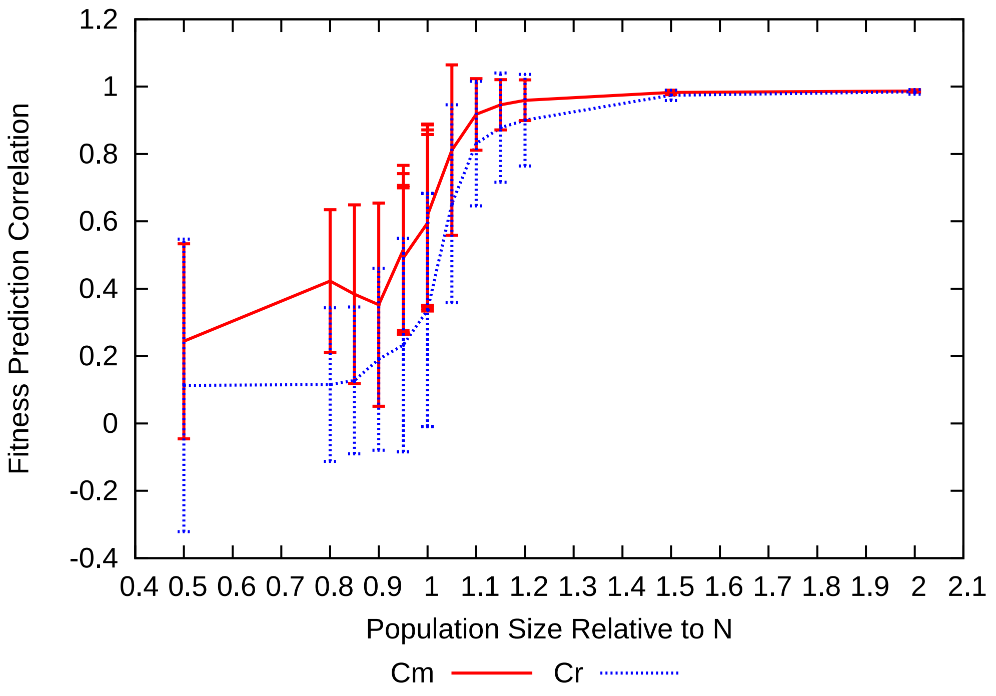


Figure B.2: FPC vs population siz fore 20 bit onemax



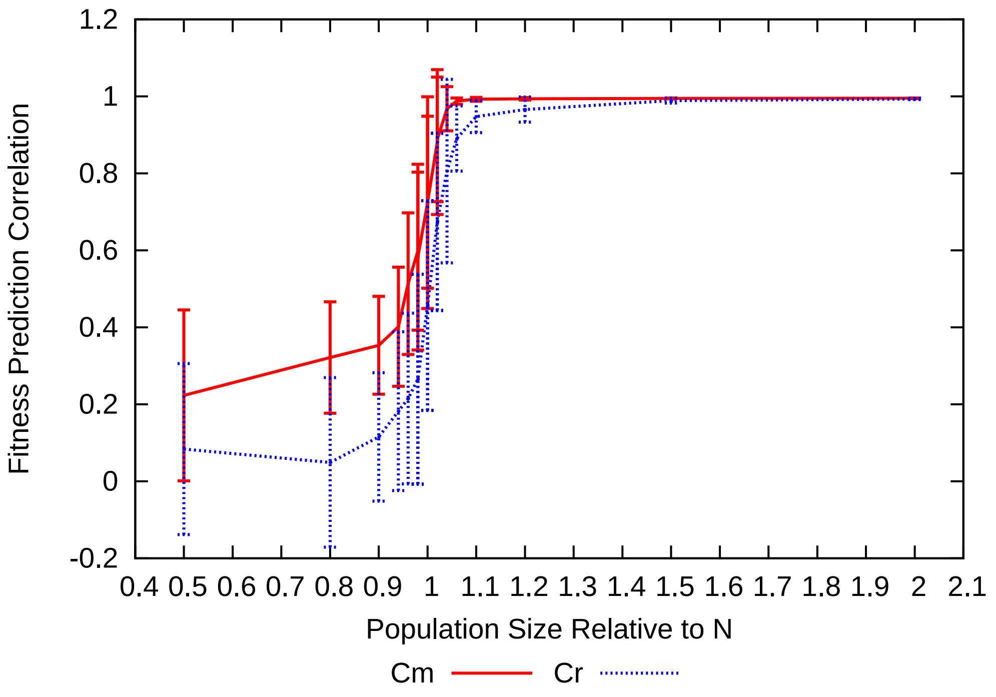


Figure B.3: FPC vs population size for 50 bit onemax

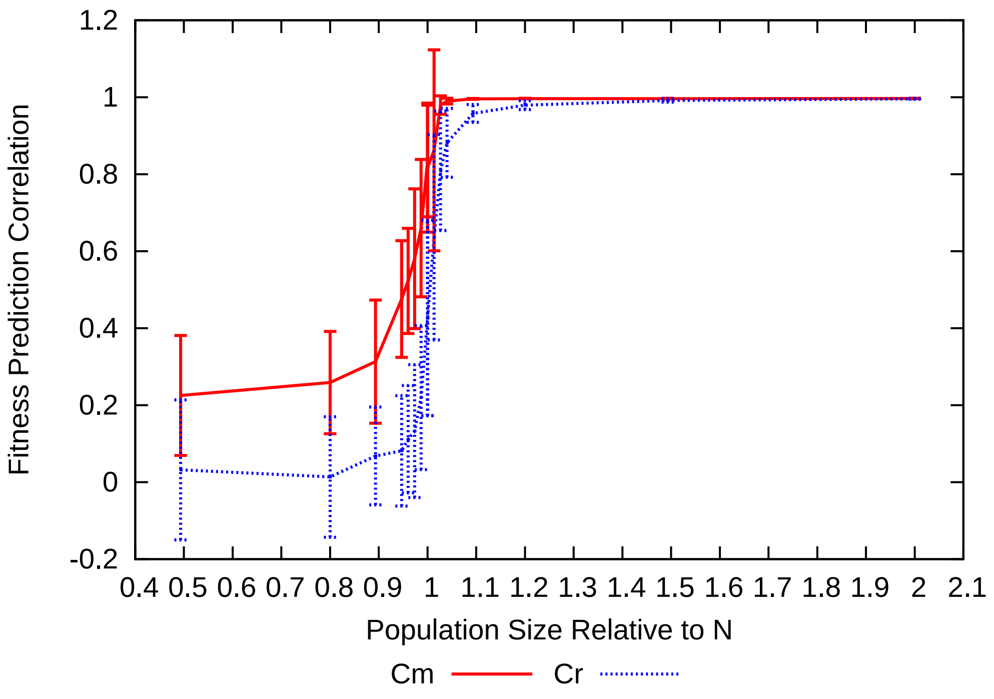


Figure B.4: FPC vs population size for 75 bit onemax

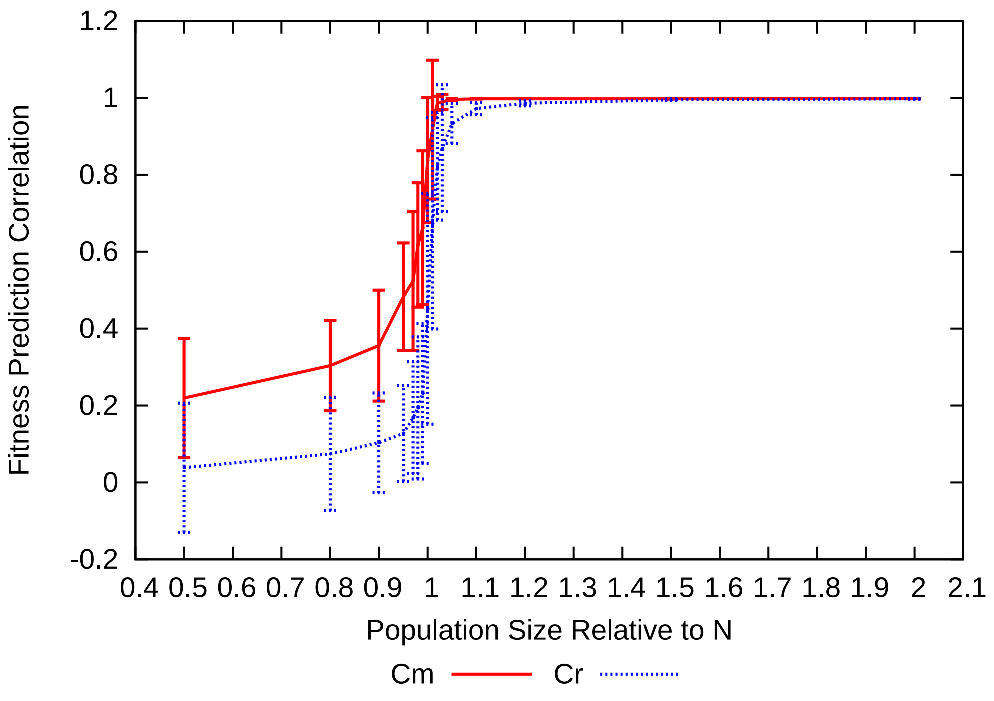


Figure B.5: FPC vs population size for 100 bit onemax

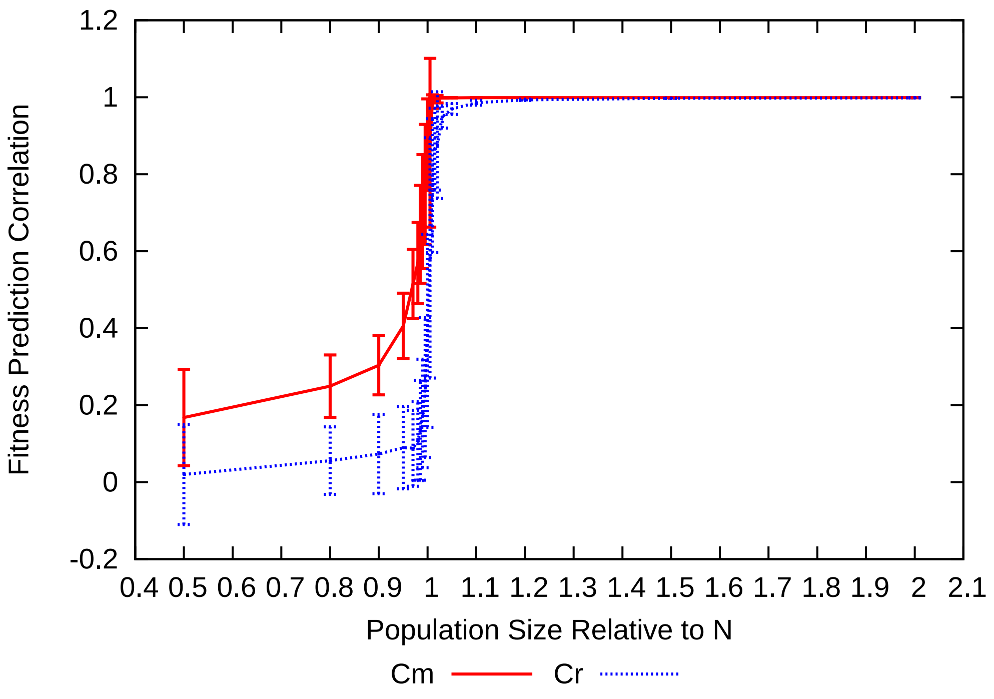


Figure B.6: FPC vs population size for 200 bit onemax

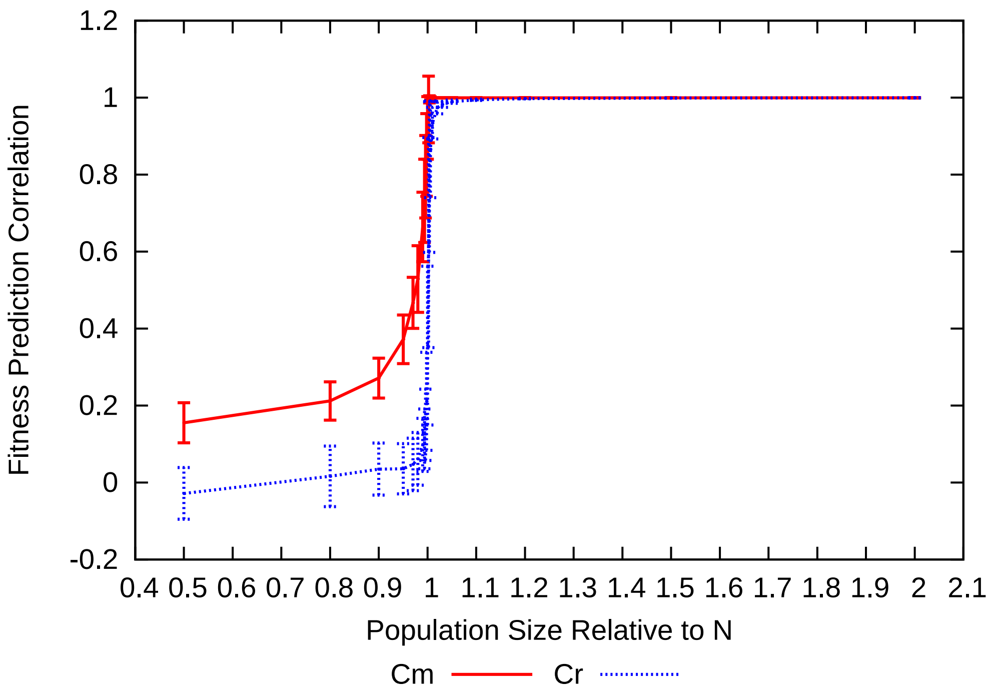


Figure B.7: FPC vs population size for 500 bit onemax

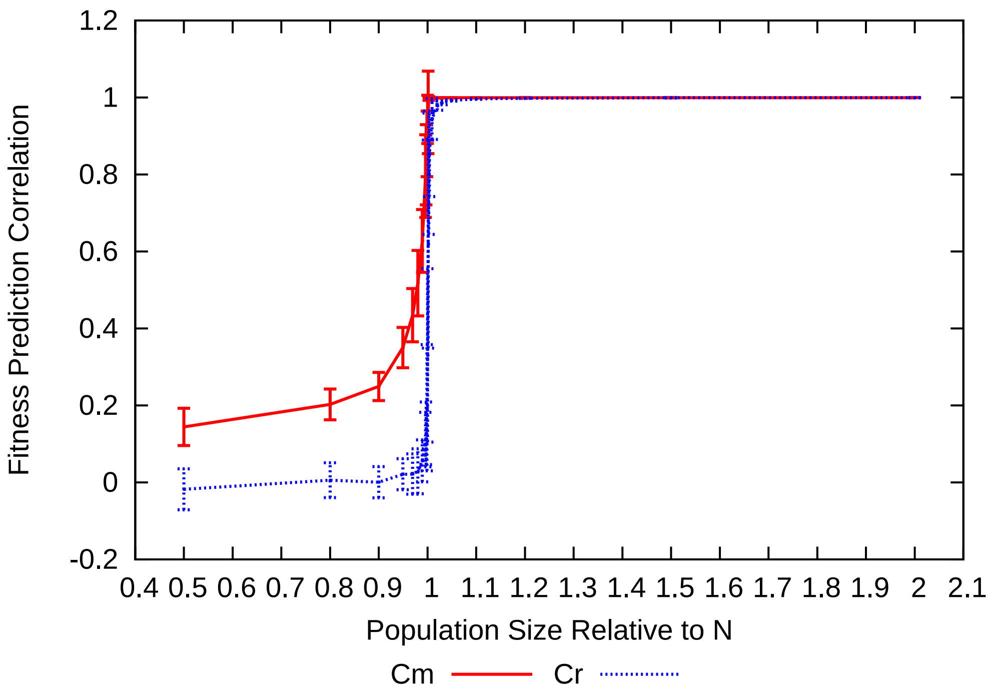


Figure B.8: FPC vs population size for 750 bit onemax

Figure B.9: FPC vs population size for 1000 bit onemax

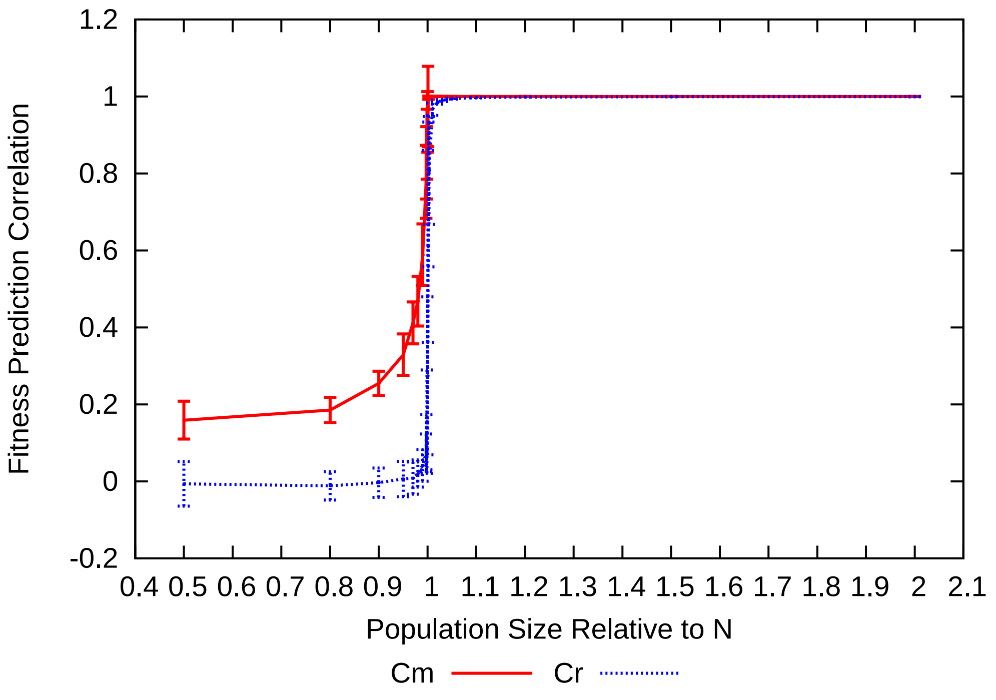
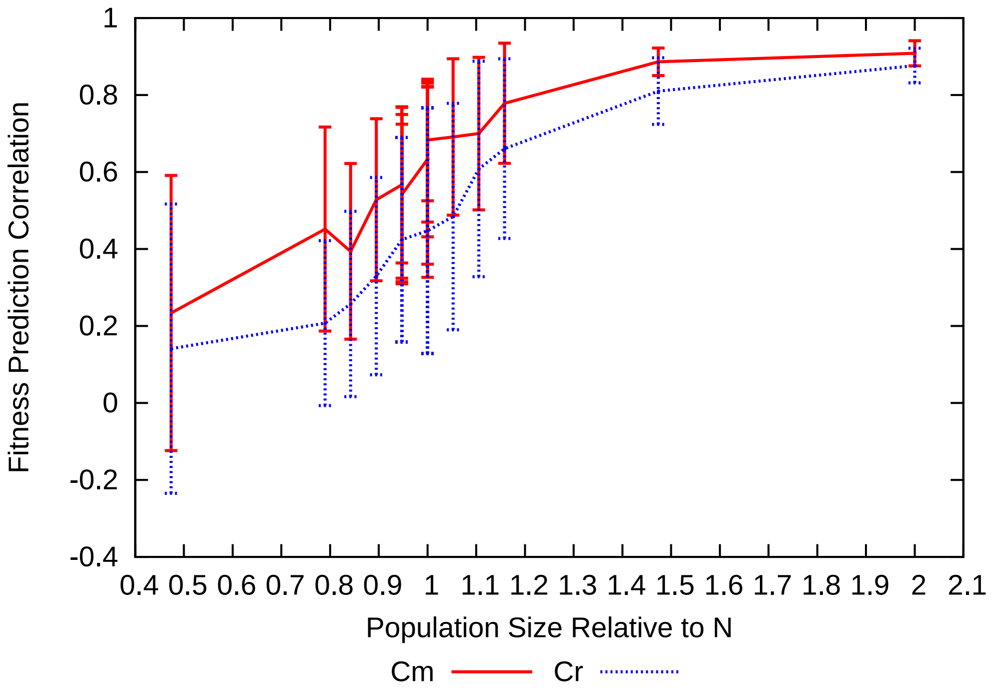


Figure B.10: FPC vs population size for 10 bit 1D Checkerboard problem using a perfect model structure



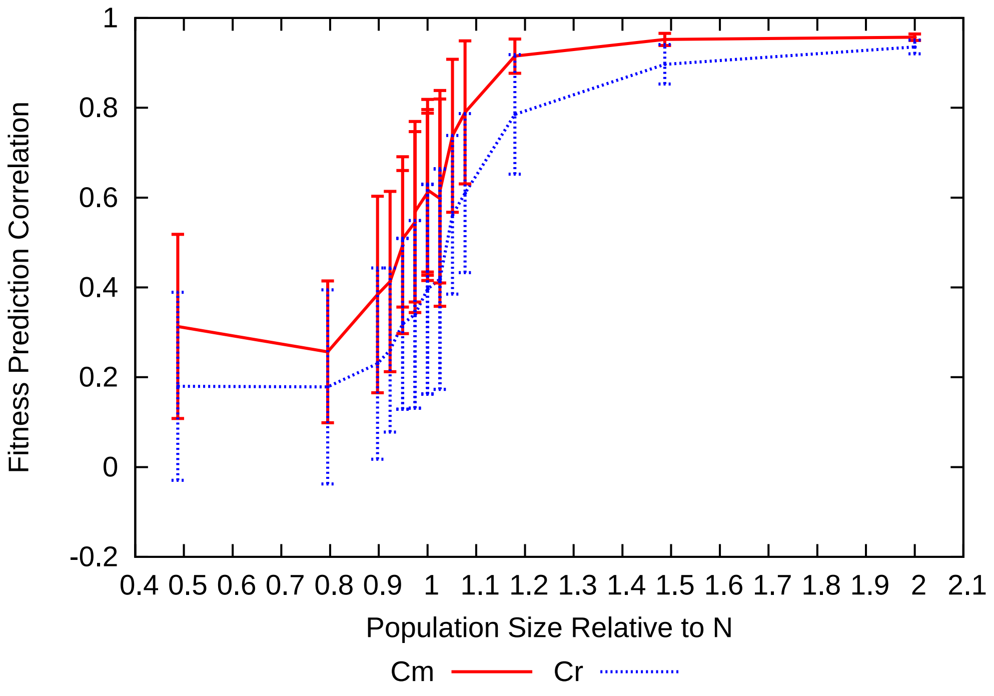


Figure B.11: FPC vs population size for 20 bit 1D Checkerboard problem using a perfect model structure

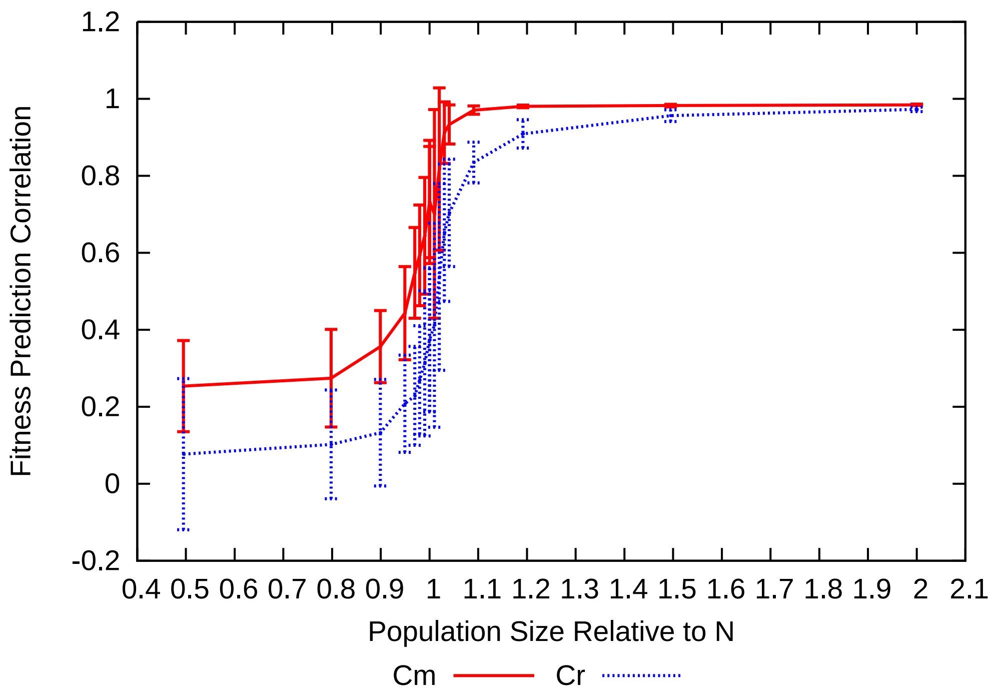


Figure B.12: FPC vs population size for 50 bit 1D Checkerboard problem using a perfect model structure

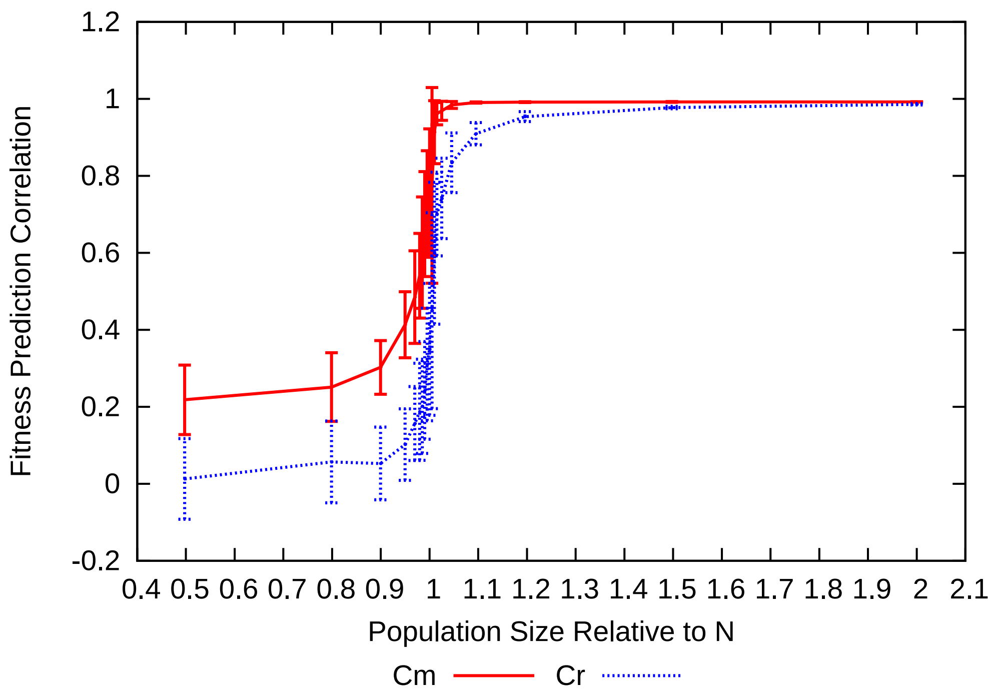


Figure B.13: FPC vs population size for 100 bit 1D Checkerboard problem using a perfect model structure

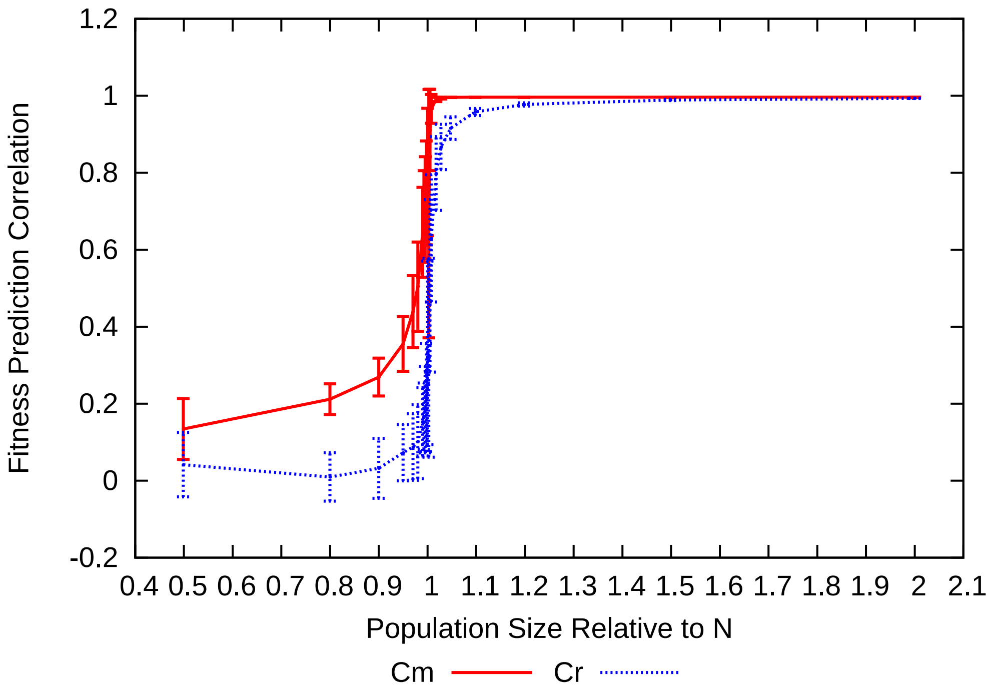


Figure B.14: FPC vs population size for 200 bit 1D Checkerboard problem using a perfect model structure

Figure B.15: FPC vs population size for 500 bit 1D Checkerboard problem using a perfect model structure

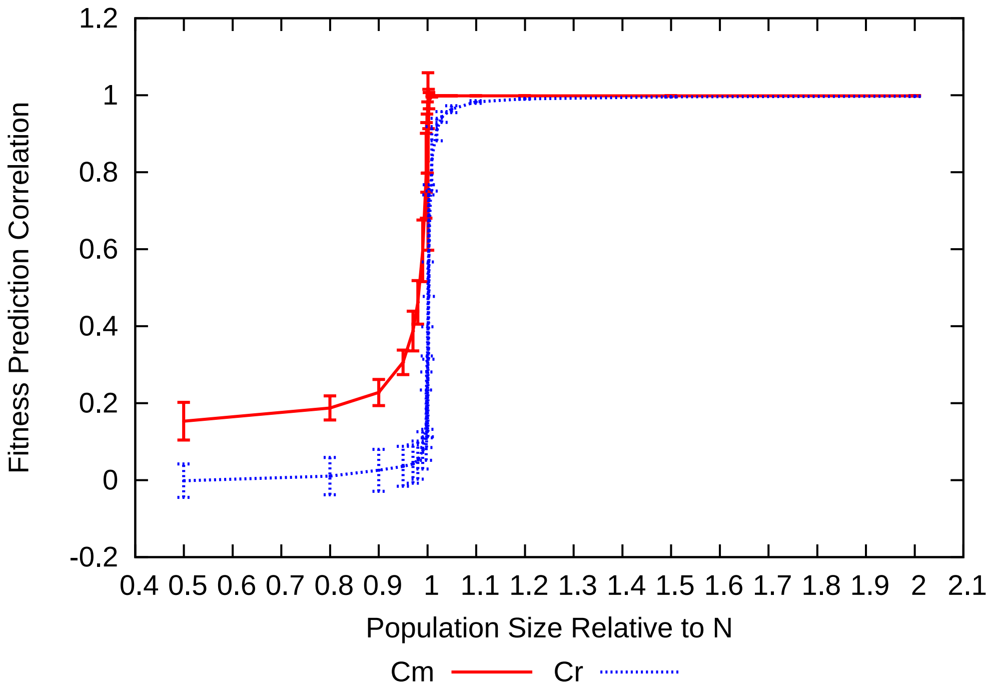
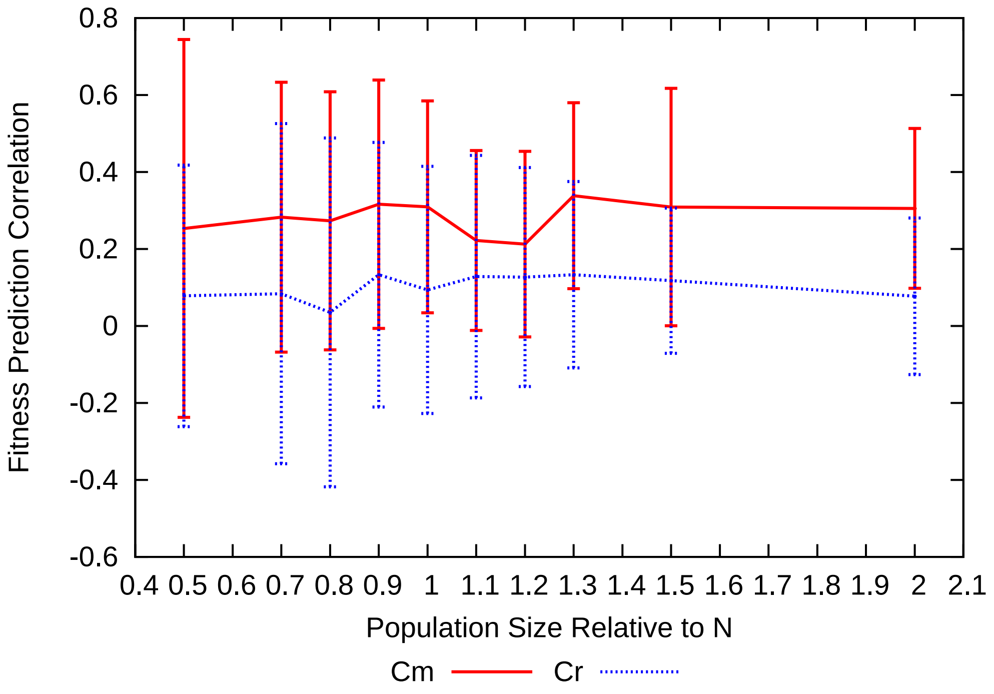


Figure B.16: FPC vs population size for 10 bit 1D Checkerboard problem using a model with only univariate terms



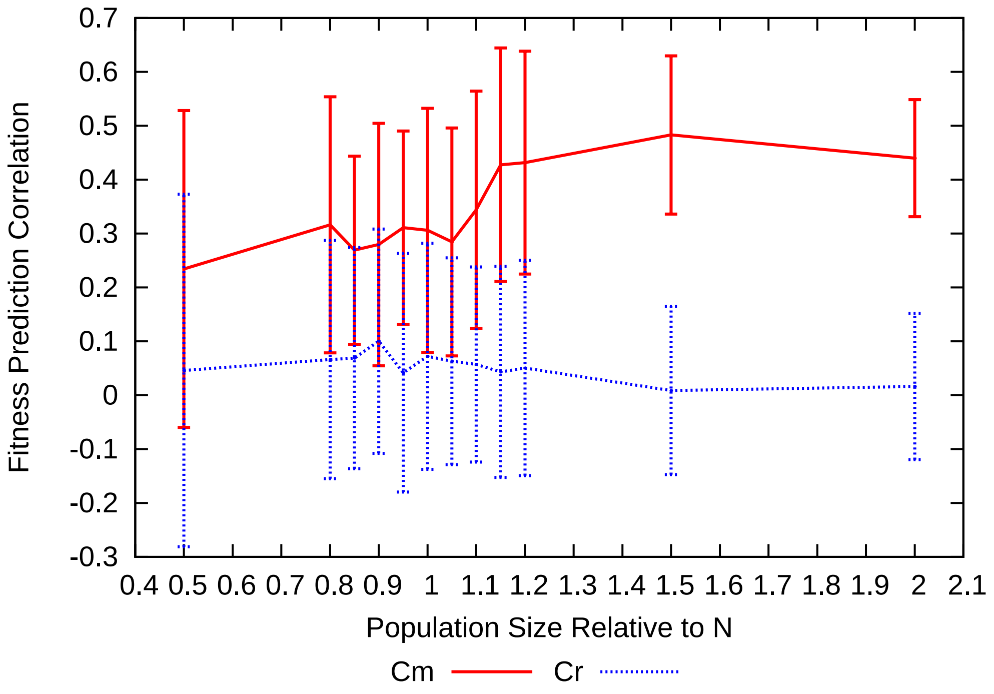


Figure B.17: FPC vs population size for 20 bit 1D Checkerboard problem using a model with only univariate terms

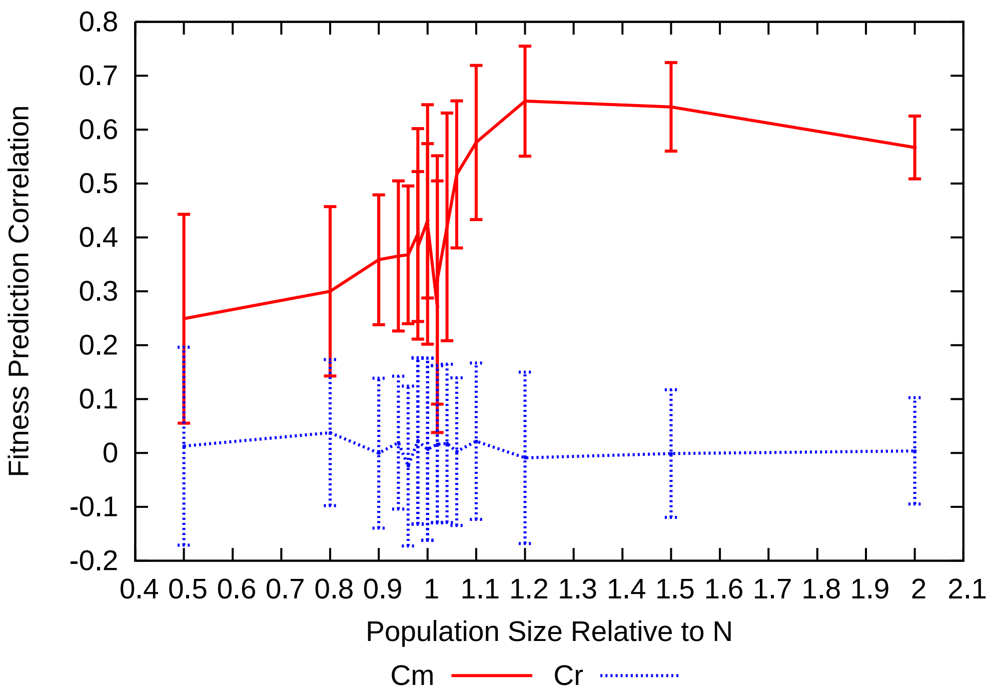


Figure B.18: FPC vs population size for 50 bit 1D Checkerboard problem using a model with only univariate terms

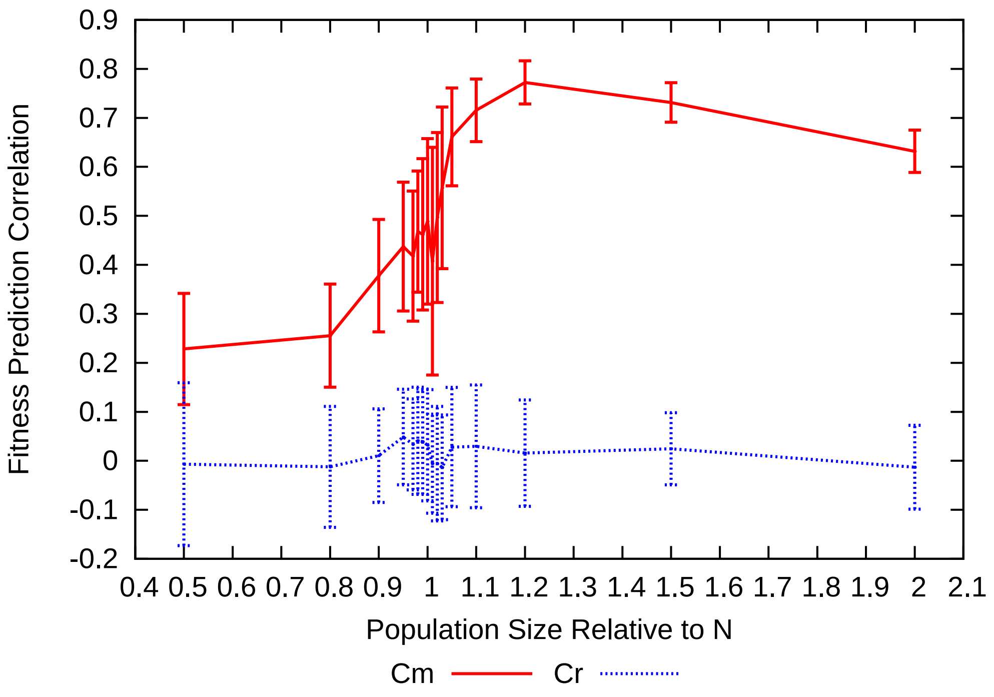


Figure B.19: FPC vs population size for 100 bit 1D Checkerboard problem using a model with only univariate terms

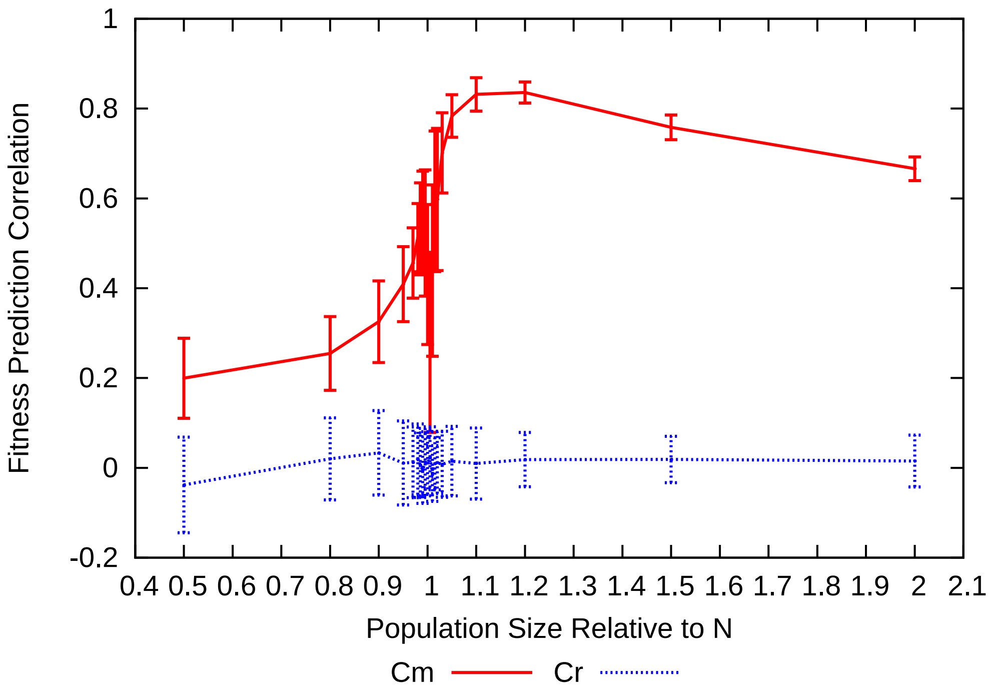


Figure B.20: FPC vs population size for 200 bit 1D Checkerboard problem using a model with only univariate terms

Figure B.21: FPC vs population size for 500 bit 1D Checkerboard problem using a model with only univariate terms

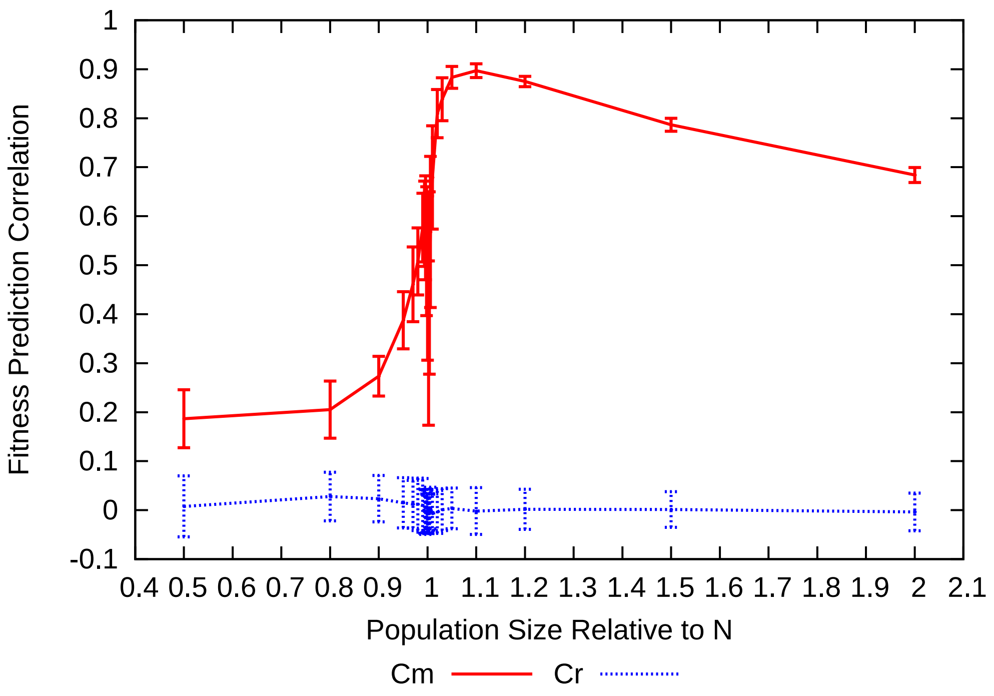
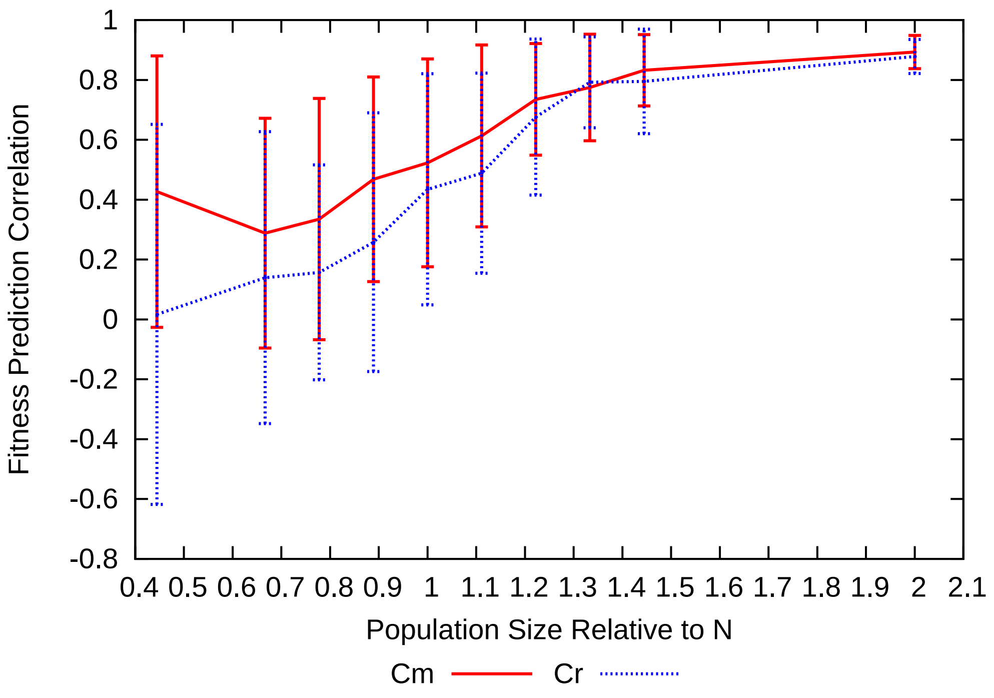


Figure B.22: FPC vs population size for 10 bit 1D Checkerboard problem using a model with only bivariate terms



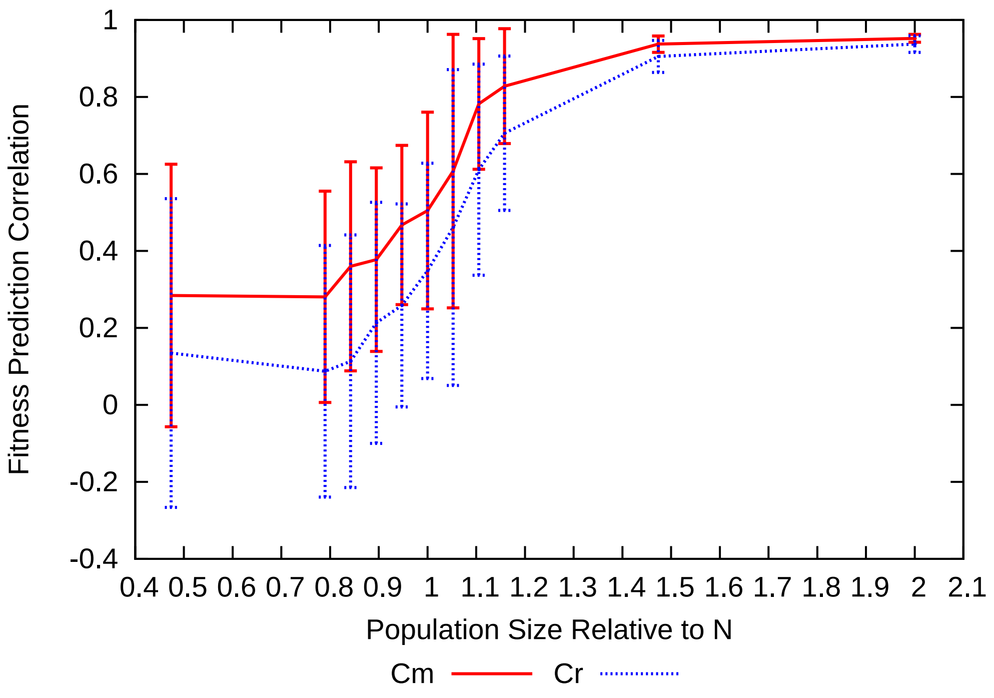


Figure B.23: FPC vs population size for 20 bit 1D Checkerboard problem using a model with only bivariate terms

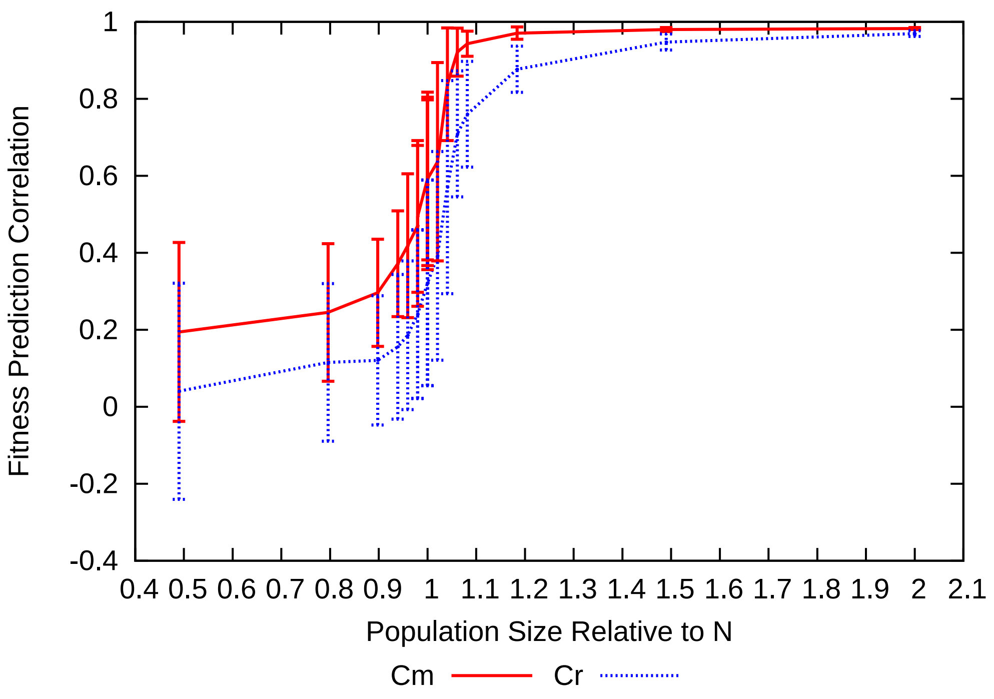


Figure B.24: FPC vs population size for 50 bit 1D Checkerboard problem using a model with only bivariate terms

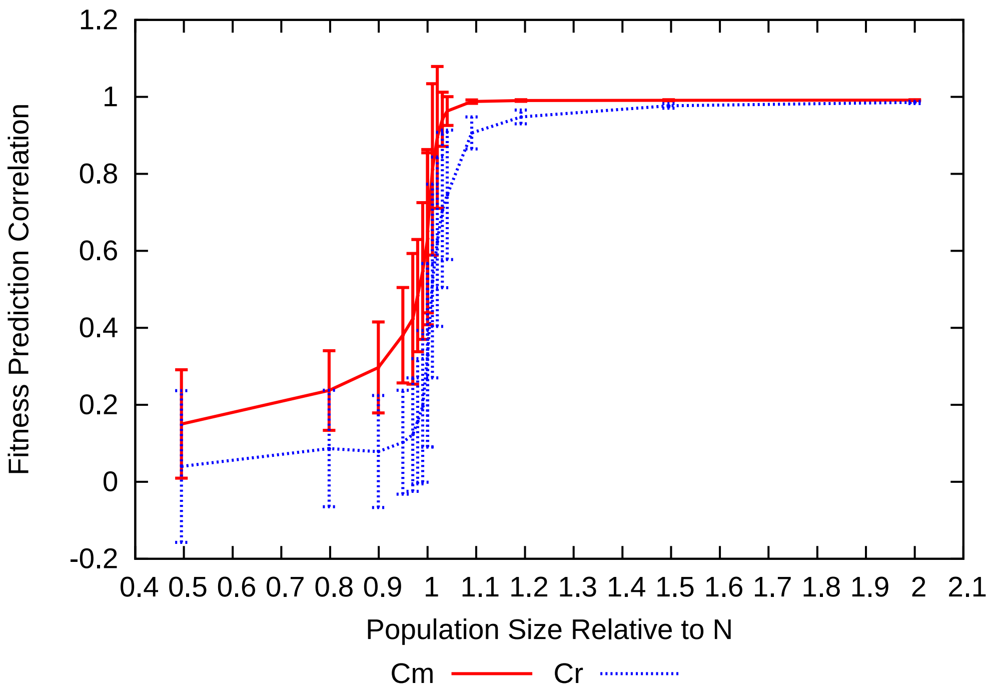


Figure B.25: FPC vs population size for 100 bit 1D Checkerboard problem using a model with only bivariate terms

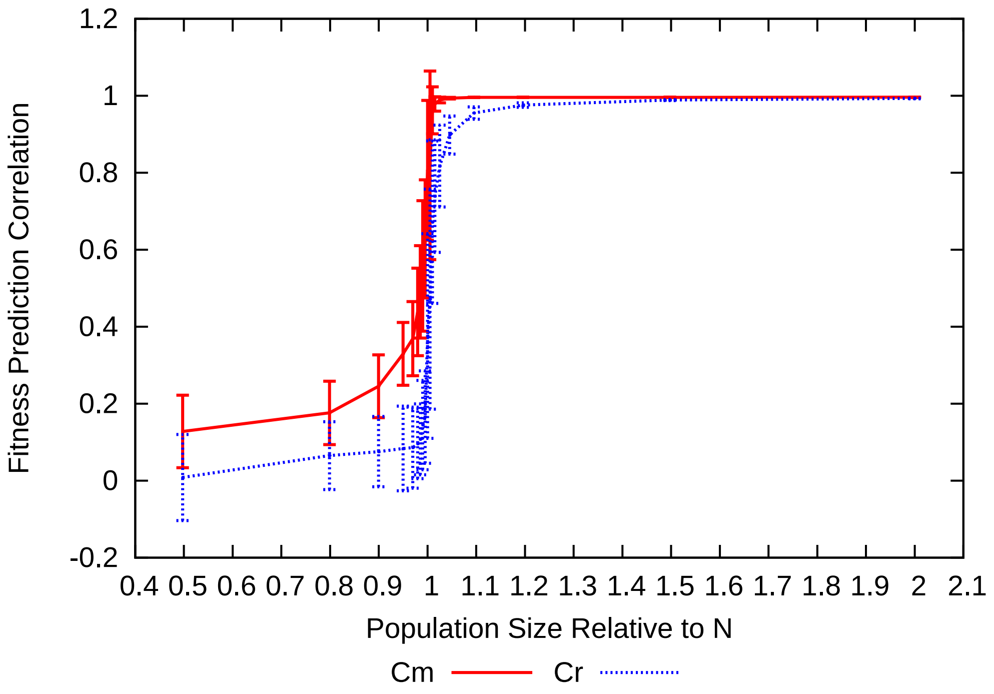


Figure B.26: FPC vs population size for 200 bit 1D Checkerboard problem using a model with only bivariate terms

Figure B.27: FPC vs population size for 500 bit 1D Checkerboard problem using a model with only bivariate terms

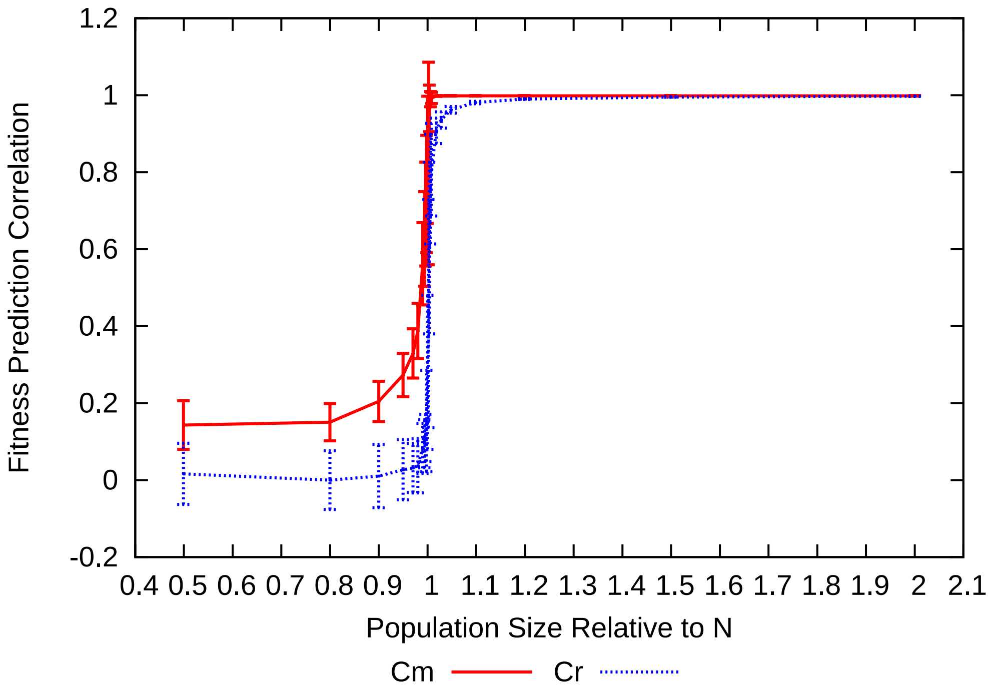
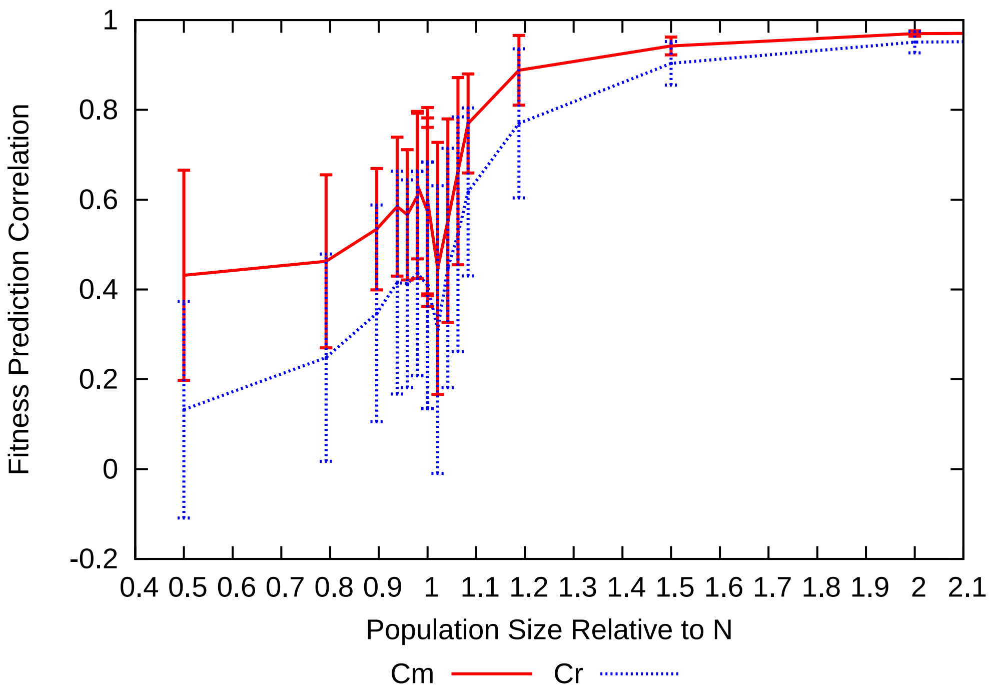


Figure B.28: FPC vs population size for 16 bit 2D Ising problem using a perfect model structure



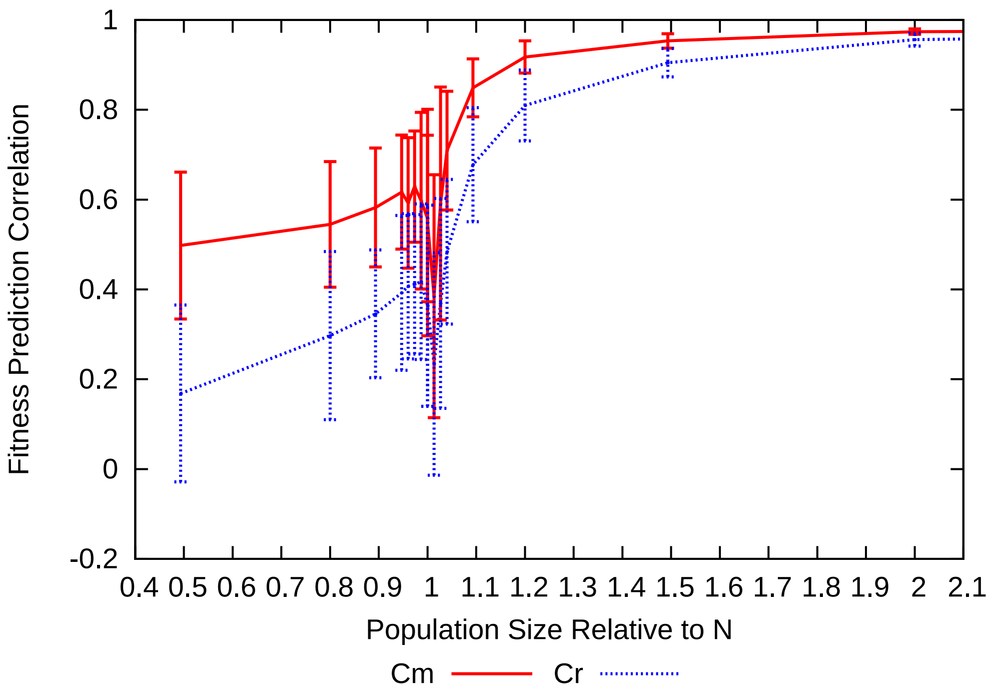


Figure B.29: FPC vs population size for 25 bit 2D Ising problem using a perfect model structure

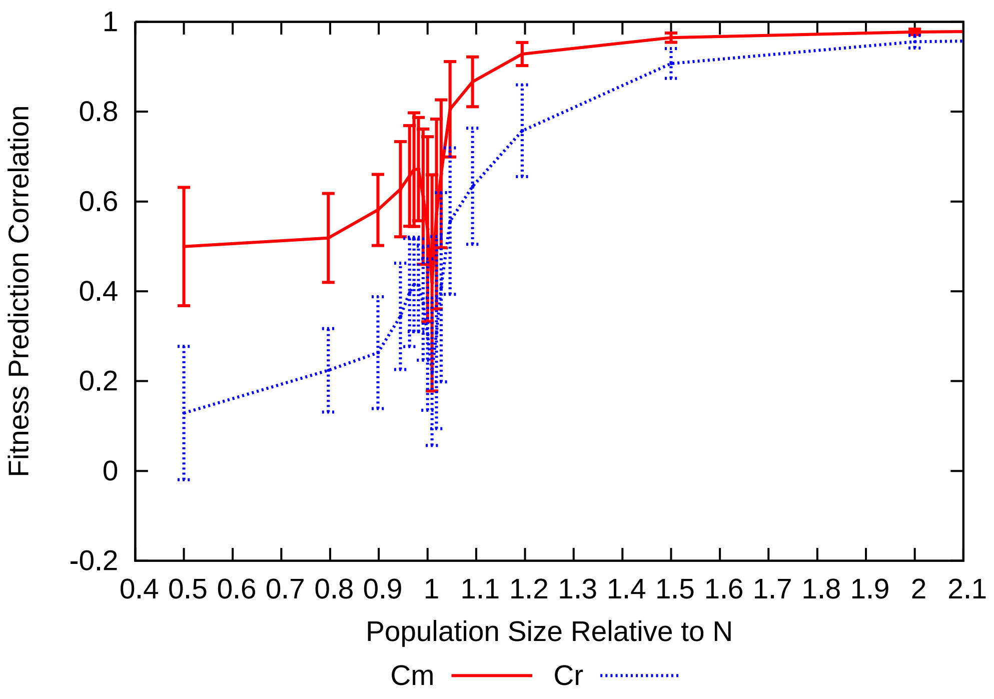


Figure B.30: FPC vs population size for 36 bit 2D Ising problem using a perfect model structure

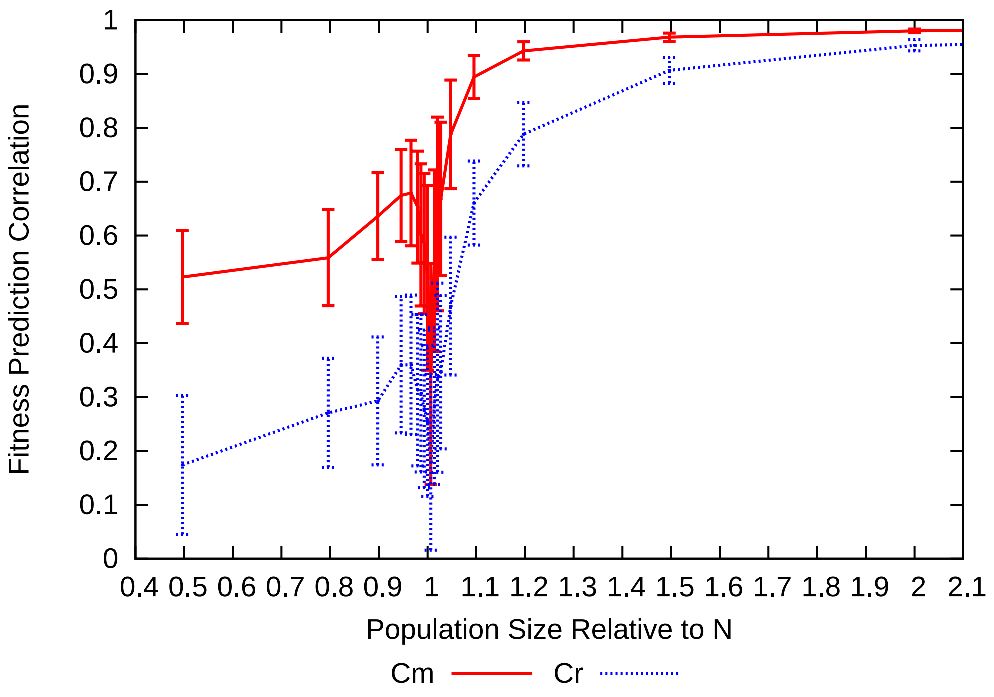


Figure B.31: FPC vs population size for 49 bit 2D Ising problem using a perfect model structure

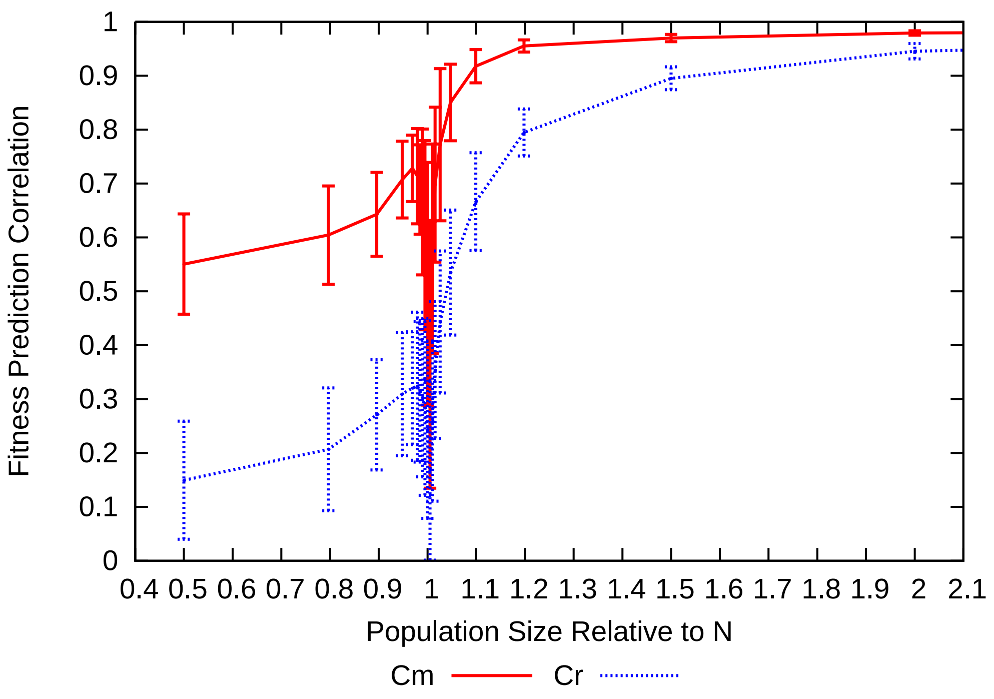


Figure B.32: FPC vs population size for 64 bit 2D Ising problem using a perfect model structure

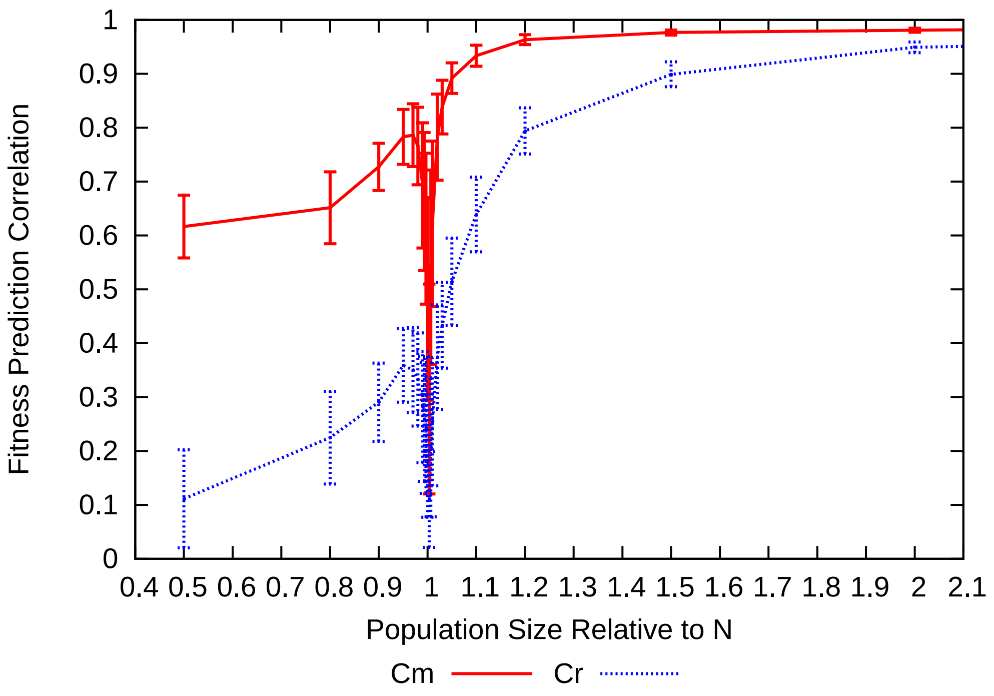


Figure B.33: FPC vs population size for 100 bit 2D Ising problem using a perfect model structure

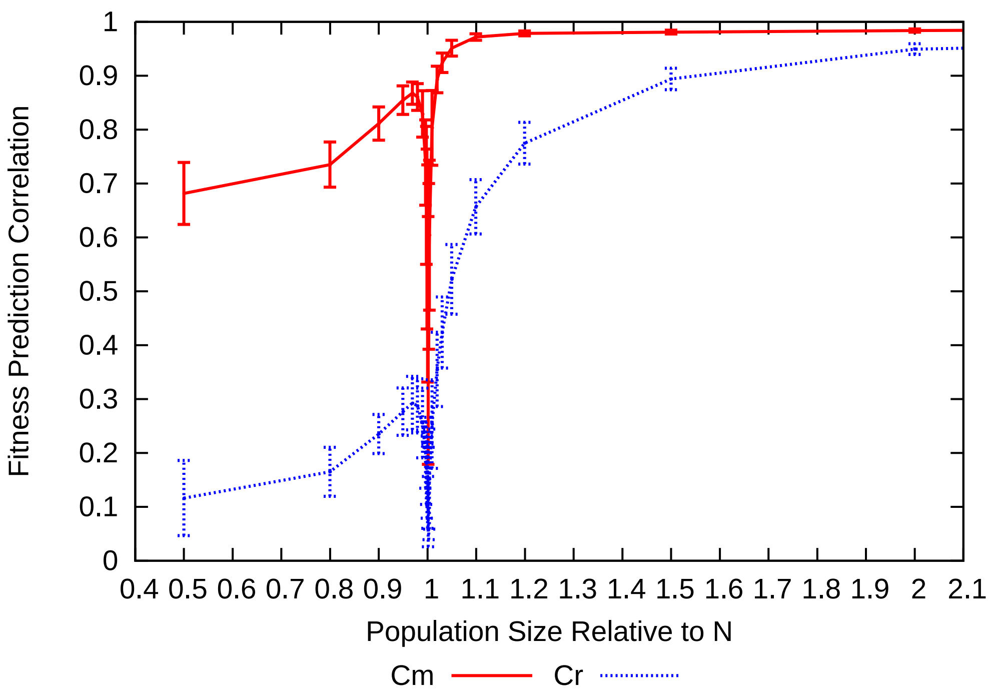


Figure B.34: FPC vs population size for 256 bit 2D Ising problem using a perfect model structure

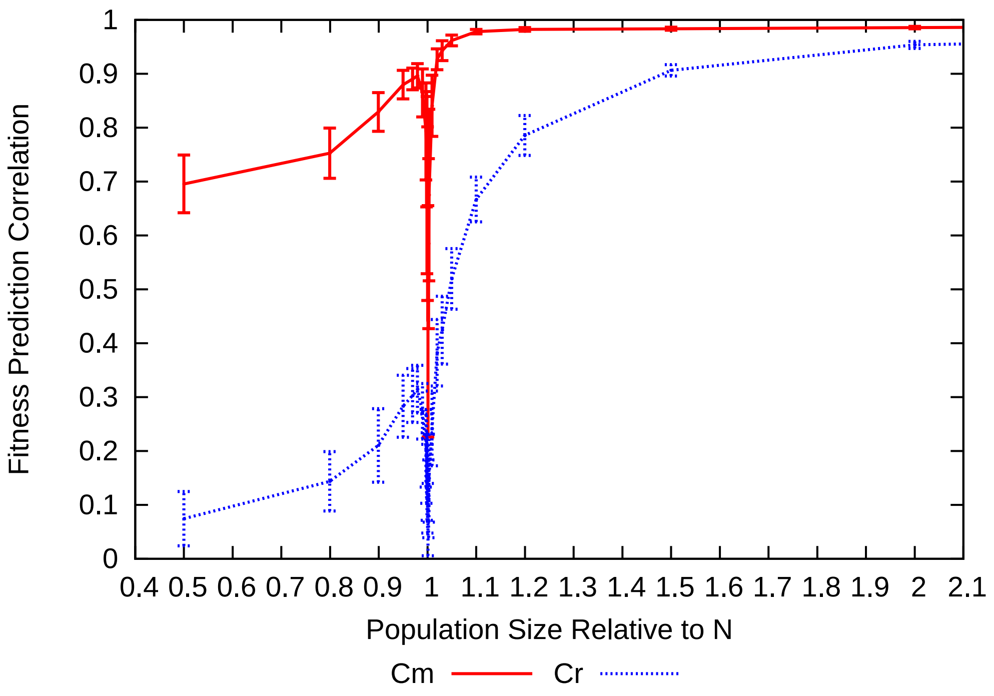


Figure B.35: FPC vs population size for 324 bit 2D Ising problem using a perfect model structure

Figure B.36: FPC vs population size for 400 bit 2D Ising problem using a perfect model structure

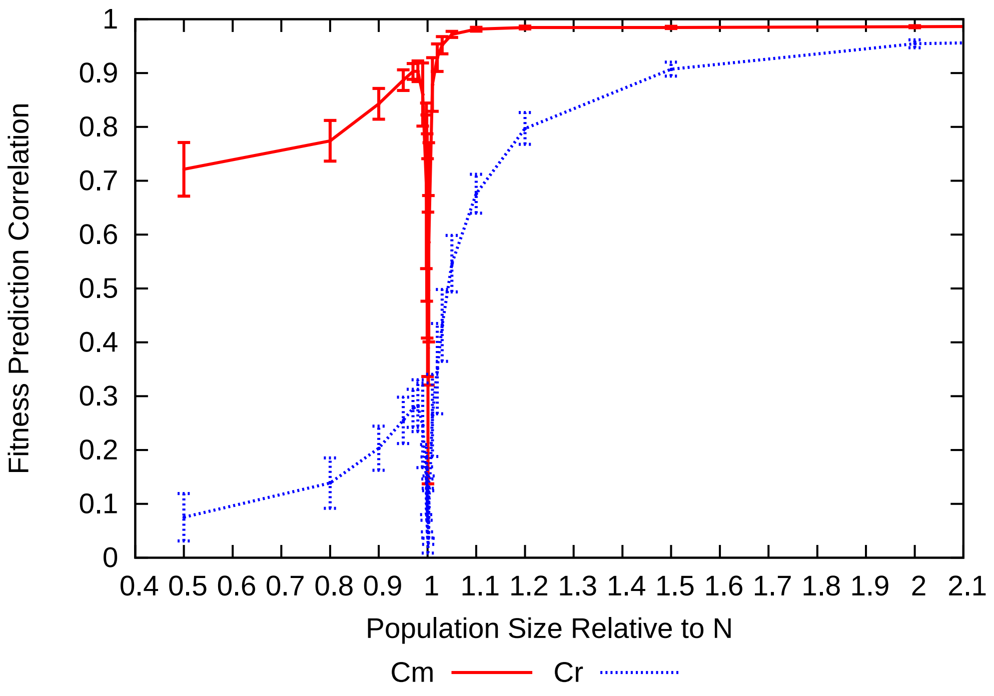
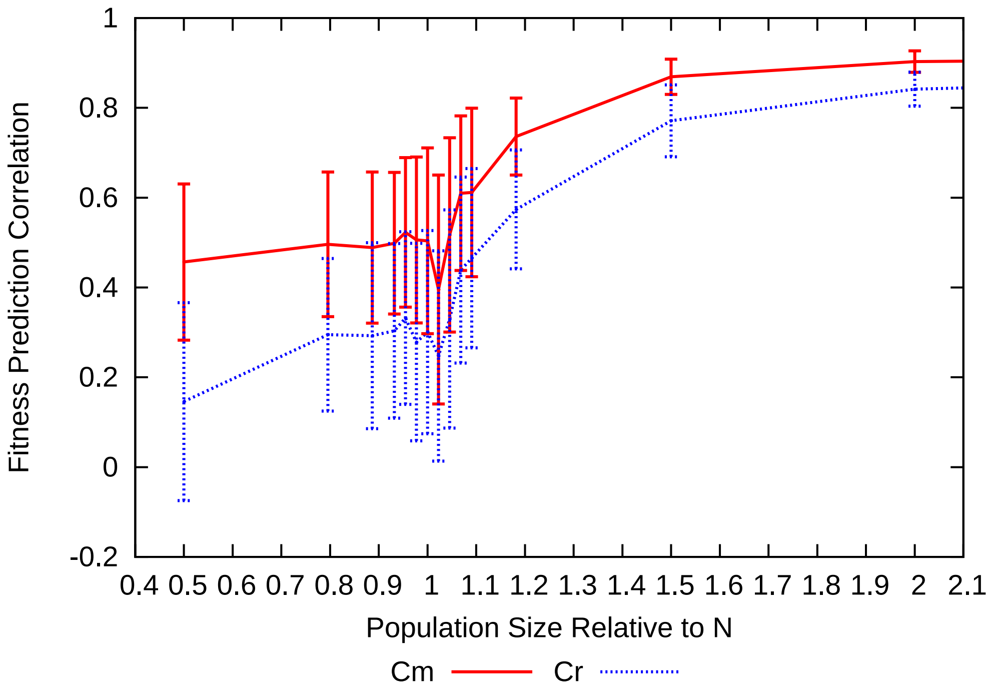


Figure B.37: FPC against population size for 16 bit 2D Ising problem using a 10% decimated model



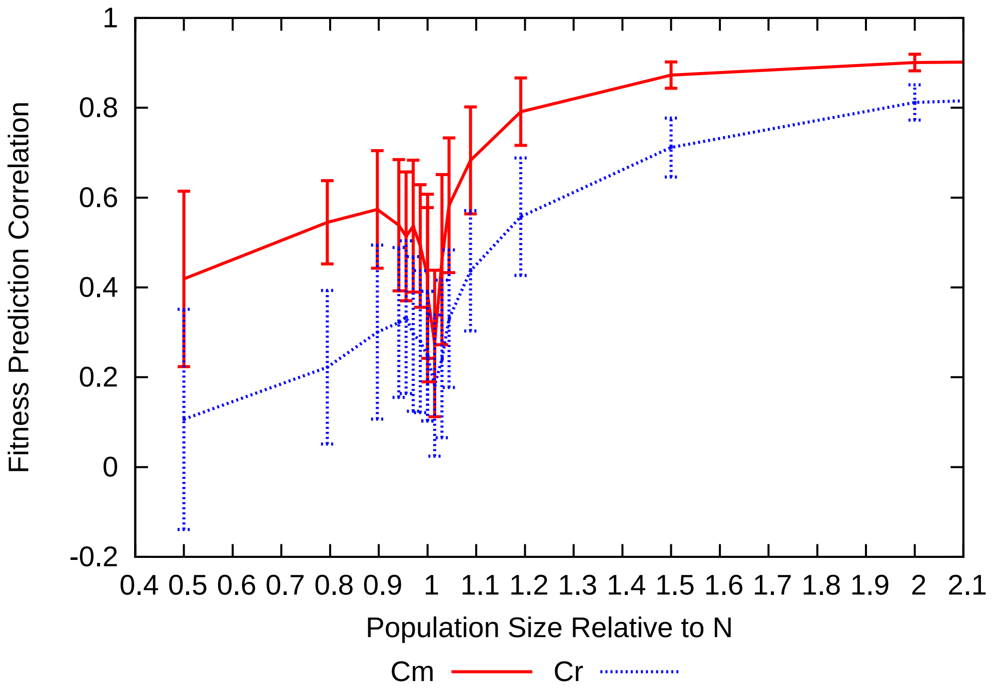


Figure B.38: FPC against population size for 25 bit 2D Ising problem using a 10% decimated model

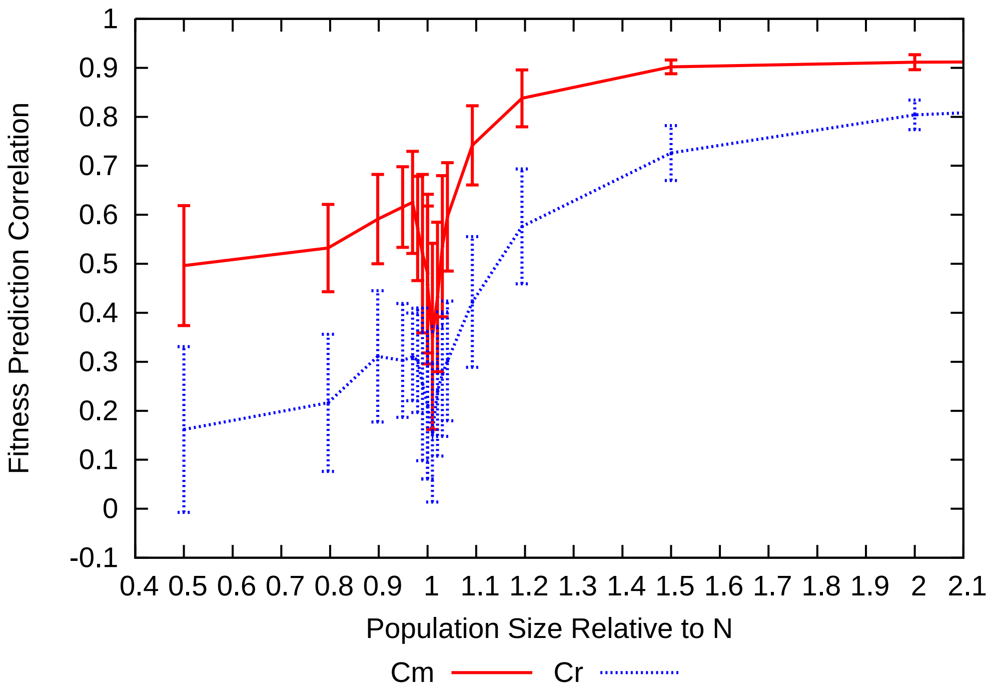


Figure B.39: FPC against population size for 36 bit 2D Ising problem using a 10% decimated model

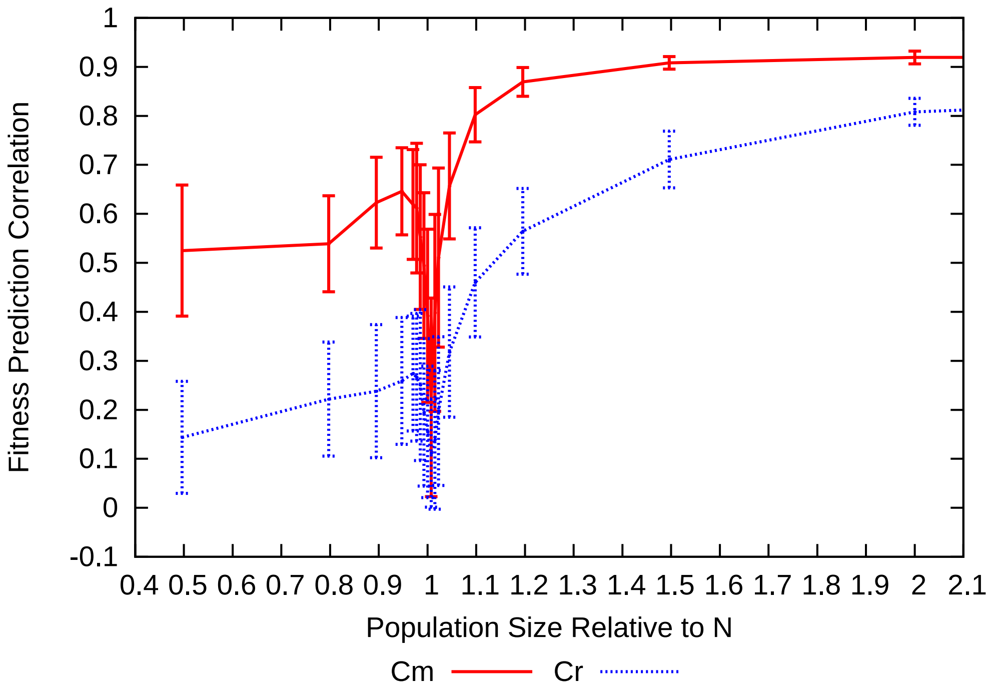


Figure B.40: FPC against population size for 49 bit 2D Ising problem using a 10% decimated model

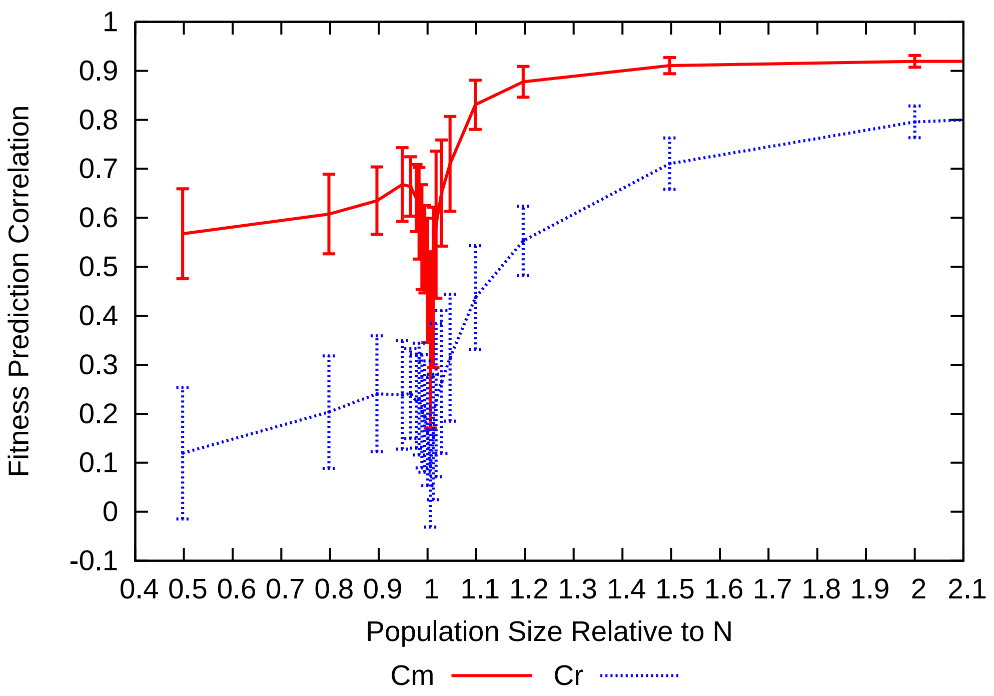


Figure B.41: FPC against population size for 64 bit 2D Ising problem using a 10% decimated model

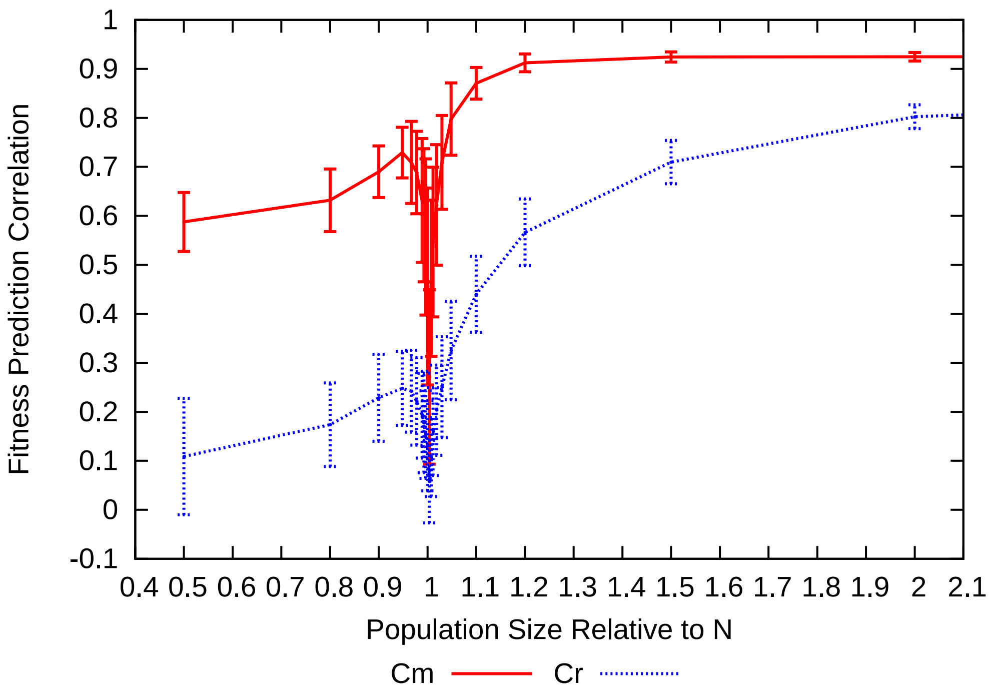


Figure B.42: FPC against population size for 100 bit 2D Ising problem using a 10% decimated model

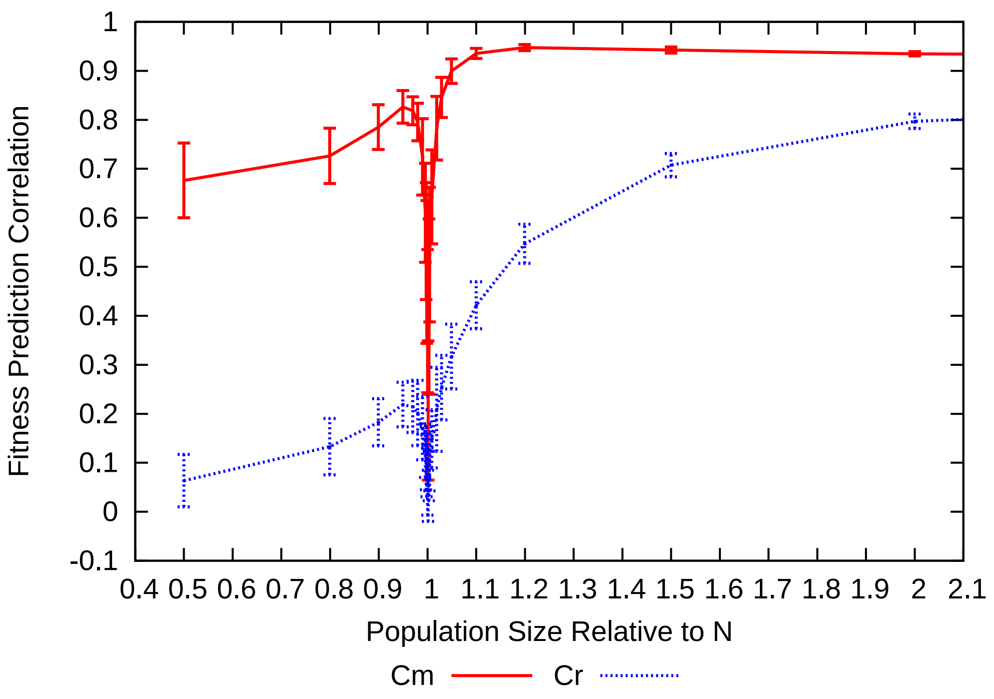


Figure B.43: FPC against population size for 256 bit 2D Ising problem using a 10% decimated model

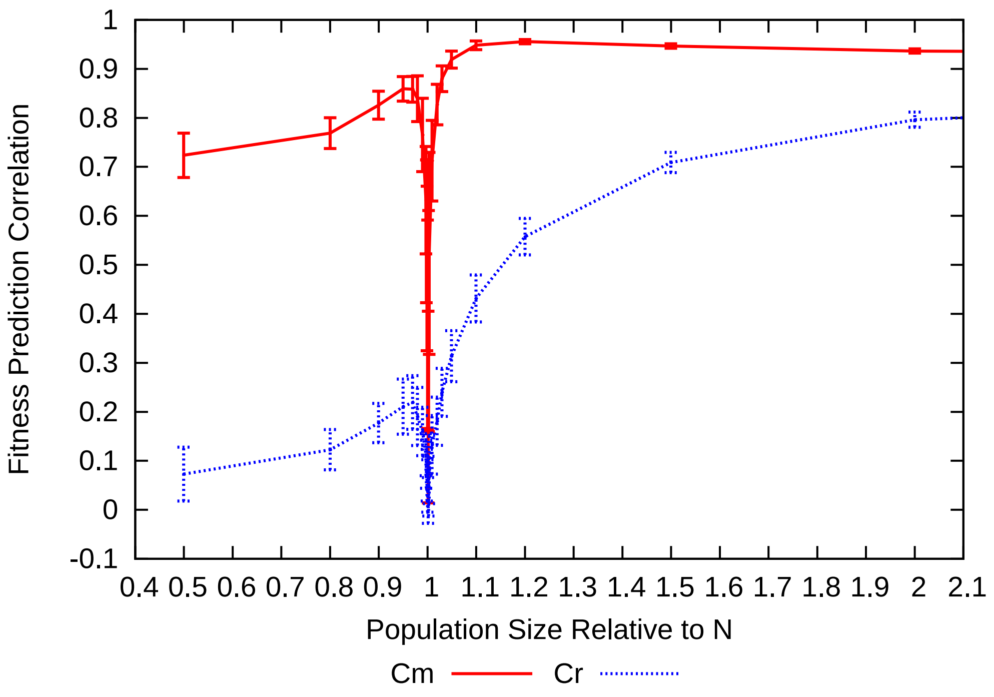


Figure B.44: FPC against population size for 324 bit 2D Ising problem using a 10% decimated model

Figure B.45: FPC against population size for 400 bit 2D Ising problem using a 10% decimated model

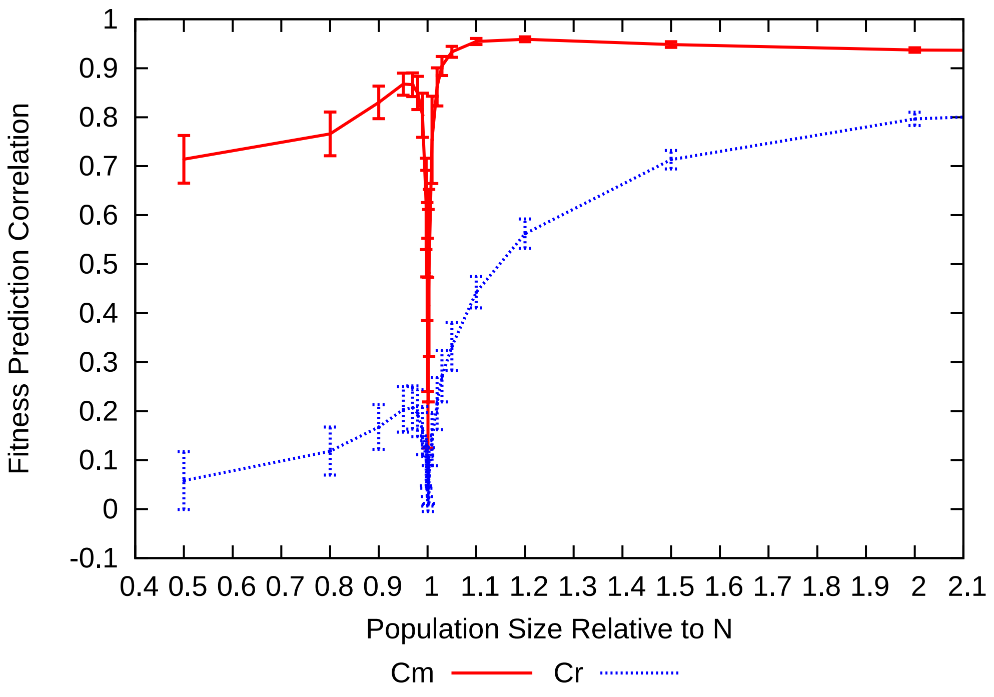
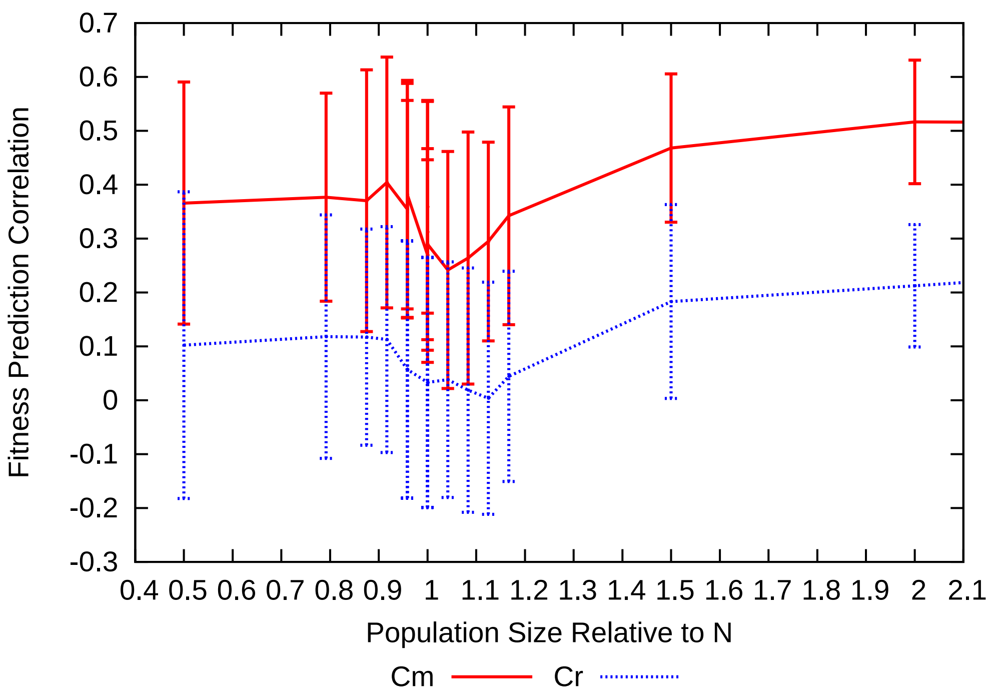


Figure B.46: FPC against population size for 16 bit 2D Ising problem using a 50% decimated model



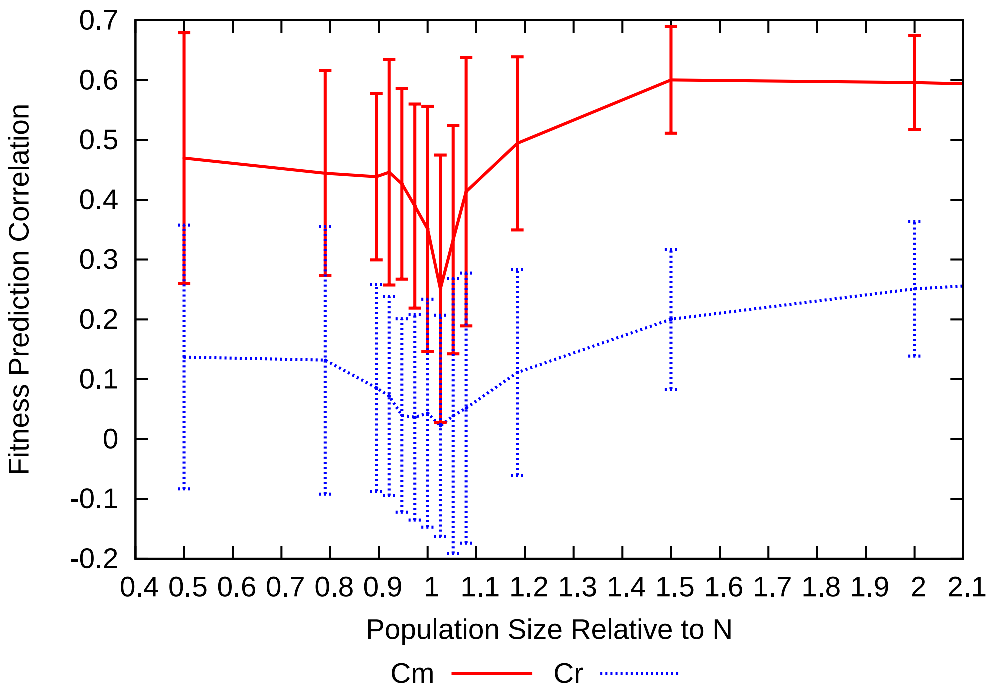


Figure B.47: FPC against population size for 25 bit 2D Ising problem using a 50% decimated model

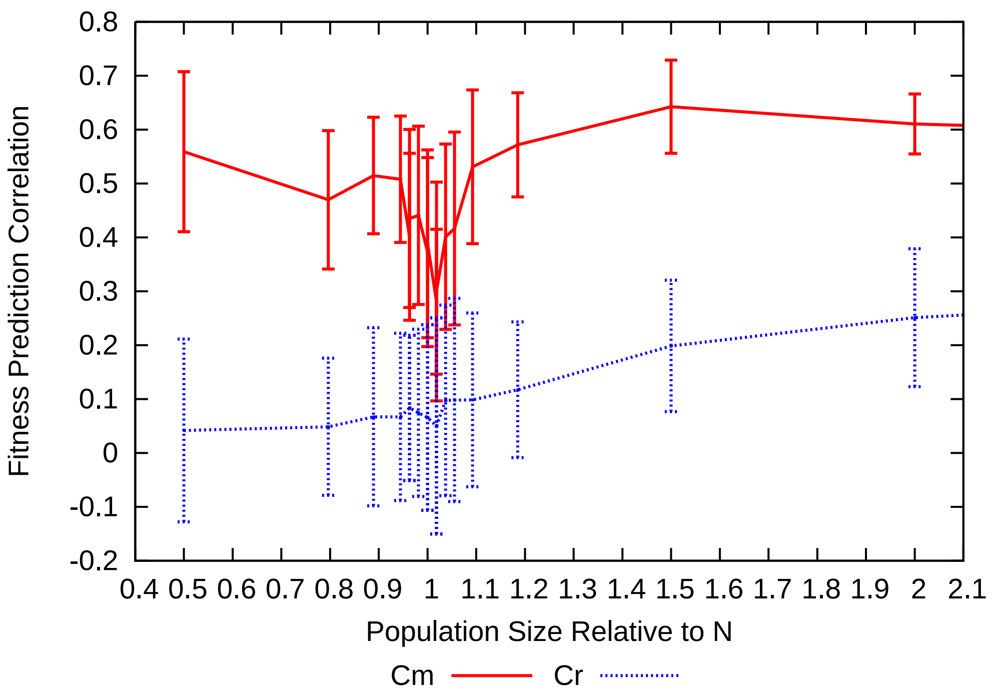


Figure B.48: FPC against population size for 36 bit 2D Ising problem using a 50% decimated model

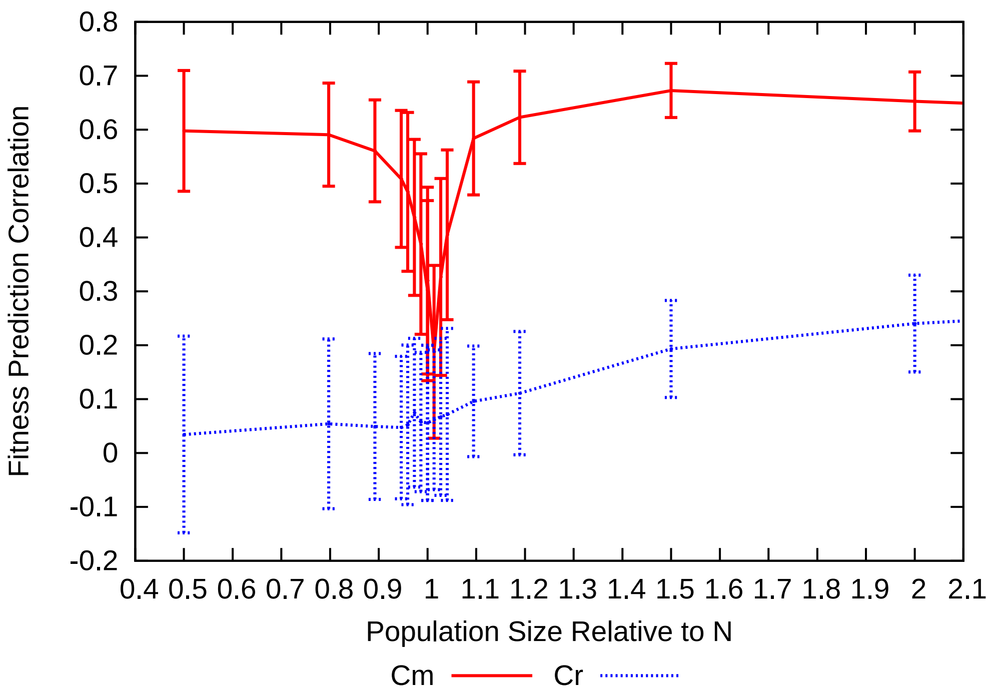


Figure B.49: FPC against population size for 49 bit 2D Ising problem using a 50% decimated model

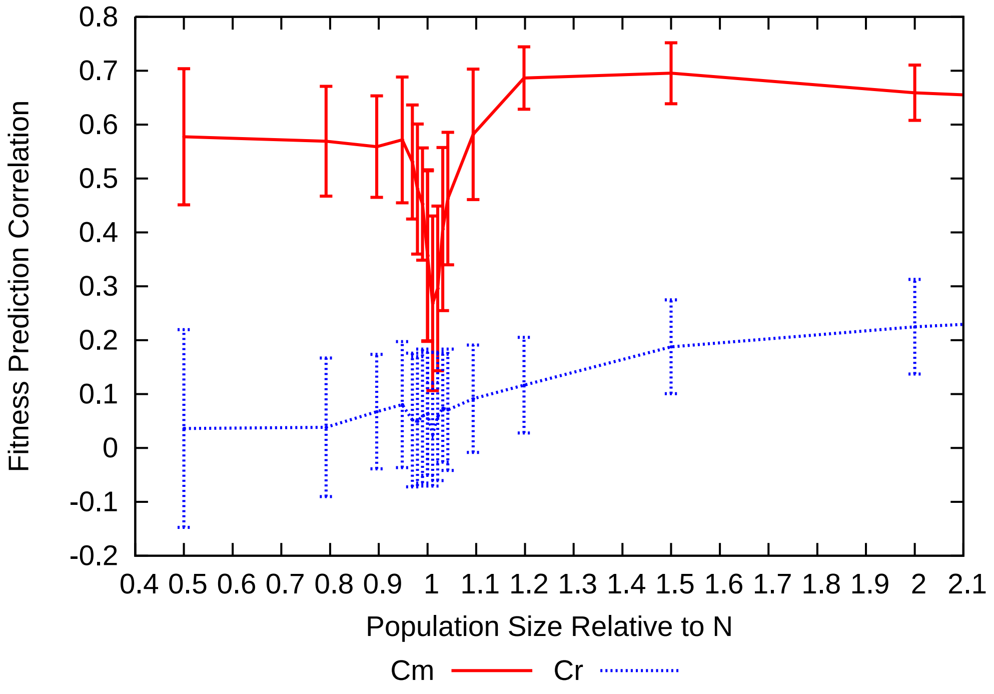


Figure B.50: FPC against population size for 64 bit 2D Ising problem using a 50% decimated model

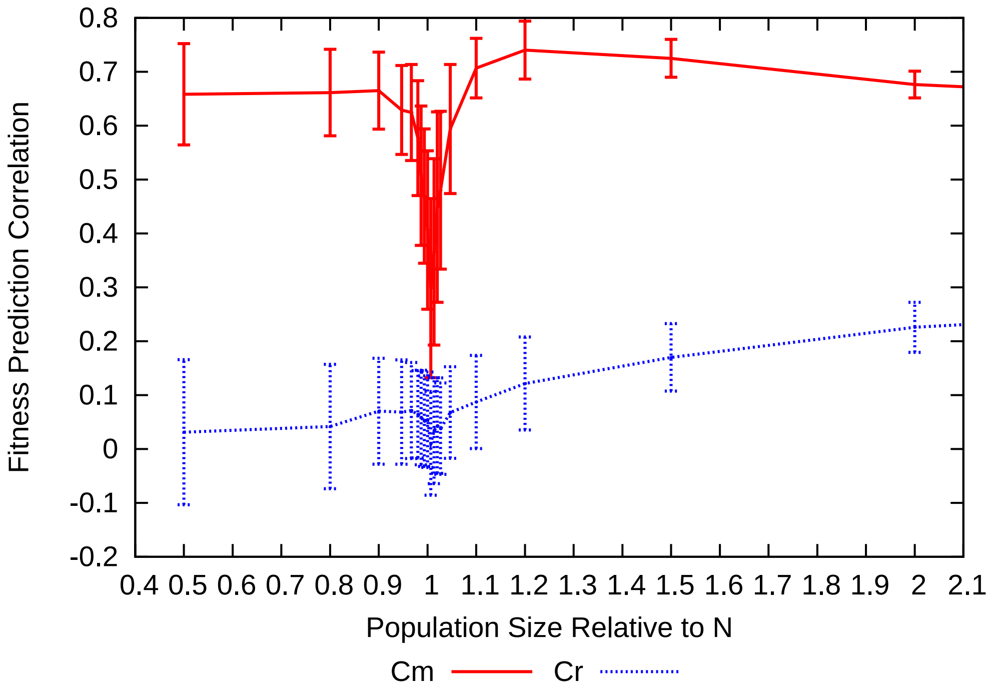


Figure B.51: FPC against population size for 100 bit 2D Ising problem using a 50% decimated model

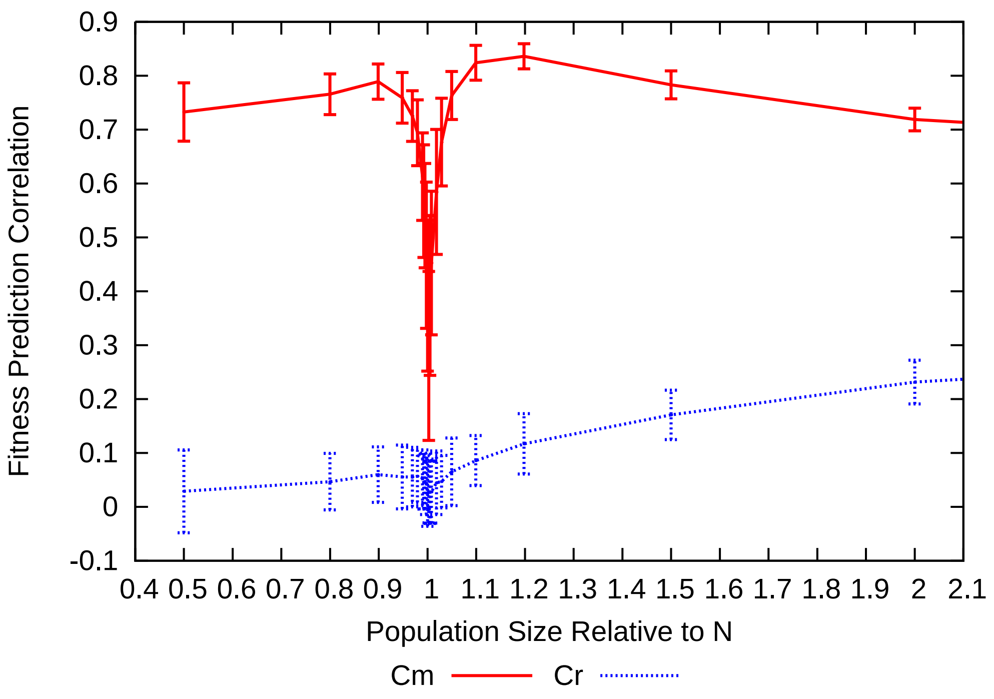


Figure B.52: FPC against population size for 256 bit 2D Ising problem using a 50% decimated model

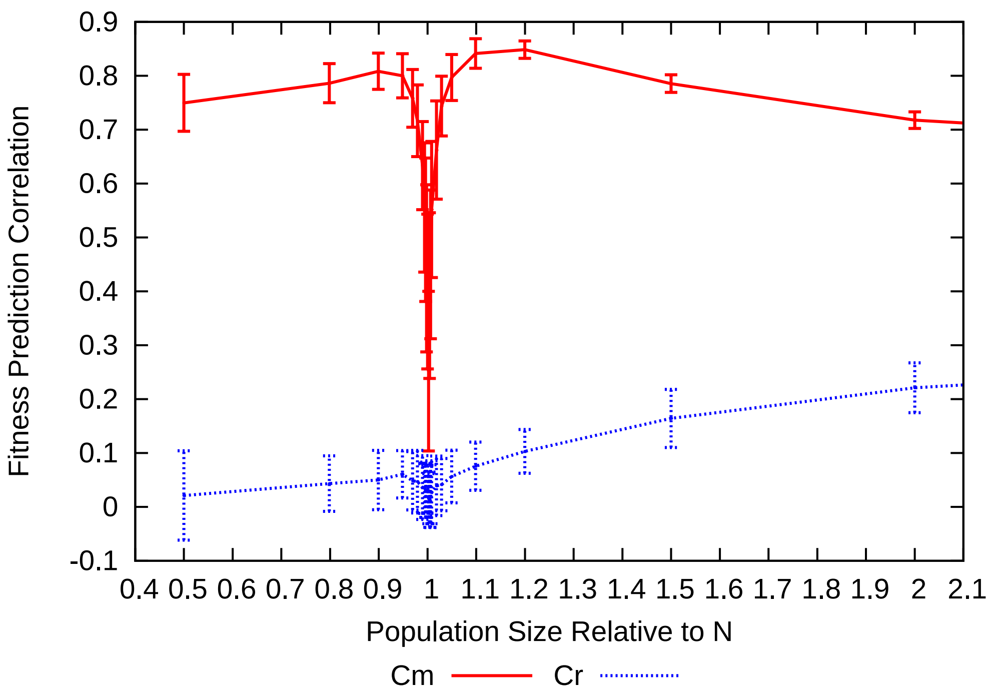


Figure B.53: FPC against population size for 324 bit 2D Ising problem using a 50% decimated model

Figure B.54: FPC against population size for 400 bit 2D Ising problem using a 50% decimated model

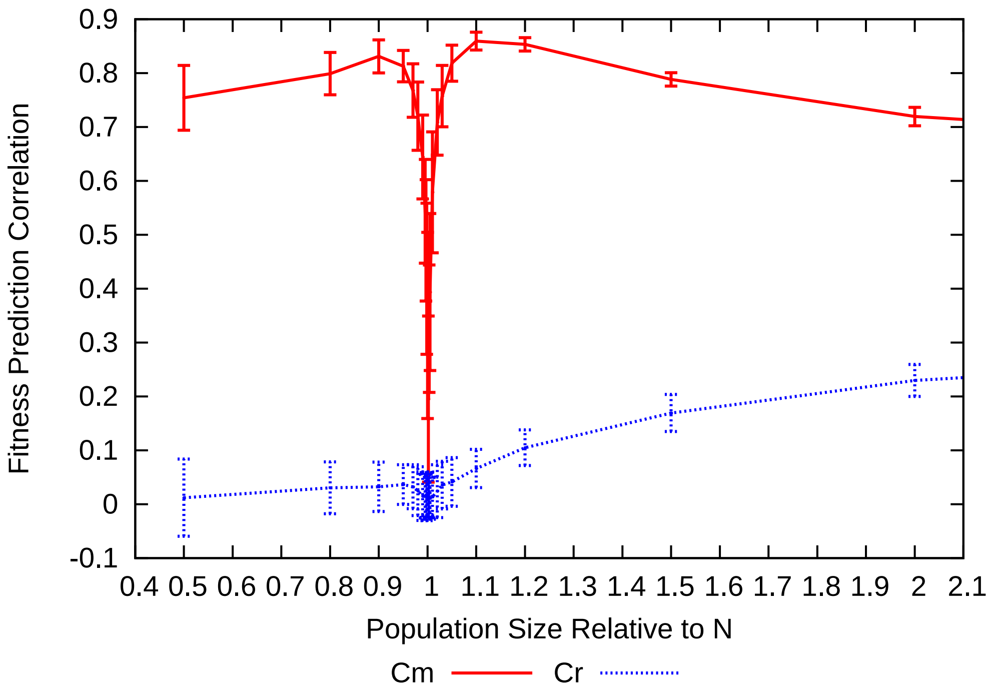
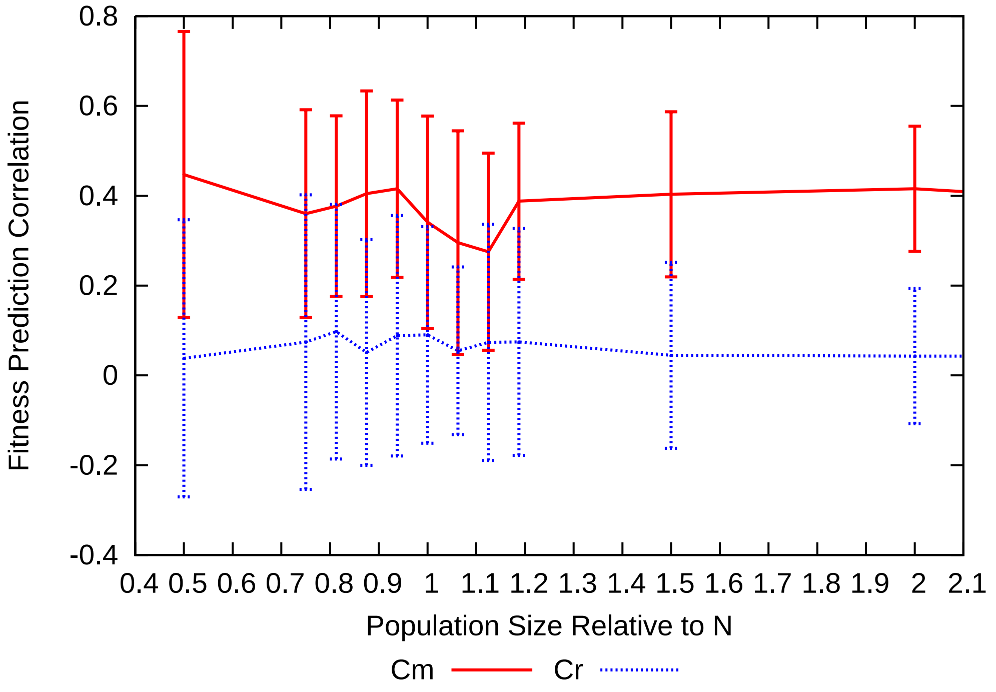


Figure B.55: FPC vs population size for 16 bit 2D Ising problem using a univariate model



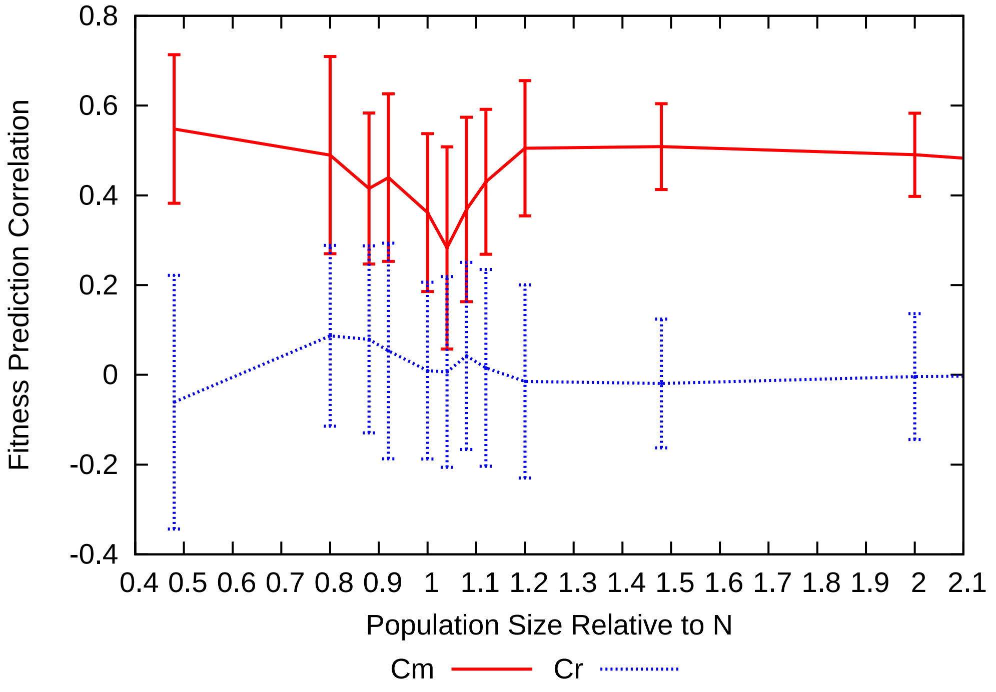


Figure B.56: FPC vs population size for 25 bit 2D Ising problem using a univariate model

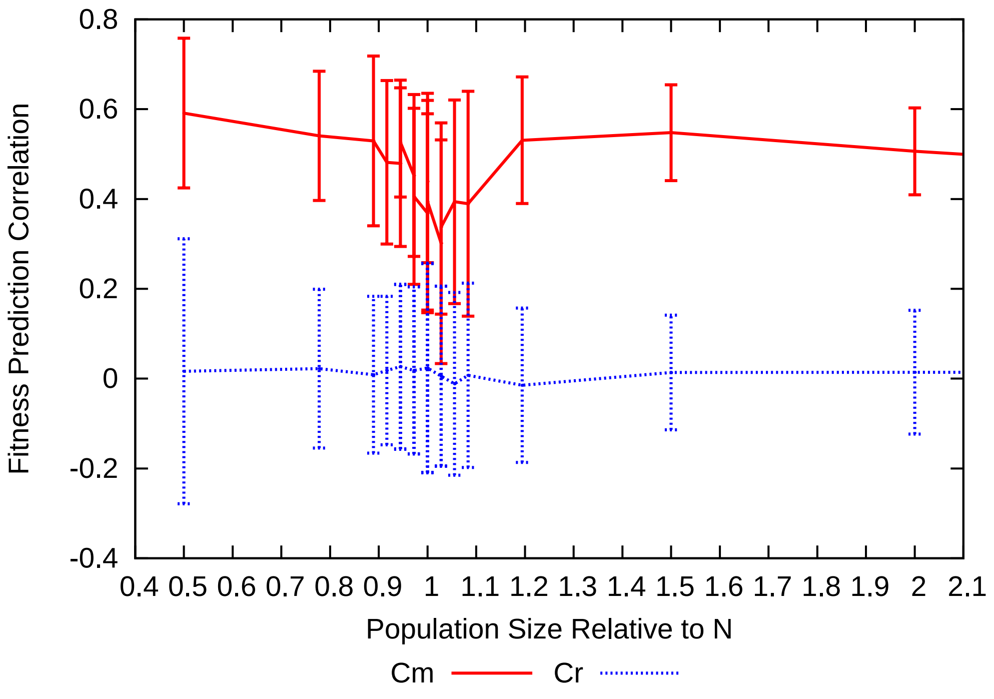


Figure B.57: FPC vs population size for 36 bit 2D Ising problem using a univariate model

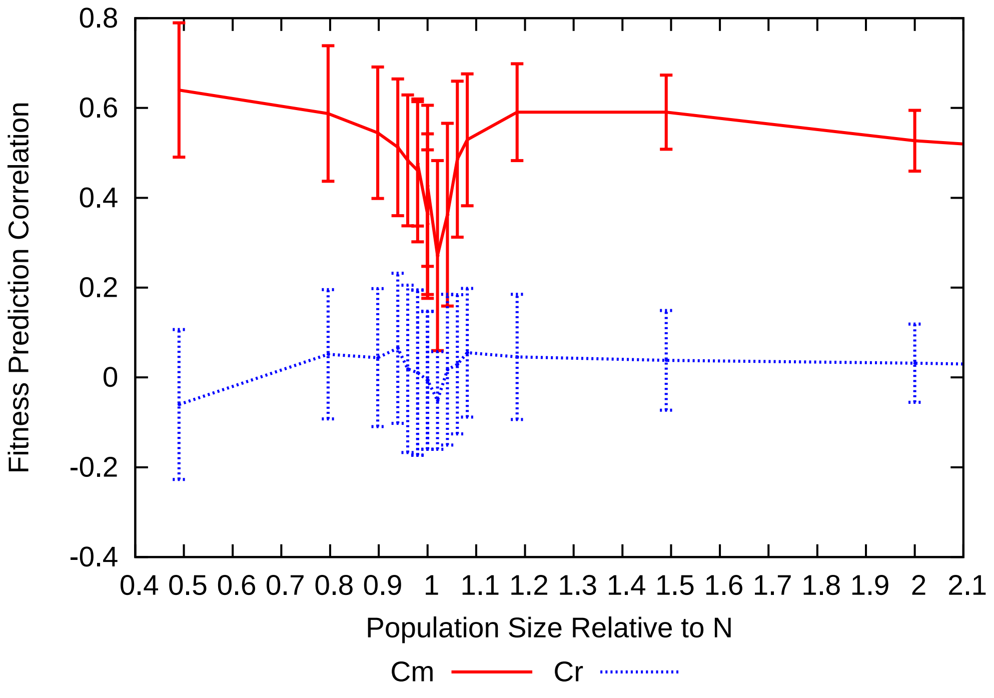


Figure B.58: FPC vs population size for 49 bit 2D Ising problem using a univariate model

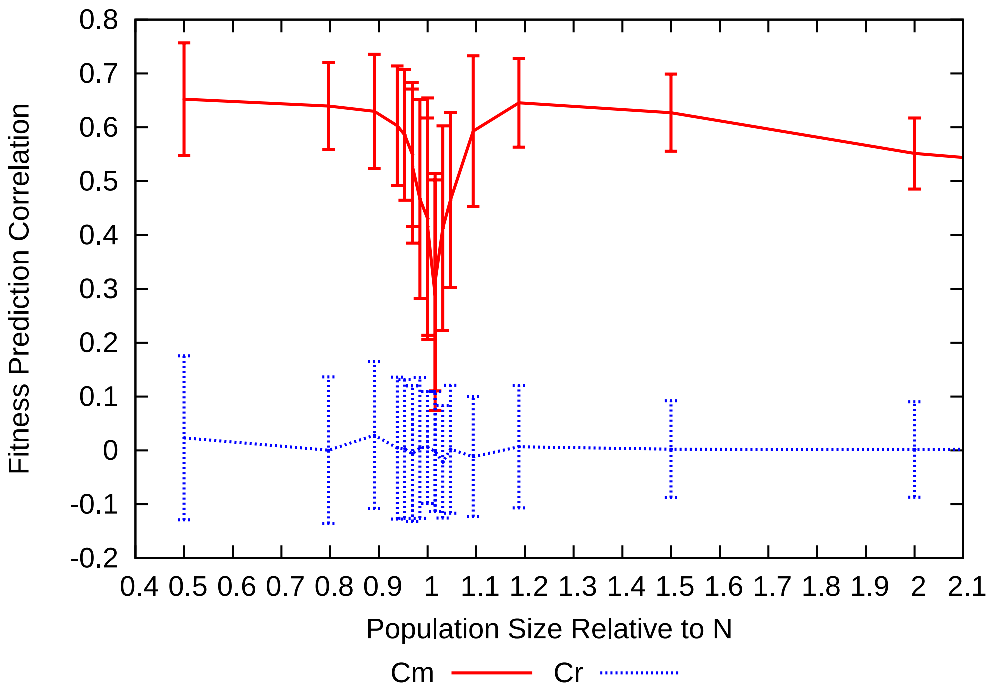


Figure B.59: FPC vs population size for 64 bit 2D Ising problem using a univariate model

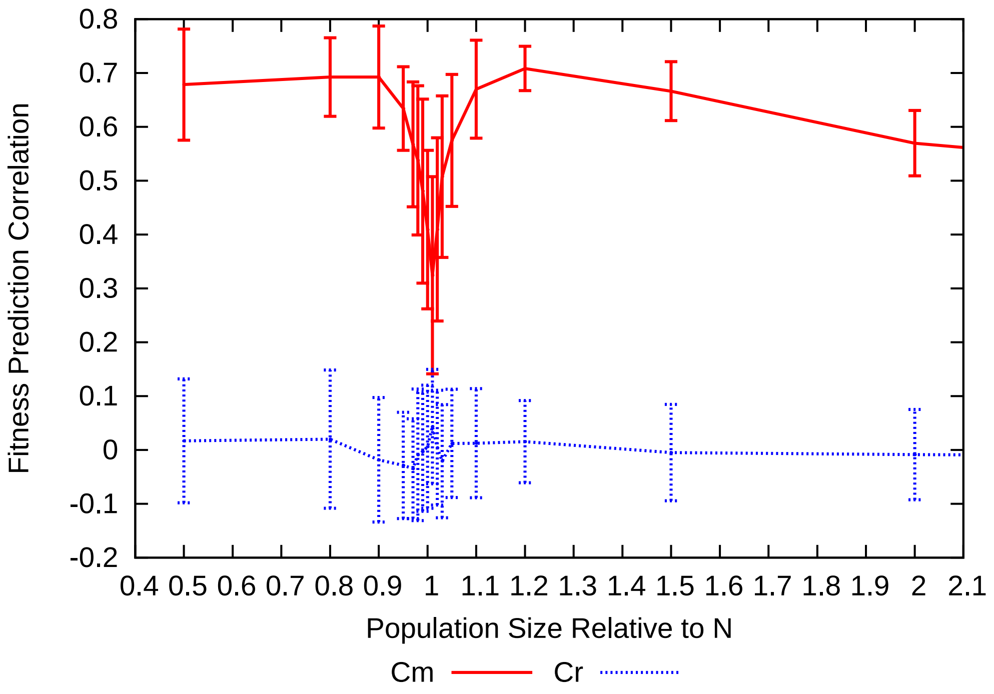


Figure B.60: FPC vs population size for 100 bit 2D Ising problem using a univariate model

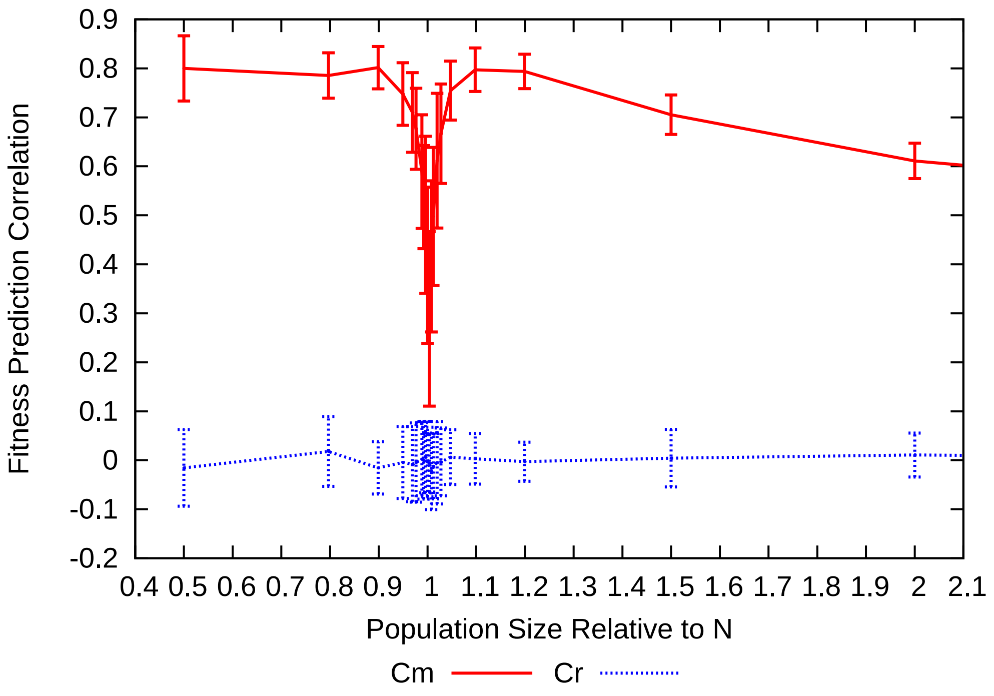


Figure B.61: FPC vs population size for 256 bit 2D Ising problem using a univariate model

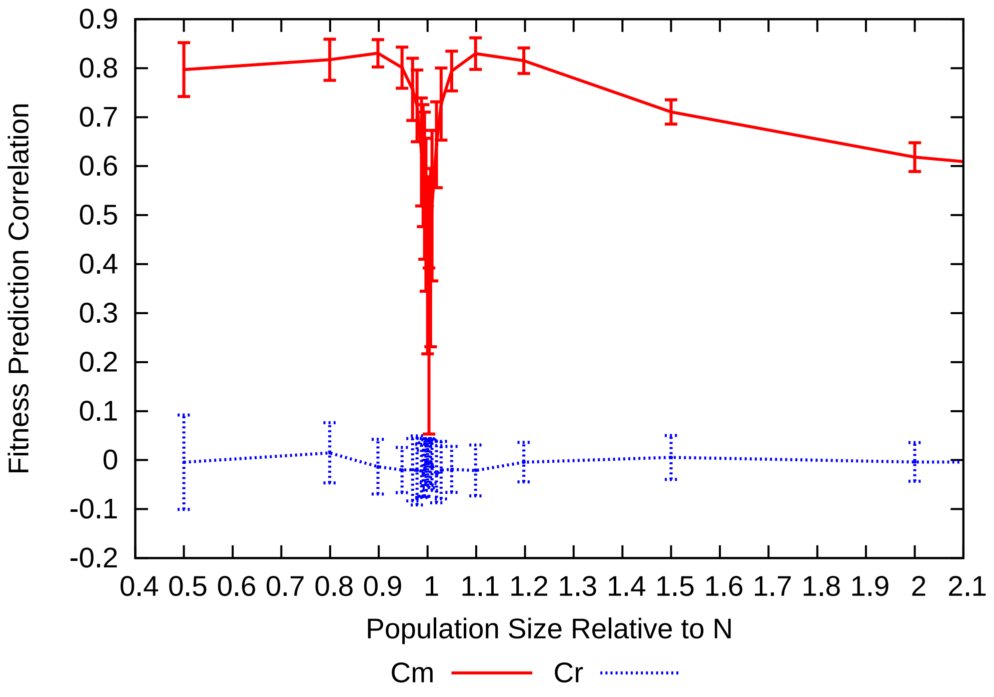


Figure B.62: FPC vs population size for 324 bit 2D Ising problem using a univariate model

Figure B.63: FPC vs population size for 400 bit 2D Ising problem using a univariate model

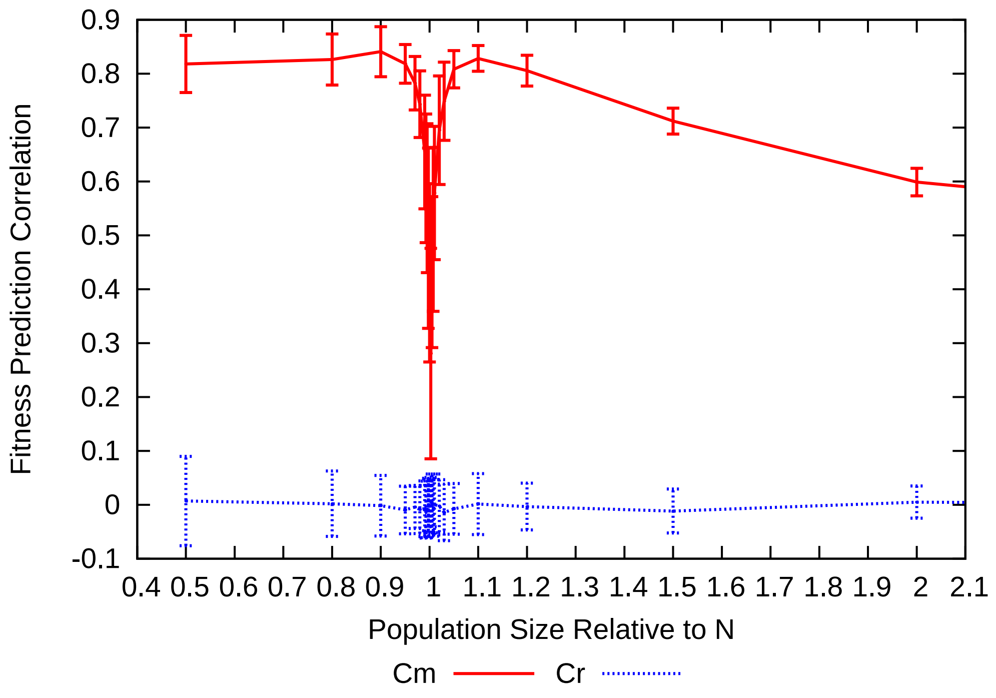
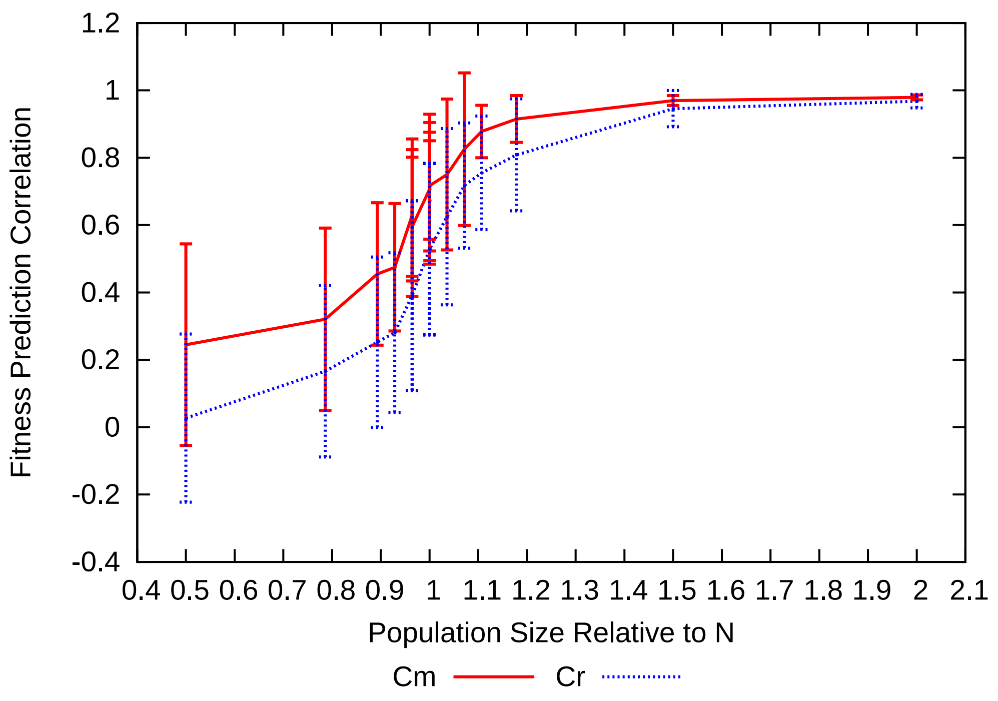


Figure B.64: FPC vs population size for 16 bit 2D Checkerboard problem using a perfect model structure



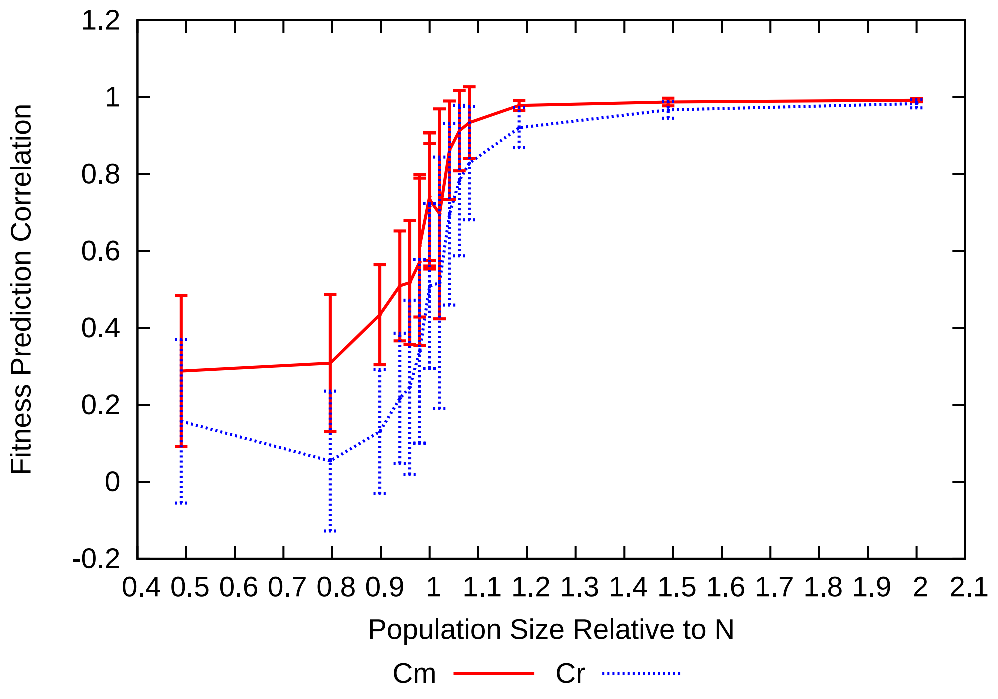


Figure B.65: FPC vs population size for 25 bit 2D Checkerboard problem using a perfect model structure

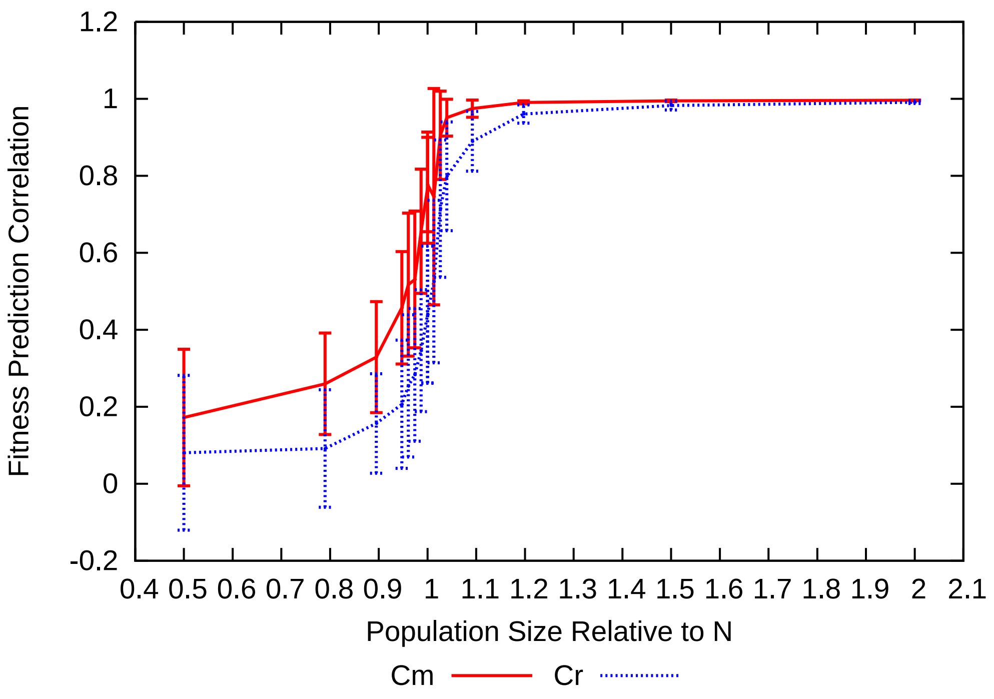


Figure B.66: FPC vs population size for 36 bit 2D Checkerboard problem using a perfect model structure

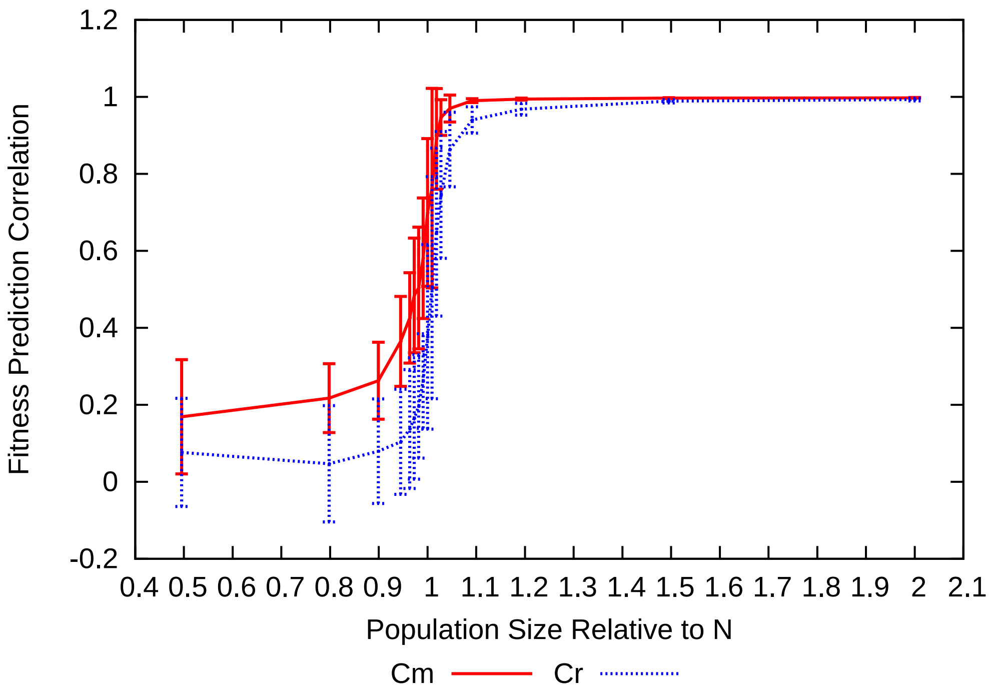


Figure B.67: FPC vs population size for 49 bit 2D Checkerboard problem using a perfect model structure

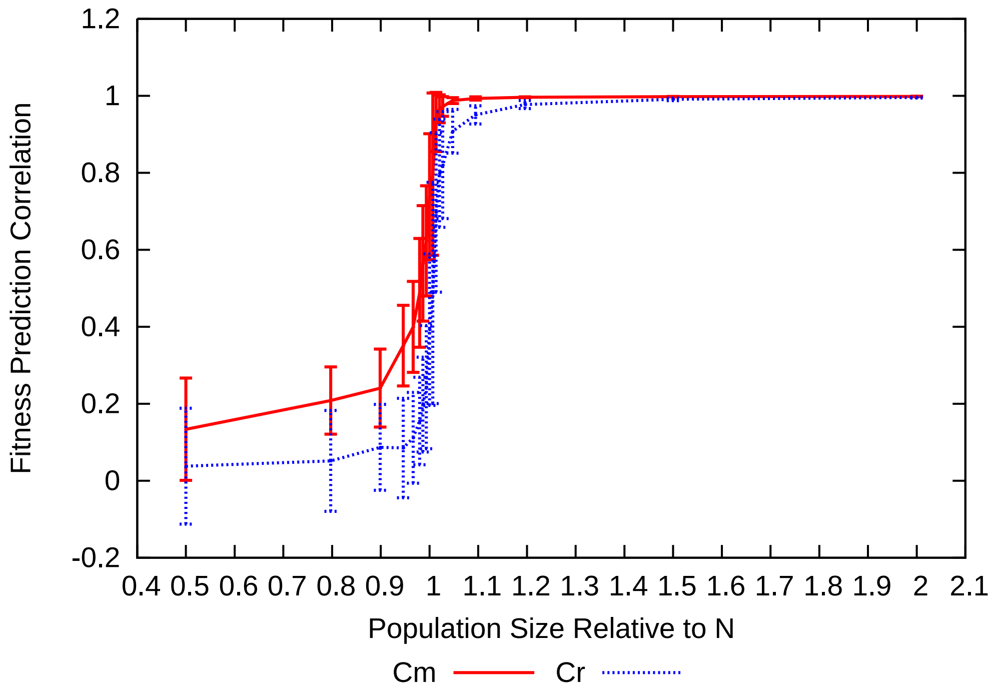


Figure B.68: FPC vs population size for 64 bit 2D Checkerboard problem using a perfect model structure

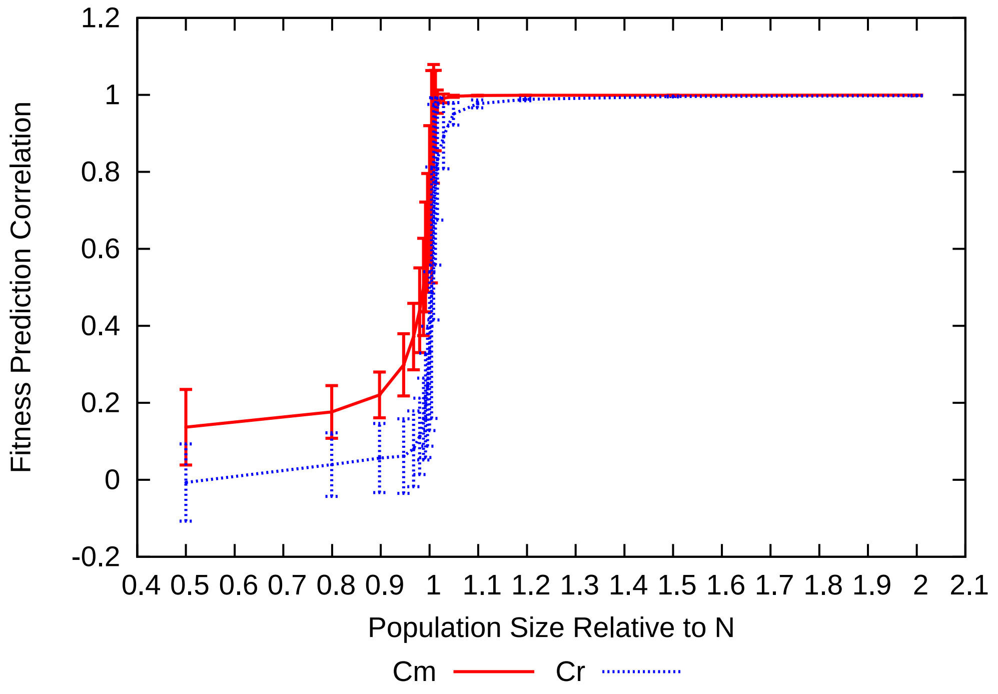


Figure B.69: FPC vs population size for 100 bit 2D Checkerboard problem using a perfect model structure

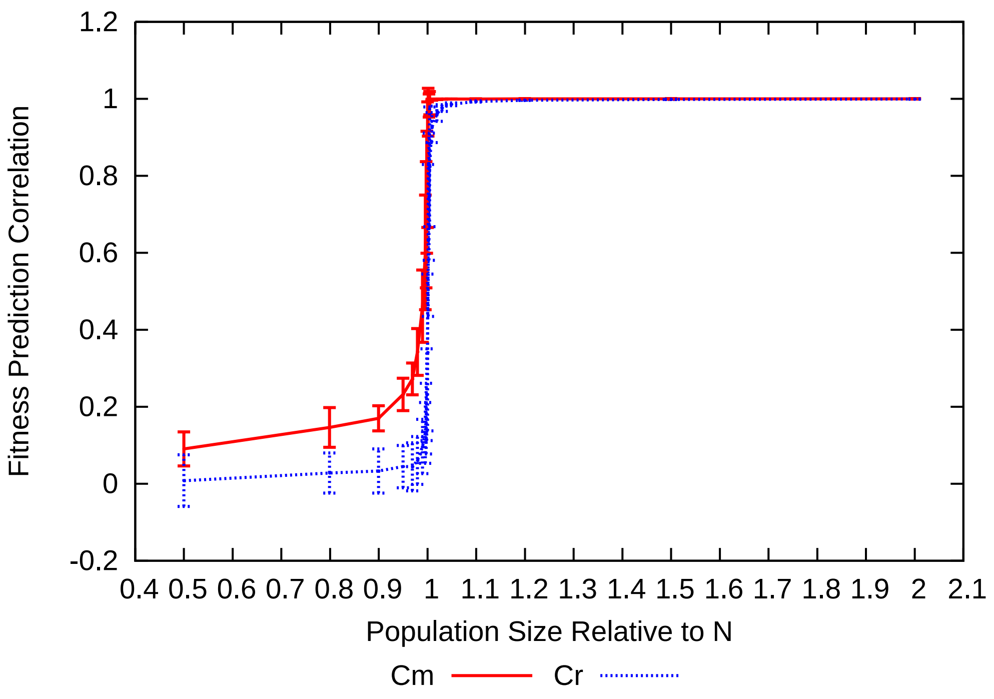


Figure B.70: FPC vs population size for 256 bit 2D Checkerboard problem using a perfect model structure

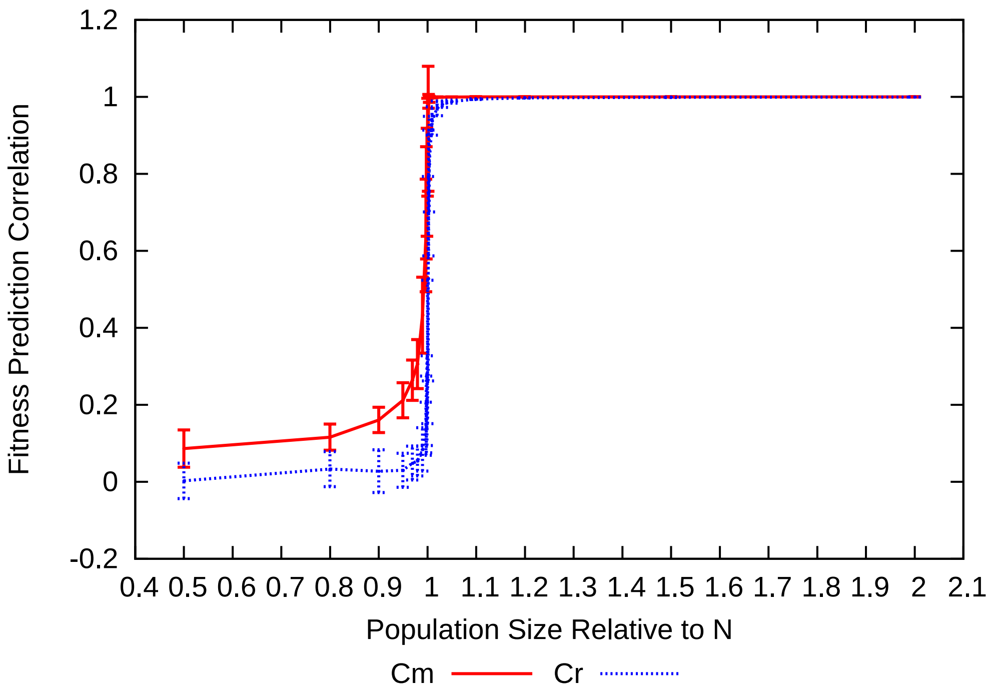


Figure B.71: FPC vs population size for 324 bit 2D Checkerboard problem using a perfect model structure

Figure B.72: FPC vs population size for 400 bit 2D Checkerboard problem using a perfect model structure

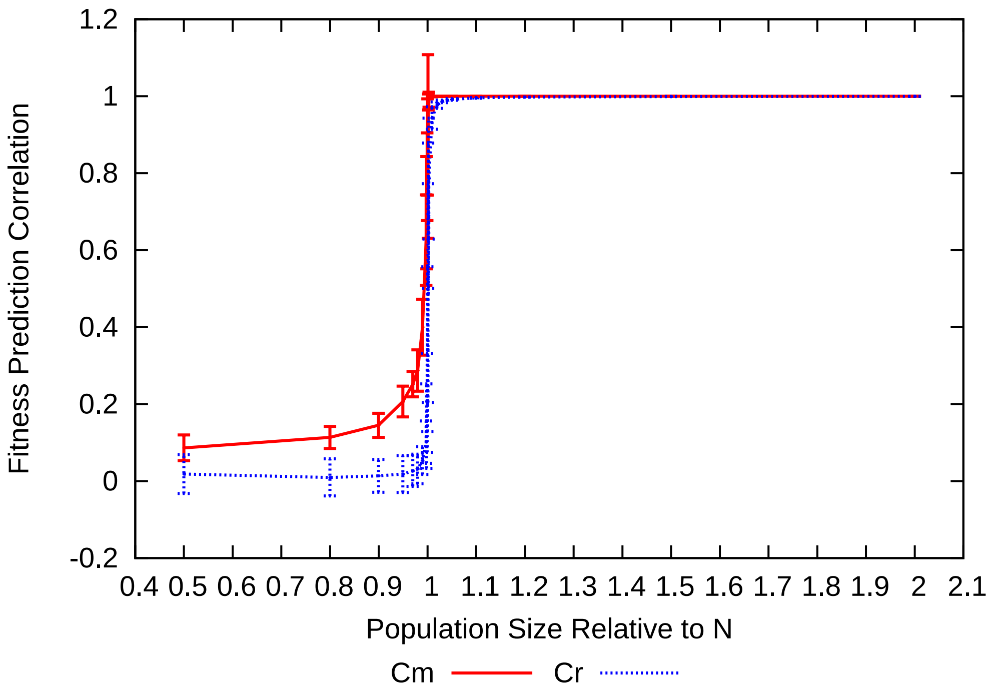
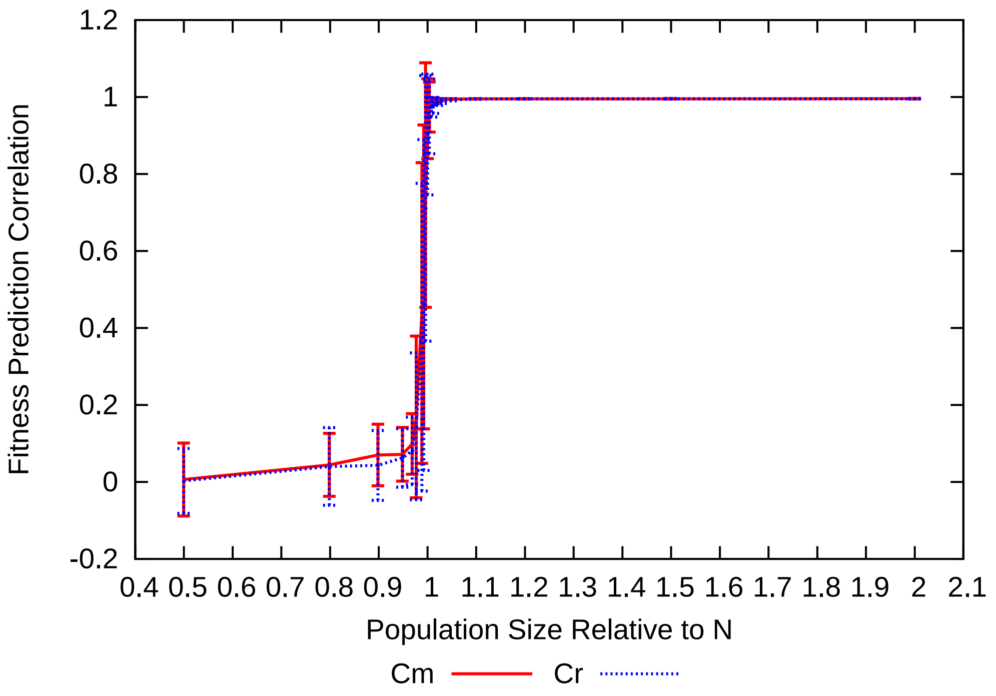


Figure B.73: FPC vs population size for 20 bit MaxSAT problem using a perfect model structure



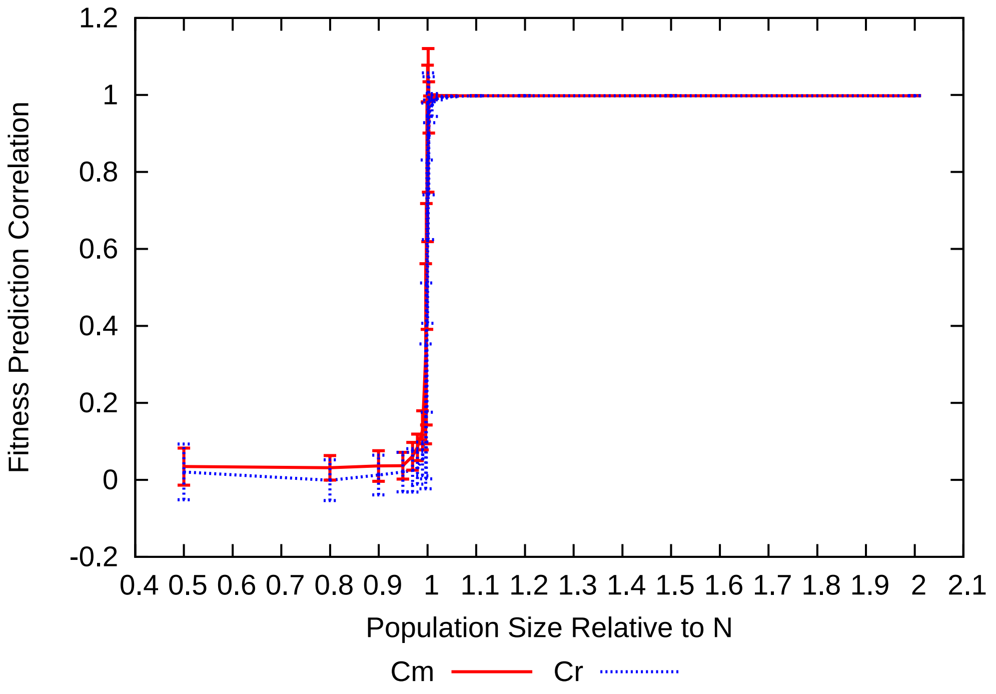


Figure B.74: FPC vs population size for 50 bit MaxSAT problem using a perfect model structure

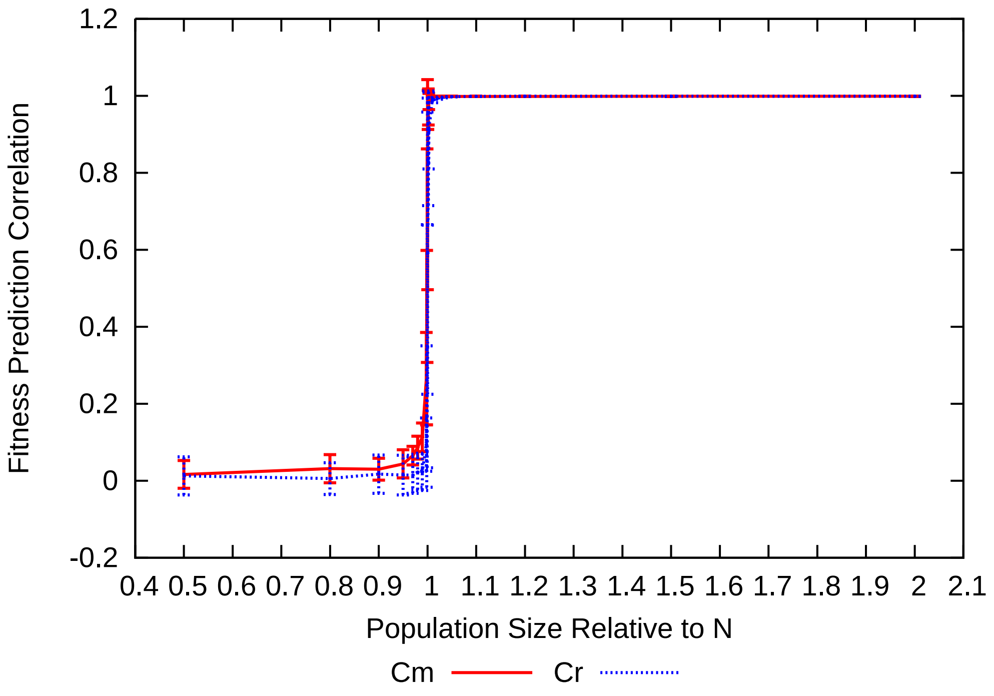


Figure B.75: FPC vs population size for 75 bit MaxSAT problem using a perfect model structure

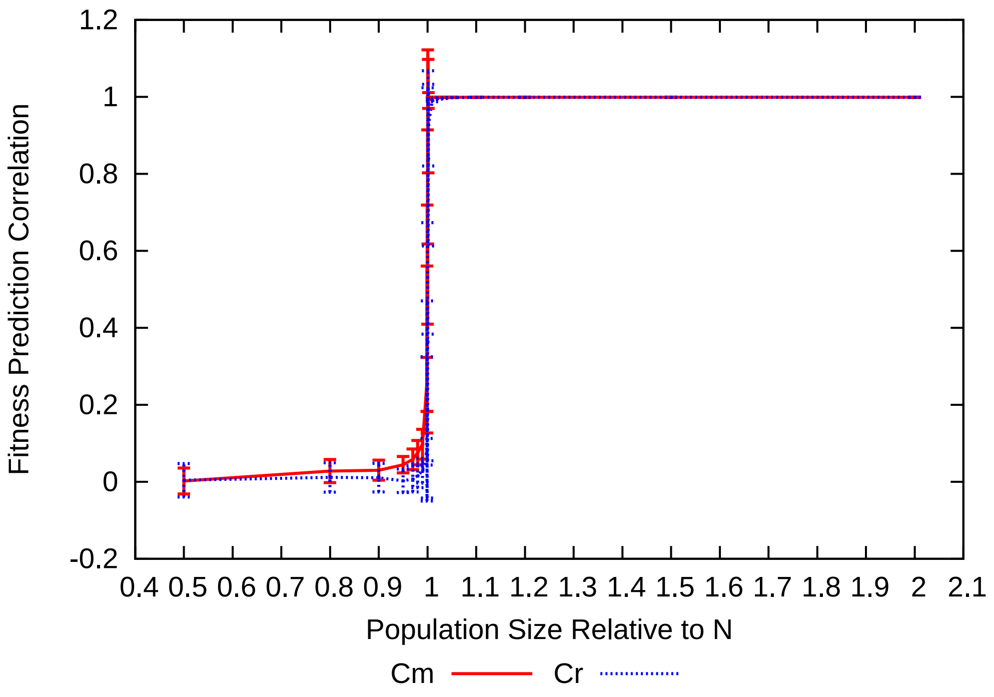


Figure B.76: FPC vs population size for 100 bit MaxSAT problem using a perfect model structure

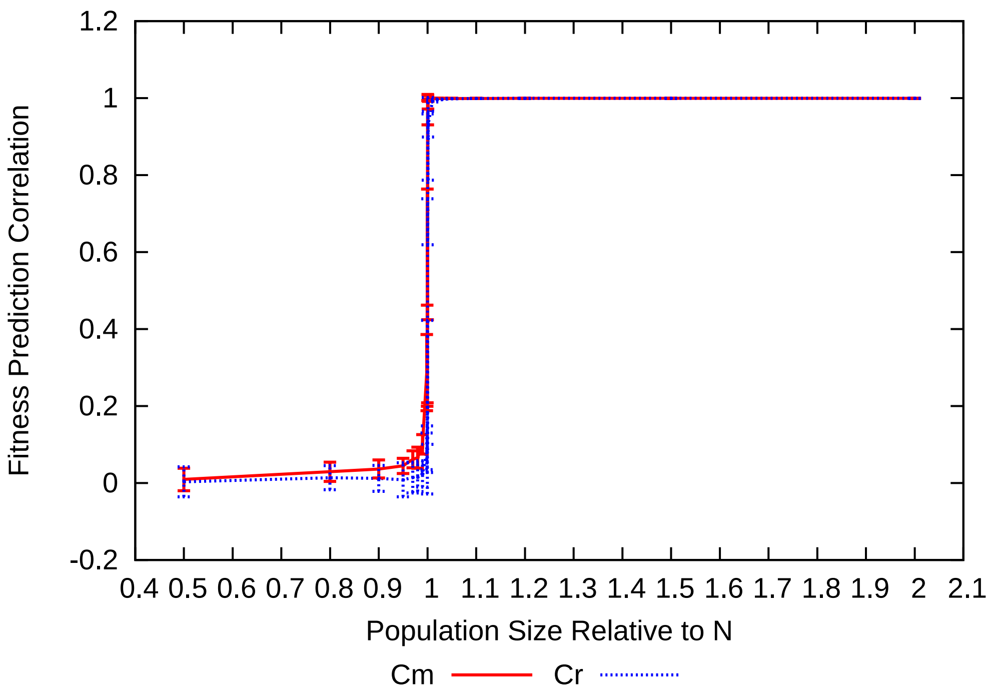
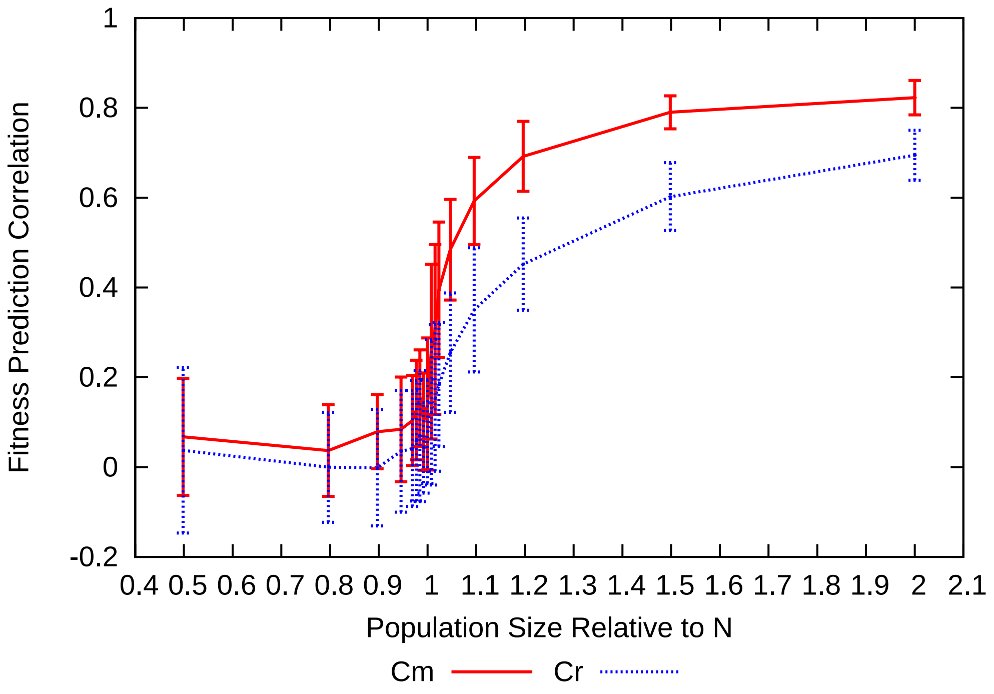


Figure B.77: FPC vs population size for 125 bit MaxSAT problem using a perfect model structure

Figure B.78: FPC vs population size for 20 bit MaxSAT problem using a 50% decimated model



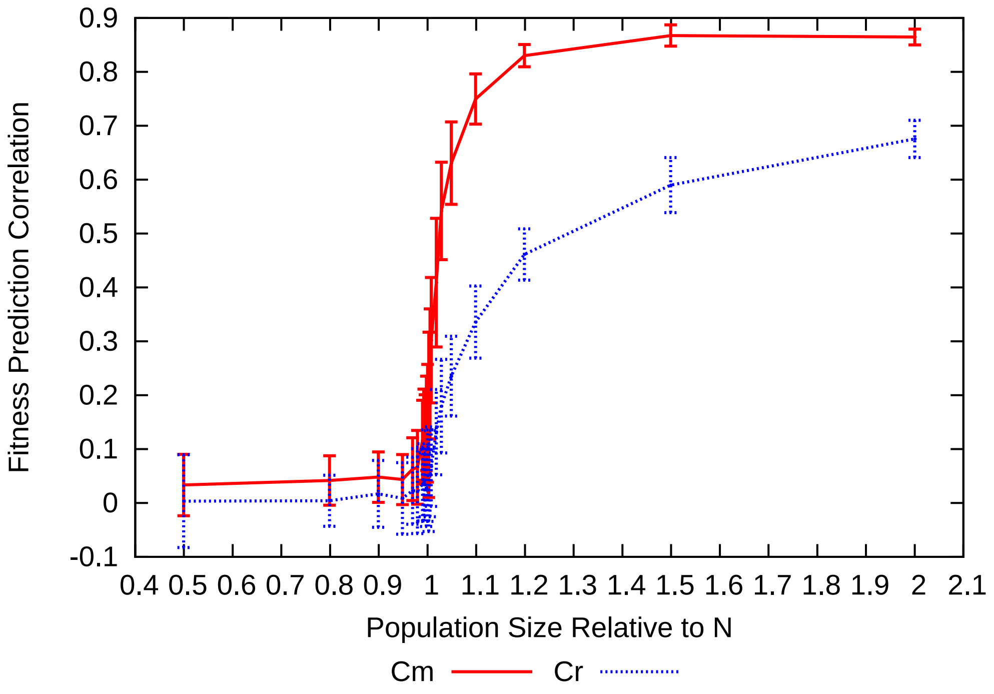


Figure B.79: FPC vs population size for 50 bit MaxSAT problem using a 50% decimated model

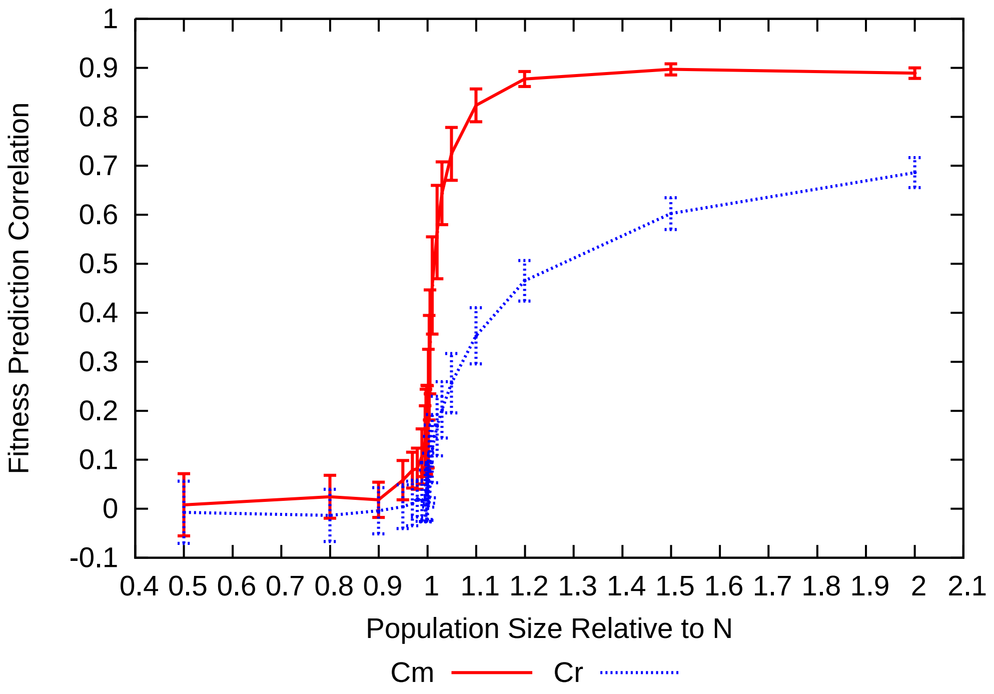


Figure B.80: FPC vs population size for 75 bit MaxSAT problem using a 50% decimated model

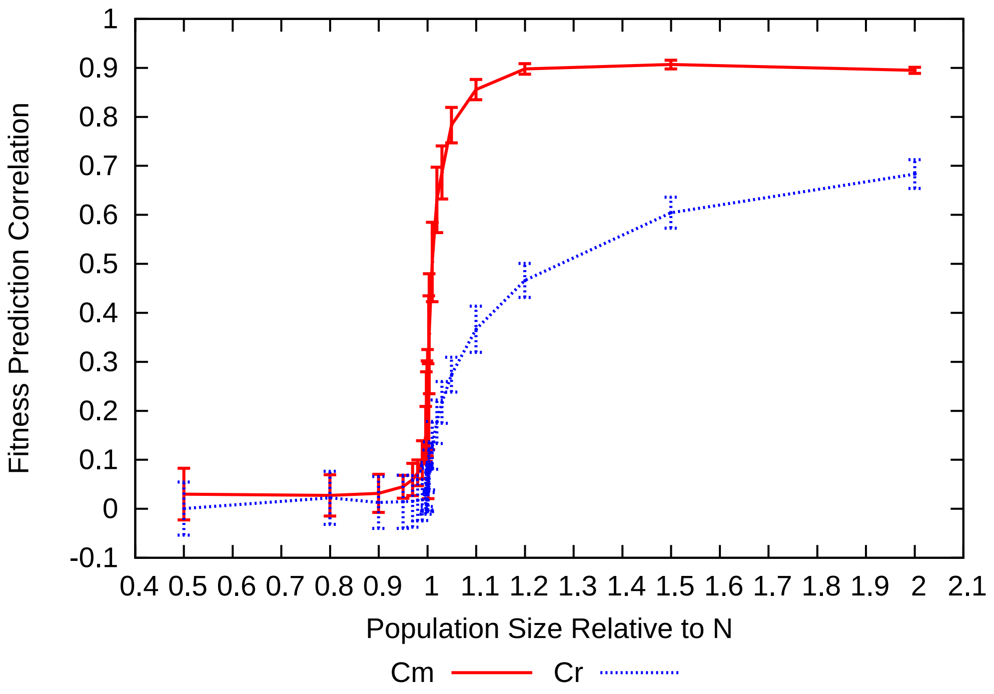


Figure B.81: FPC vs population size for 100 bit MaxSAT problem using a 50% decimated model

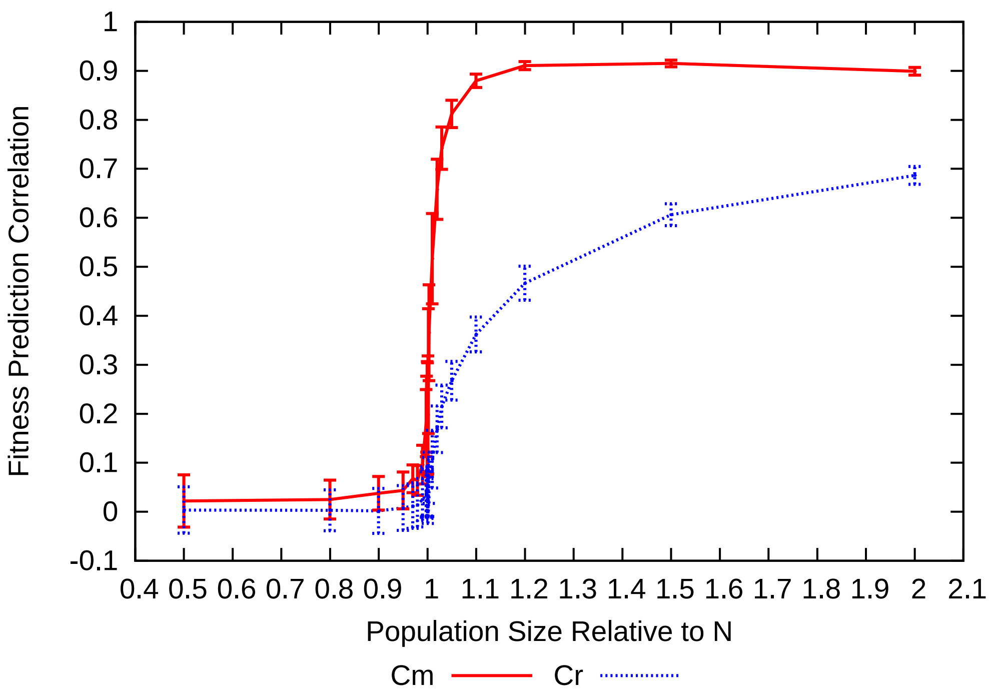


Figure B.82: FPC vs population size for 125 bit MaxSAT problem using a 50% decimated model

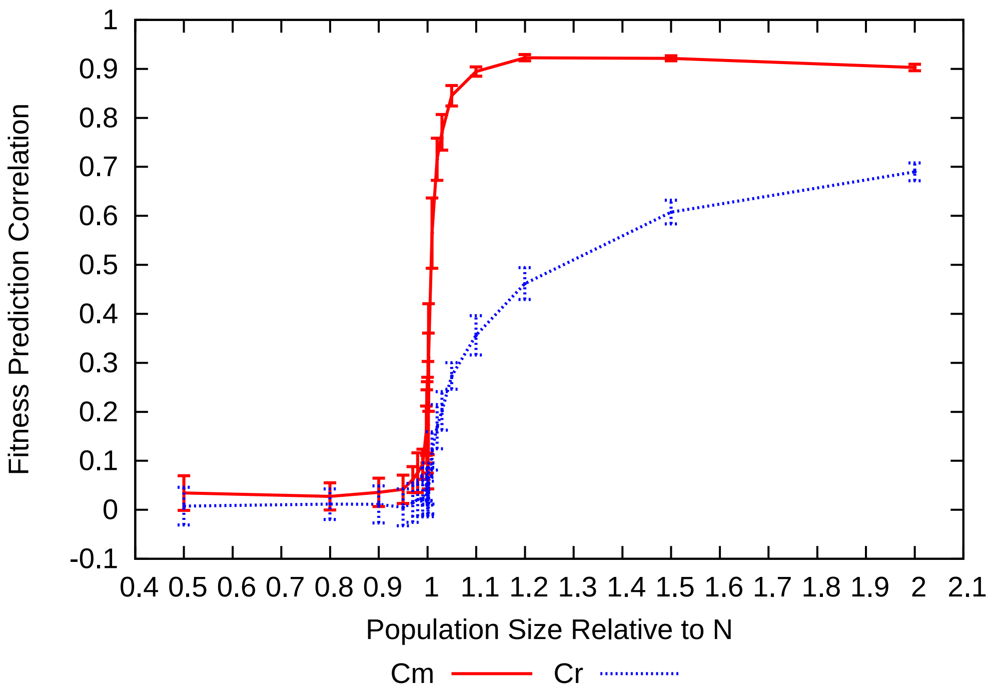


Figure B.83: FPC vs population size for 150 bit MaxSAT problem using a 50% decimated model

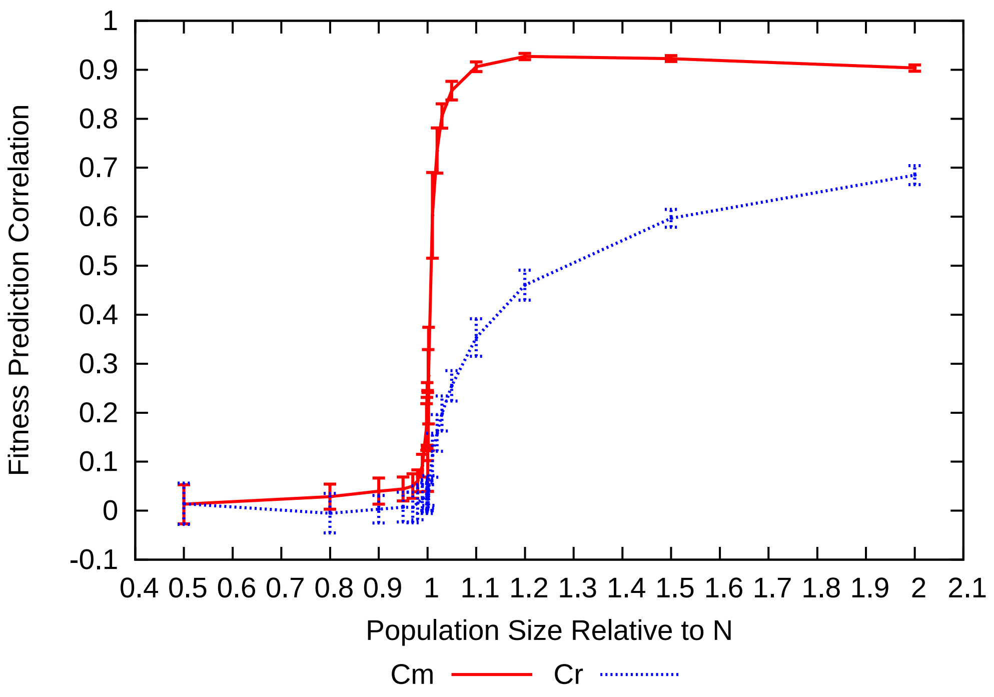


Figure B.84: FPC vs population size for 175 bit MaxSAT problem using a 50% decimated model

Figure B.85: FPC vs population size for 200 bit MaxSAT problem using a 50% decimated model

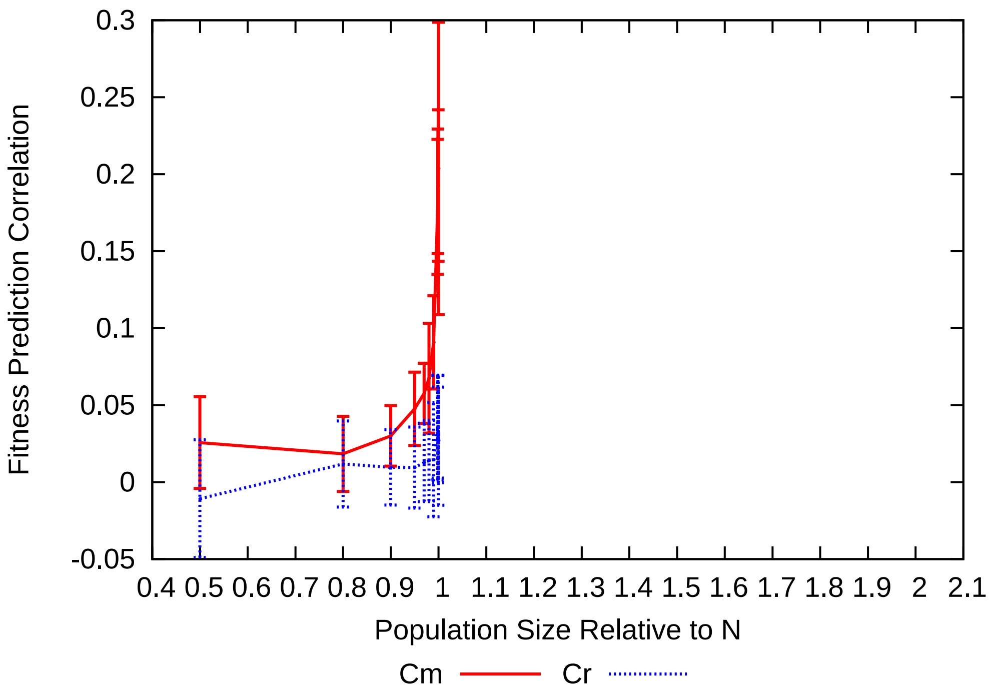
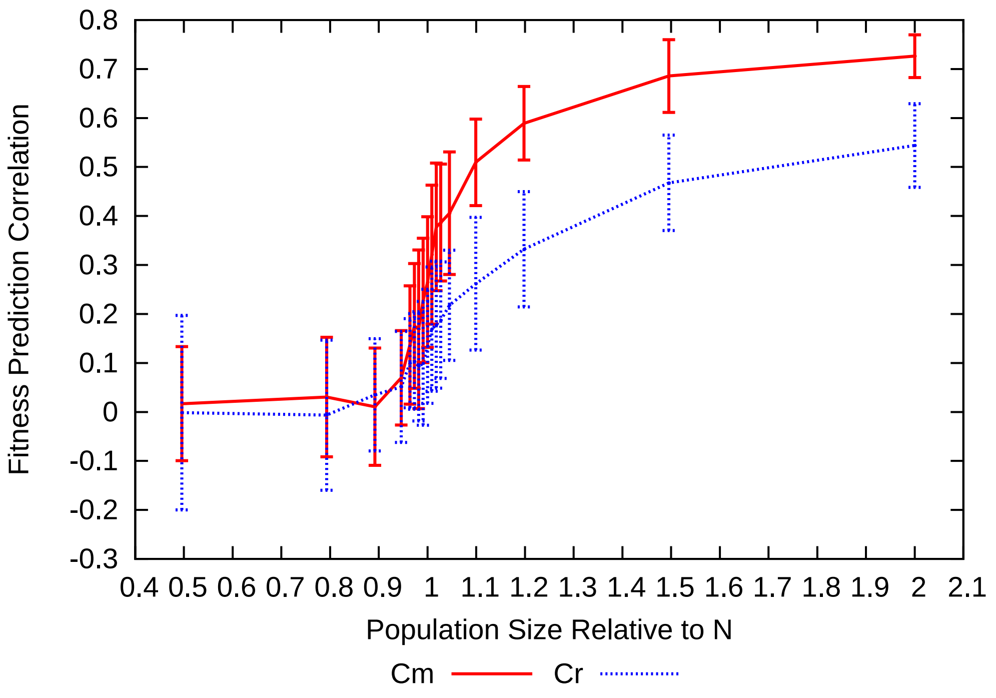


Figure B.86: FPC vs population size for 20 bit MaxSAT problem using a structure with trivariate and univariate terms



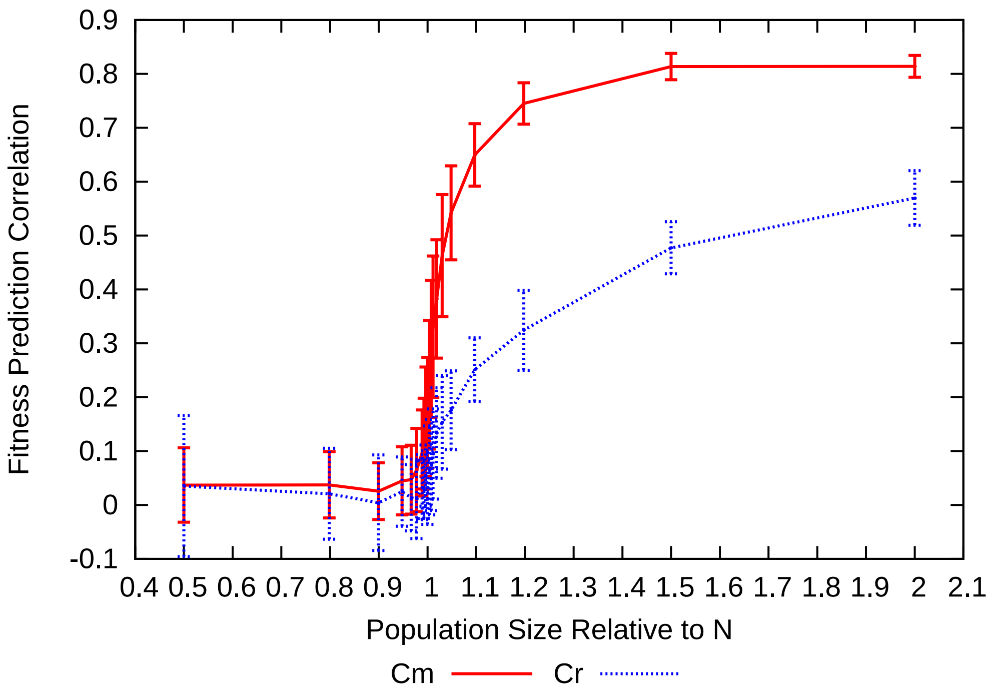


Figure B.87: FPC vs population size for 50 bit MaxSAT problem using a structure with trivariate and univariate terms

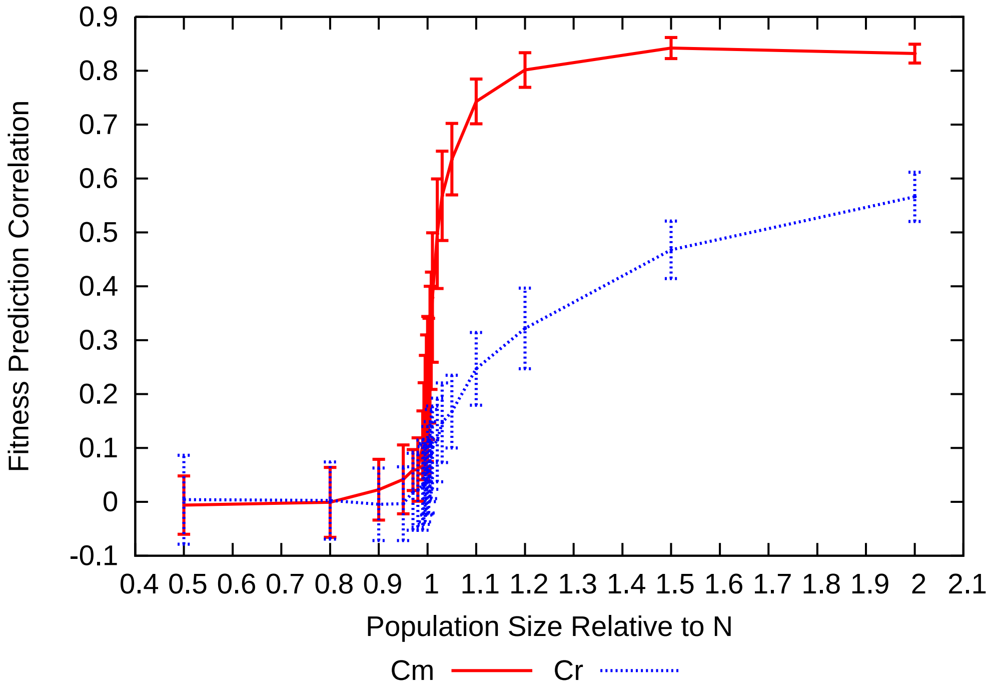


Figure B.88: FPC vs population size for 75 bit MaxSAT problem using a structure with trivariate and univariate terms

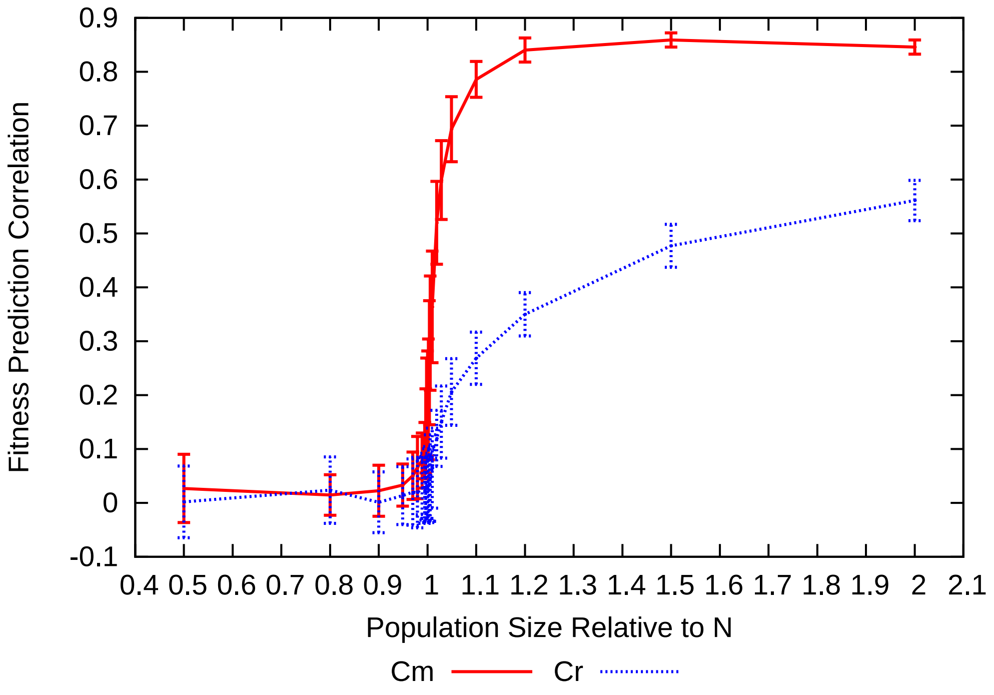


Figure B.89: FPC vs population size for 100 bit MaxSAT problem using a structure with trivariate and univariate terms

Figure B.90: FPC vs population size for 125 bit MaxSAT problem using a structure with trivariate and univariate terms

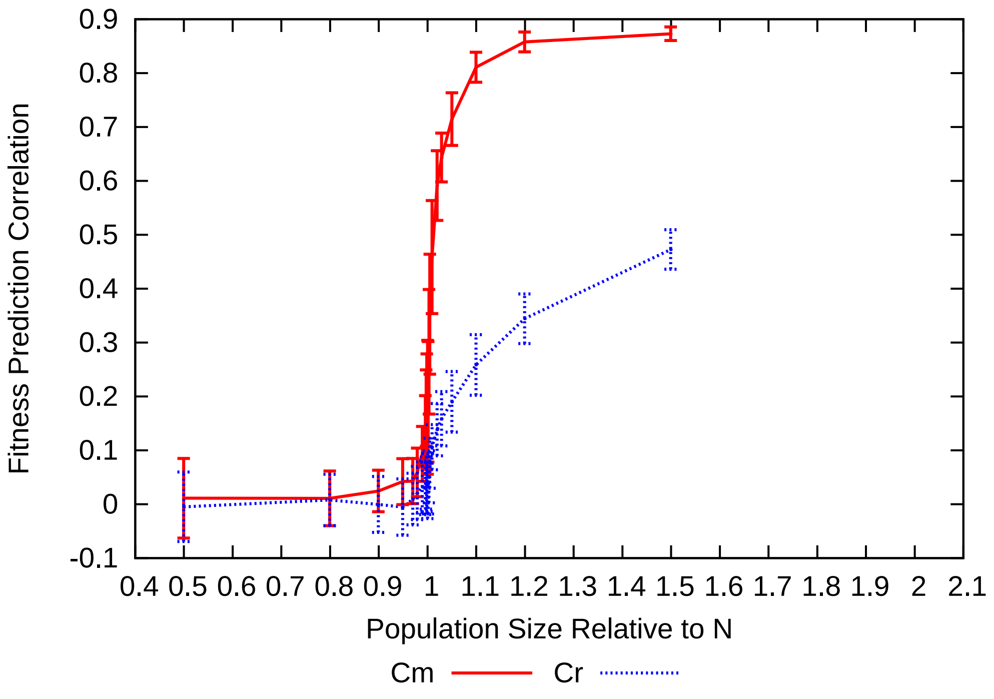
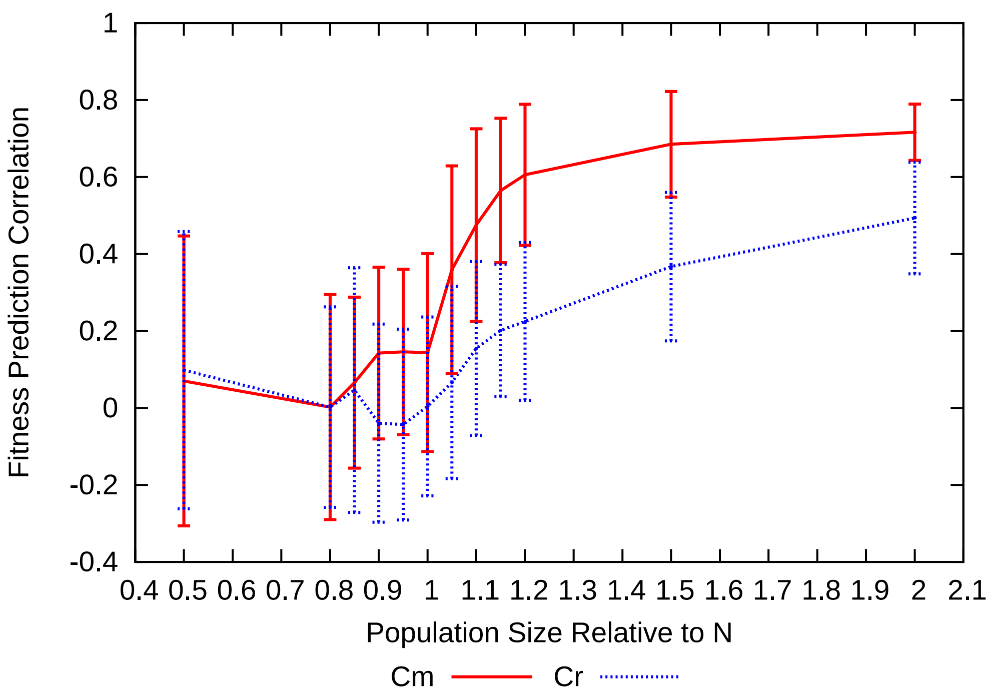


Figure B.91: FPC vs population size for 20 bit MaxSAT problem using a univariate structure



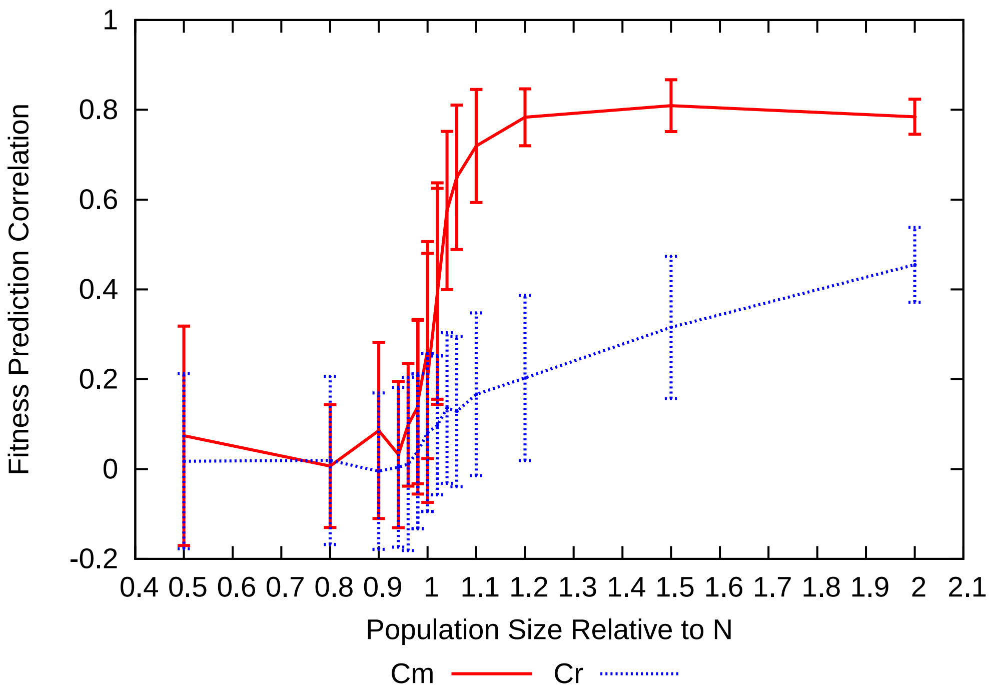


Figure B.92: FPC vs population size for 50 bit MaxSAT problem using a univariate structure

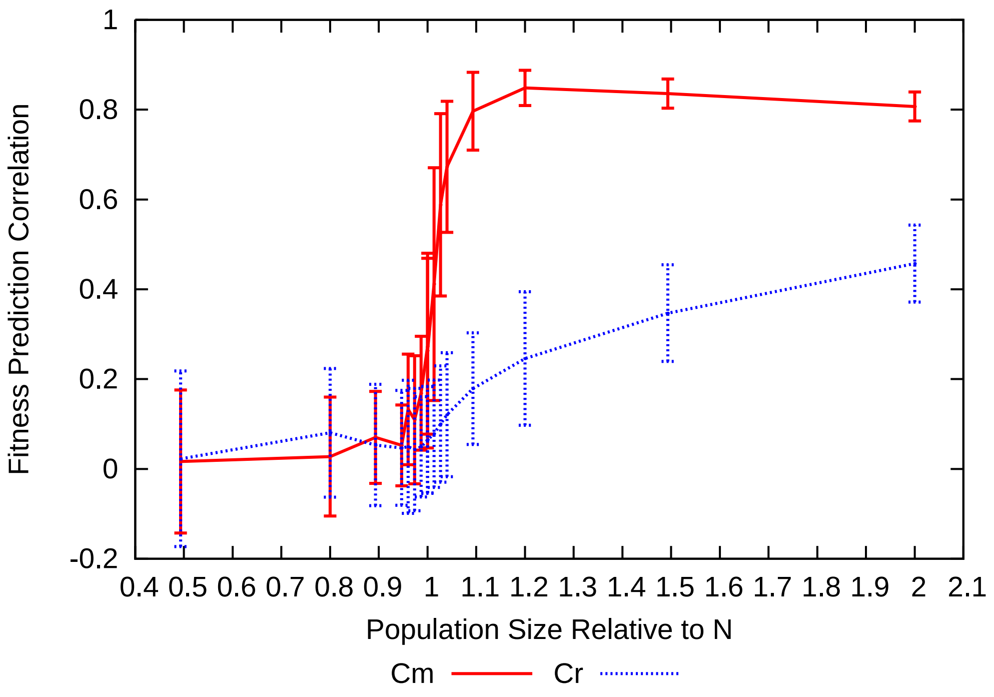


Figure B.93: FPC vs population size for 75 bit MaxSAT problem using a univariate structure

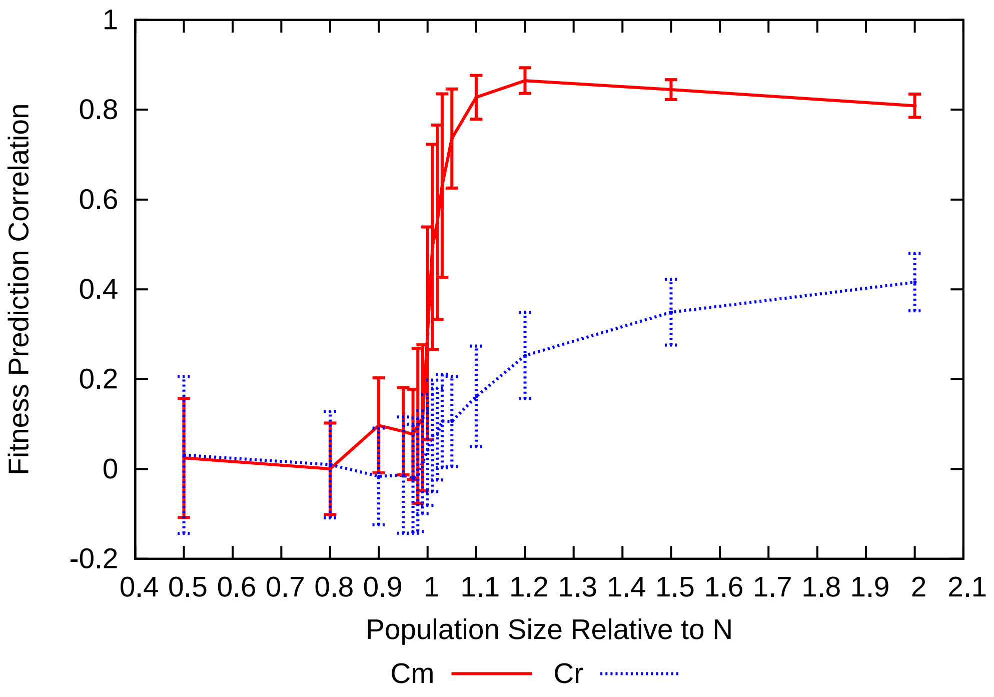


Figure B.94: FPC vs population size for 100 bit MaxSAT problem using a univariate structure

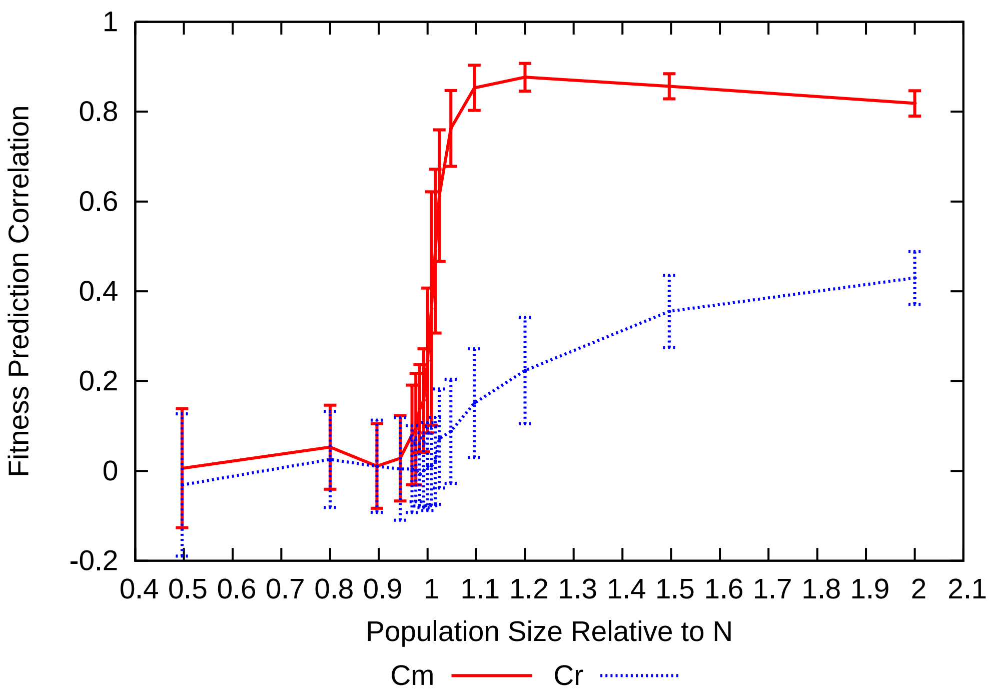


Figure B.95: FPC vs population size for 125 bit MaxSAT problem using a univariate structure

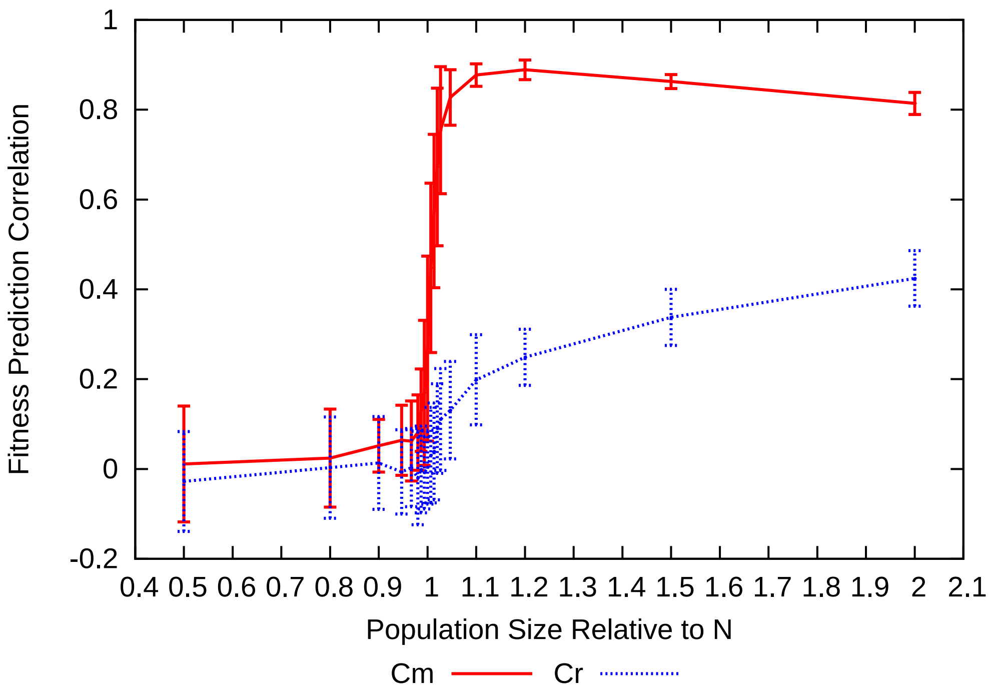


Figure B.96: FPC vs population size for 150 bit MaxSAT problem using a univariate structure

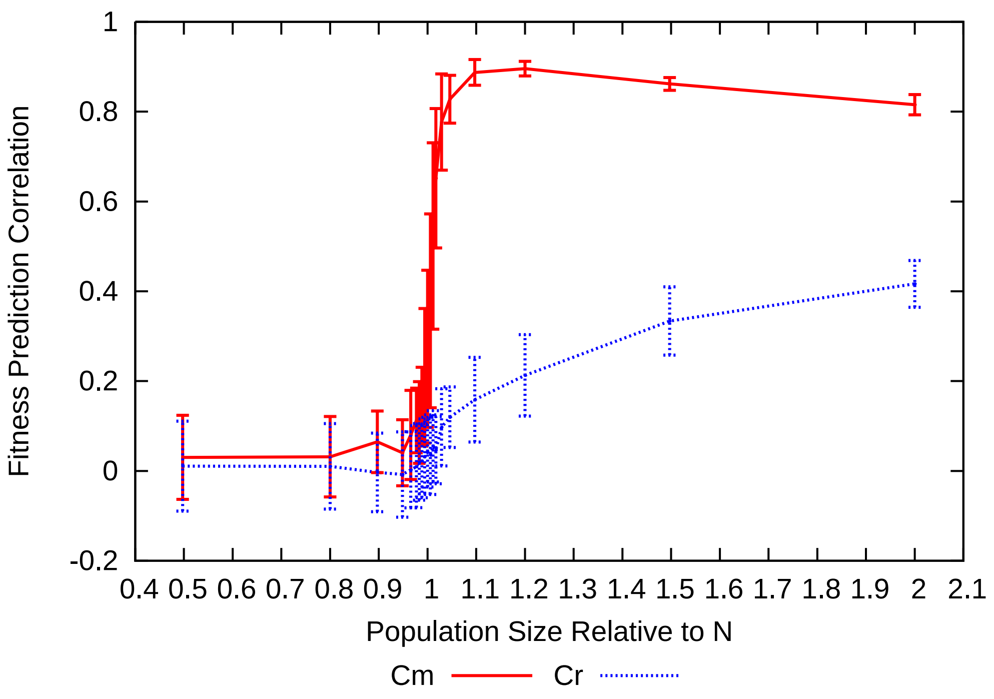


Figure B.97: FPC vs population size for 175 bit MaxSAT problem using a univariate structure

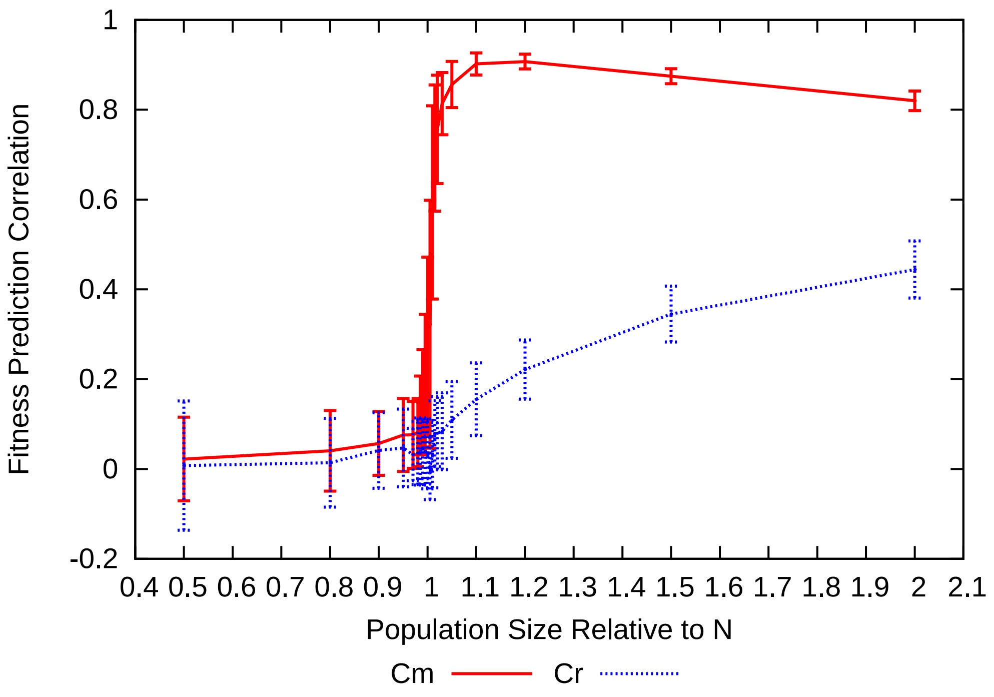
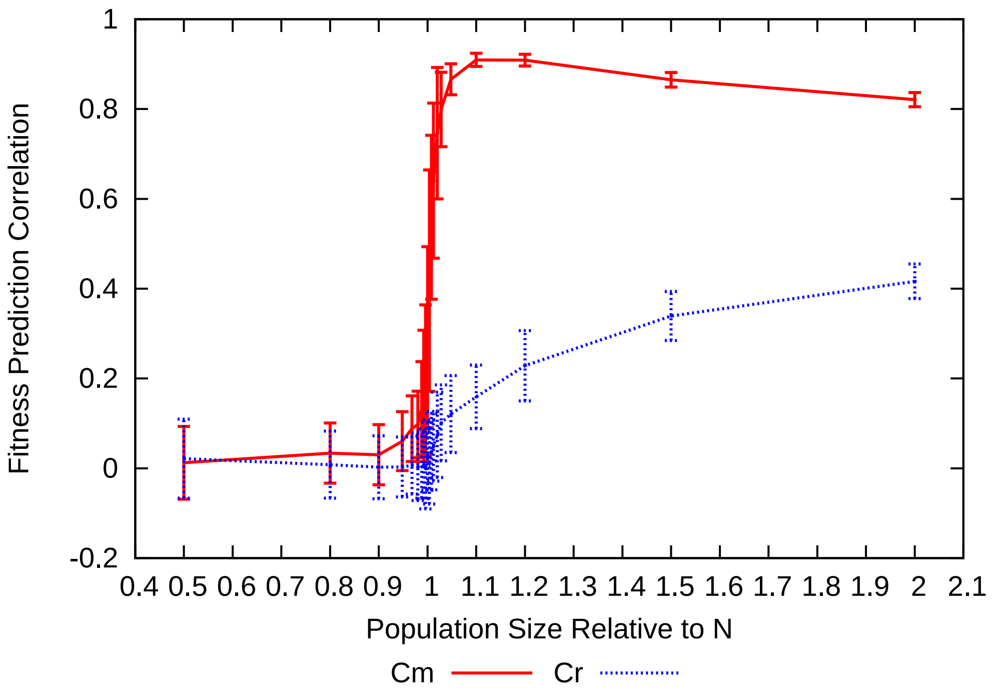


Figure B.98: FPC vs population size for 200 bit MaxSAT problem using a univariate structure

Figure B.99: FPC vs population size for 250 bit MaxSAT problem using a univariate structure



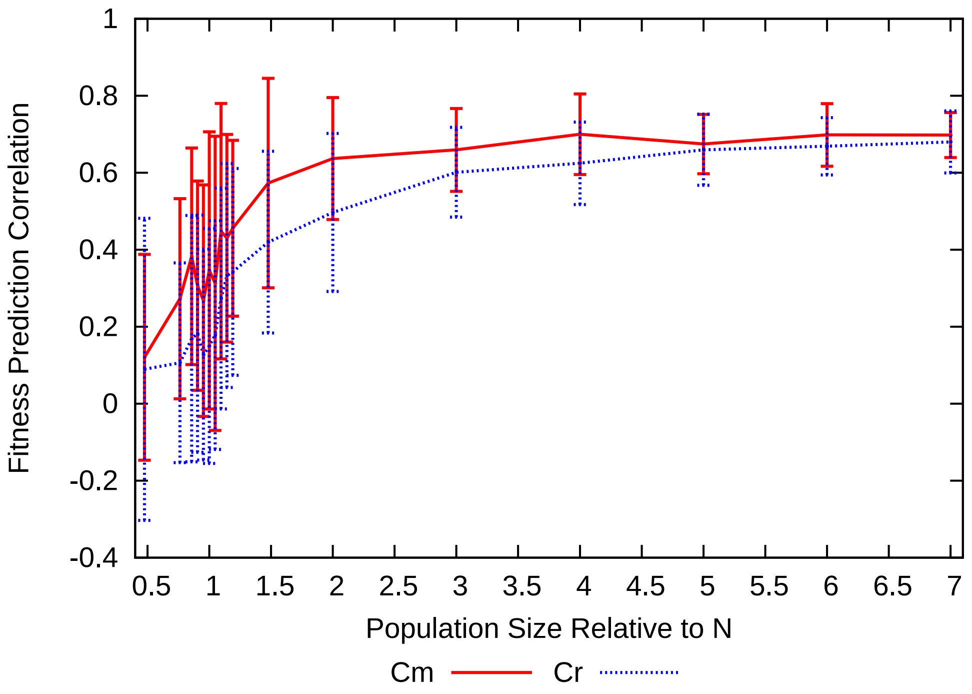


Figure B.100: FPC vs population size for 10 bit Trap-5 problem using a chain model with both univariate and bivariate terms, plus terms for each group of 5 variables

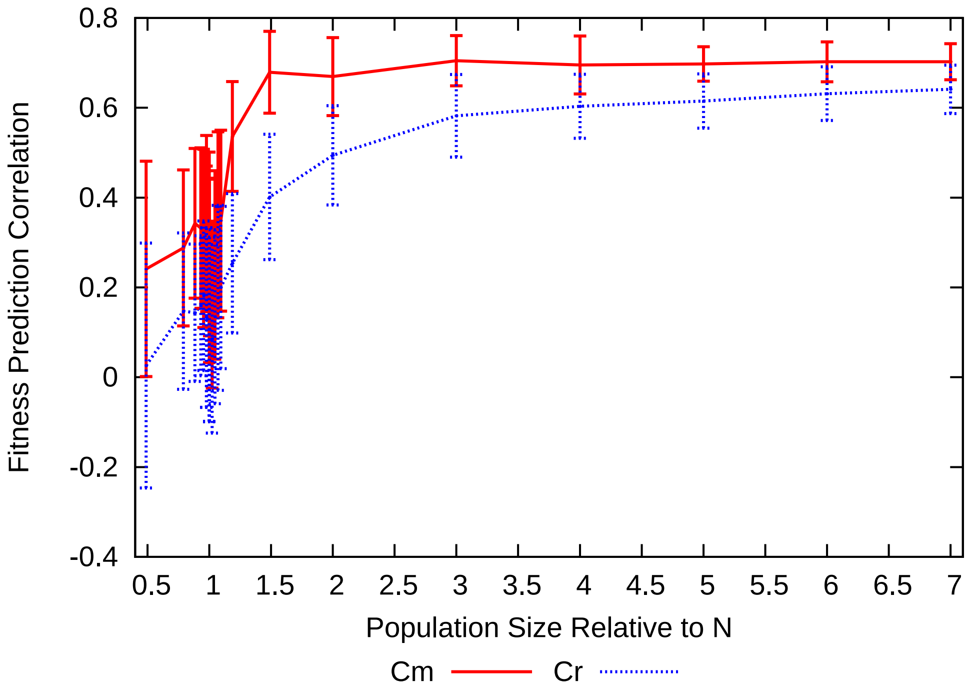


Figure B.101: FPC vs population size for 20 bit Trap-5 problem using a chain model with both univariate and bivariate terms, plus terms for each group of 5 variables

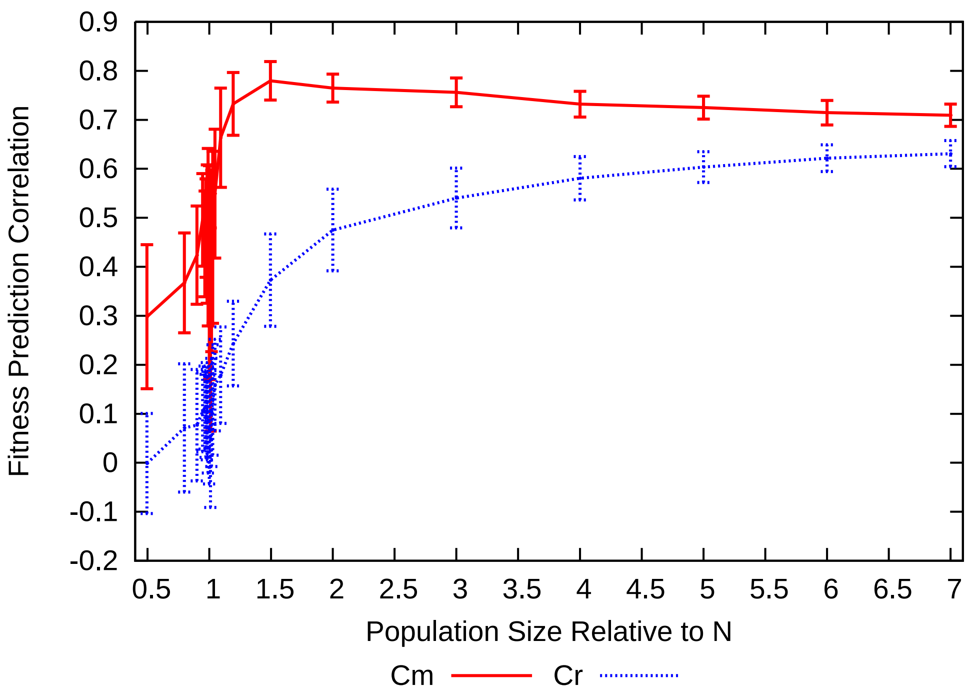


Figure B.102: FPC vs population size for 50 bit Trap-5 problem using a chain model with both univariate and bivariate terms, plus terms for each group of 5 variables

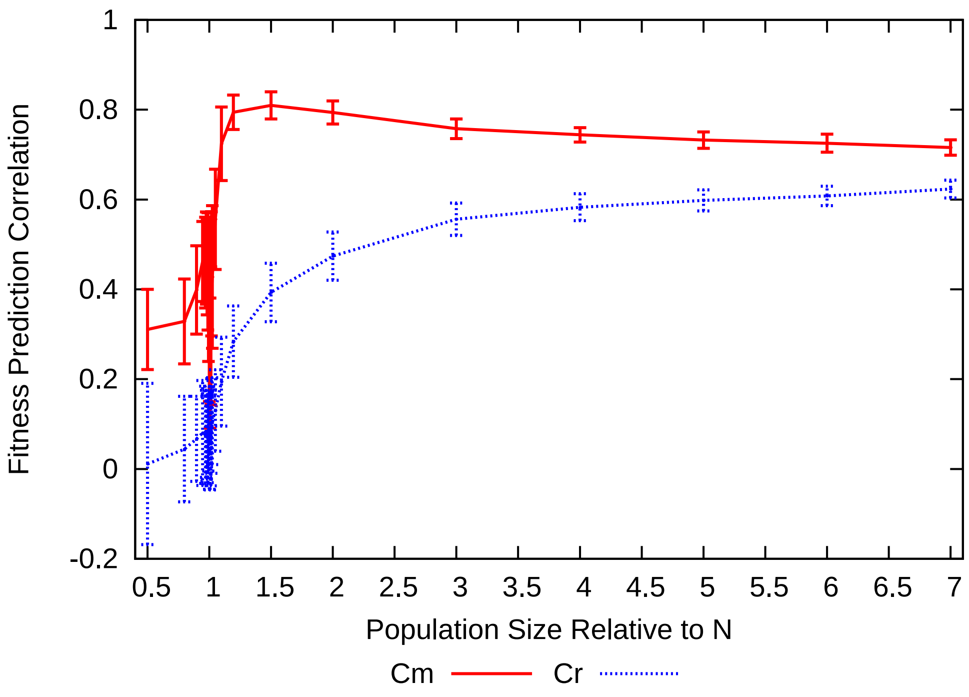


Figure B.103: FPC vs population size for 75 bit Trap-5 problem using a chain model with both univariate and bivariate terms, plus terms for each group of 5 variables

Figure B.104: FPC vs population size for 100 bit Trap-5 problem using a chain model with both univariate and bivariate terms, plus terms for each group of 5 variables

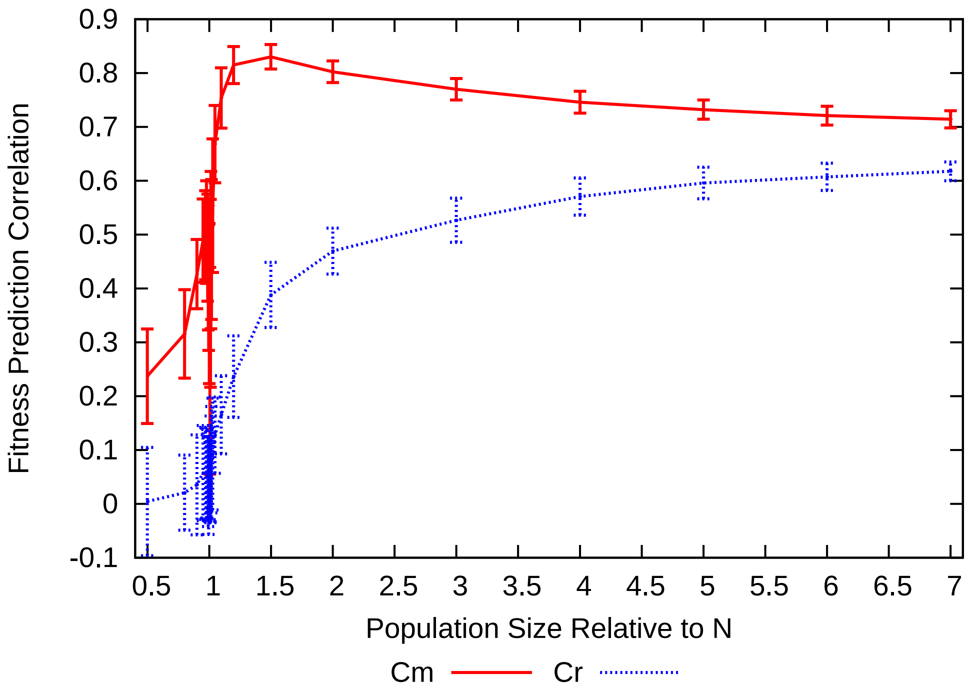
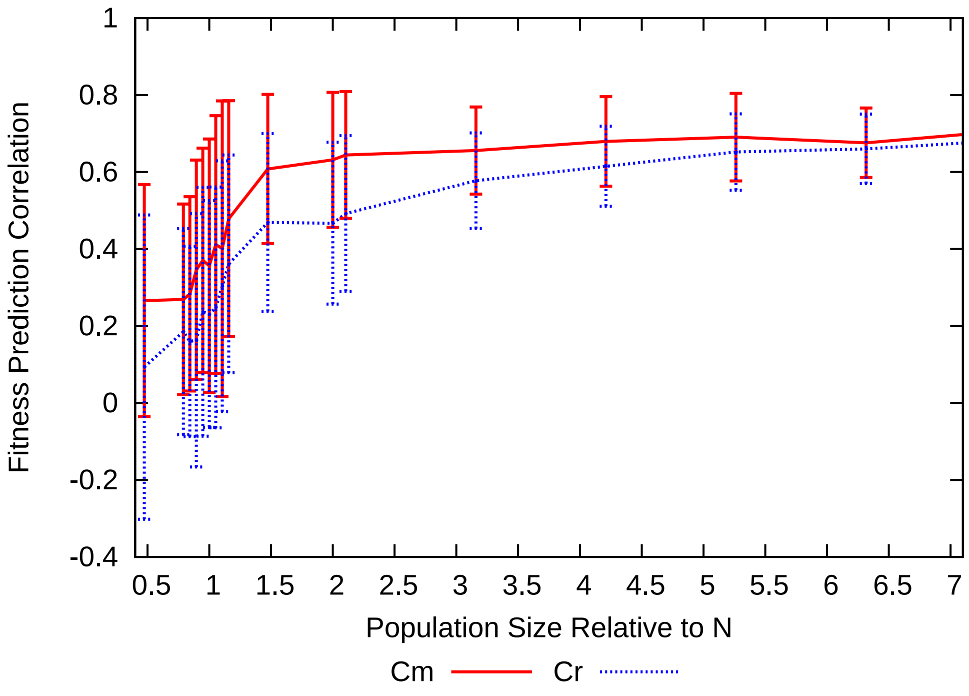


Figure B.105: FPC vs population size for 10 bit Trap-5 problem using a chain model with both univariate and bivariate terms



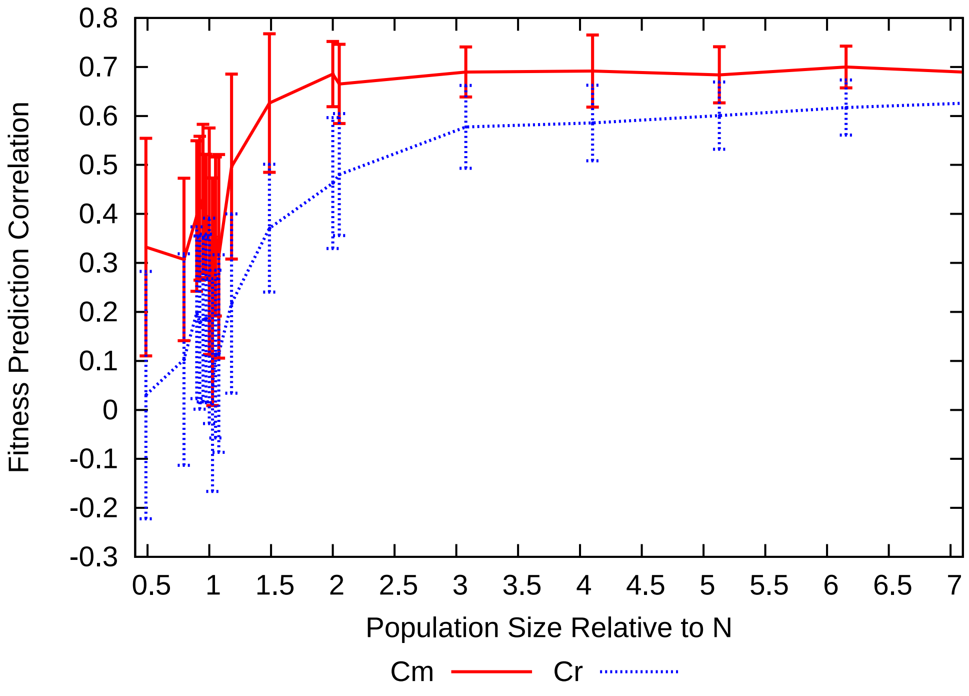


Figure B.106: FPC vs population size for 20 bit Trap-5 problem using a chain model with both univariate and bivariate terms

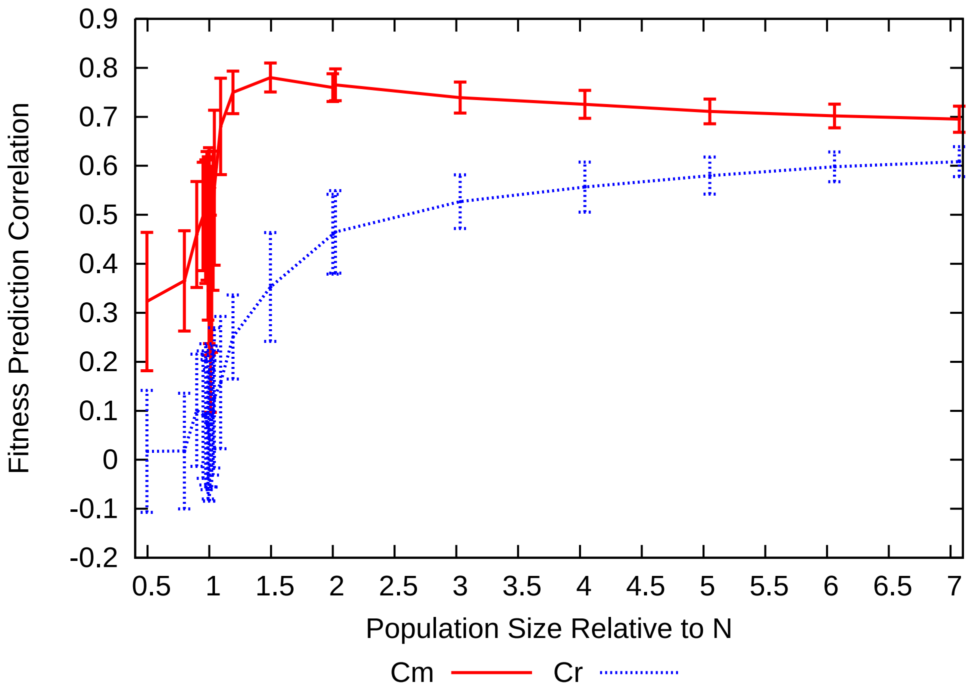


Figure B.107: FPC vs population size for 50 bit Trap-5 problem using a chain model with both univariate and bivariate terms

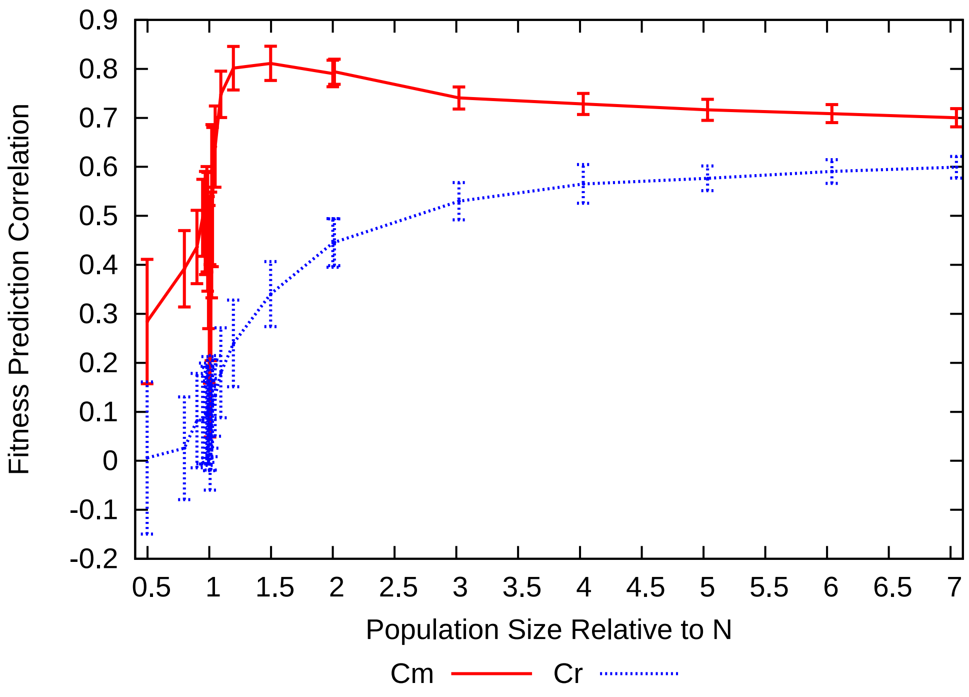


Figure B.108: FPC vs population size for 75 bit Trap-5 problem using a chain model with both univariate and bivariate terms

Figure B.109: FPC vs population size for 100 bit Trap-5 problem using a chain model with both univariate and bivariate terms

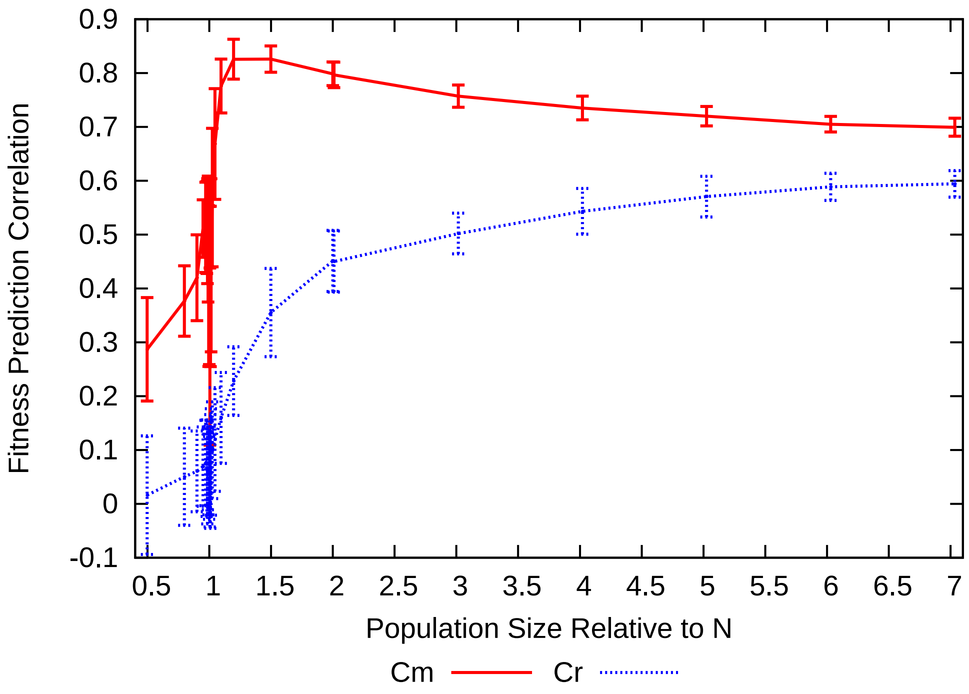
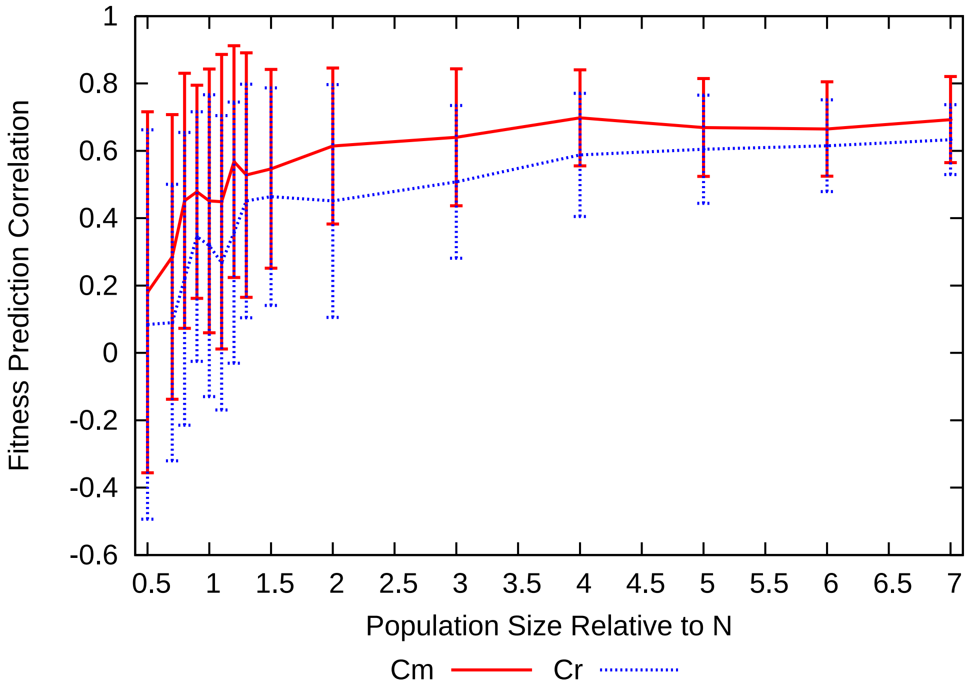


Figure B.110: FPC vs population size for 10 bit Trap-5 problem using a univariate model



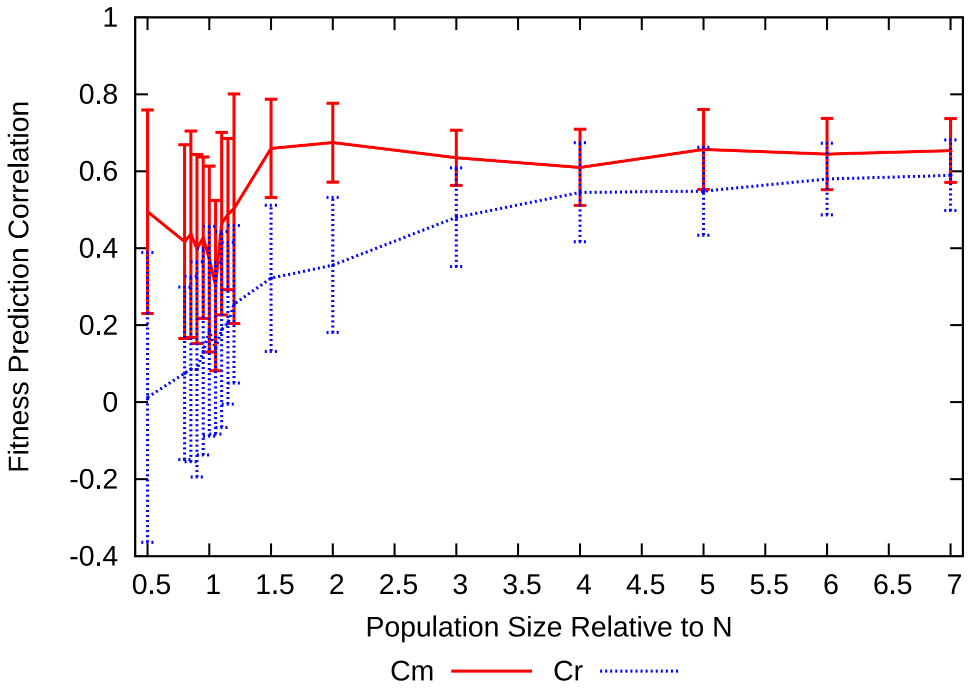


Figure B.111: FPC vs population size for 20 bit Trap-5 problem using a univariate model

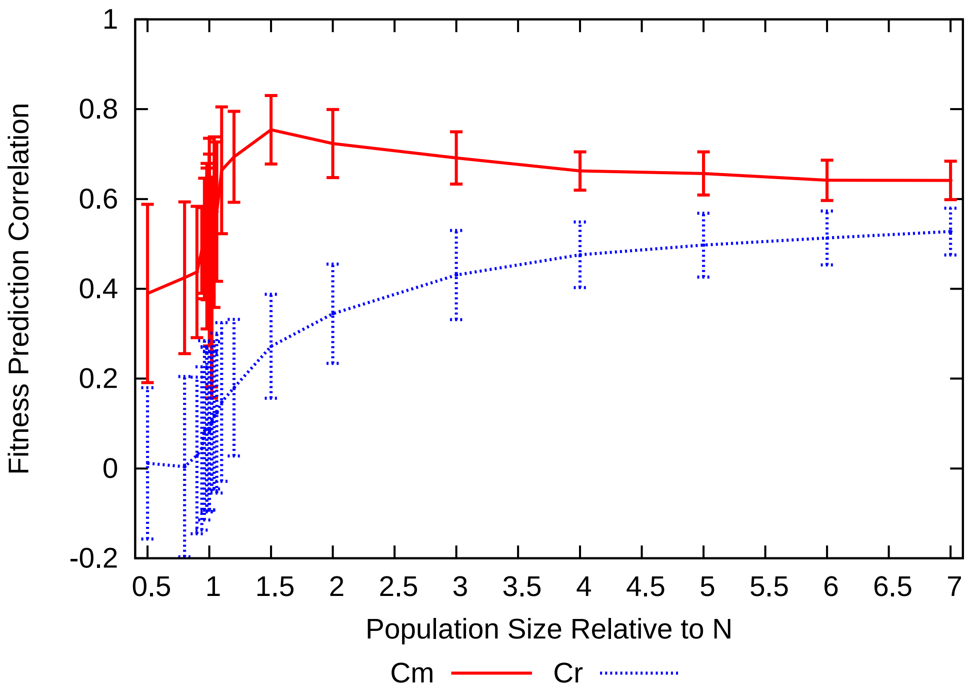


Figure B.112: FPC vs population size for 50 bit Trap-5 problem using a univariate model

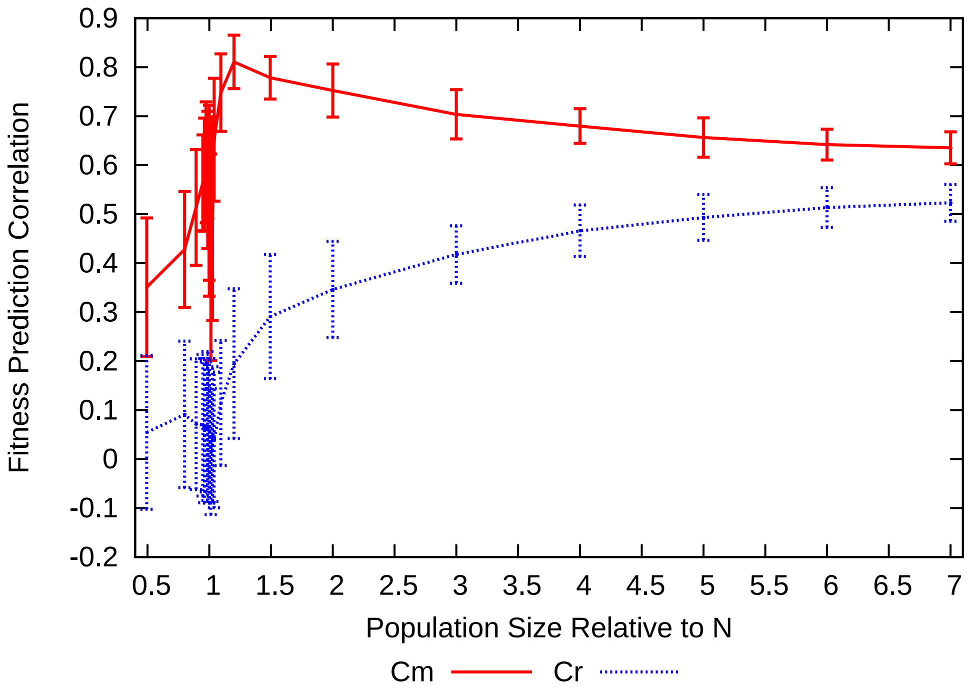


Figure B.113: FPC vs population size for 75 bit Trap-5 problem using a univariate model

Figure B.114: FPC vs population size for 100 bit Trap-5 problem using a univariate model

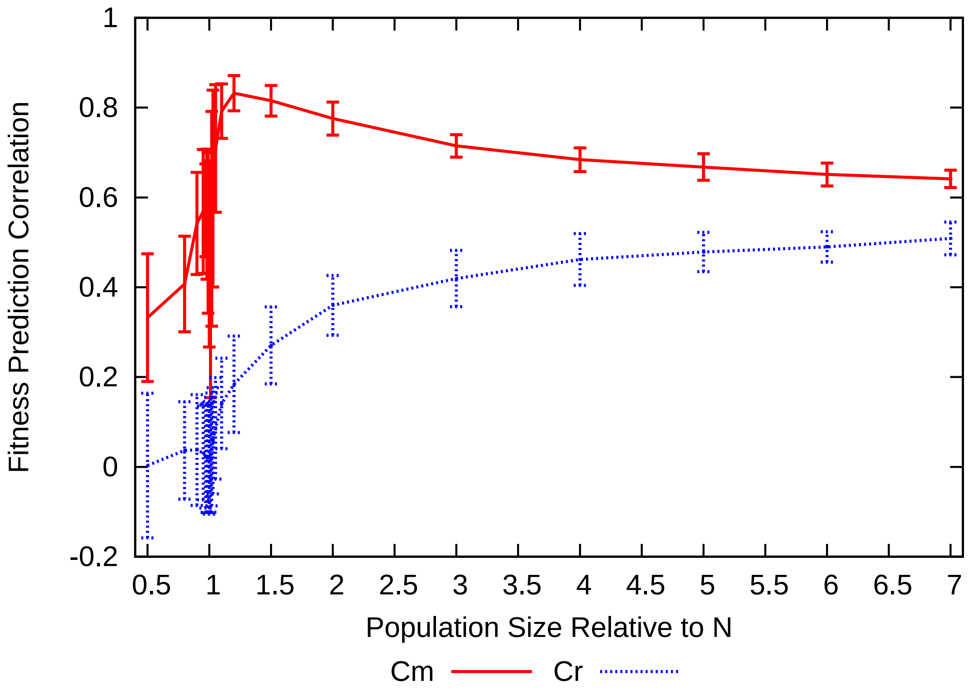
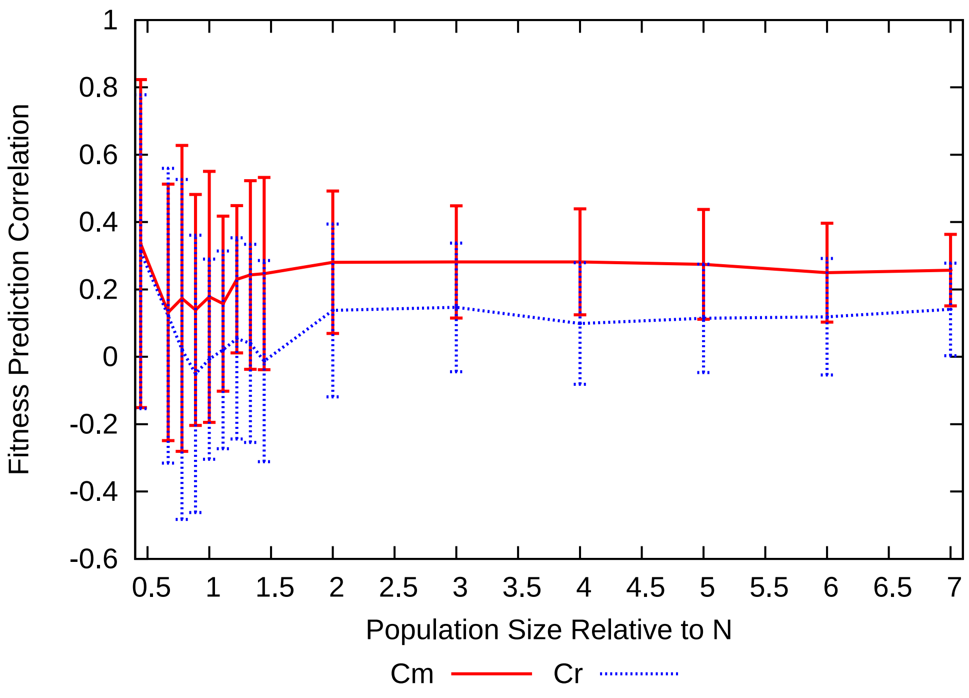


Figure B.115: FPC against population size for 10 bit problem using a chain model with no univariate terms



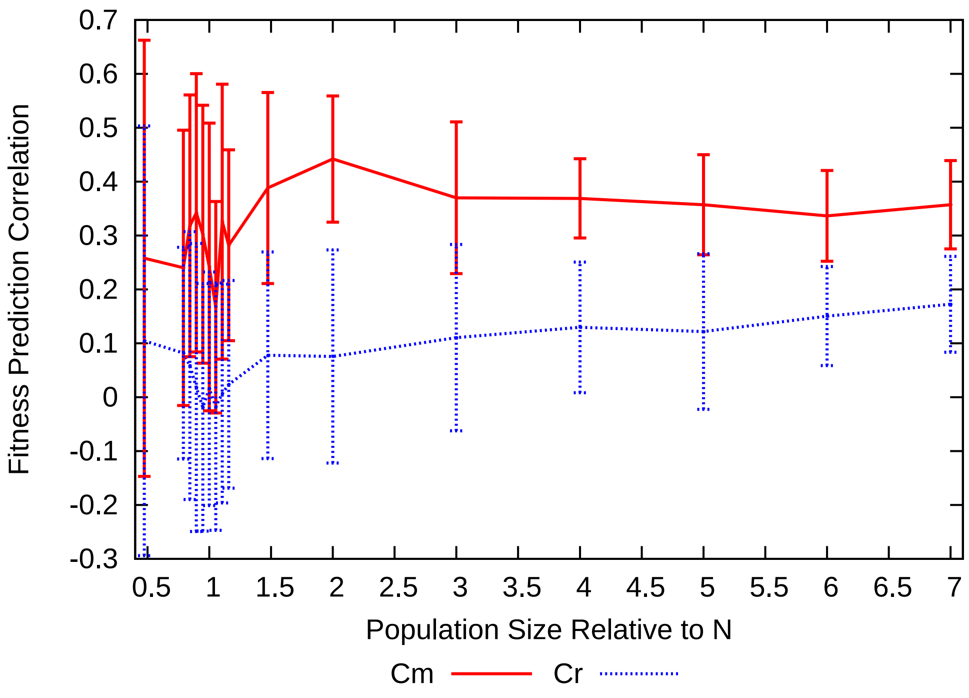


Figure B.116: FPC against population size for 20 bit Trap-5 problem using a chain model with no univariate terms

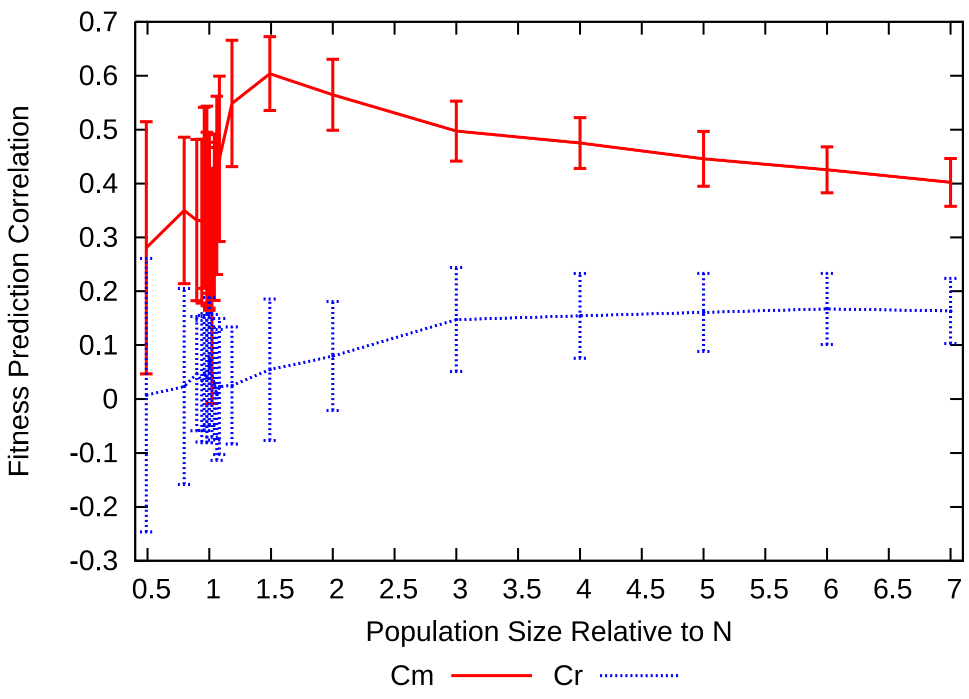


Figure B.117: FPC against population size for 50 bit Trap-5 problem using a chain model with no univariate terms

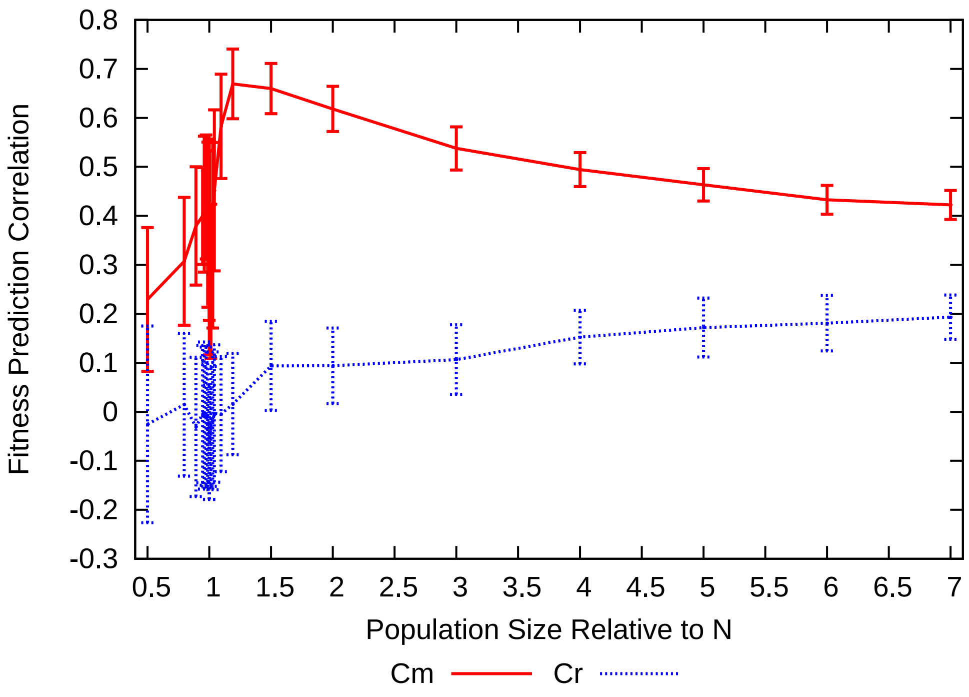
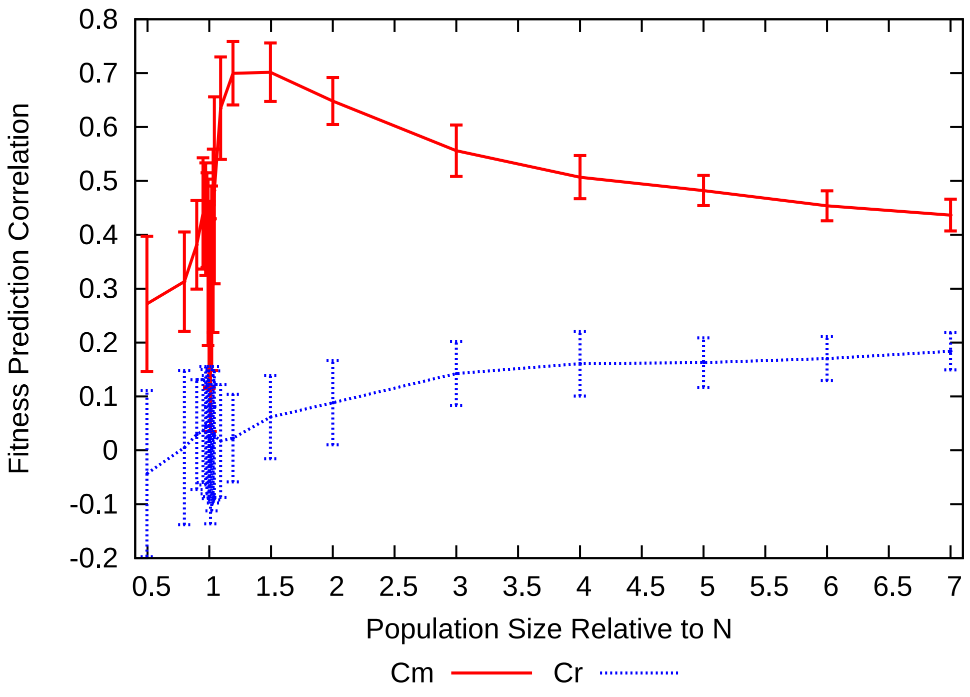


Figure B.118: FPC against population size for 75 bit Trap-5 problem using a chain model with no univariate terms

Figure B.119: FPC against population size for 100 bit Trap-5 problem using a chain model with no univariate terms



Appendix C

Figures - Selection Experiments

The figures in this chapter relate to the experiments discussed in Chapter 4.

Figure C.1: FPC against selection proportion for fully specified 10 bit OneMax

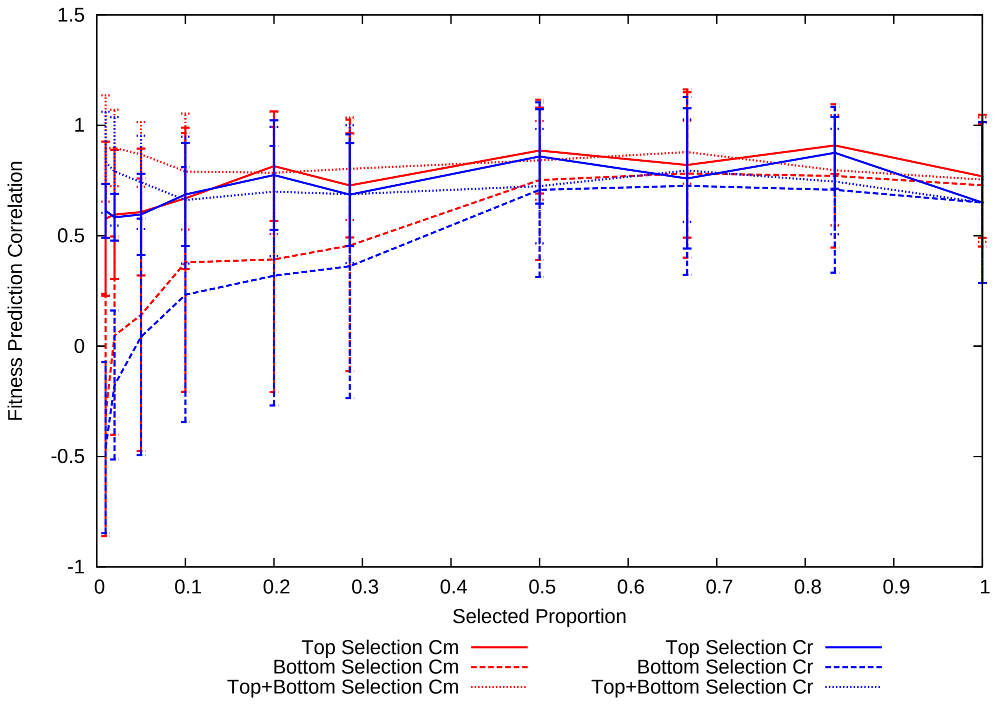


Figure C.2: FPC against selection proportion for fully specified 20 bit OneMax

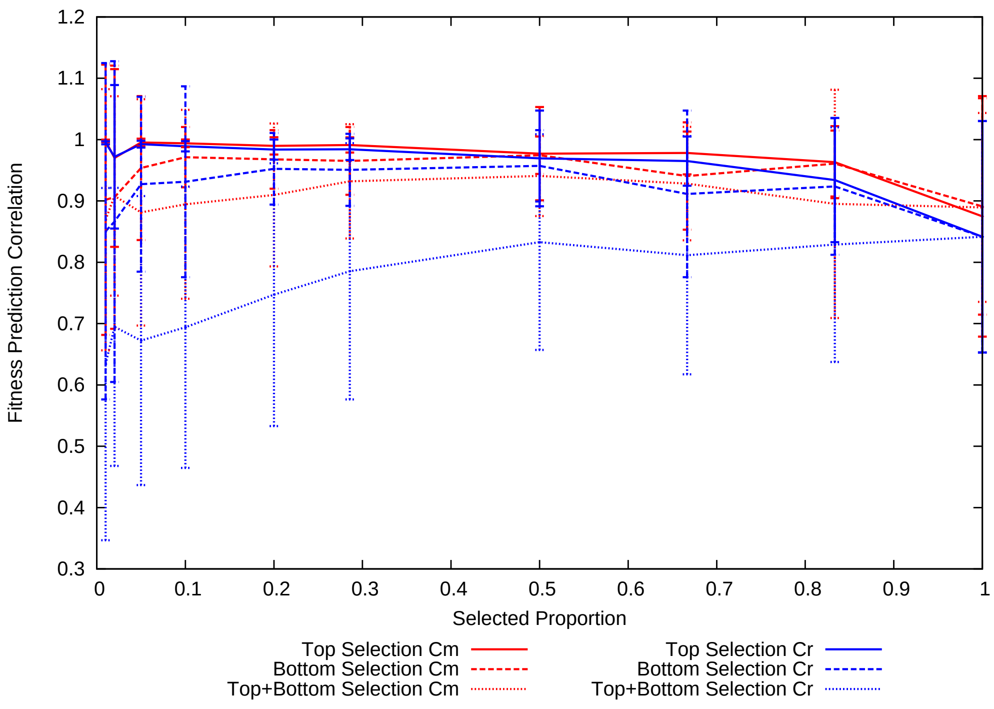


Figure C.3: FPC against selection proportion for fully specified 50 bit OneMax

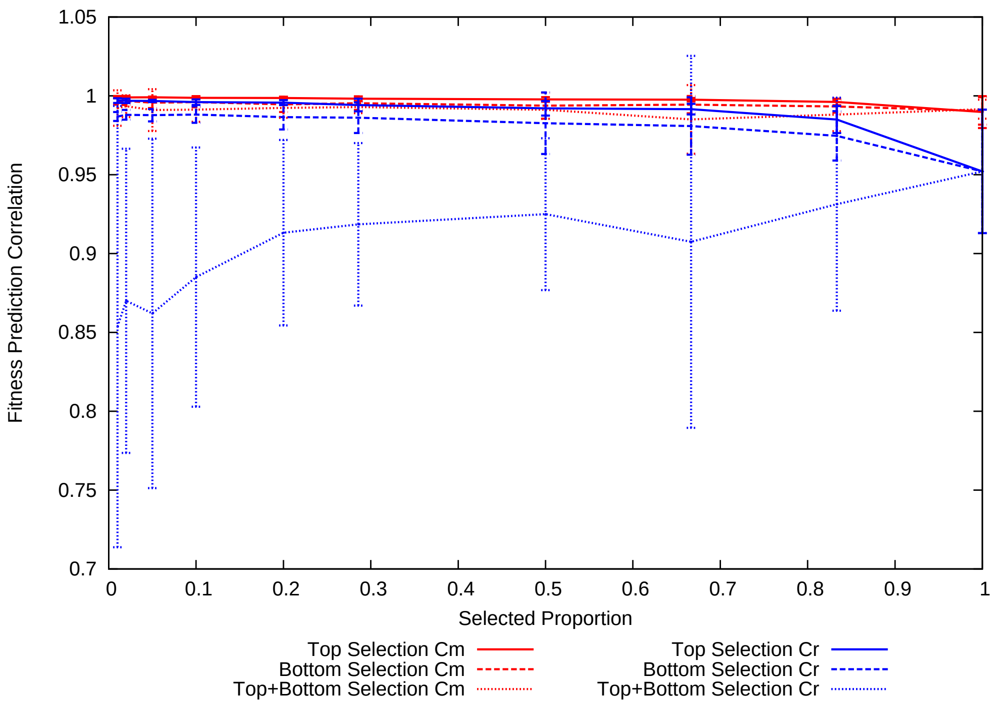


Figure C.4: FPC against selection proportion for fully specified 75 bit OneMax

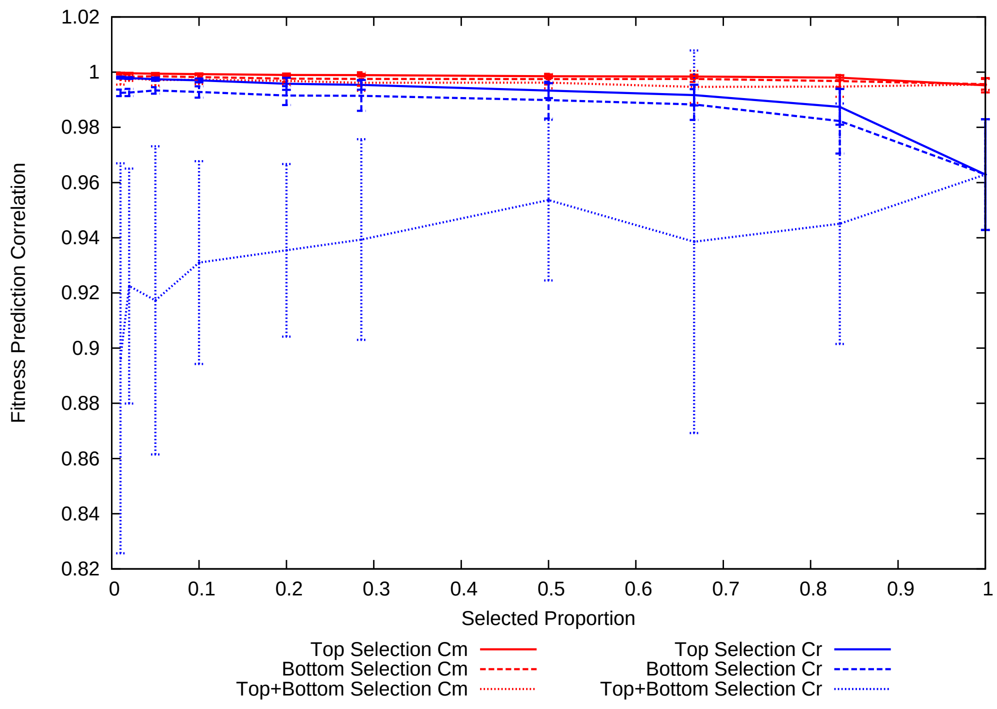


Figure C.5: FPC against selection proportion for fully specified 100 bit OneMax

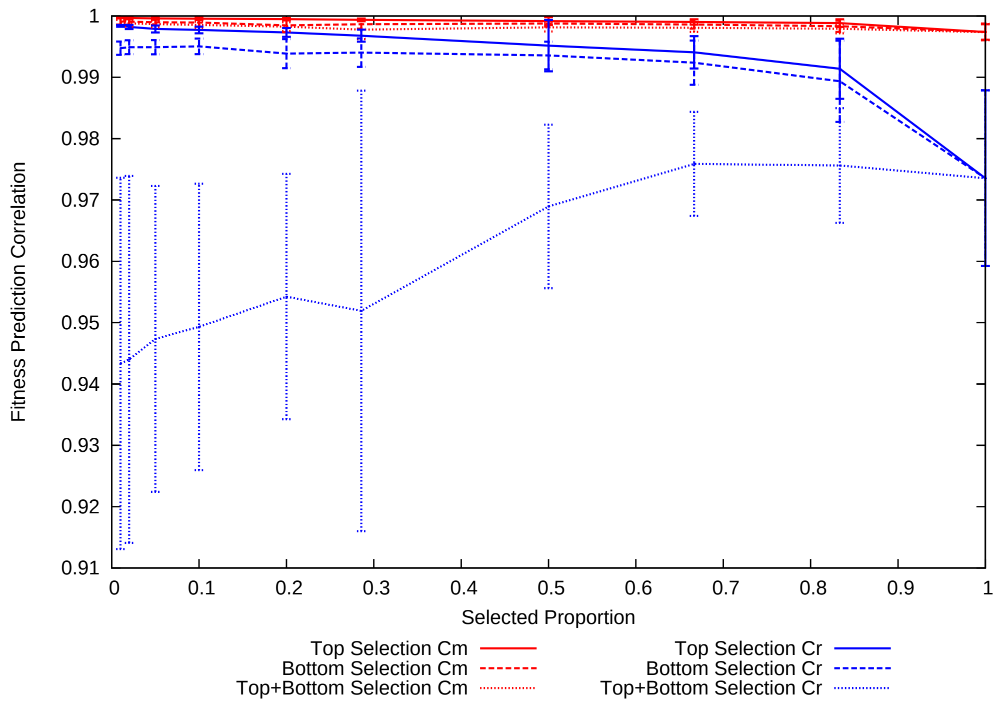


Figure C.6: FPC against selection proportion for fully specified 200 bit OneMax

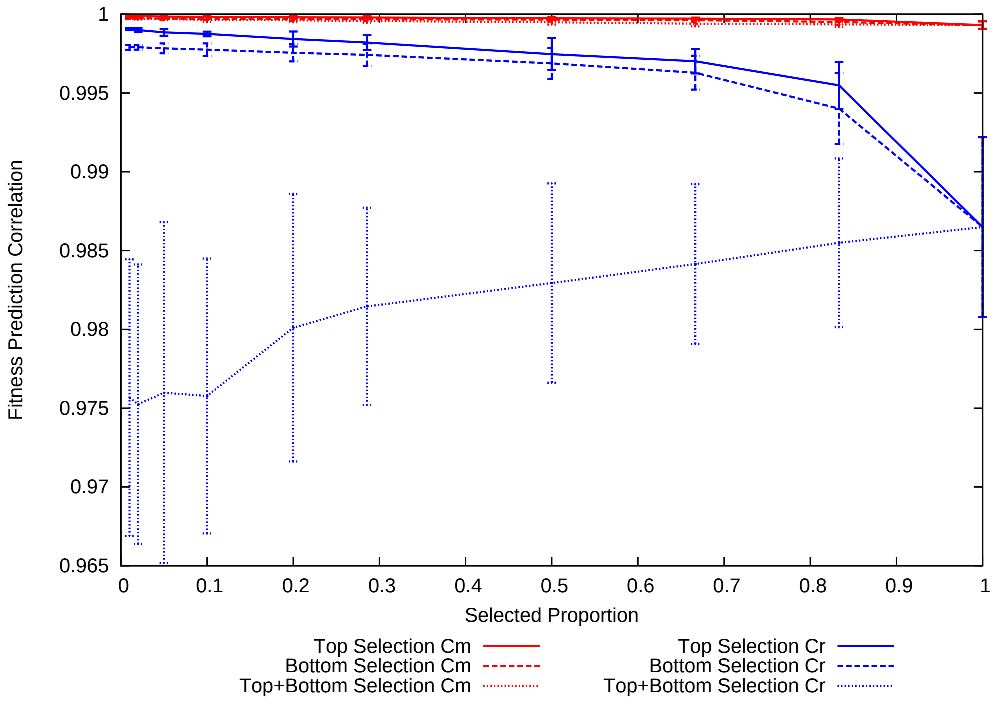


Figure C.7: FPC against selection proportion for fully specified 500 bit OneMax

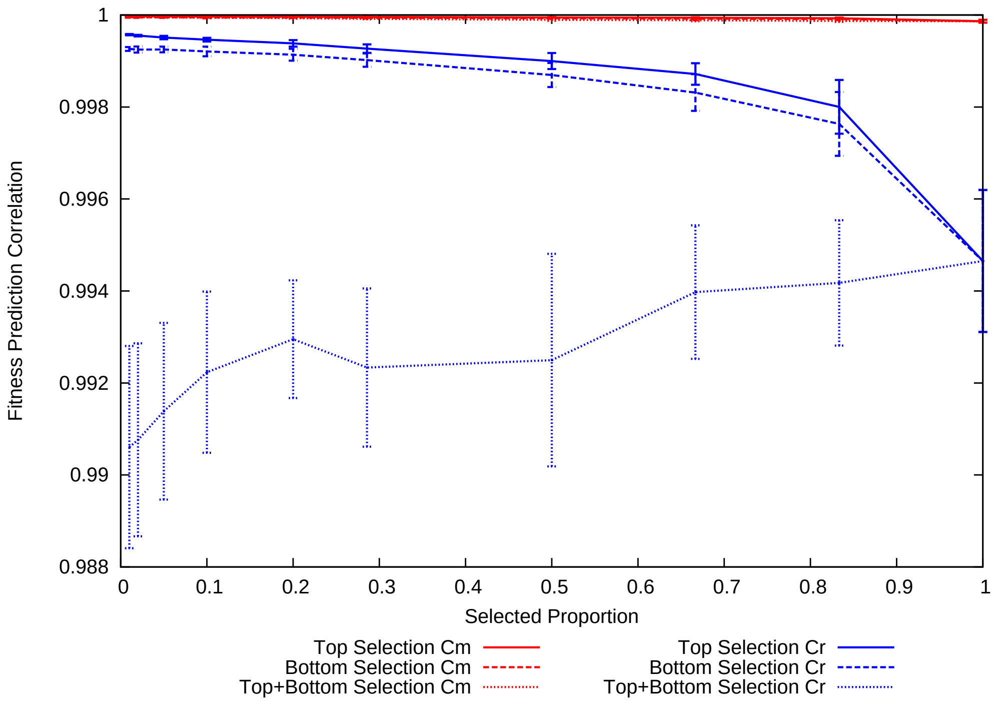


Figure C.8: FPC against selection proportion for fully specified 750 bit OneMax

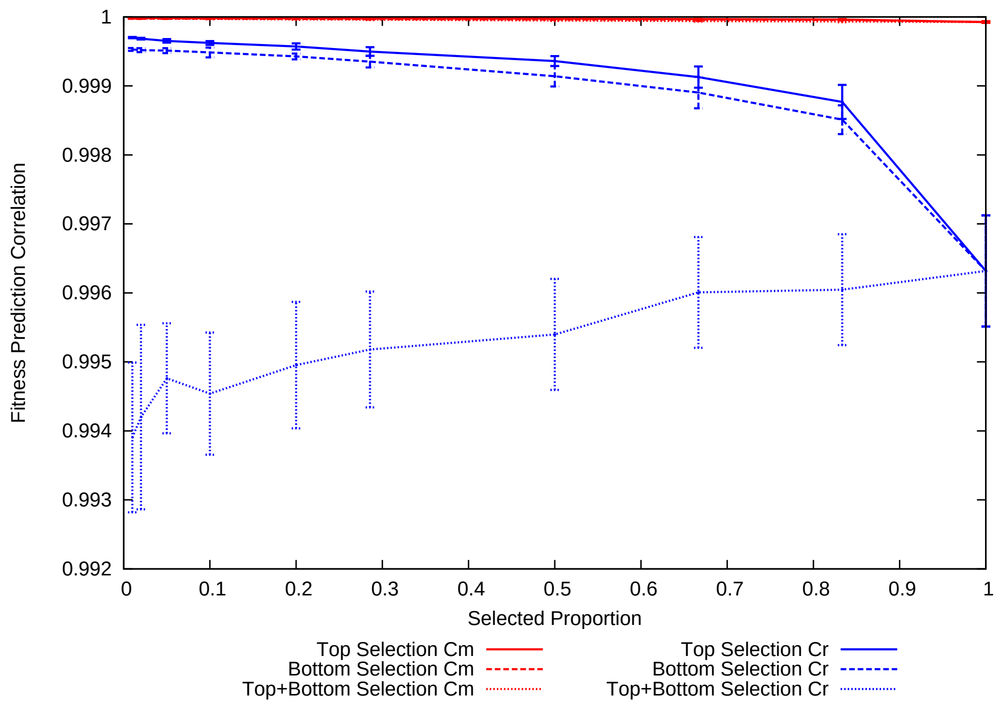


Figure C.9: FPC against selection proportion for fully specified 1000 bit OneMax

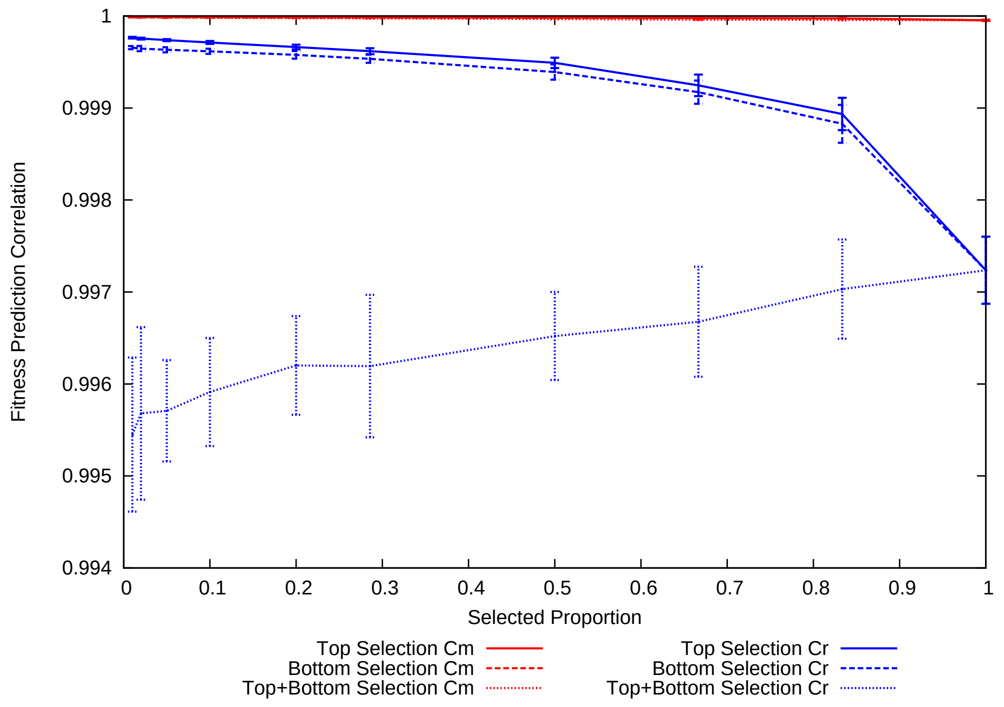


Figure C.10: FPC against selection proportion for fully specified 16 bit 2D Ising

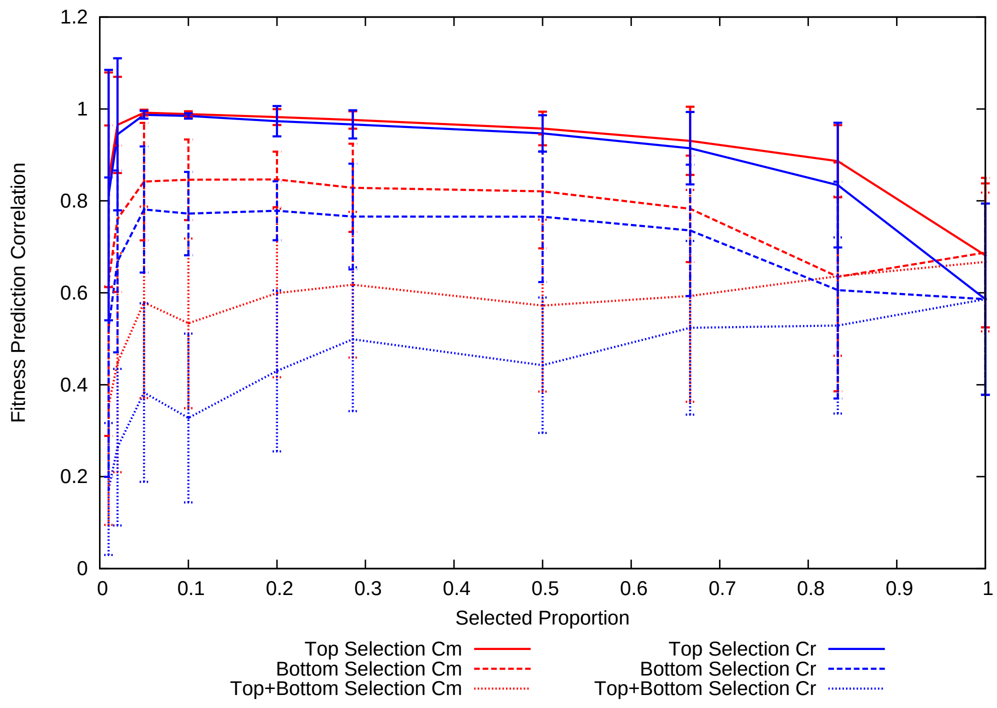


Figure C.11: FPC against selection proportion for fully specified 25 bit 2D Ising

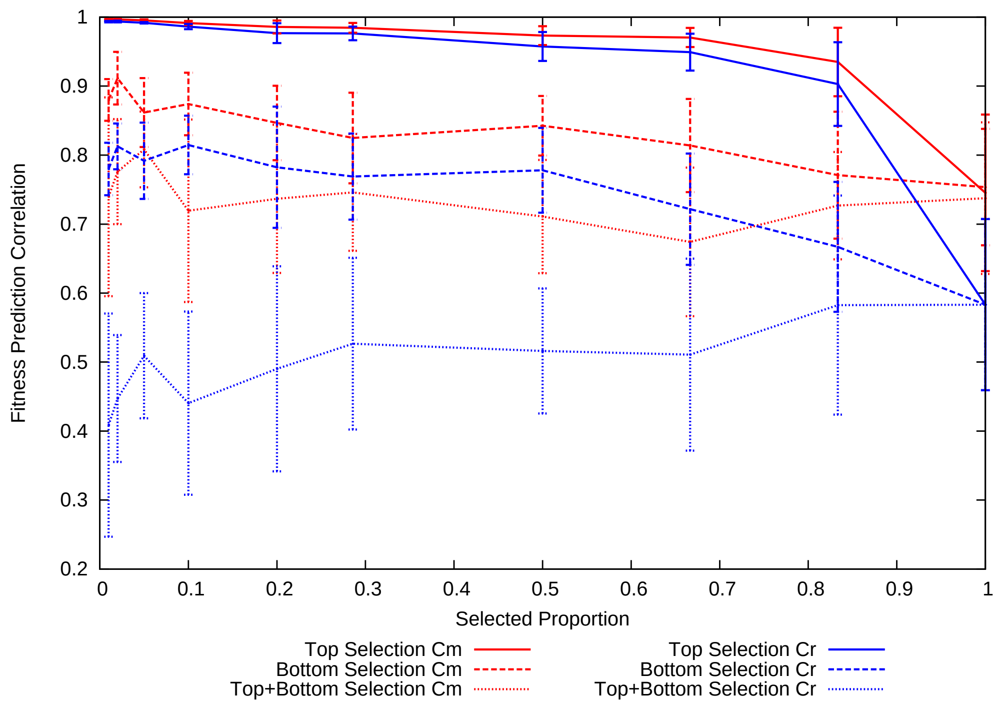


Figure C.12: FPC against selection proportion for fully specified 36 bit 2D Ising

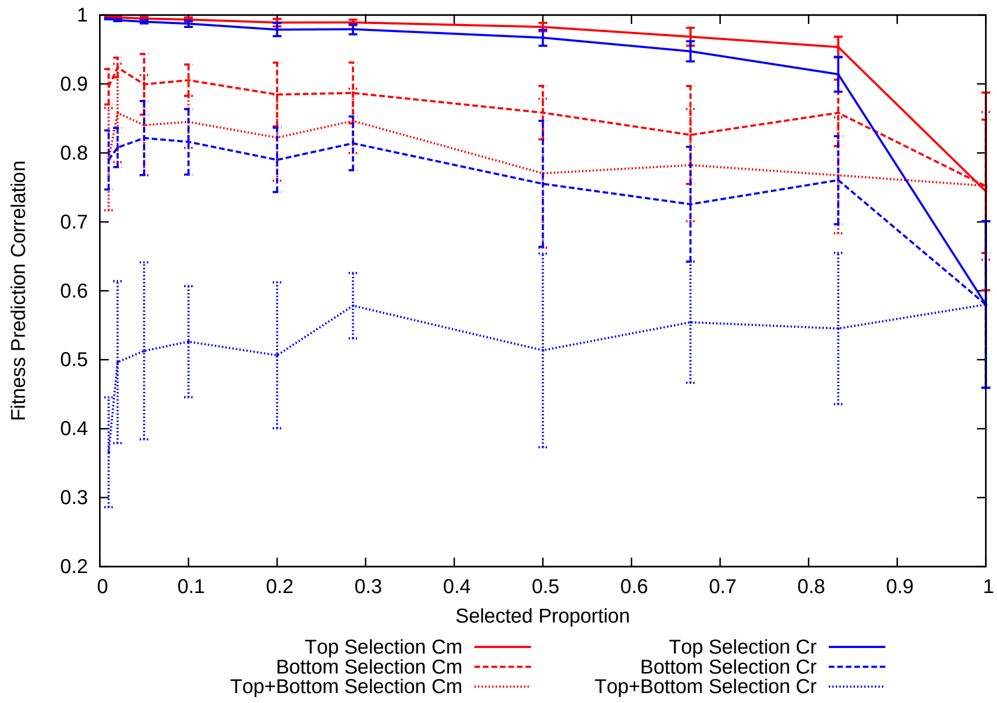


Figure C.13: FPC against selection proportion for fully specified 49 bit 2D Ising

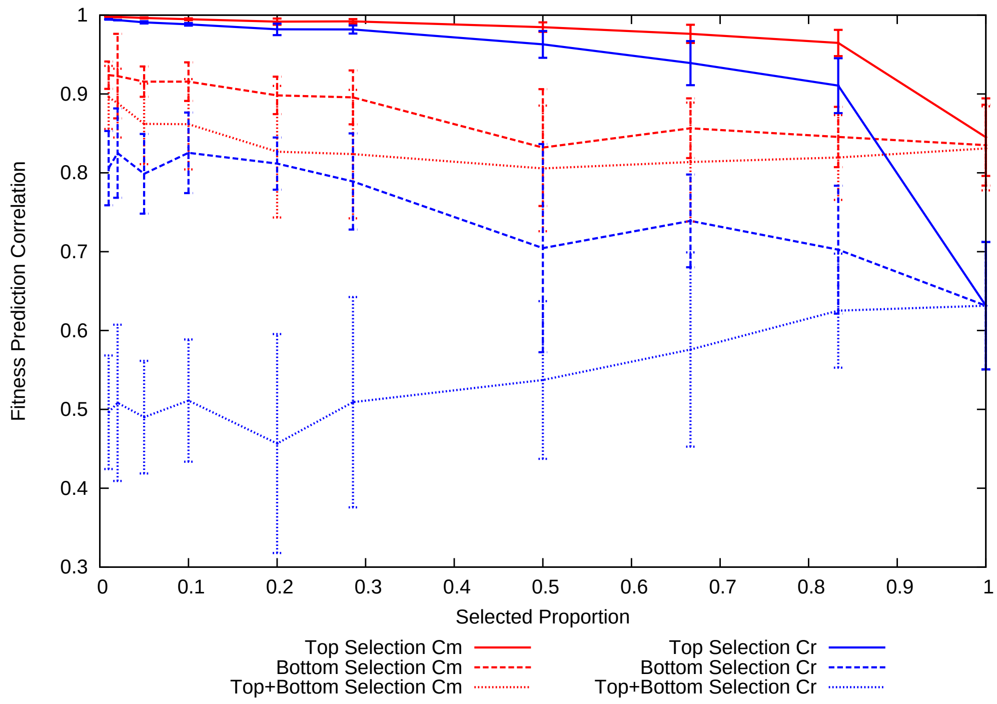


Figure C.14: FPC against selection proportion for fully specified 64 bit 2D Ising

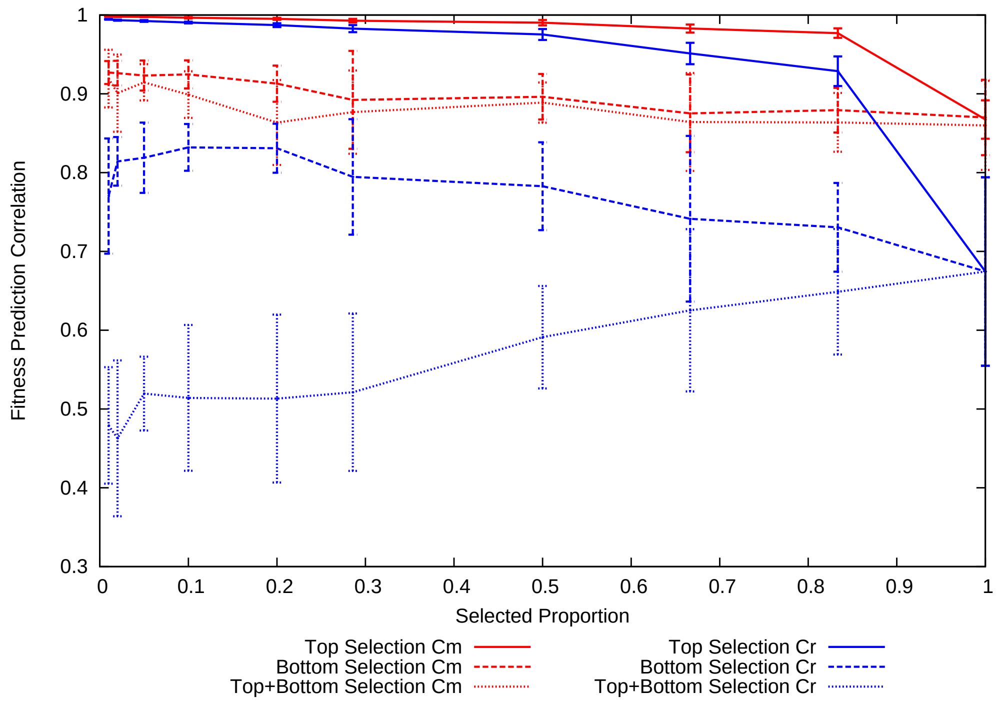


Figure C.15: FPC against selection proportion for fully specified 100 bit 2D Ising

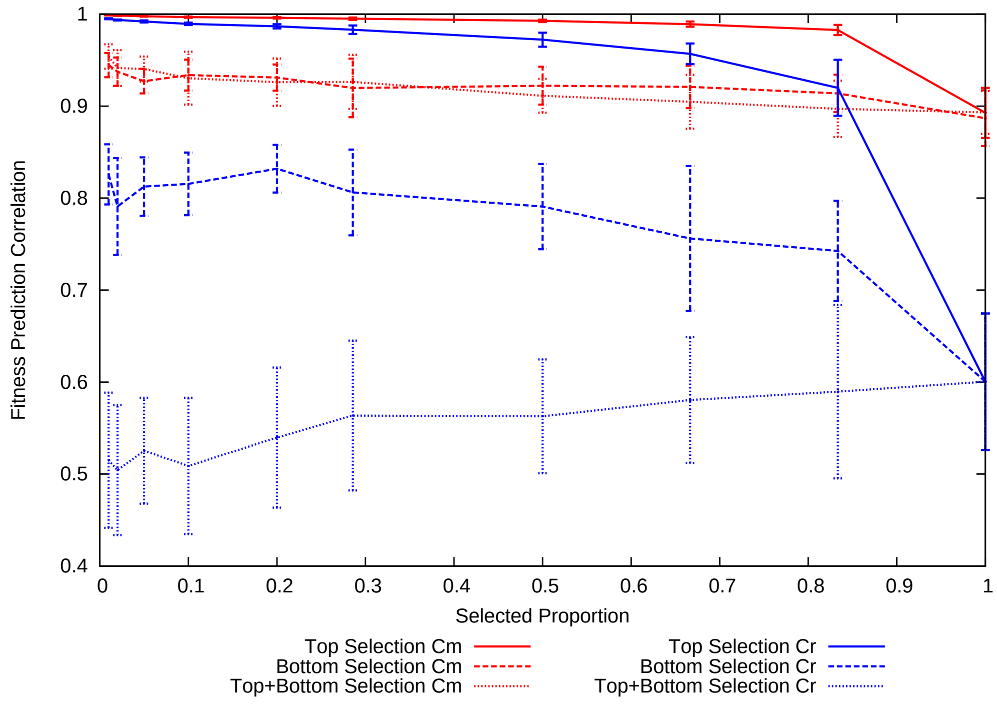


Figure C.16: FPC against selection proportion for fully specified 256 bit 2D Ising

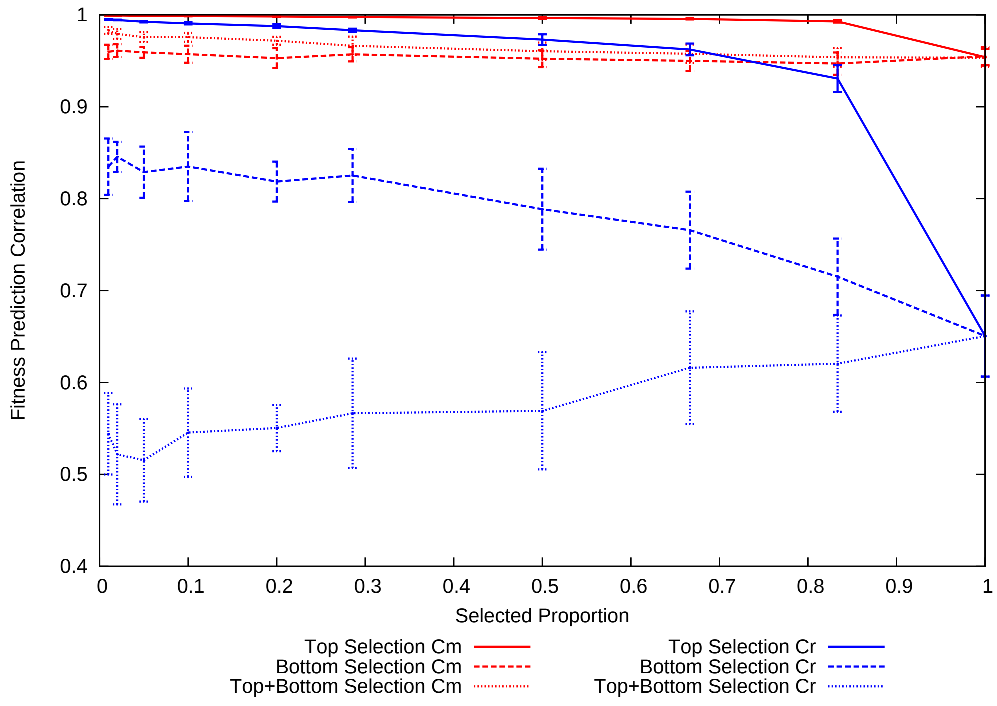


Figure C.17: FPC against selection proportion for fully specified 324 bit 2D Ising

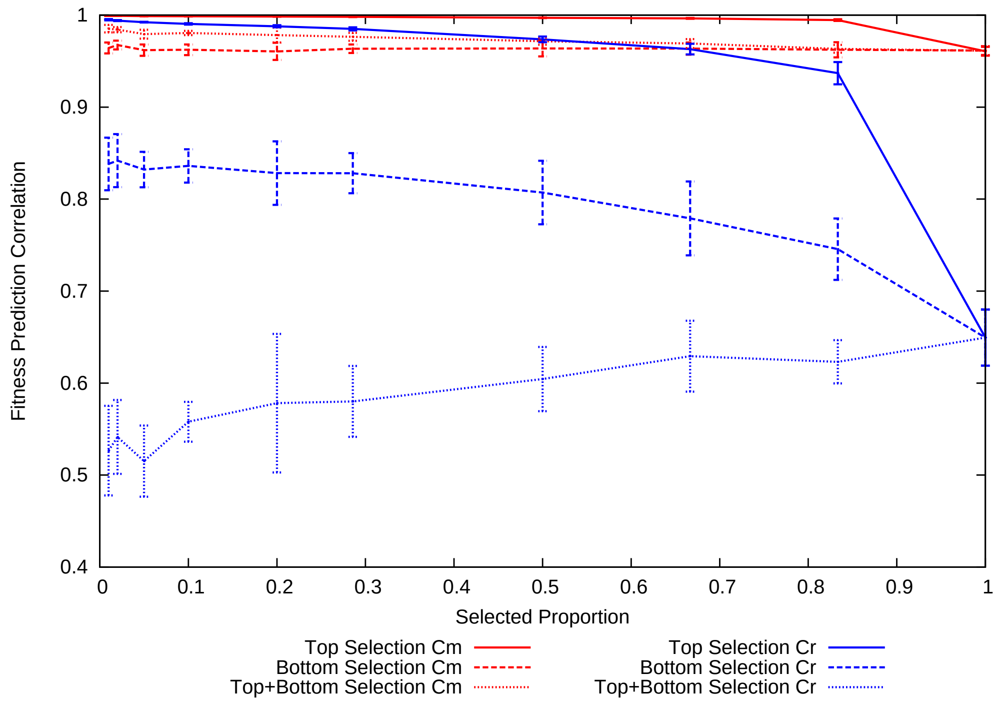


Figure C.18: FPC against selection proportion for fully specified 400 bit 2D Ising

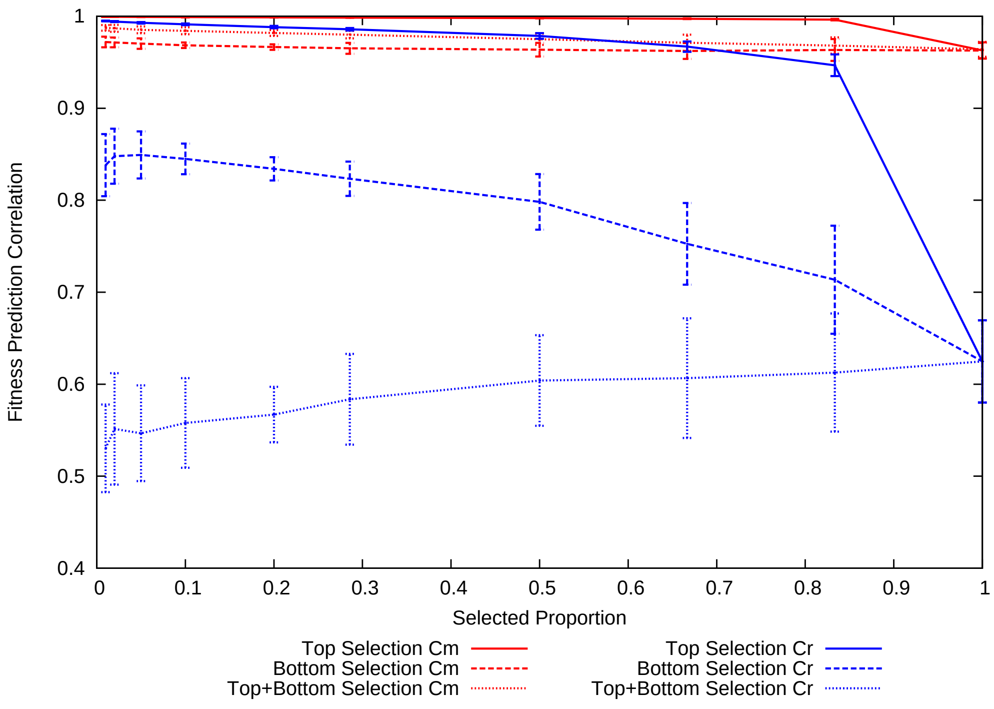


Figure C.19: FPC against selection proportion for fully specified 20 bit MaxSAT

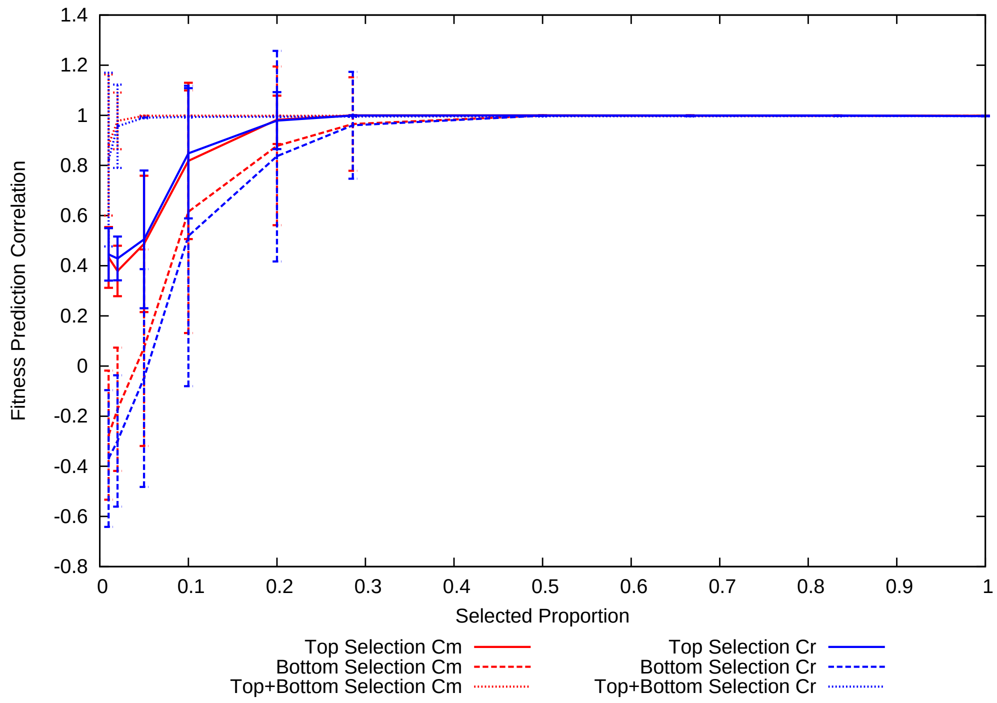


Figure C.20: FPC against selection proportion for fully specified 50 bit MaxSAT

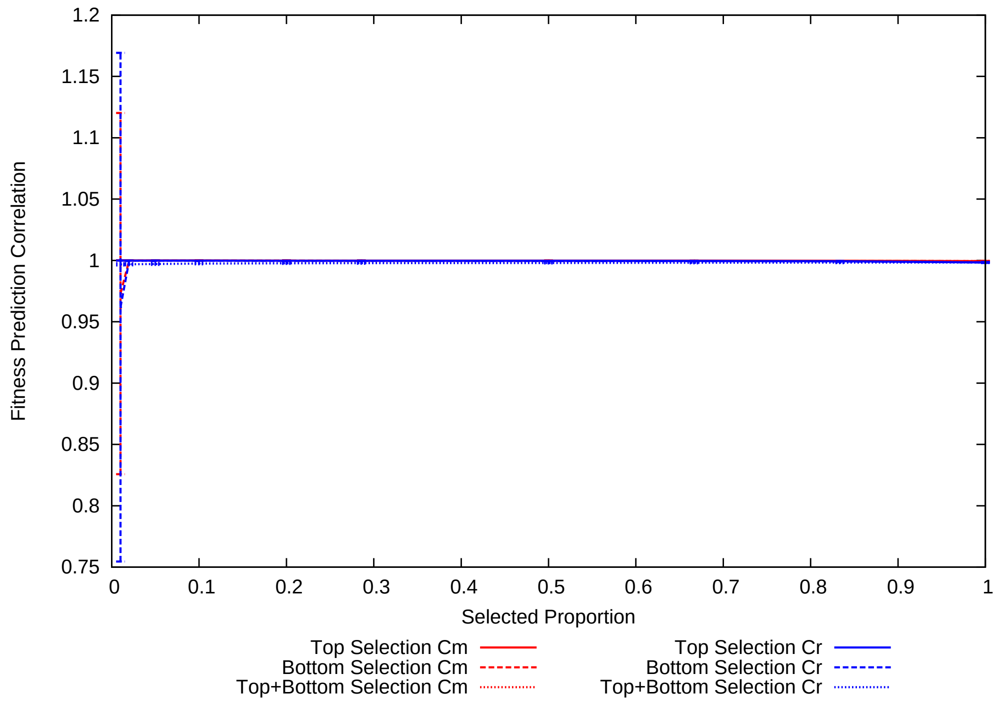


Figure C.21: FPC against selection proportion for fully specified 75 bit MaxSAT

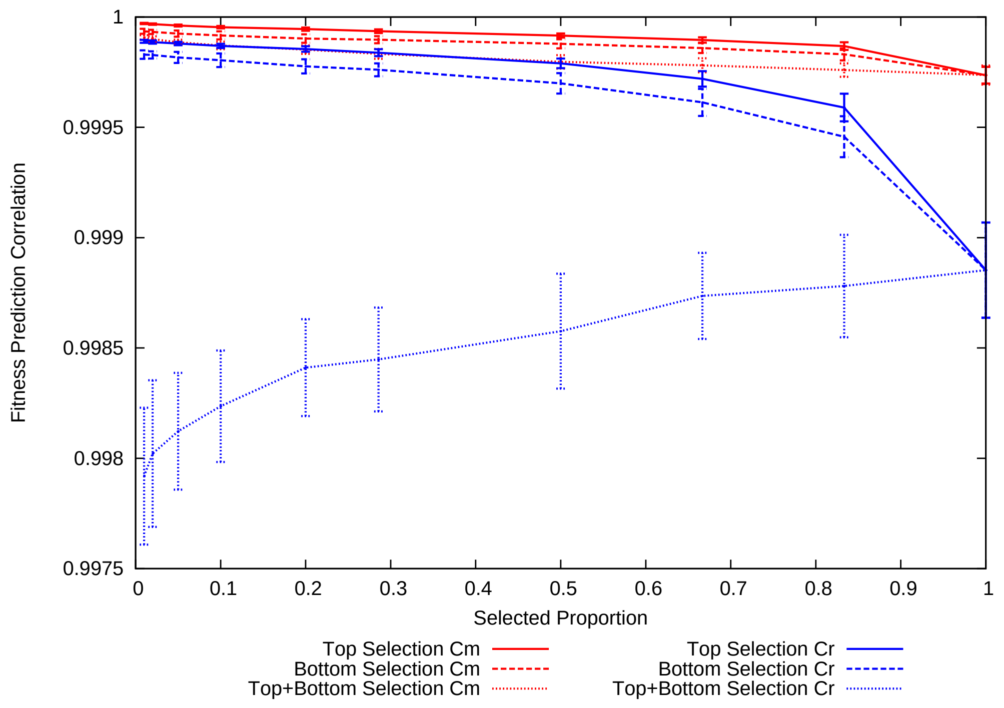


Figure C.22: FPC against selection proportion for fully specified 100 bit MaxSAT

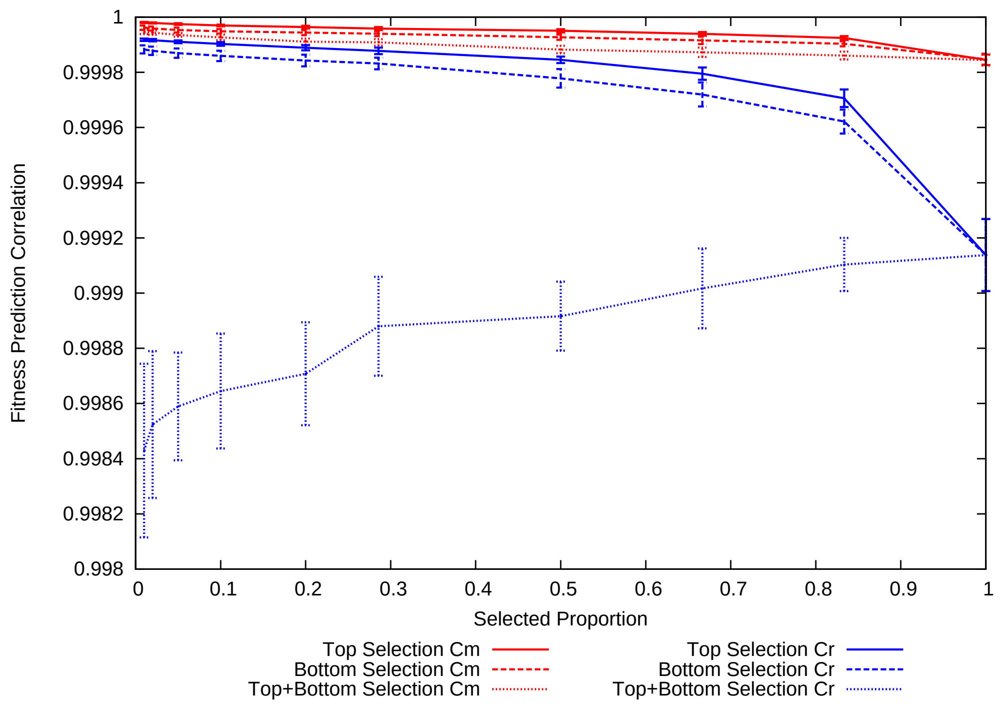


Figure C.23: FPC against selection proportion for fully specified 125 bit MaxSAT

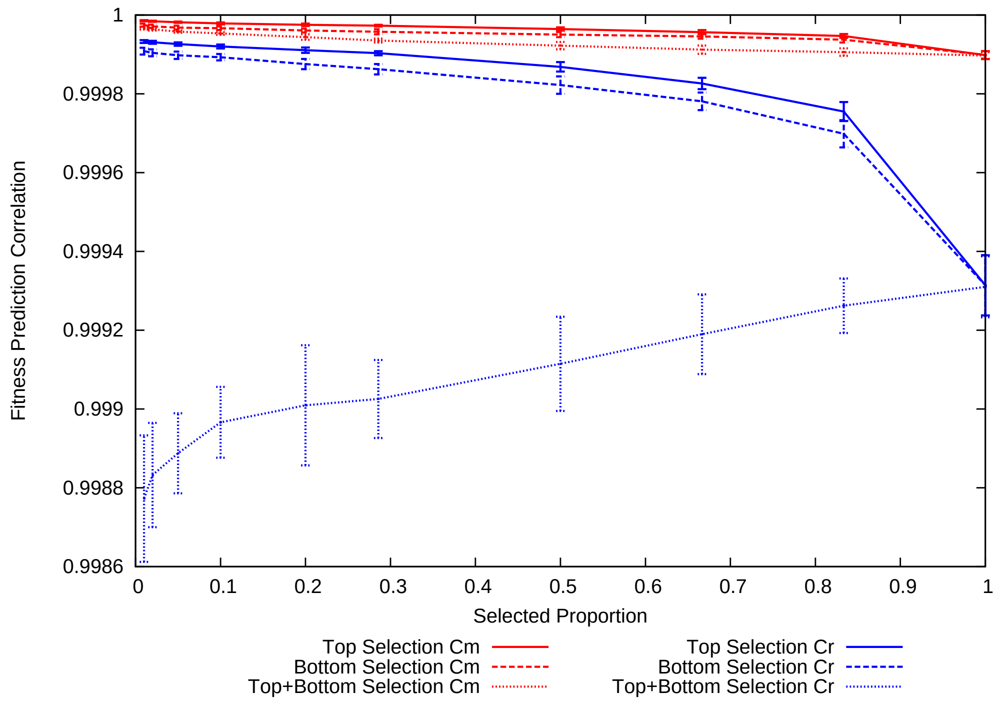


Figure C.24: FPC against selection proportion for fully specified 150 bit MaxSAT

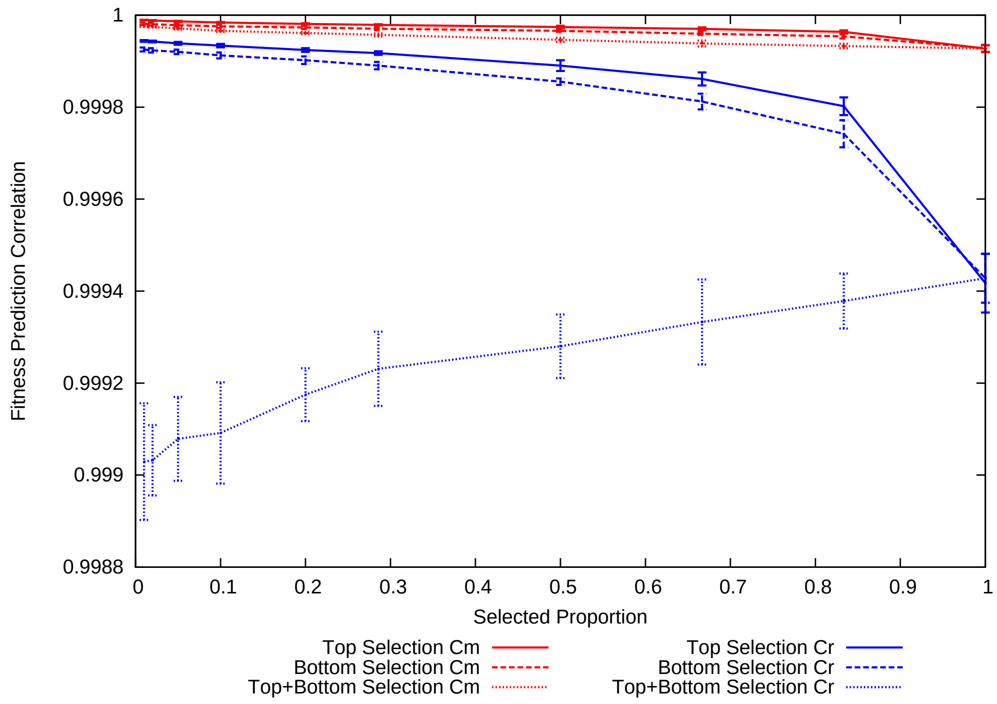


Figure C.25: FPC against selection proportion for fully specified 175 bit MaxSAT

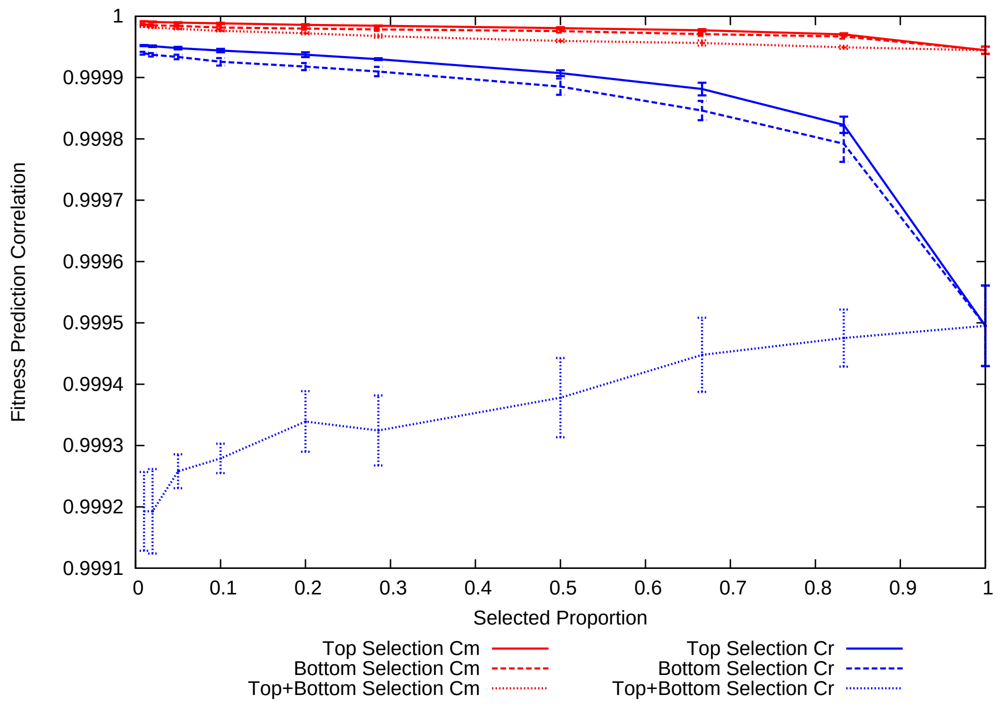


Figure C.26: FPC against selection proportion for 0.1N under specified 20 bit onemax

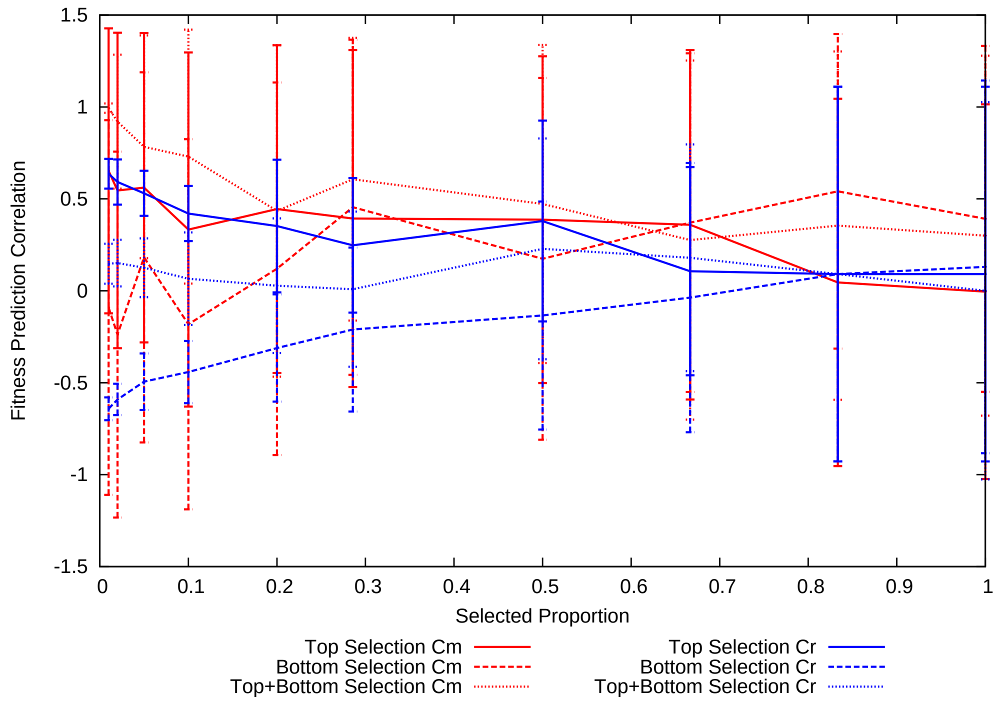


Figure C.27: FPC against selection proportion for 0.1N under specified 50 bit onemax

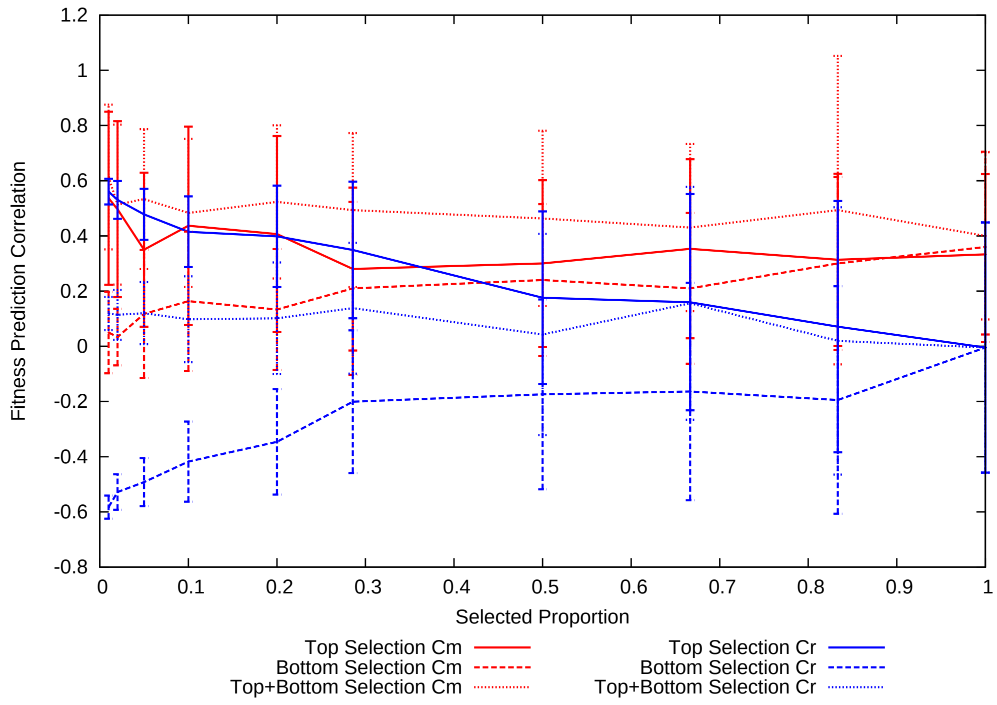


Figure C.28: FPC against selection proportion for 0.1N under specified 75 bit onemax

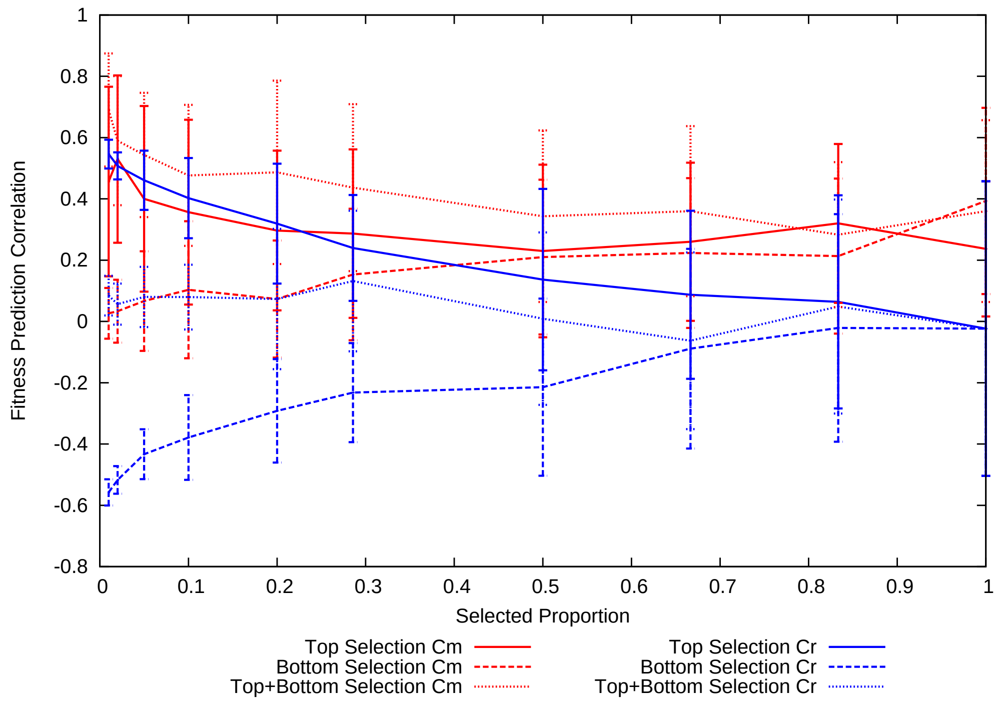


Figure C.29: FPC against selection proportion for 0.1N under specified 100 bit onemax

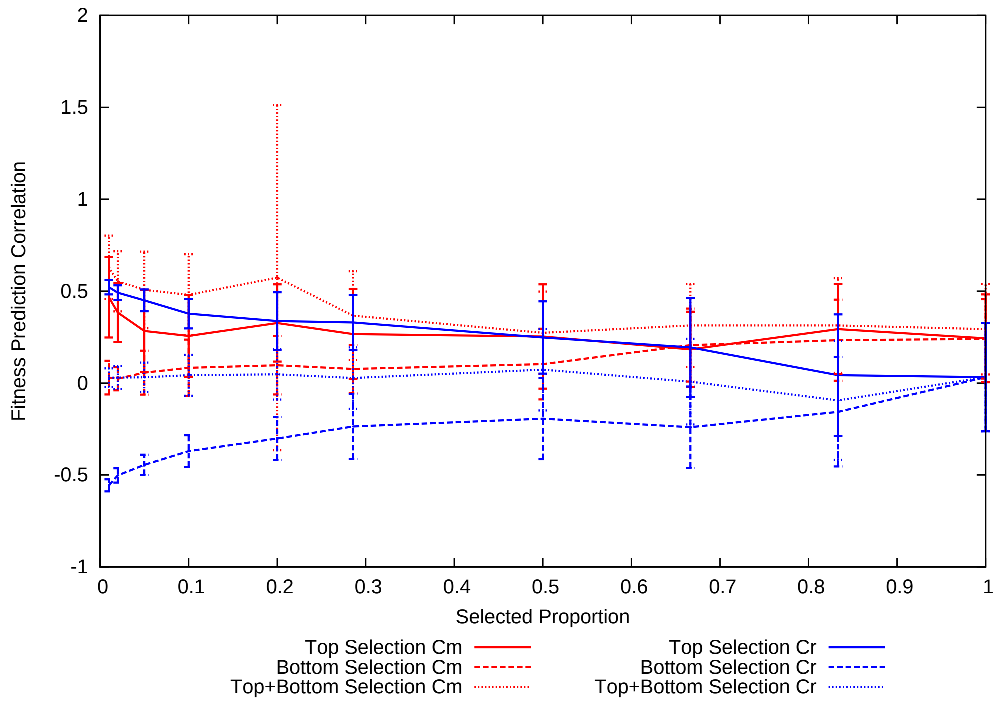


Figure C.30: FPC against selection proportion for 0.1N under specified 200 bit onemax

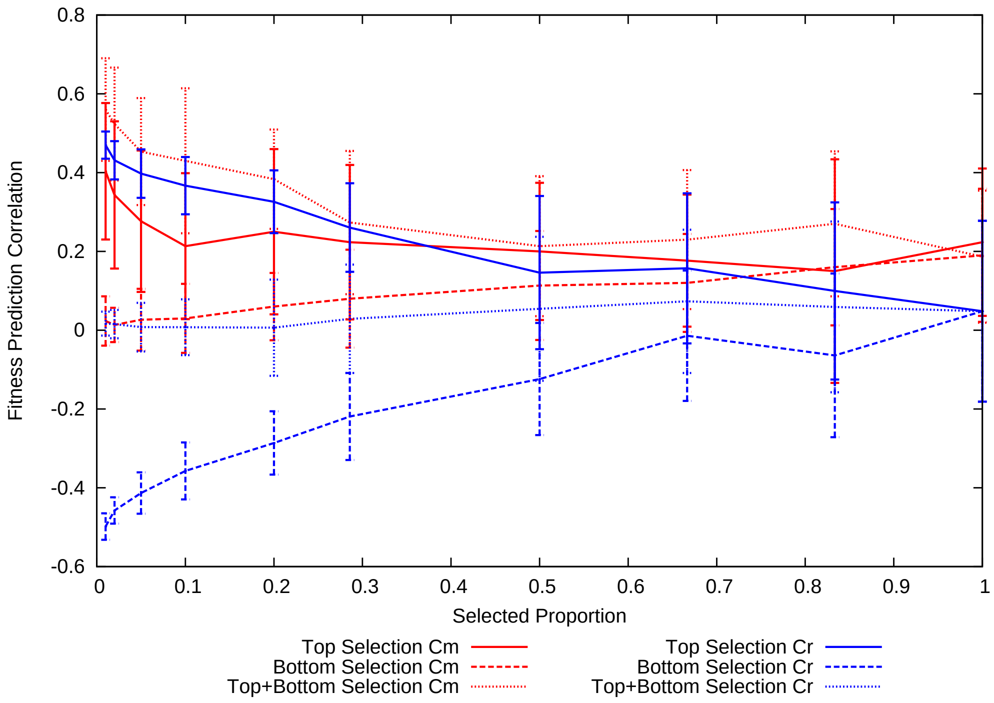


Figure C.31: FPC against selection proportion for 0.1N under specified 500 bit onemax

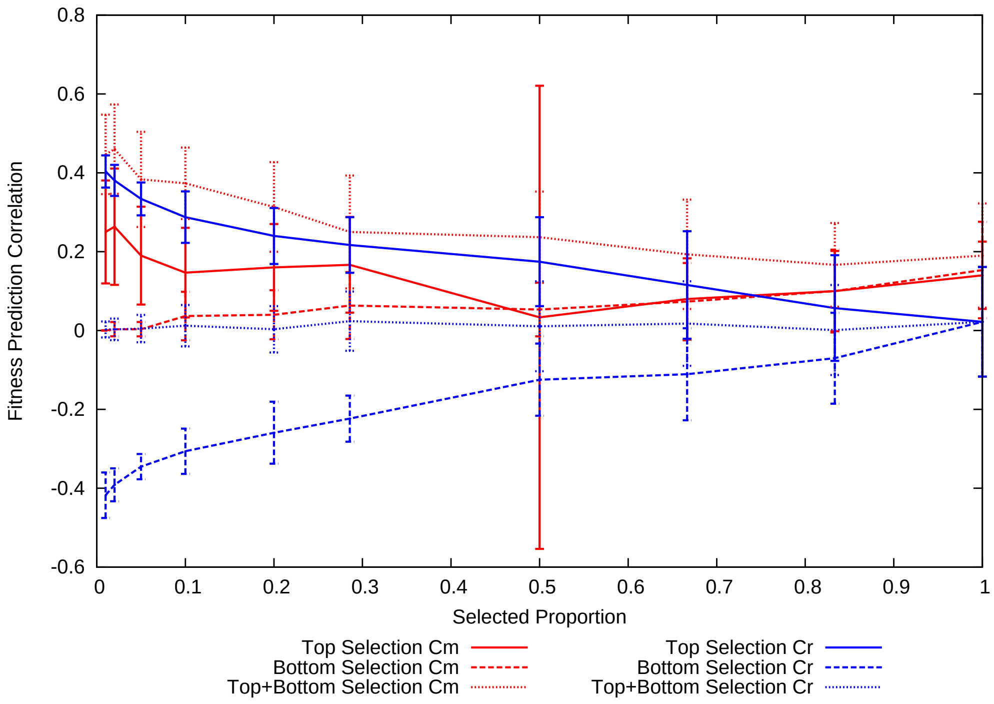


Figure C.32: FPC against selection proportion for 0.1N under specified 750 bit onemax

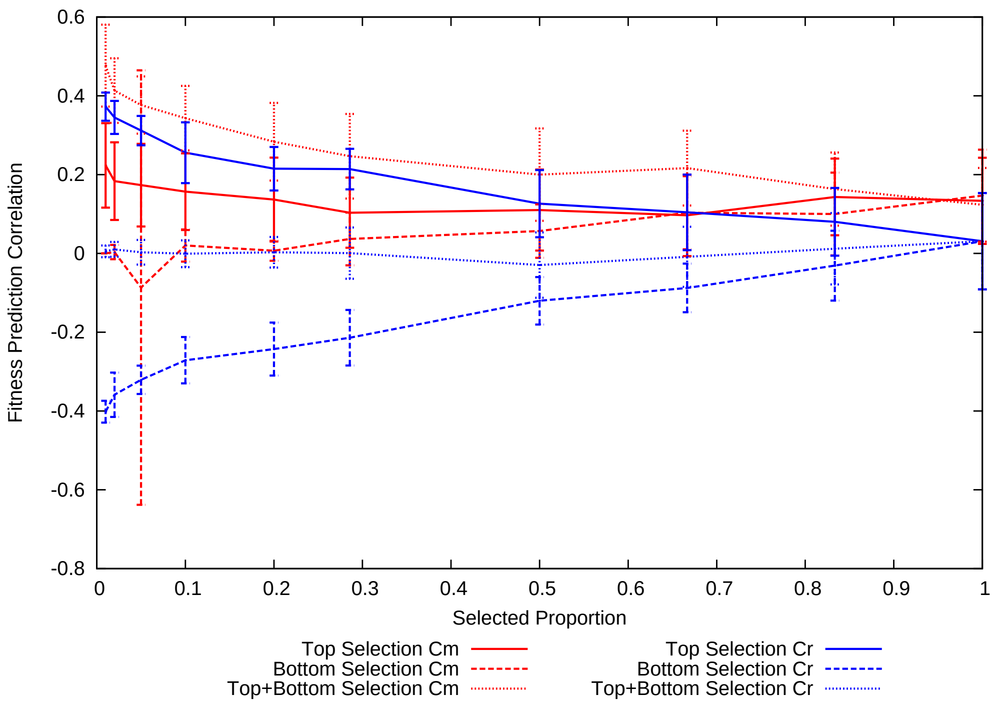


Figure C.33: FPC against selection proportion for 0.1N under specified 1000 bit onemax

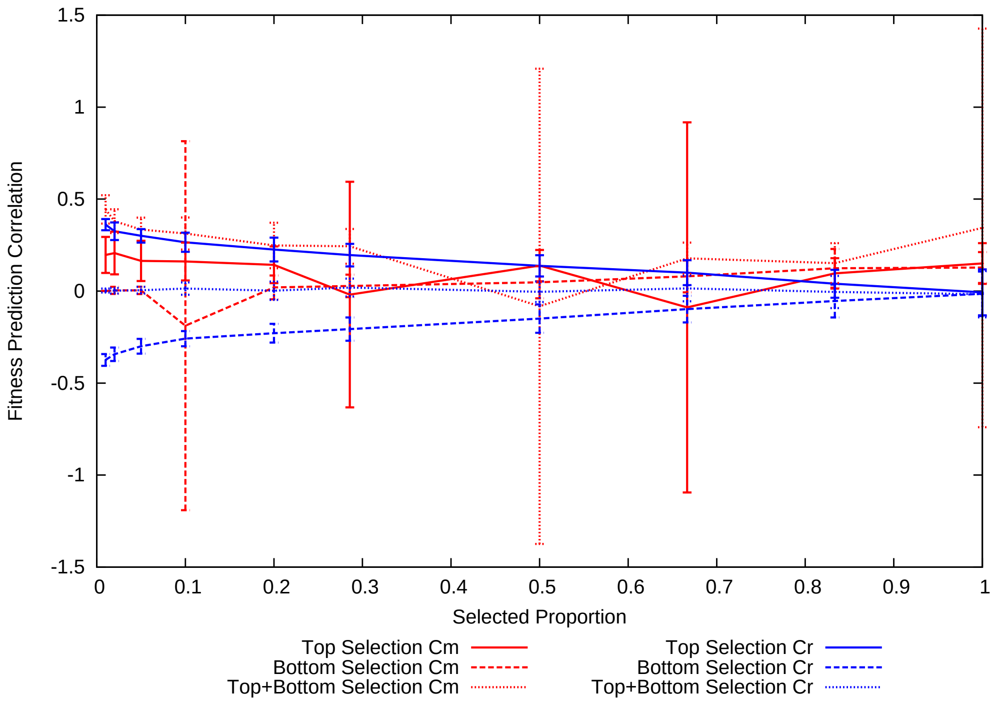


Figure C.34: FPC against selection proportion for 0.1N under specified 16 bit 2D Ising

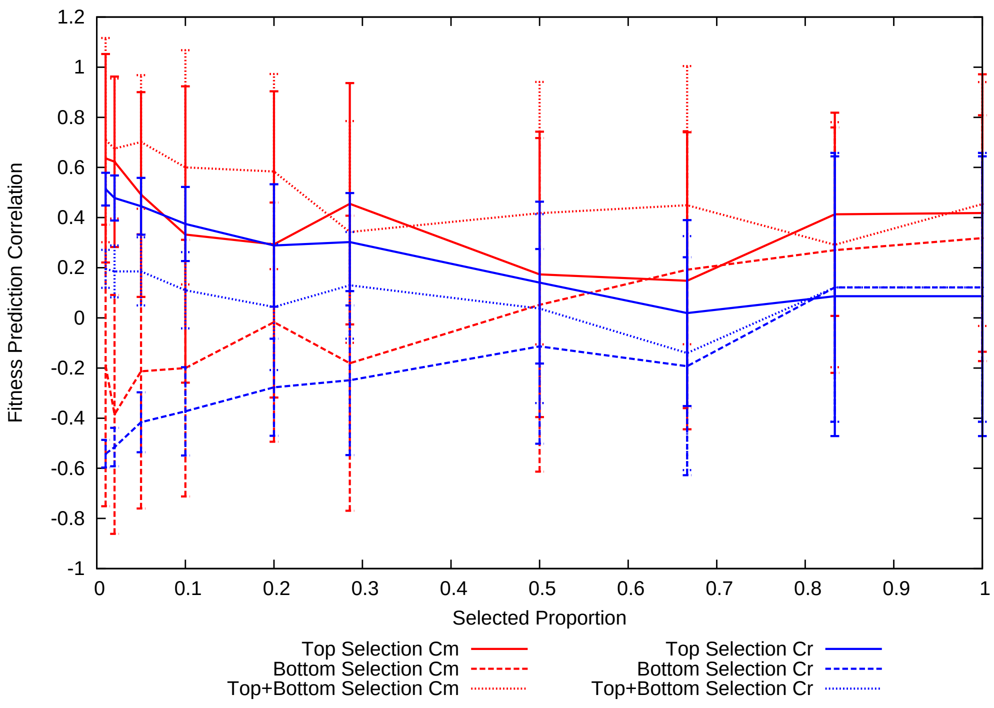


Figure C.35: FPC against selection proportion for 0.1N under specified 25 bit 2D Ising

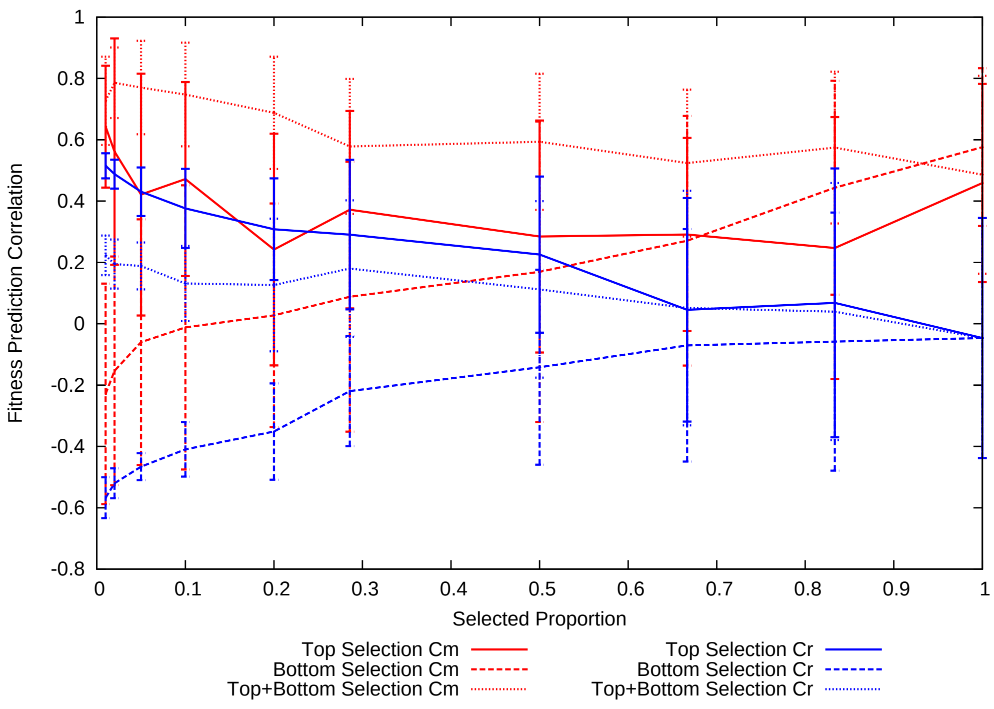


Figure C.36: FPC against selection proportion for 0.1N under specified 36 bit 2D Ising

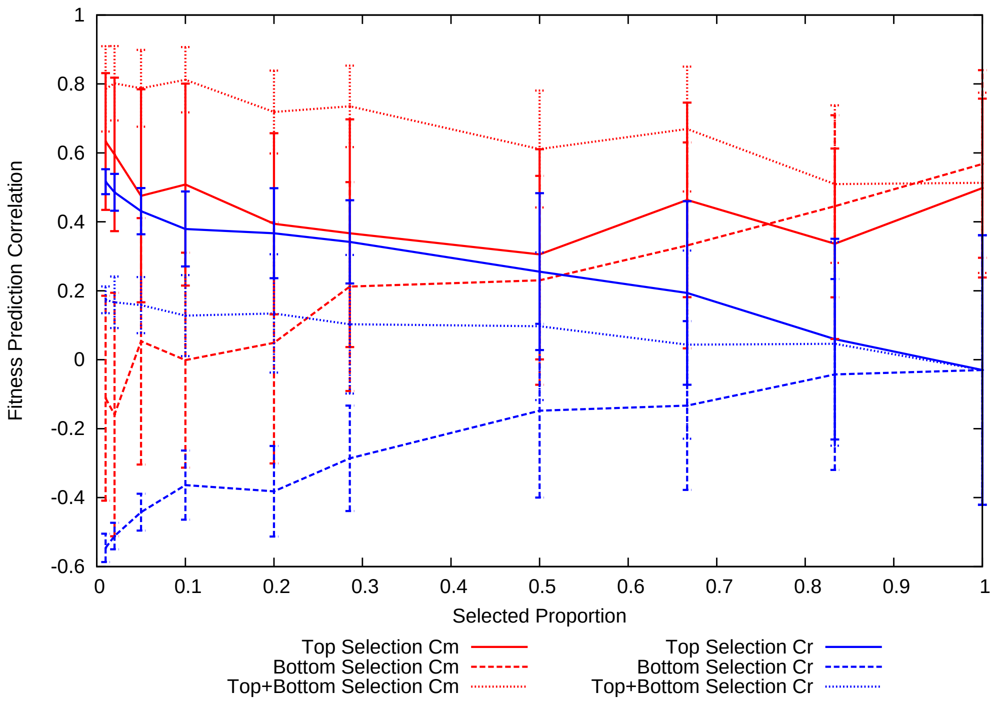


Figure C.37: FPC against selection proportion for 0.1N under specified 49 bit 2D Ising

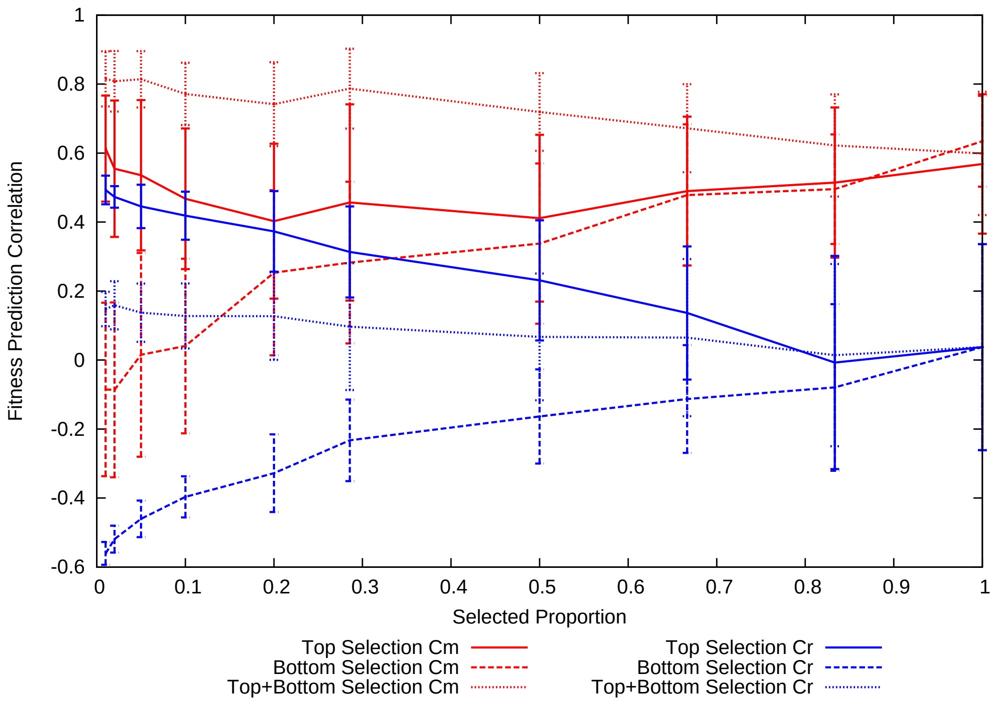


Figure C.38: FPC against selection proportion for 0.1N under specified 64 bit 2D Ising

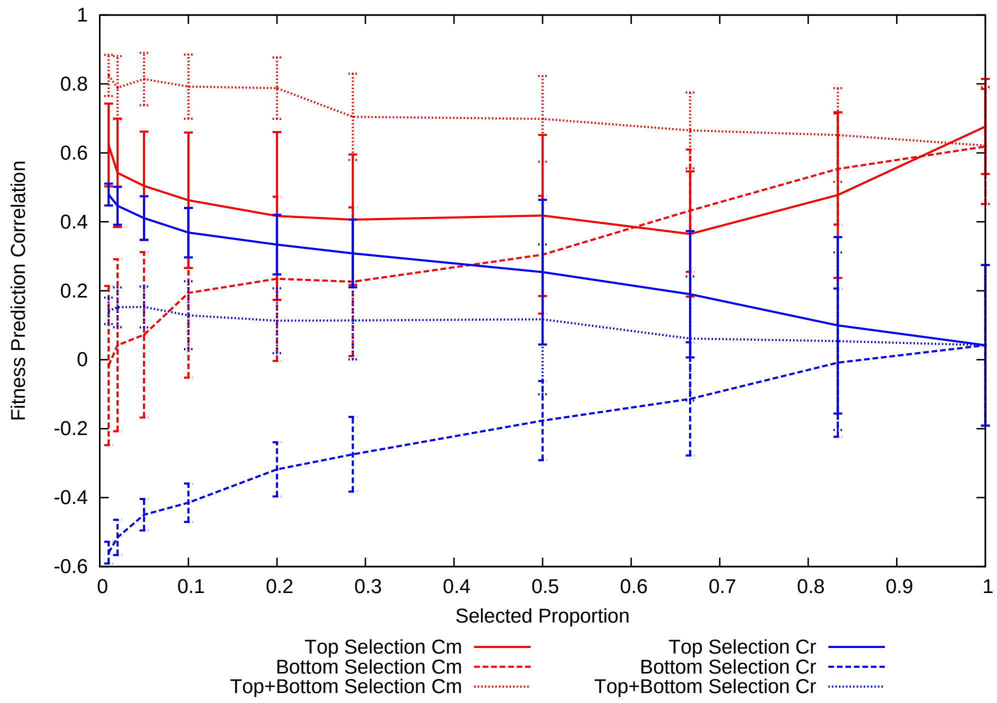


Figure C.39: FPC against selection proportion for 0.1N under specified 100 bit 2D Ising

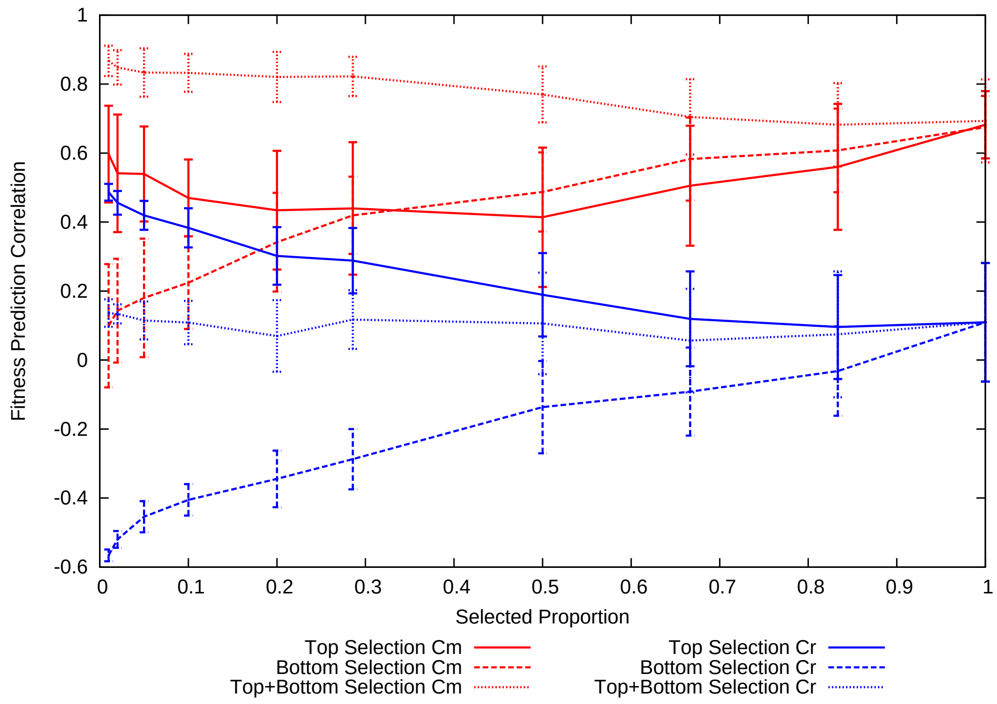


Figure C.40: FPC against selection proportion for 0.1N under specified 256 bit 2D Ising

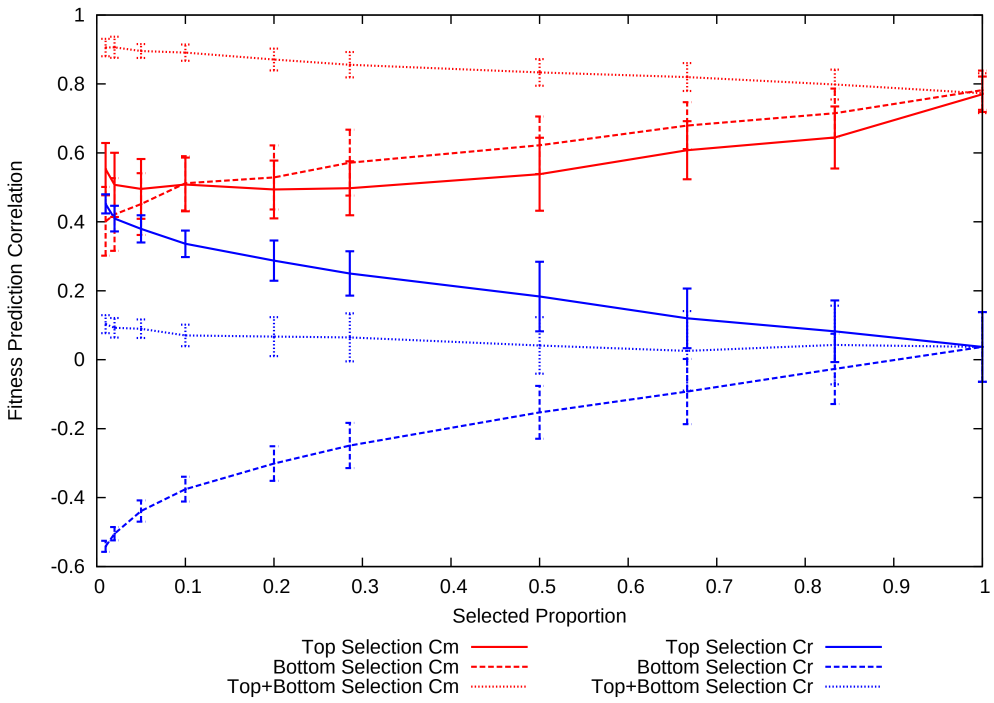


Figure C.41: FPC against selection proportion for 0.1N under specified 324 bit 2D Ising

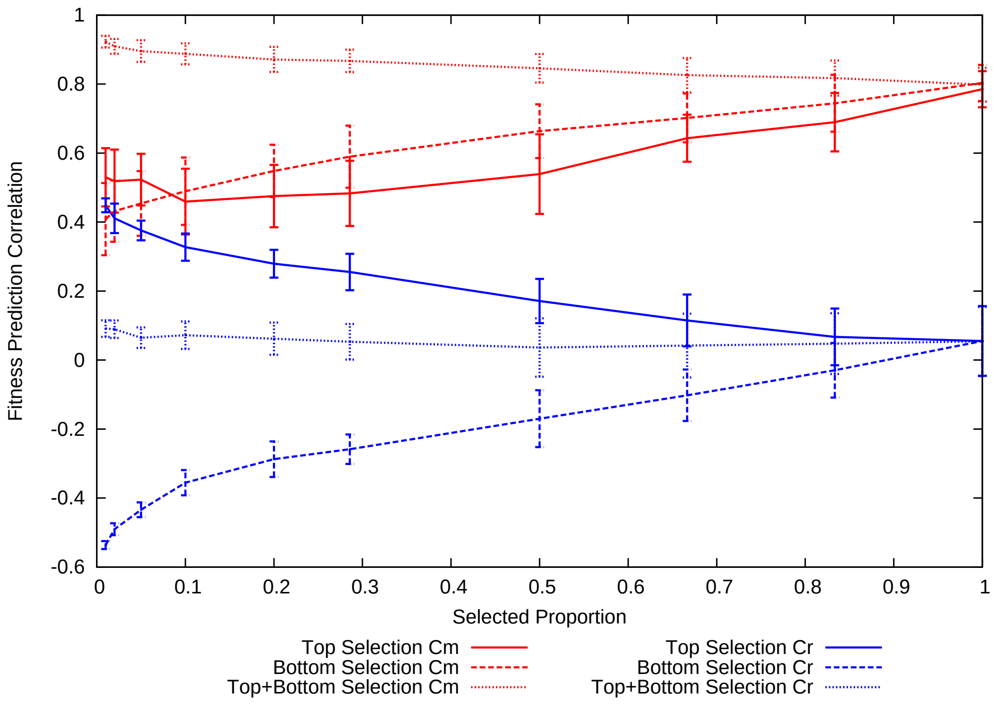


Figure C.42: FPC against selection proportion for 0.1N under specified 400 bit 2D Ising

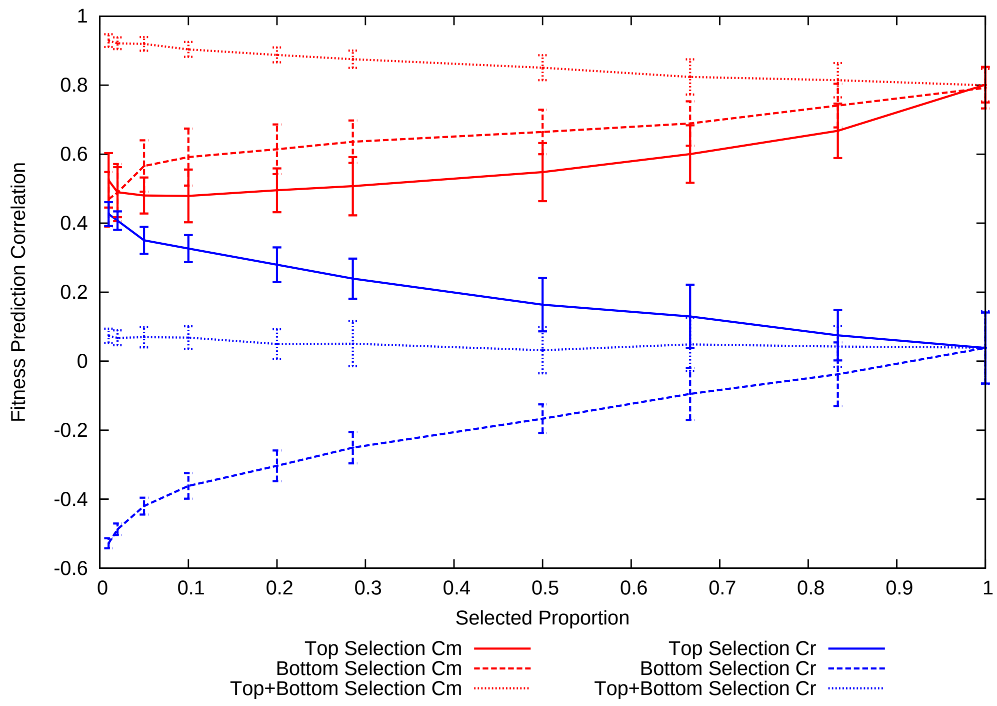


Figure C.43: FPC against selection proportion for 0.1N under specified 20 bit MaxSAT

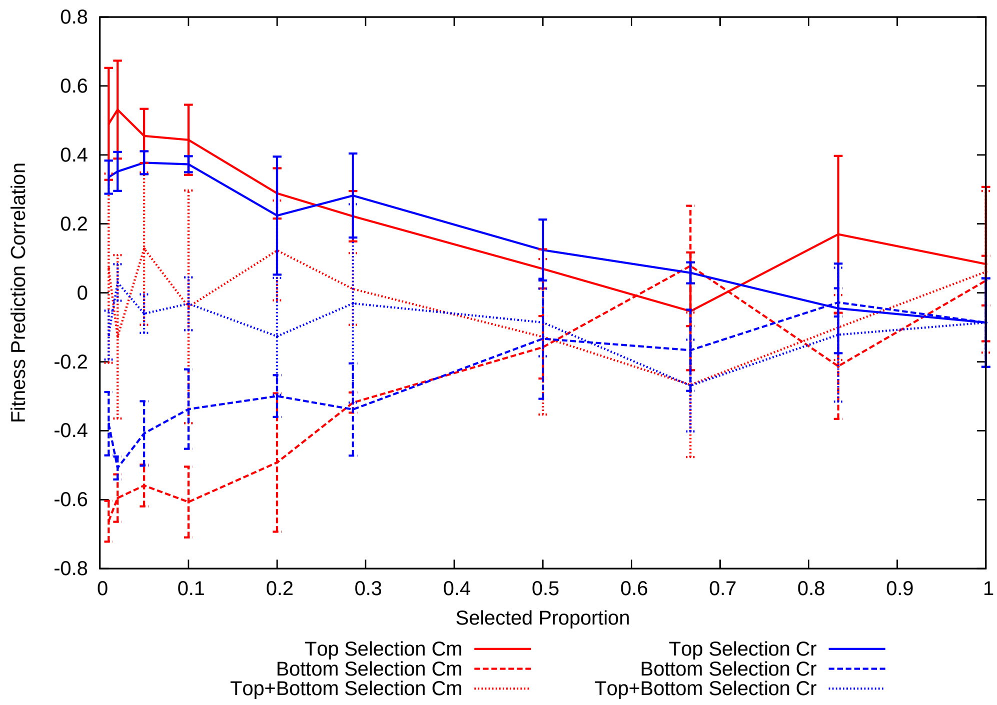


Figure C.44: FPC against selection proportion for 0.1N under specified 50 bit MaxSAT

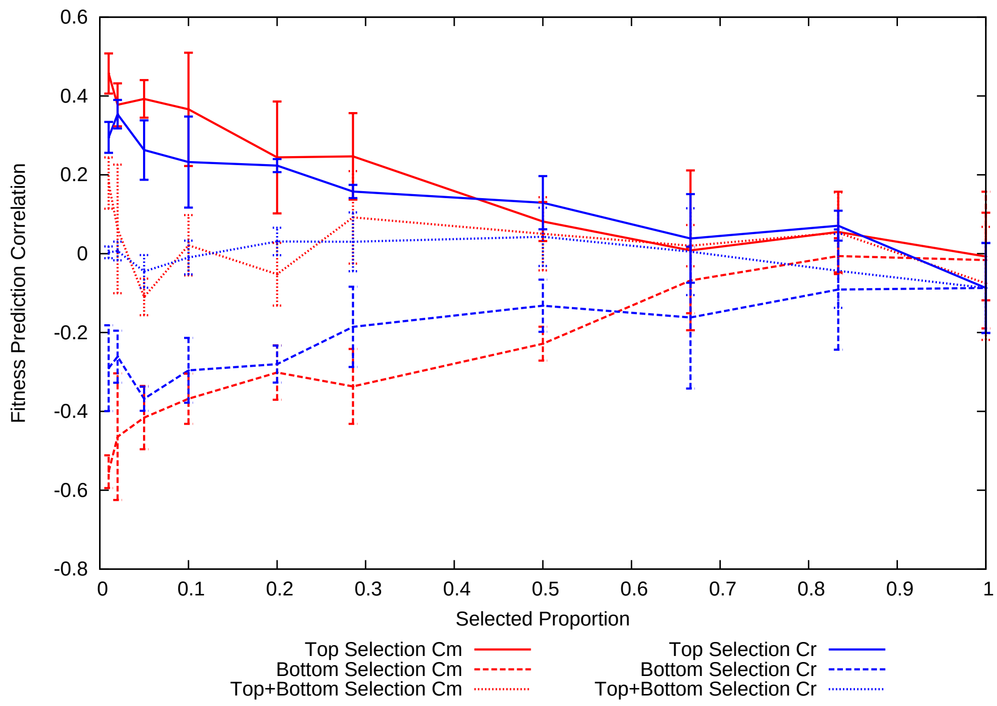


Figure C.45: FPC against selection proportion for 0.1N under specified 75 bit MaxSAT

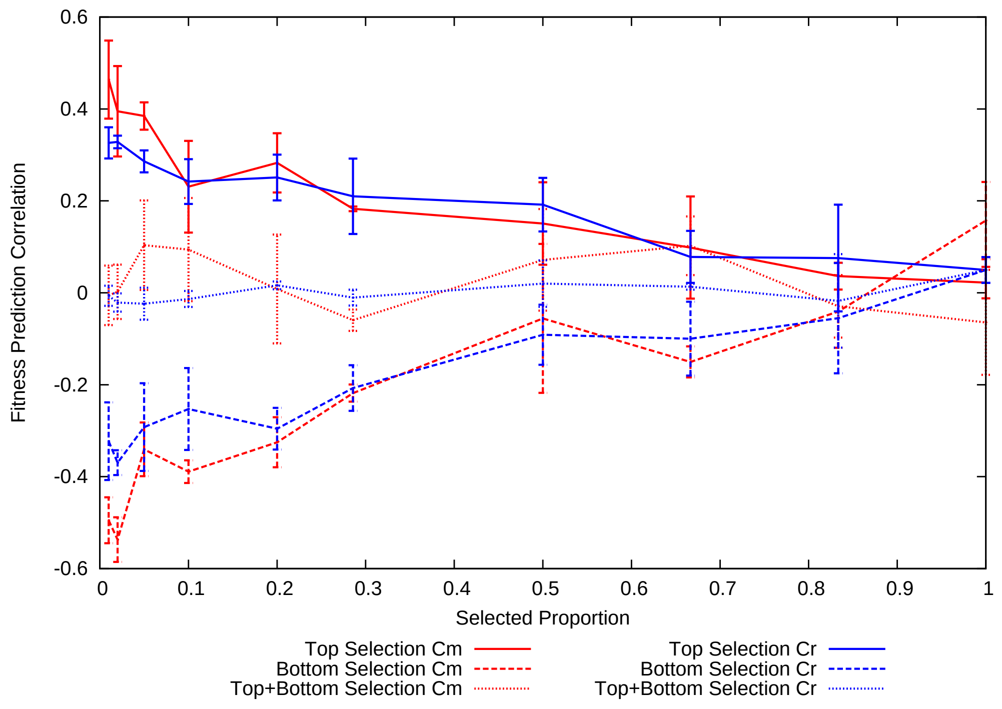


Figure C.46: FPC against selection proportion for 0.1N under specified 100 bit MaxSAT

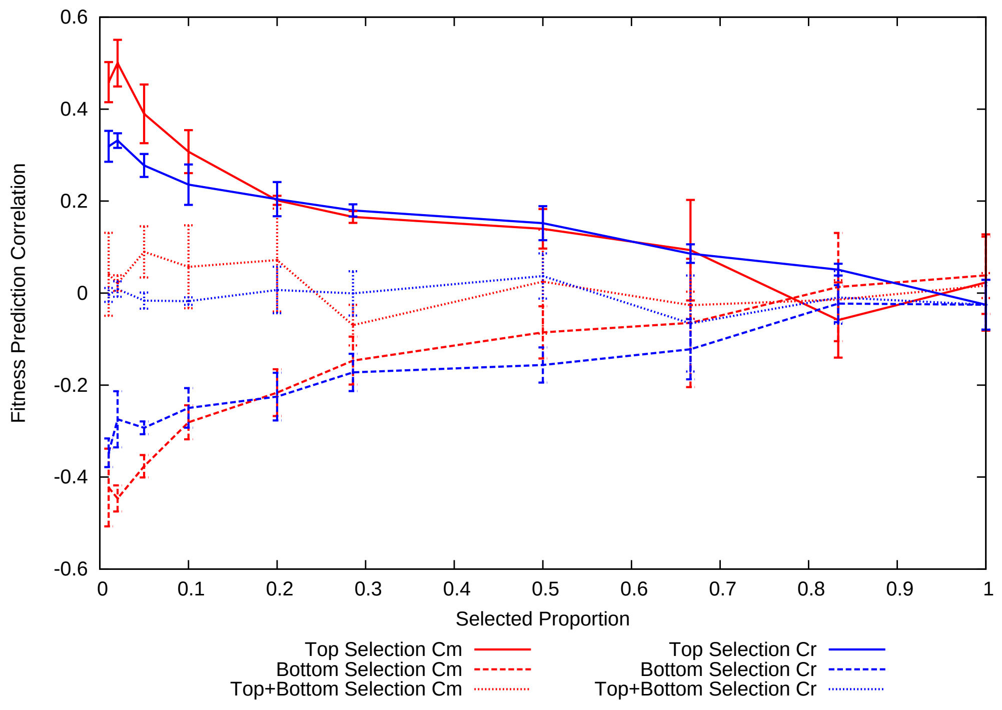


Figure C.47: FPC against selection proportion for 0.1N under specified 125 bit MaxSAT

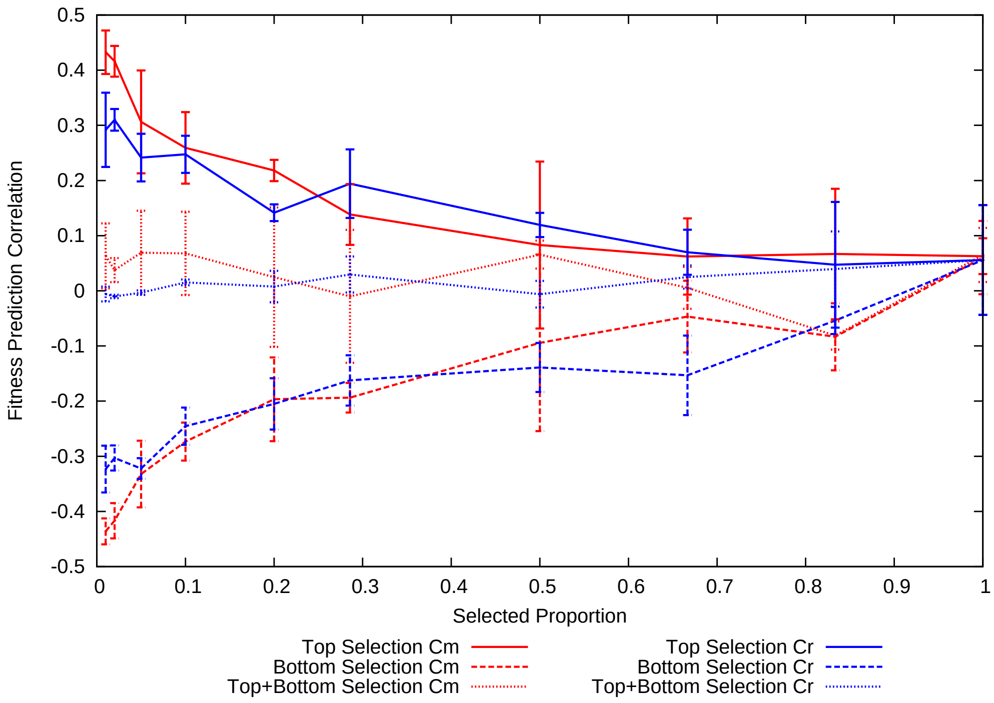


Figure C.48: FPC against selection proportion for 0.1N under specified 150 bit MaxSAT

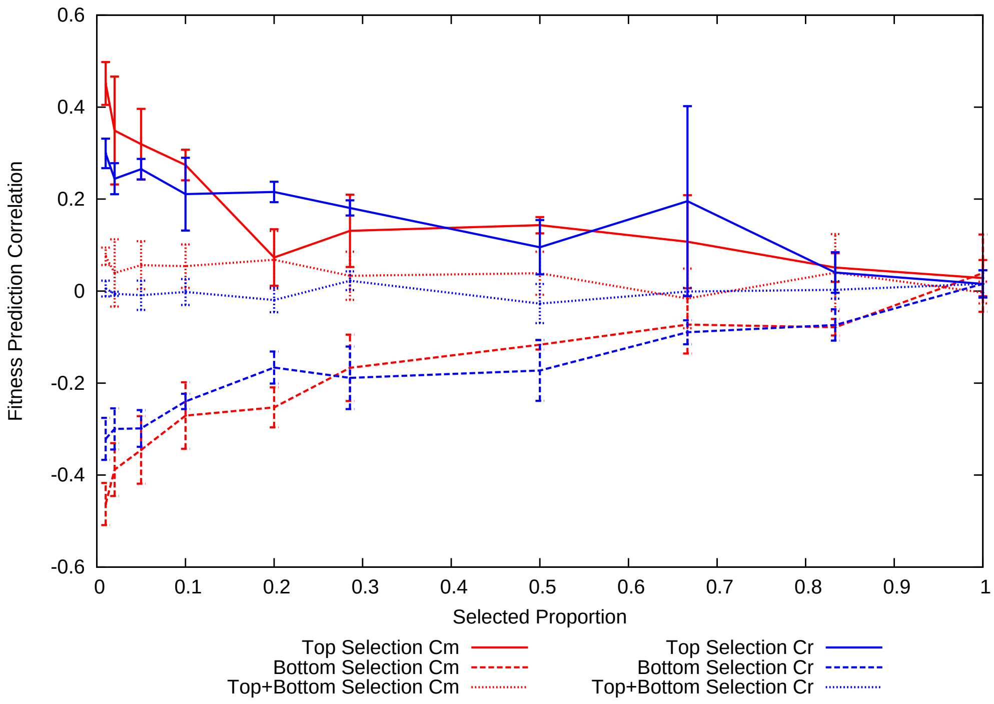


Figure C.49: FPC against selection proportion for 0.1N under specified 175 bit MaxSAT

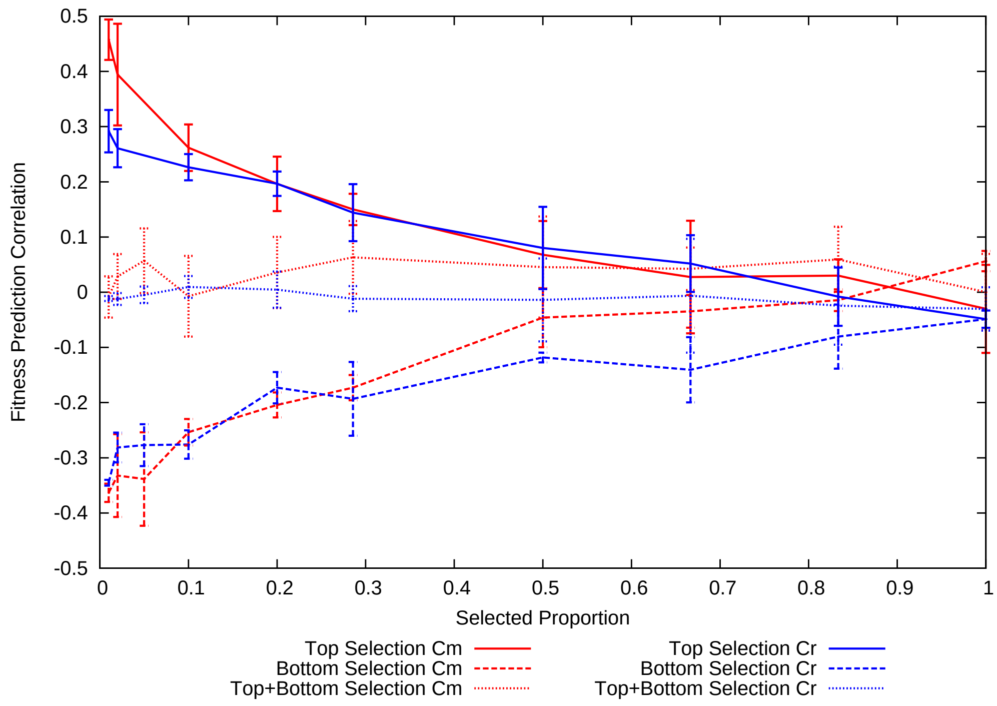


Figure C.50: FPC against selection proportion for fully specified 16 bit 2D Ising with no bivariate interactions

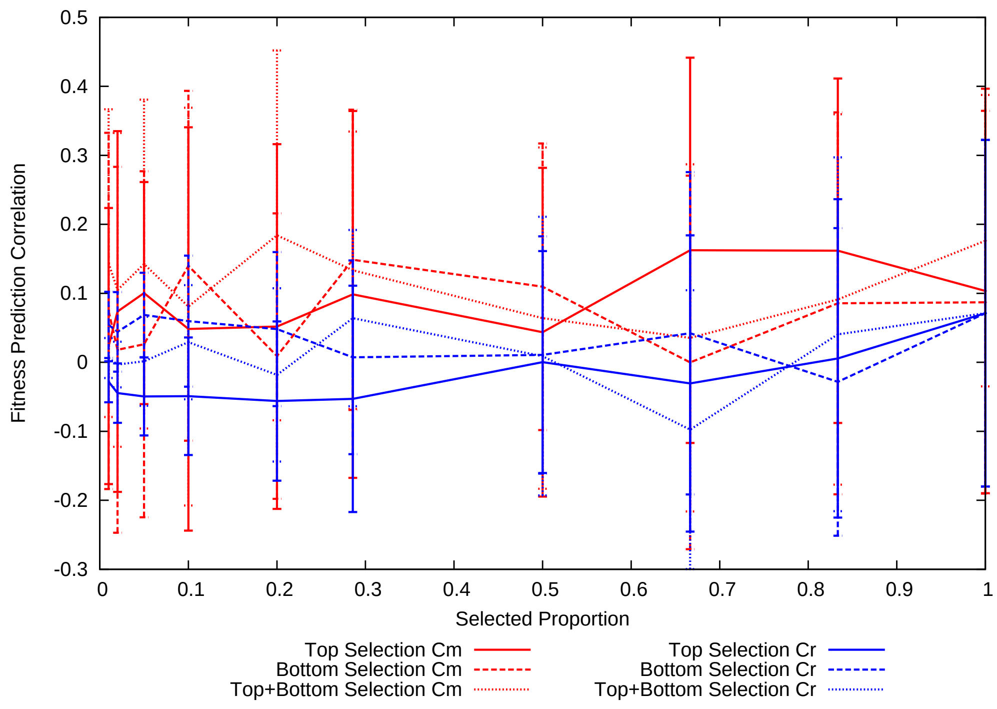


Figure C.51: FPC against selection proportion for fully specified 25 bit 2D Ising with no bivariate interactions

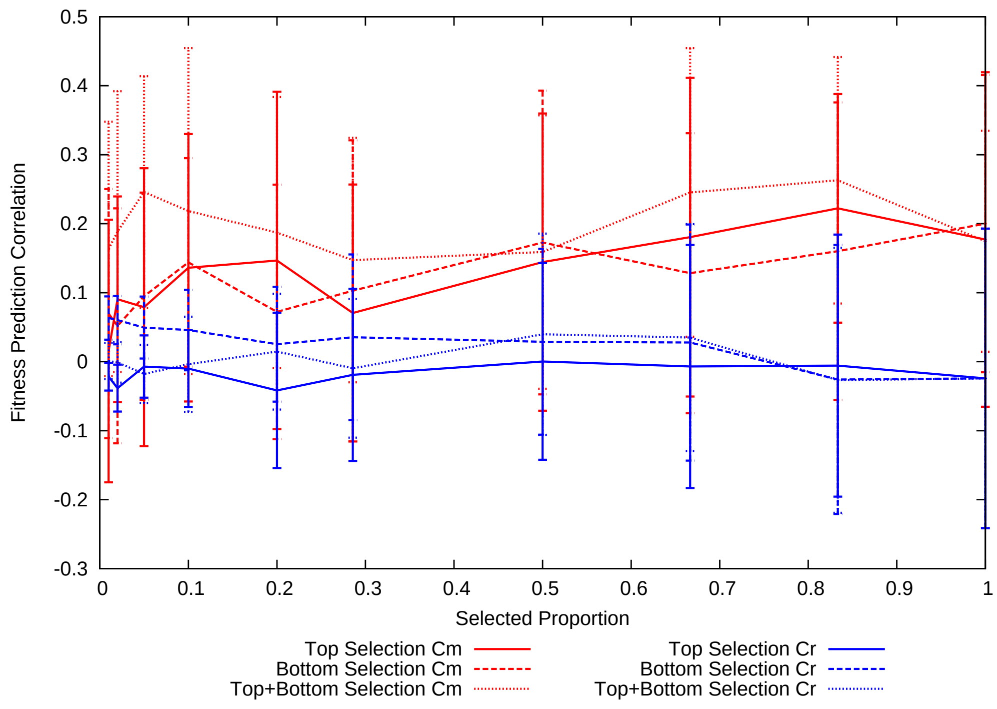


Figure C.52: FPC against selection proportion for fully specified 36 bit 2D Ising with no bivariate interactions

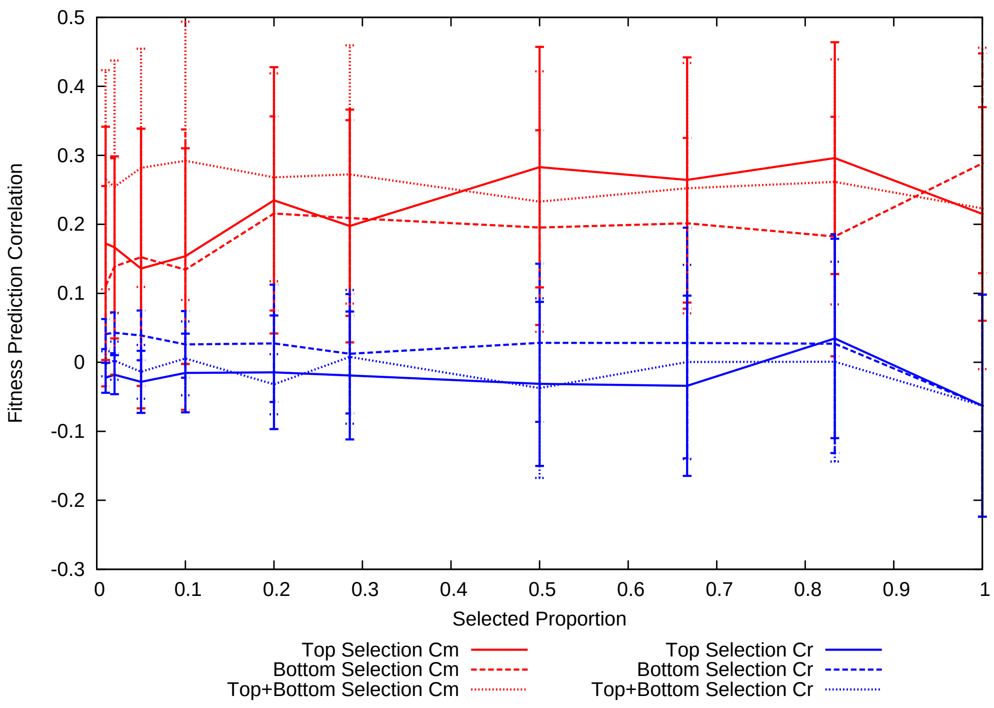


Figure C.53: FPC against selection proportion for fully specified 49 bit 2D Ising with no bivariate interactions

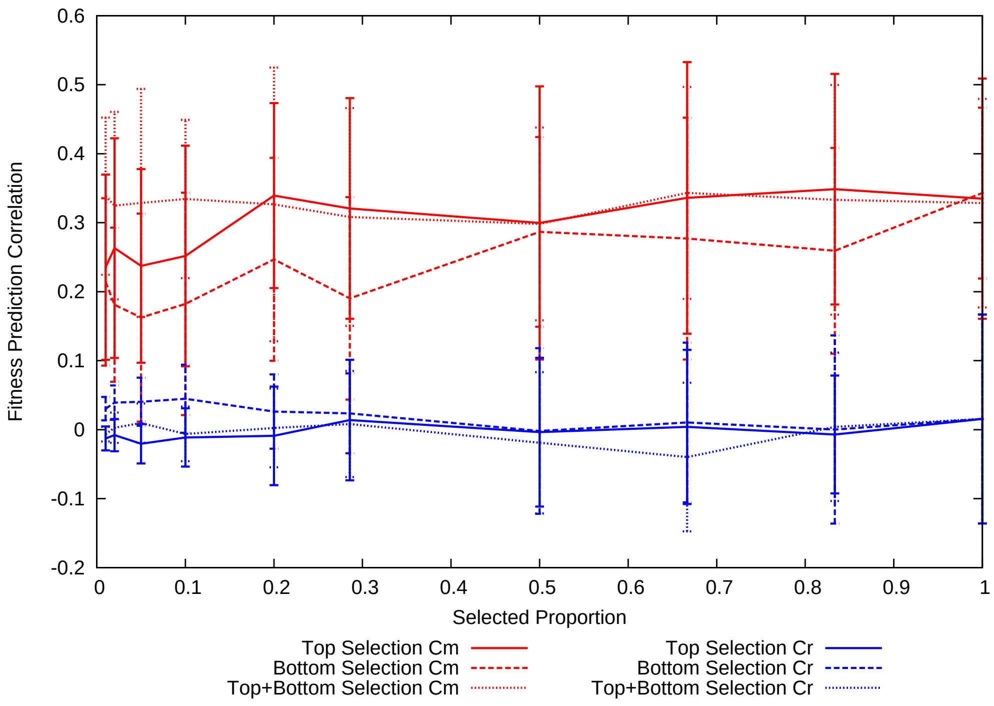


Figure C.54: FPC against selection proportion for fully specified 64 bit 2D Ising with no bivariate interactions

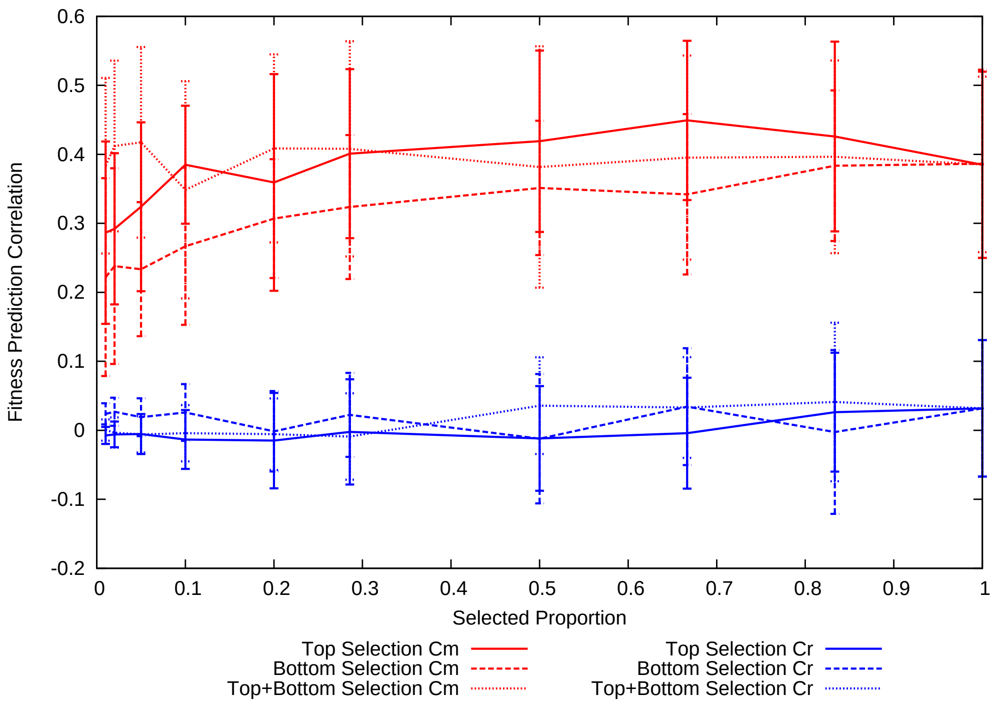


Figure C.55: FPC against selection proportion for fully specified 100 bit 2D Ising with no bivariate interactions

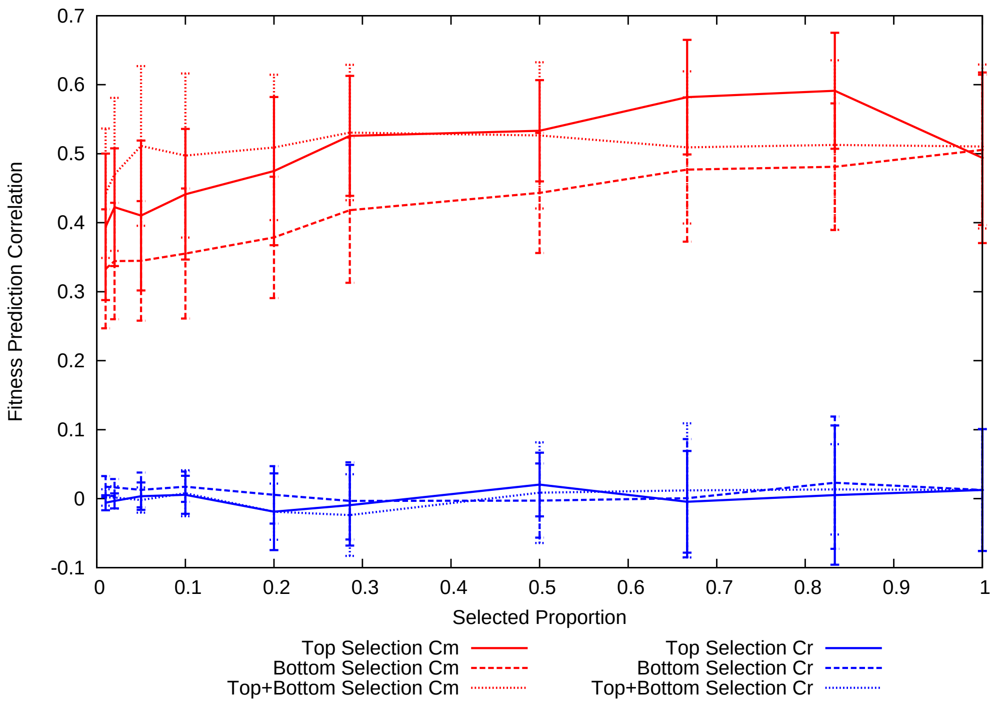


Figure C.56: FPC against selection proportion for fully specified 256 bit 2D Ising with no bivariate interactions

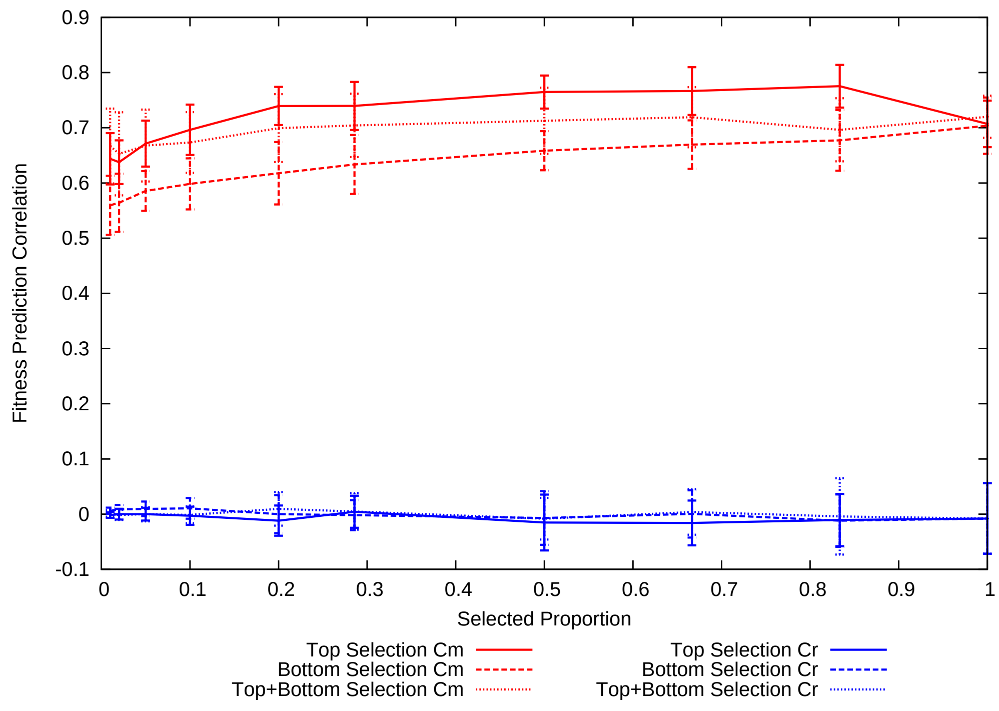


Figure C.57: FPC against selection proportion for fully specified 324 bit 2D Ising with no bivariate interactions

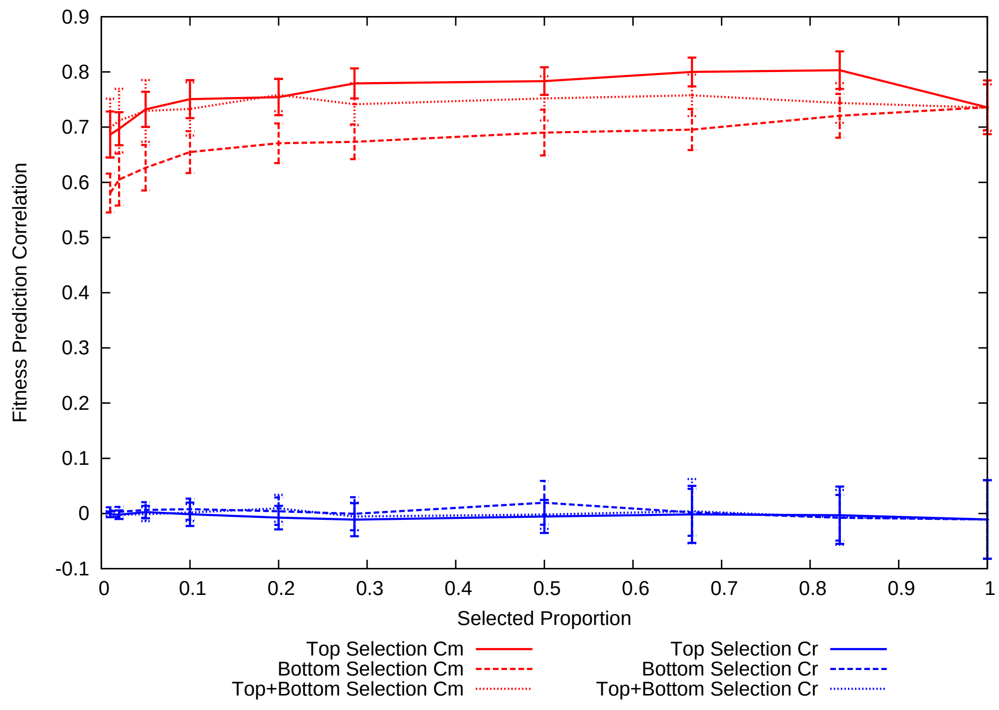


Figure C.58: FPC against selection proportion for fully specified 400 bit 2D Ising with no bivariate interactions

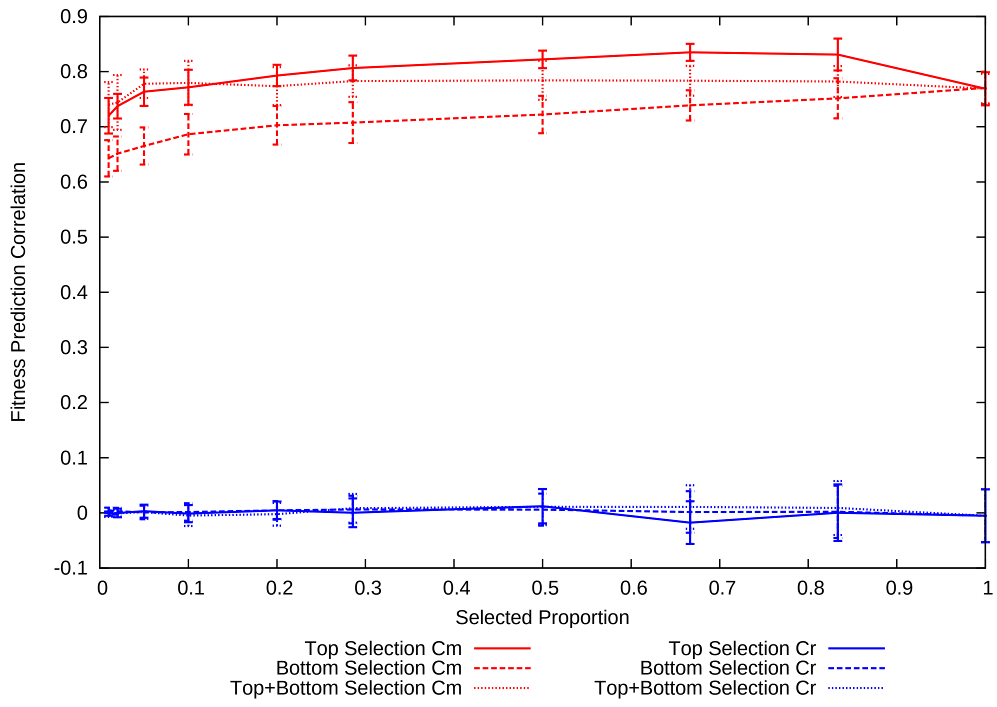


Figure C.59: FPC against selection proportion for fully specified 16 bit 2D Ising with 0.1 decimation

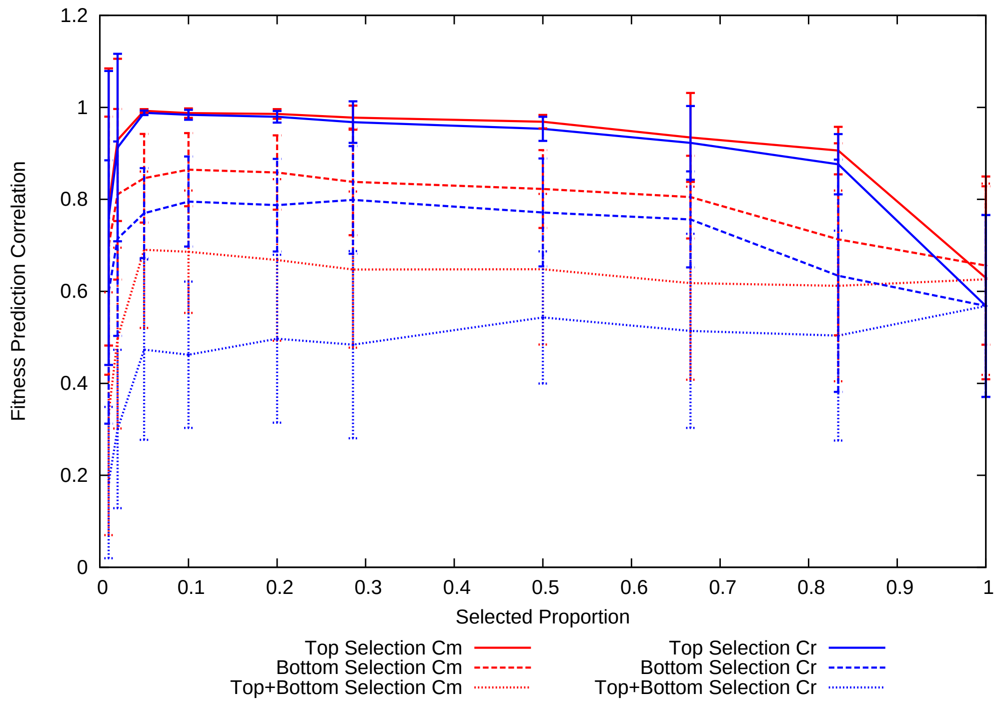


Figure C.60: FPC against selection proportion for fully specified 25 bit 2D Ising with 0.1 decimation

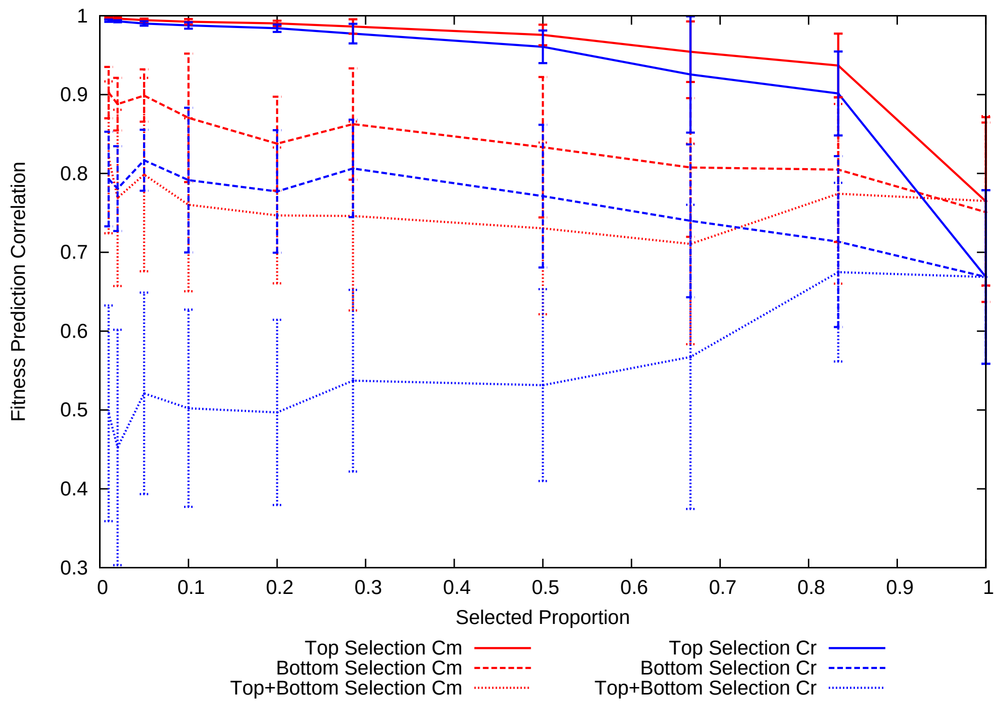


Figure C.61: FPC against selection proportion for fully specified 36 bit 2D Ising with 0.1 decimation

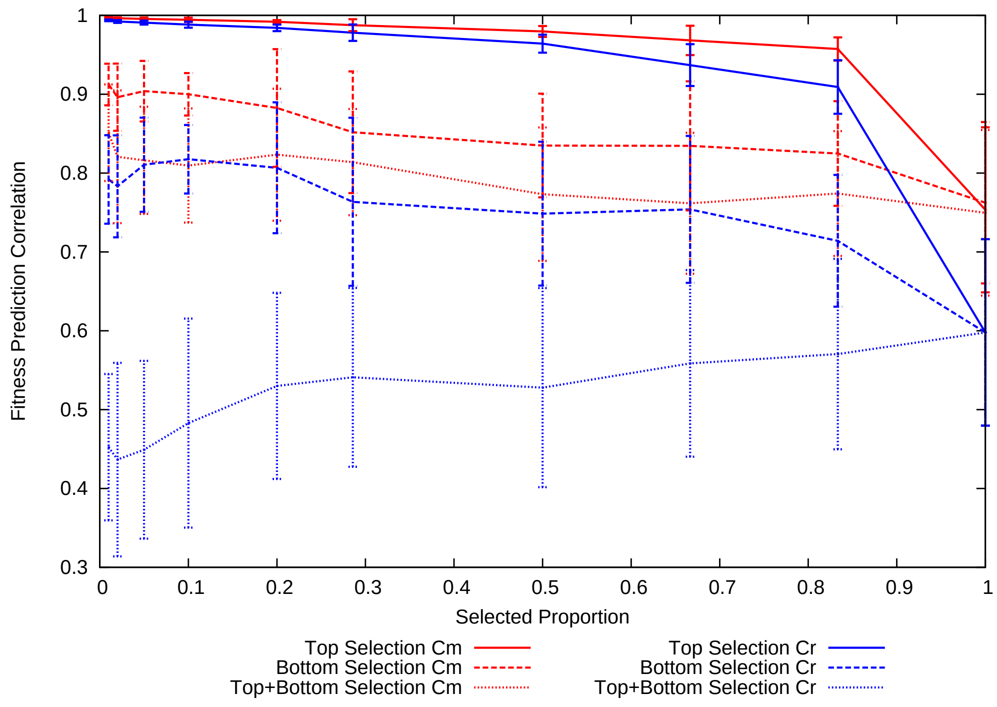


Figure C.62: FPC against selection proportion for fully specified 49 bit 2D Ising with 0.1 decimation

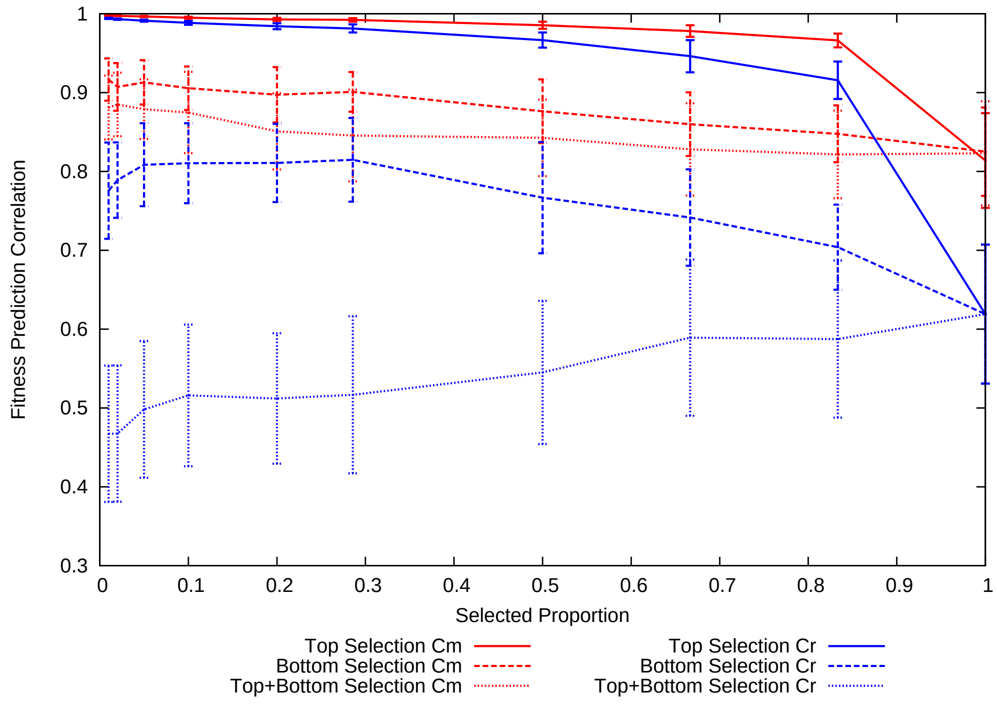


Figure C.63: FPC against selection proportion for fully specified 64 bit 2D Ising with 0.1 decimation

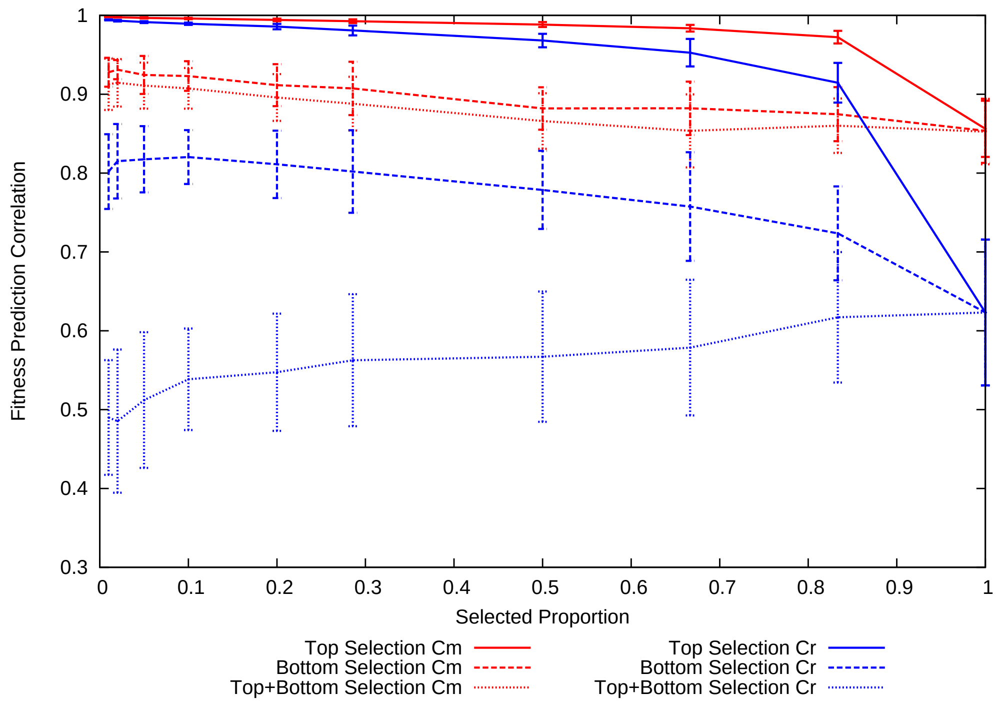


Figure C.64: FPC against selection proportion for fully specified 100 bit 2D Ising with 0.1 decimation

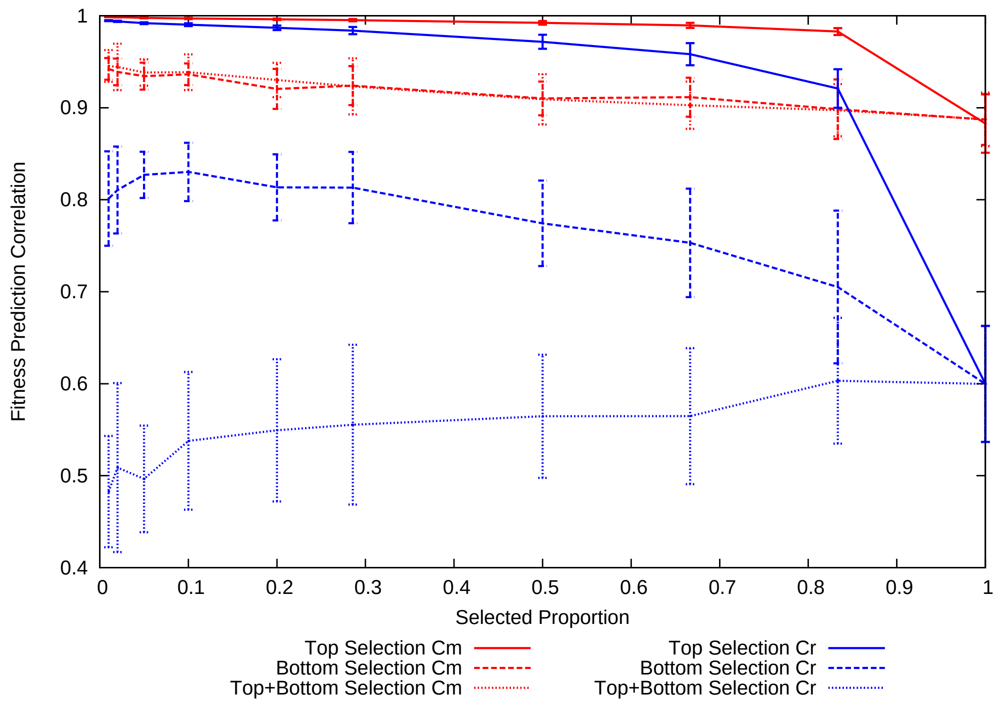


Figure C.65: FPC against selection proportion for fully specified 256 bit 2D Ising with 0.1 decimation

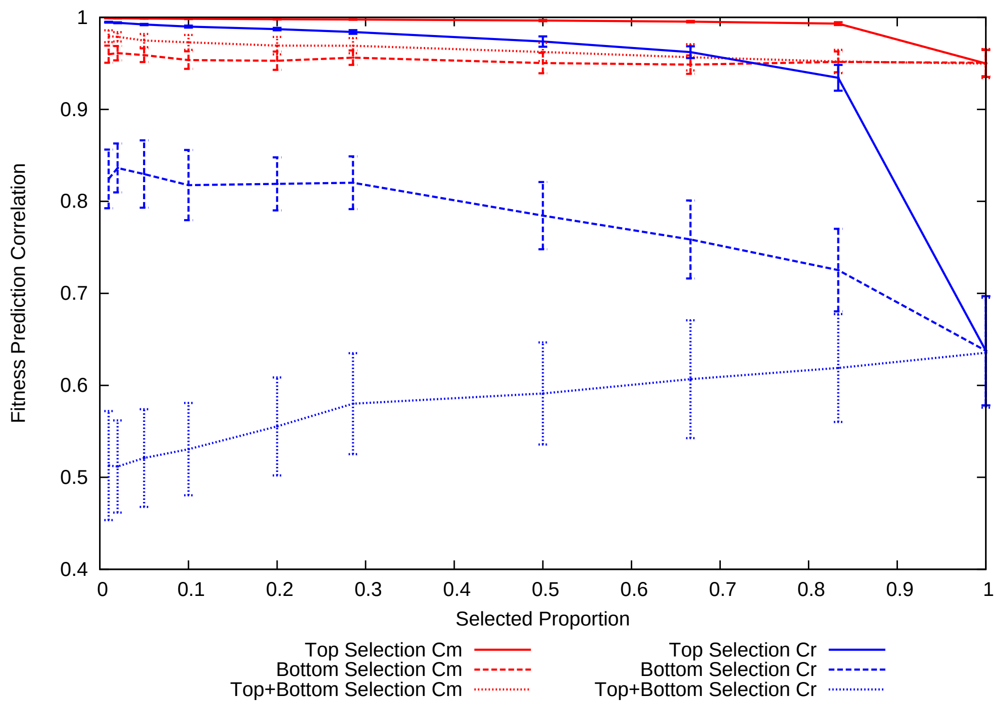


Figure C.66: FPC against selection proportion for fully specified 324 bit 2D Ising with 0.1 decimation

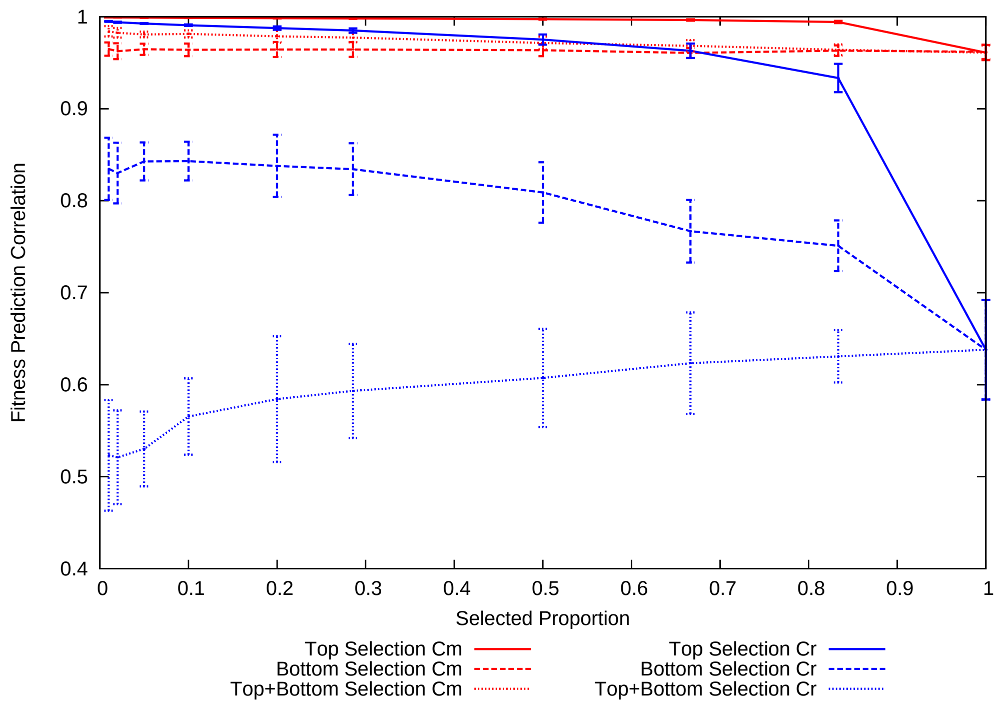


Figure C.67: FPC against selection proportion for fully specified 400 bit 2D Ising with 0.1 decimation

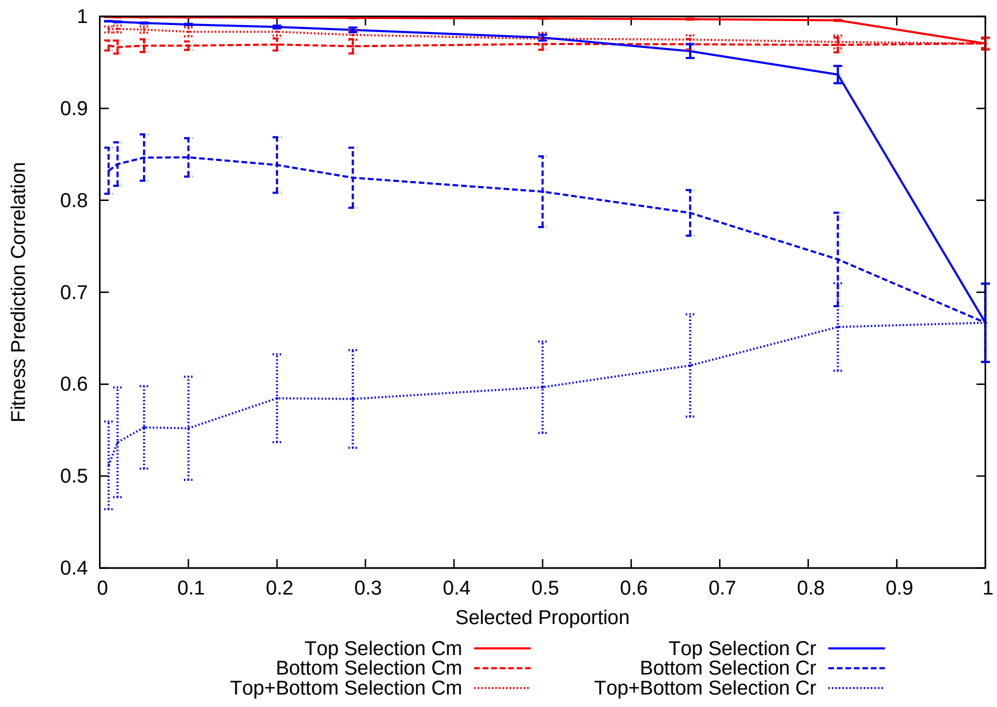


Figure C.68: FPC against selection proportion for fully specified 16 bit 2D Ising problem with 0.5 decimation

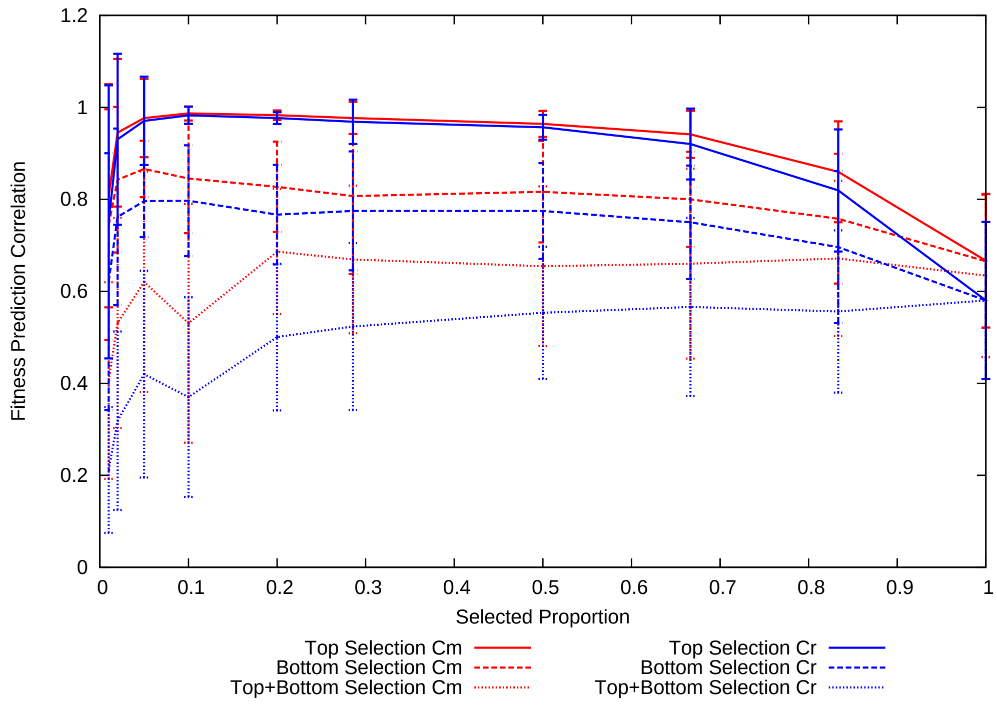


Figure C.69: FPC against selection proportion for fully specified 25 bit 2D Ising problem with 0.5 decimation

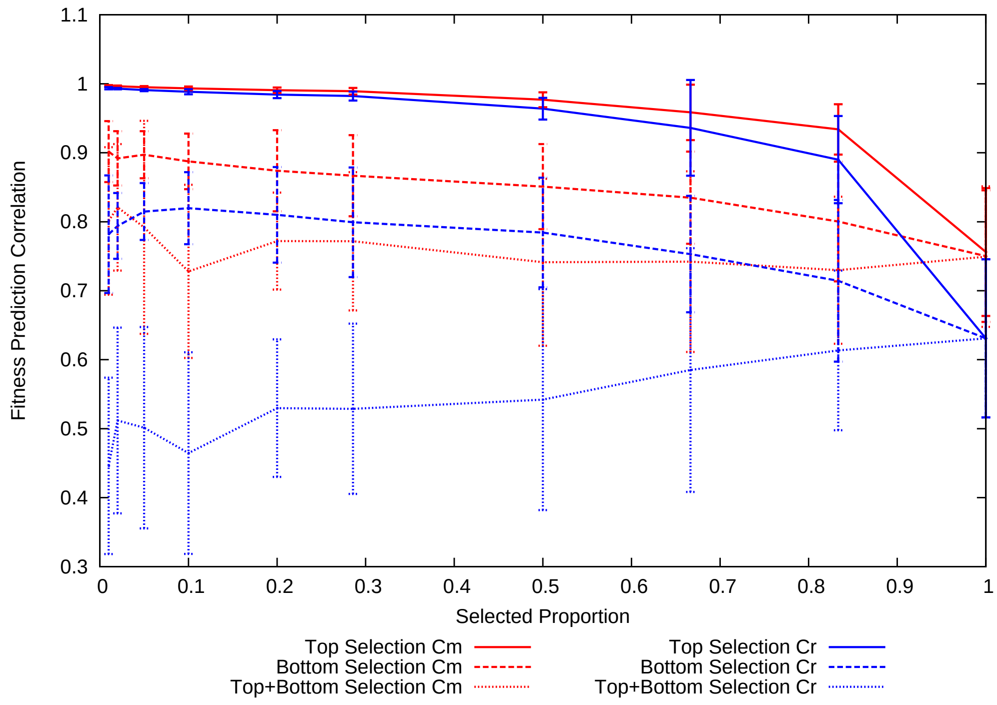


Figure C.70: FPC against selection proportion for fully specified 36 bit 2D Ising problem with 0.5 decimation

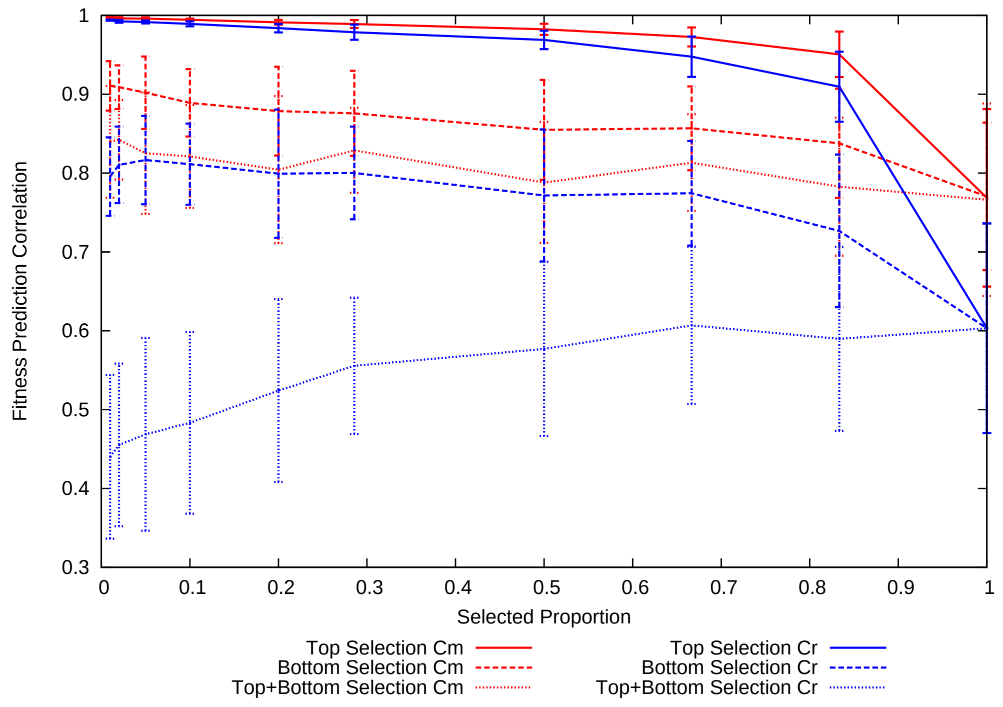


Figure C.71: FPC against selection proportion for fully specified 49 bit 2D Ising problem with 0.5 decimation

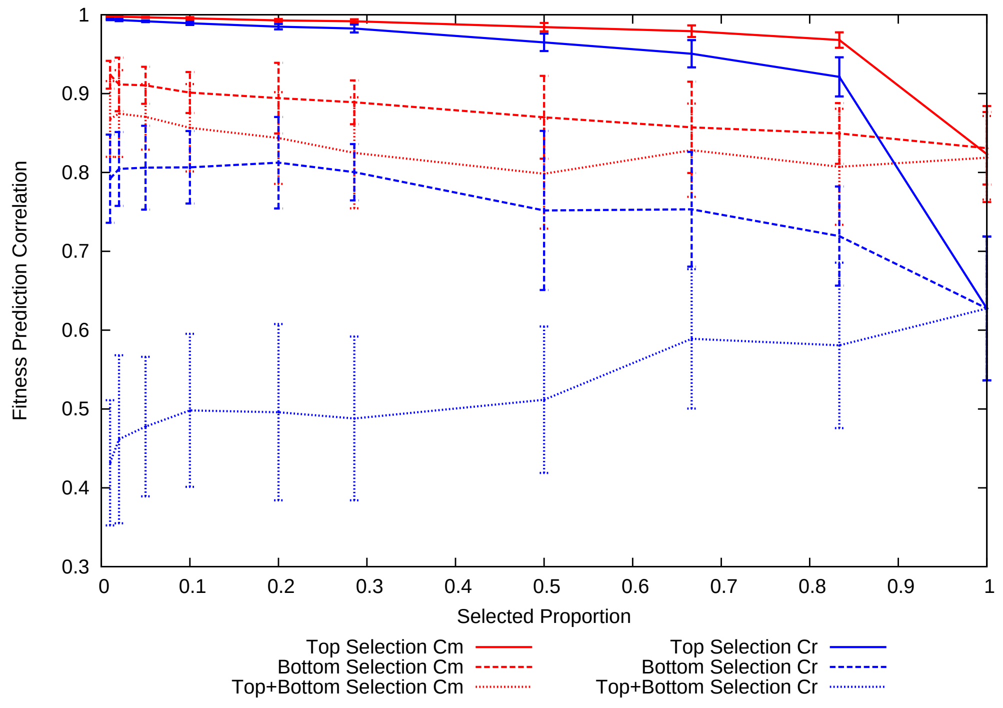


Figure C.72: FPC against selection proportion for fully specified 64 bit 2D Ising problem with 0.5 decimation

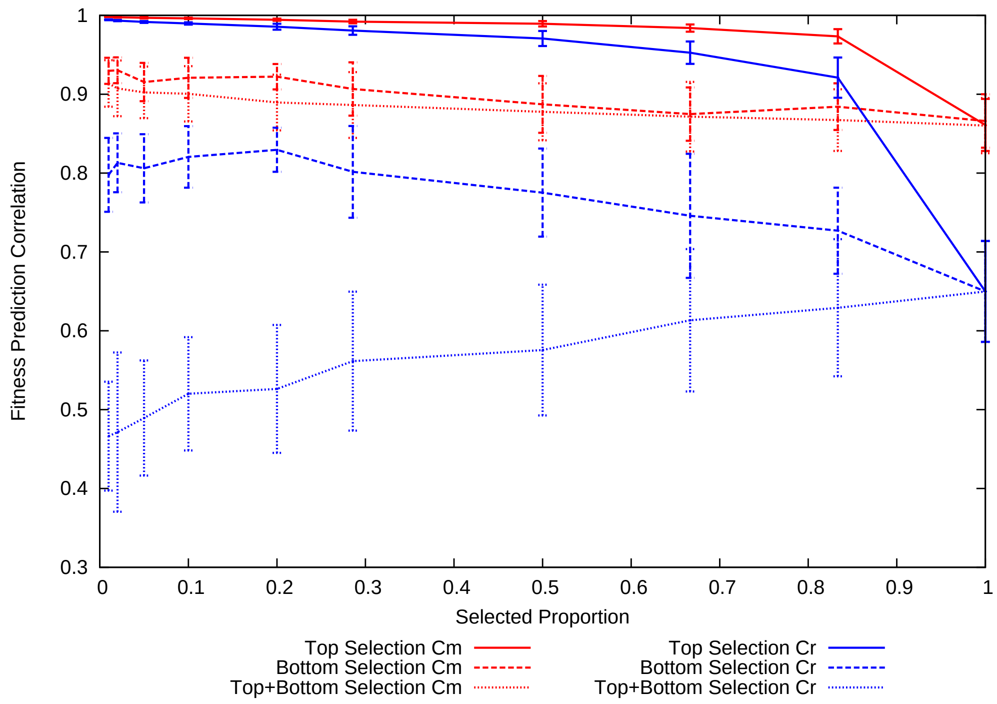


Figure C.73: FPC against selection proportion for fully specified 100 bit 2D Ising problem with 0.5 decimation

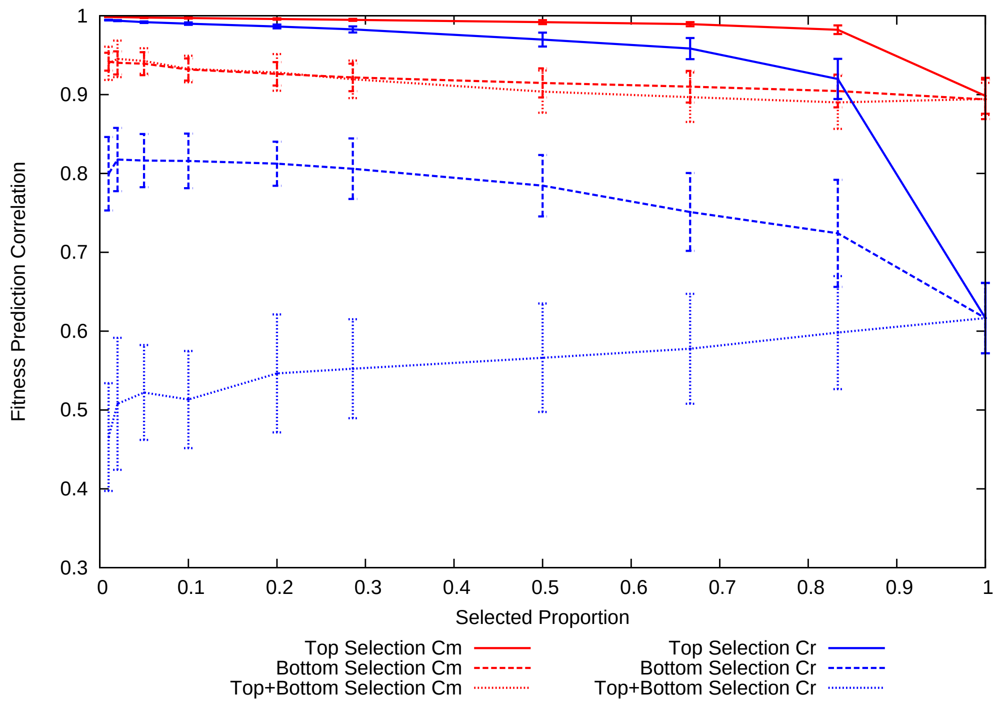


Figure C.74: FPC against selection proportion for fully specified 256 bit 2D Ising problem with 0.5 decimation

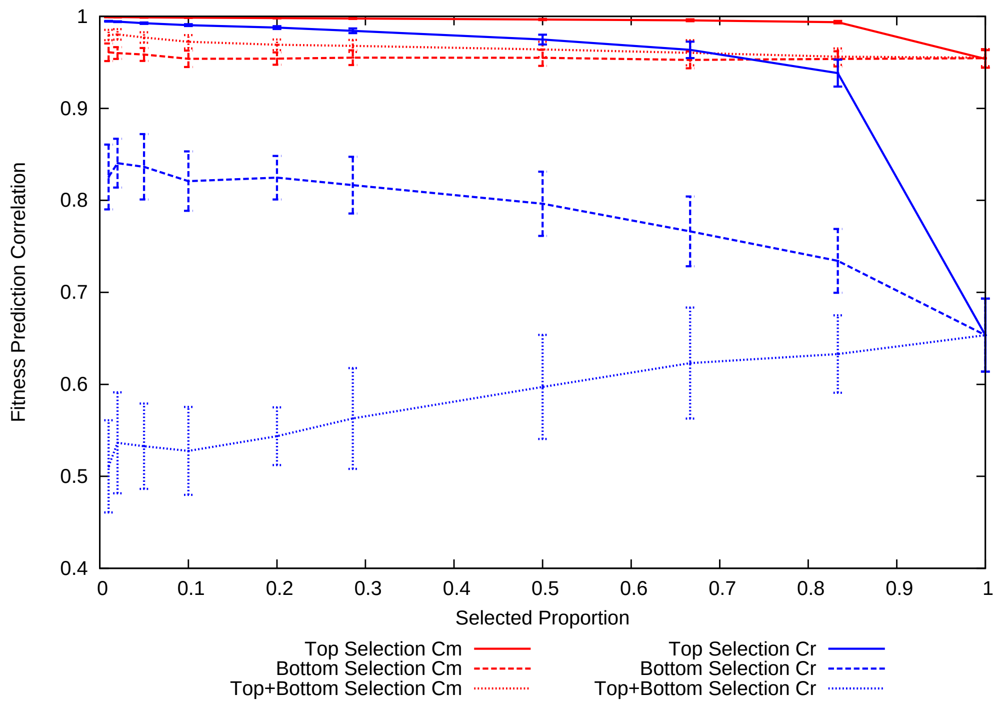


Figure C.75: FPC against selection proportion for fully specified 324 bit 2D Ising problem with 0.5 decimation

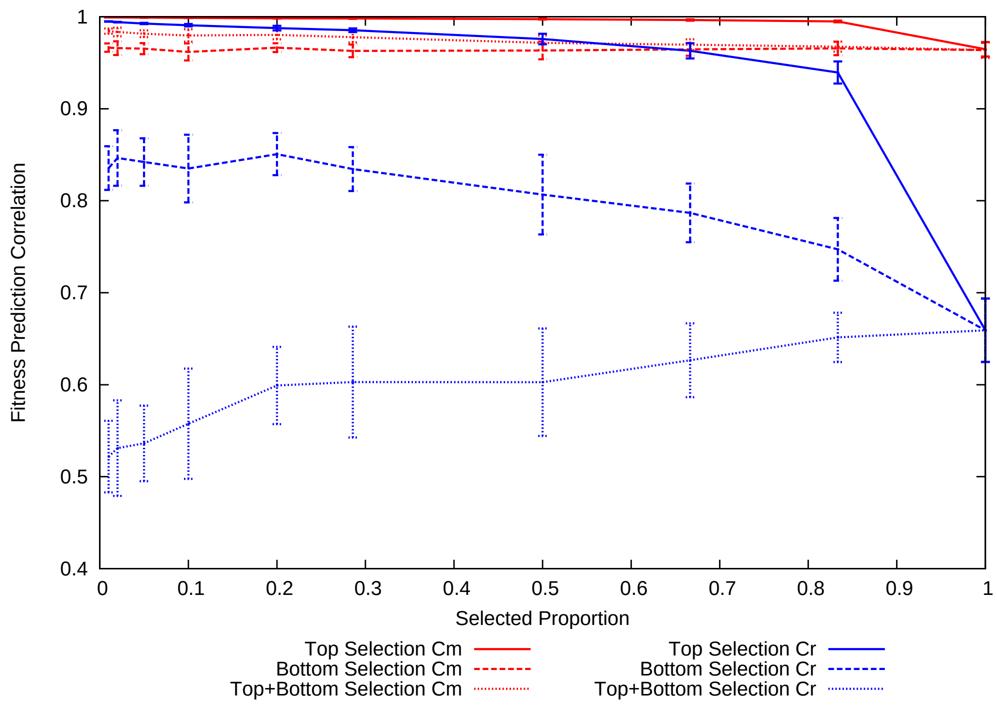


Figure C.76: FPC against selection proportion for fully specified 400 bit 2D Ising problem with 0.5 decimation

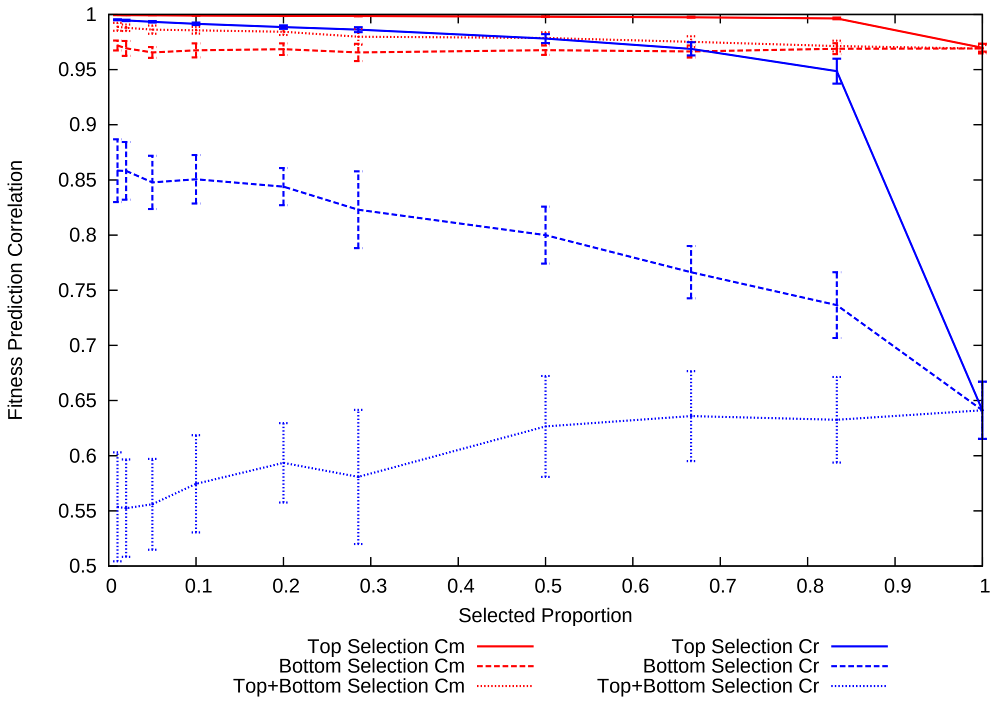


Figure C.77: FPC against selection proportion for fully specified 20 bit MaxSAT with only univariate terms in the model

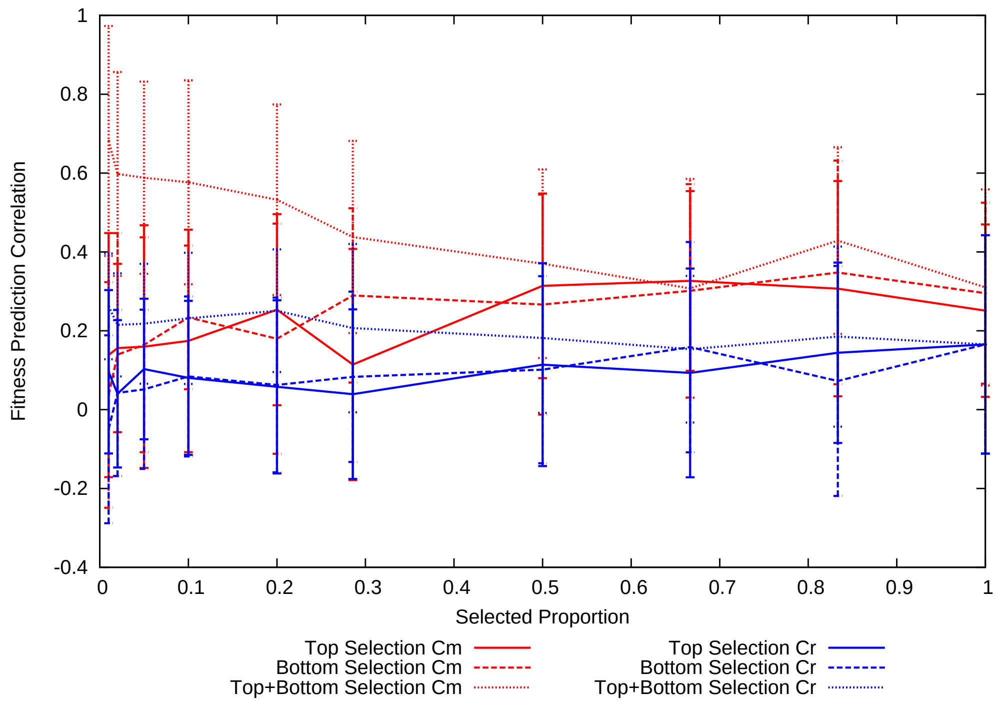


Figure C.78: FPC against selection proportion for fully specified 50 bit MaxSAT with only univariate terms in the model

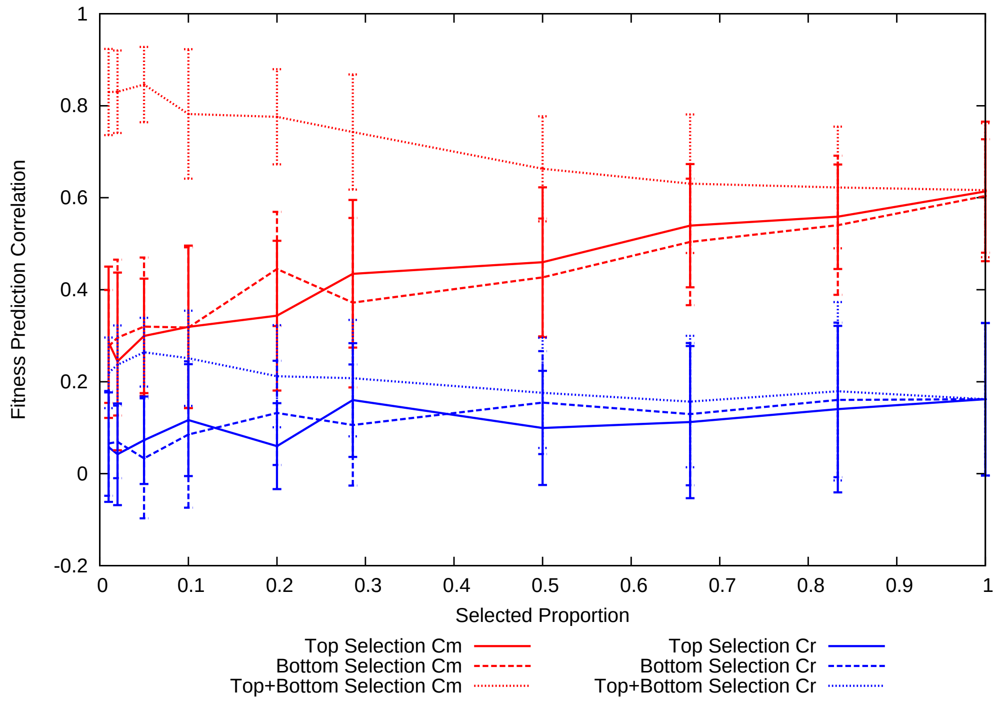


Figure C.79: FPC against selection proportion for fully specified 75 bit MaxSAT with only univariate terms in the model

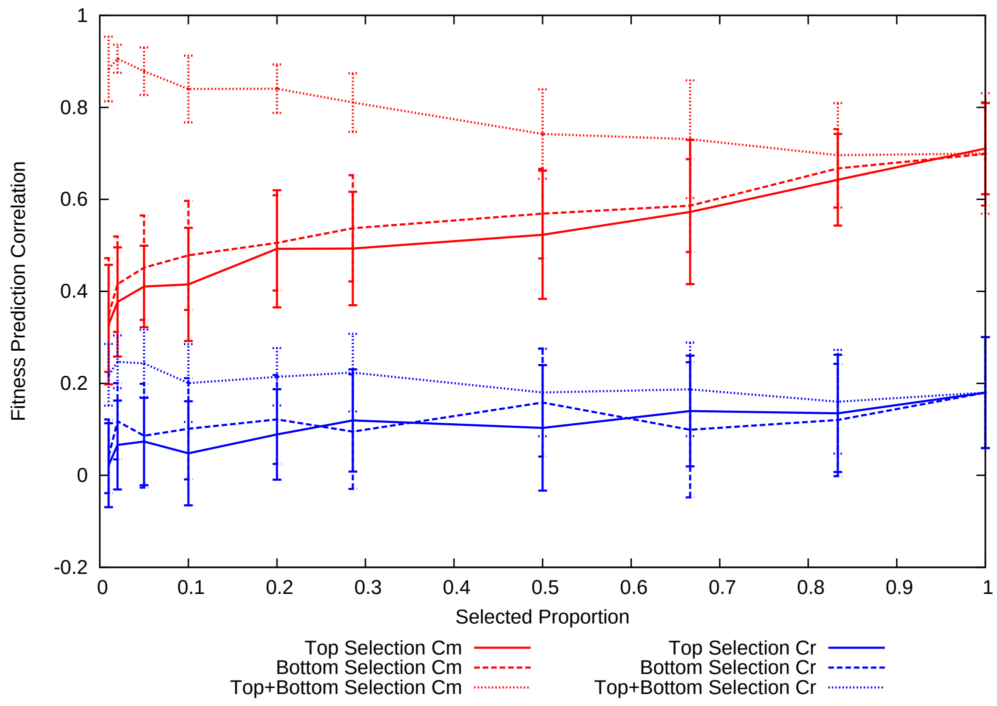


Figure C.80: FPC against selection proportion for fully specified 100 bit MaxSAT with only univariate terms in the model

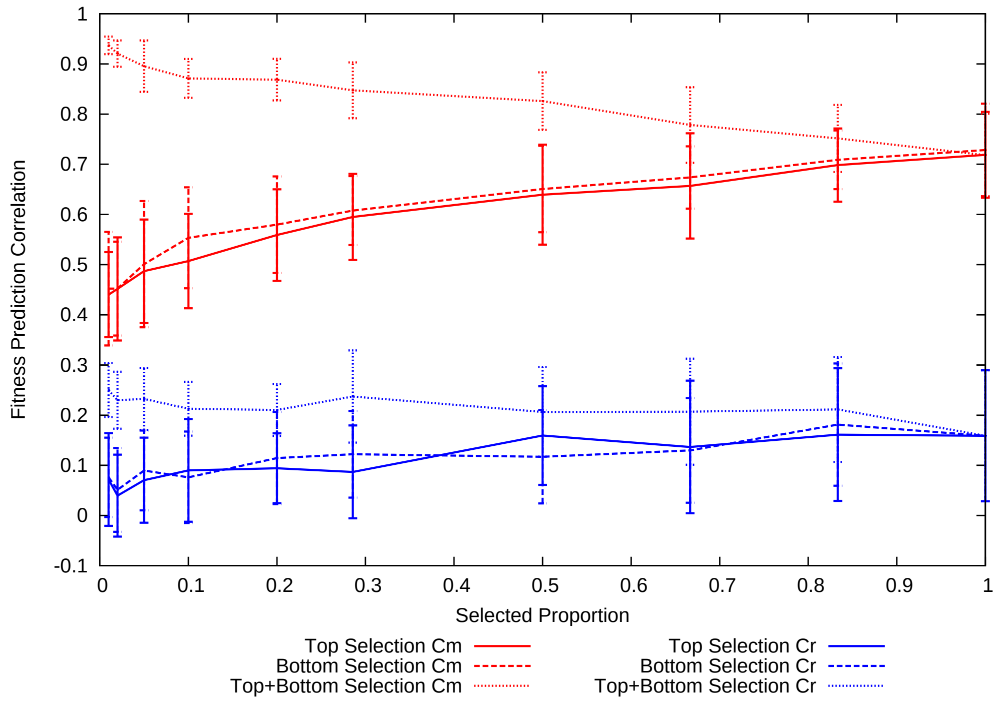


Figure C.81: FPC against selection proportion for fully specified 125 bit MaxSAT with only univariate terms in the model

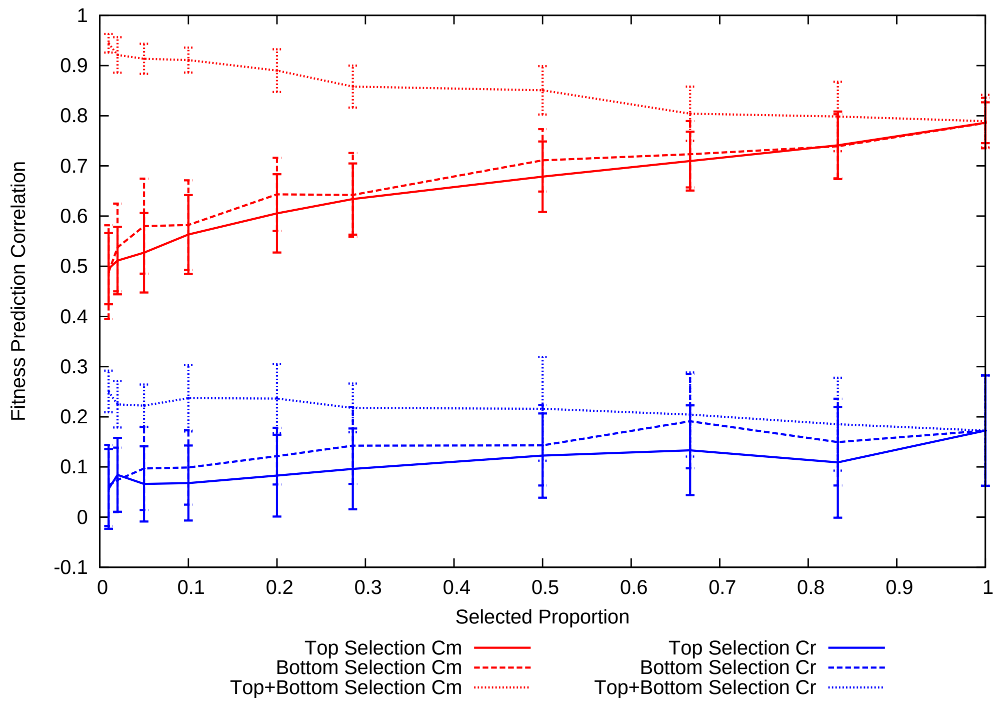


Figure C.82: FPC against selection proportion for fully specified 150 bit MaxSAT with only univariate terms in the model

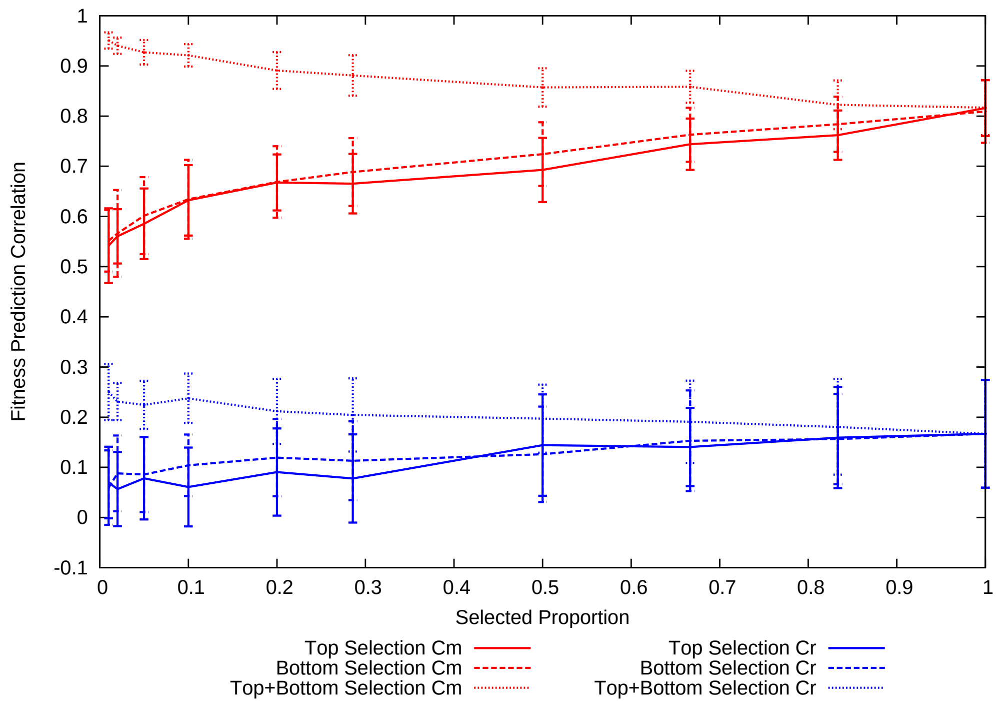


Figure C.83: FPC against selection proportion for fully specified 75 bit MaxSAT with only univariate terms in the model

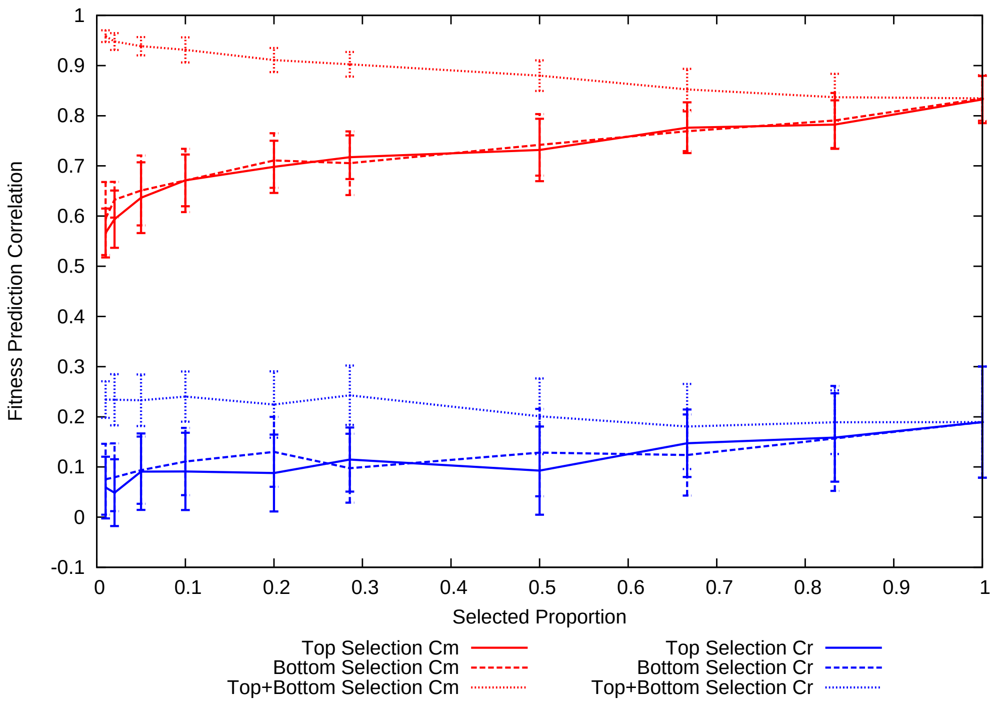


Figure C.84: FPC against selection proportion for fully specified 200 bit MaxSAT with only univariate terms in the model

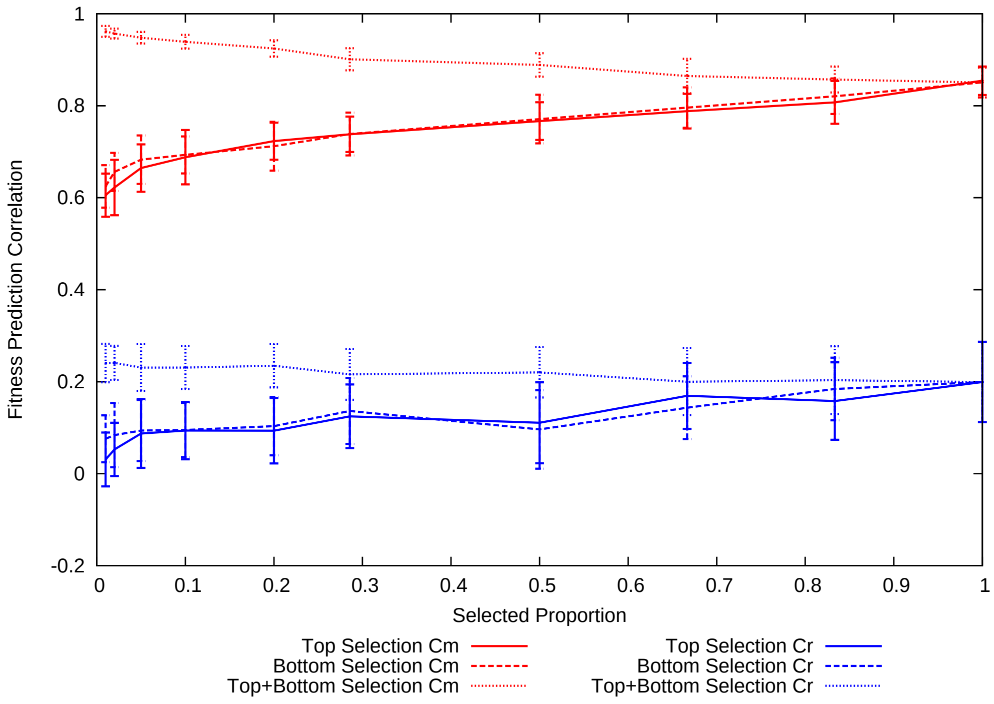


Figure C.85: FPC against selection proportion for fully specified 250 bit MaxSAT with only univariate terms in the model

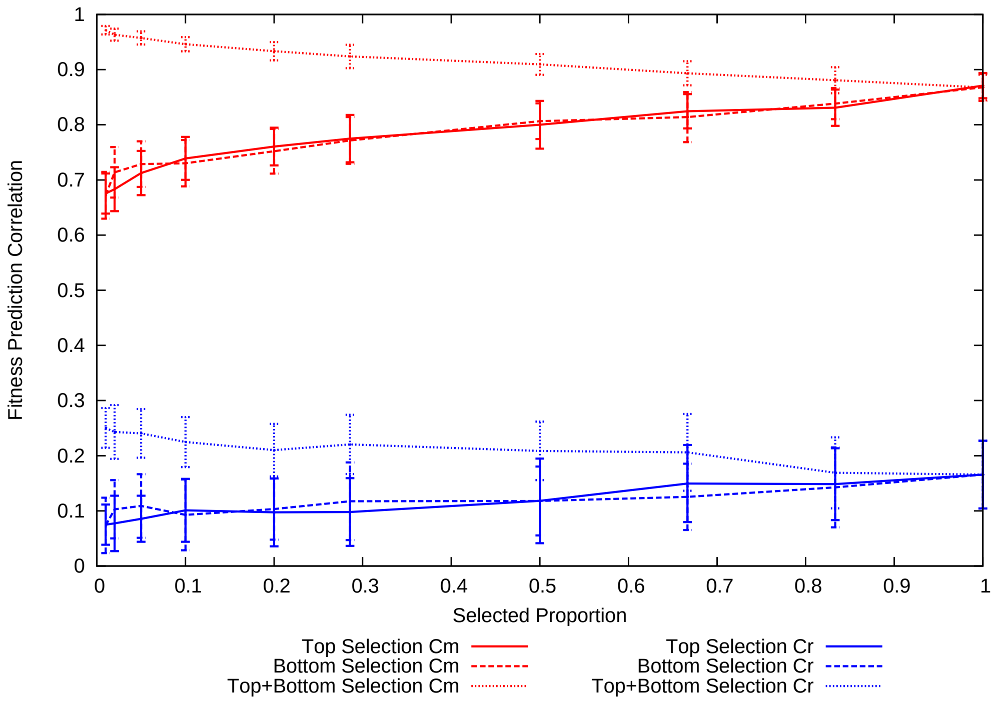


Figure C.86: FPC against selection proportion for fully specified 20 bit MaxSAT with trivariate and univariate terms in the model

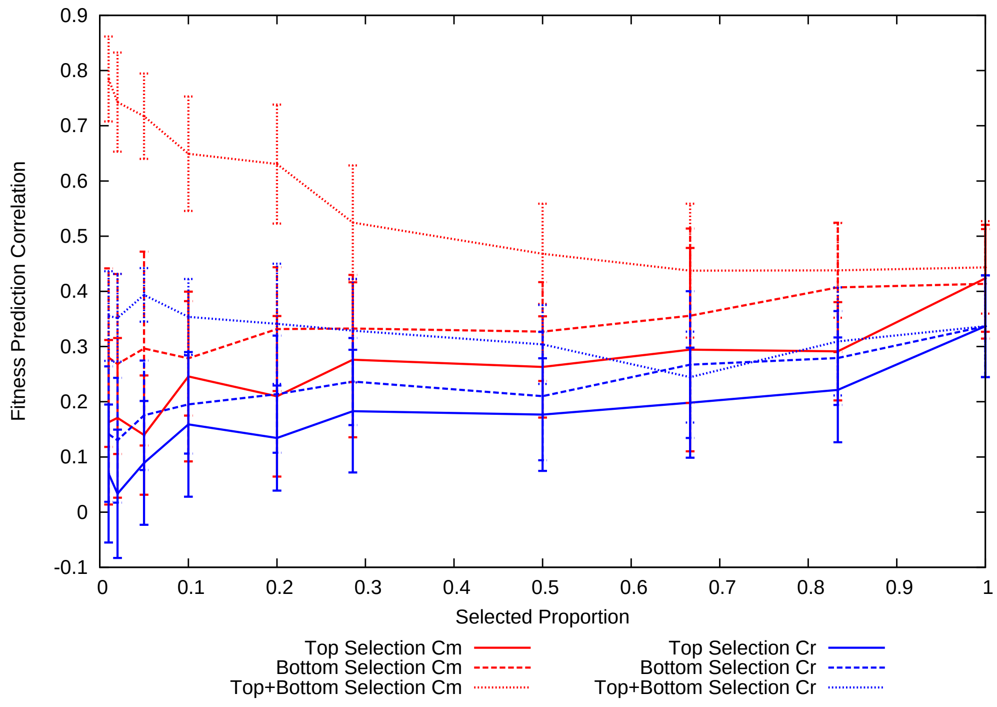


Figure C.87: FPC against selection proportion for fully specified 50 bit MaxSAT with trivariate and univariate terms in the model

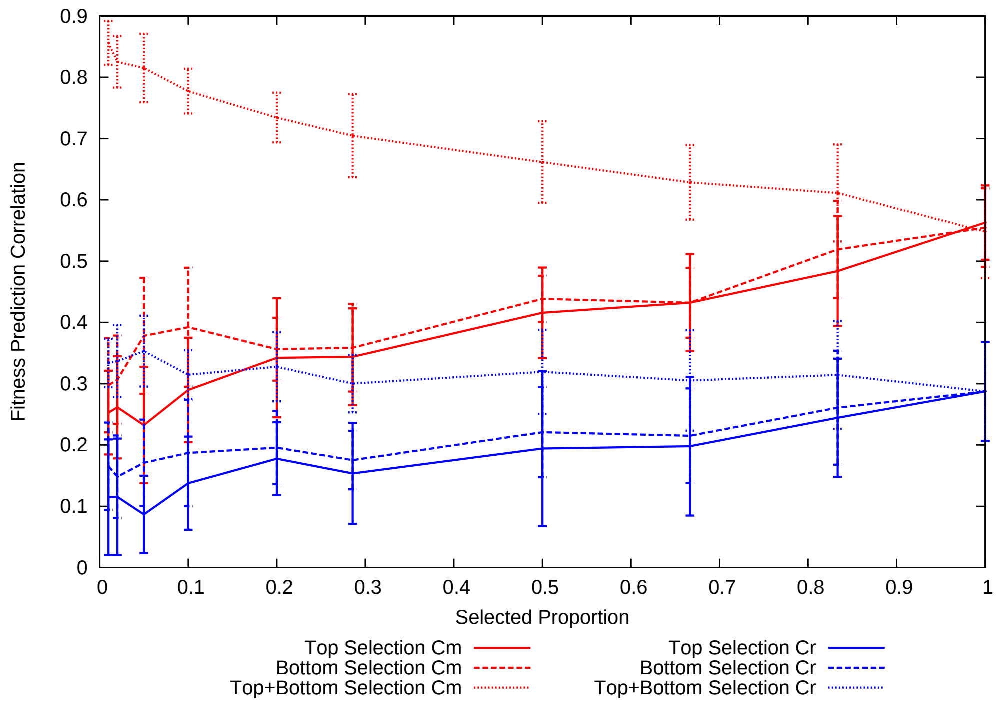


Figure C.88: FPC against selection proportion for fully specified 75 bit MaxSAT with trivariate and univariate terms in the model

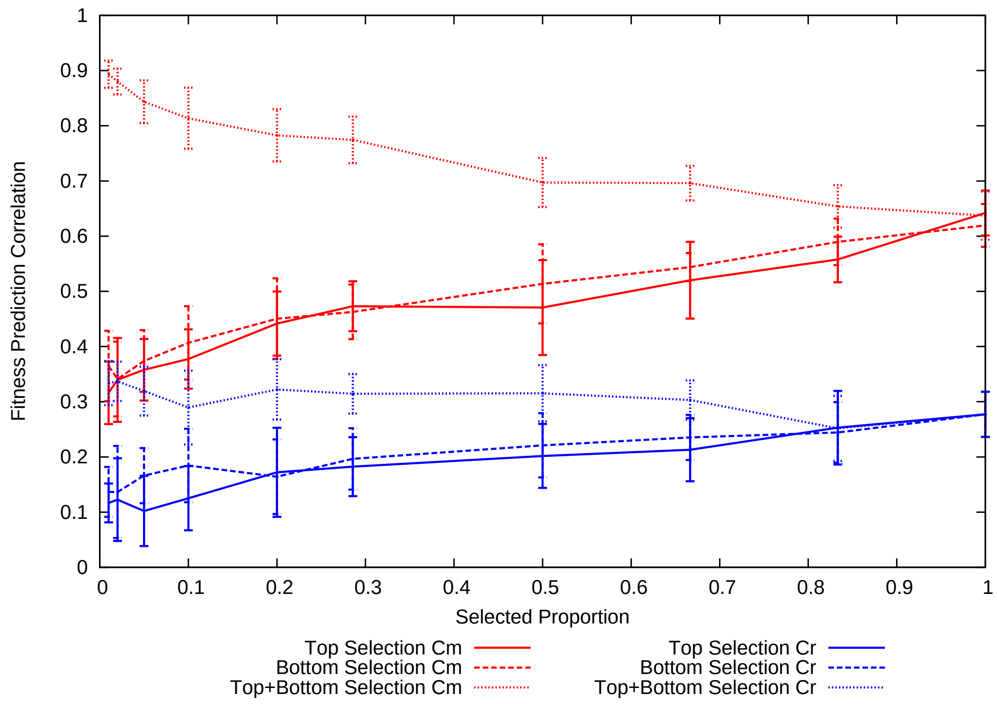


Figure C.89: FPC against selection proportion for fully specified 100 bit MaxSAT with trivariate and univariate terms in the model

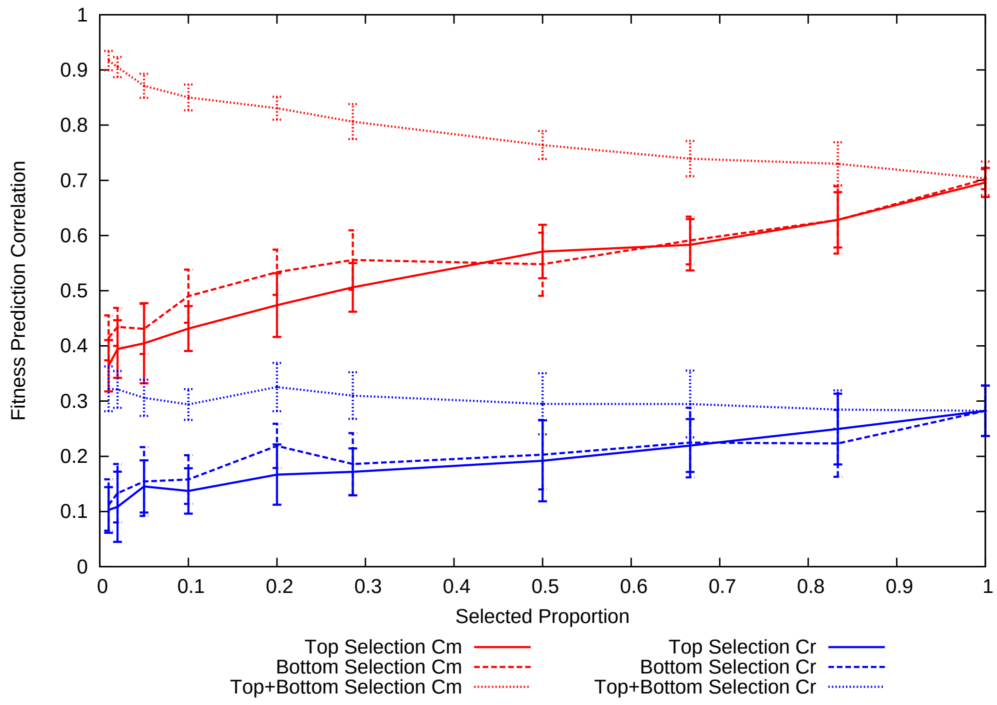


Figure C.90: FPC against selection proportion for fully specified 125 bit MaxSAT with trivariate and univariate terms in the model

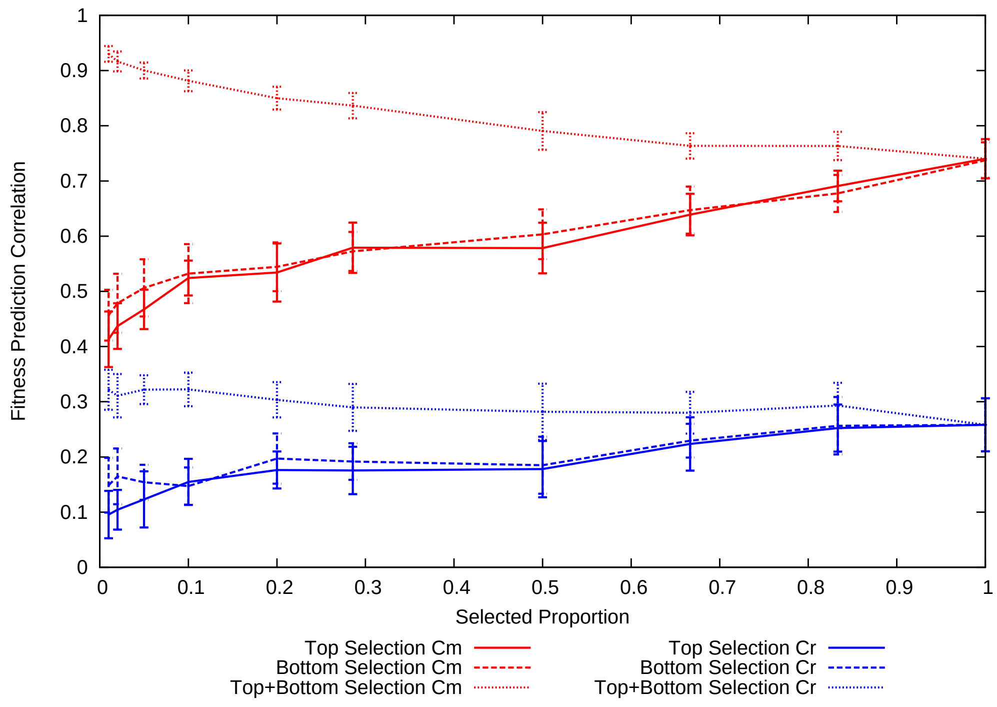


Figure C.91: FPC against selection proportion for fully specified 150 bit MaxSAT with trivariate and univariate terms in the model

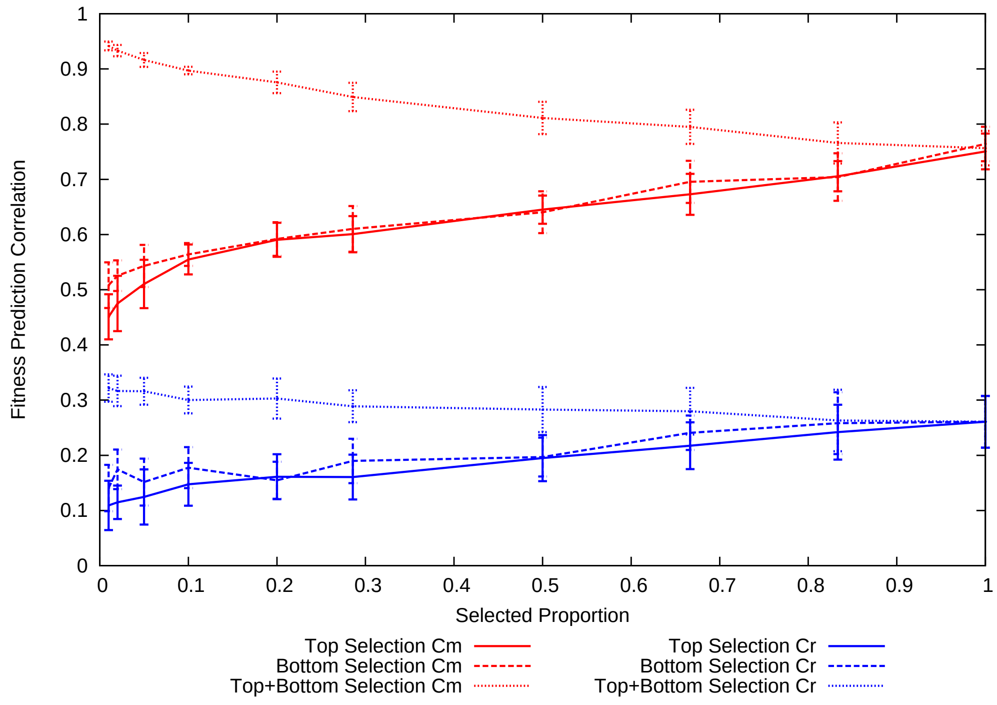


Figure C.92: FPC against selection proportion for fully specified 175 bit MaxSAT with trivariate and univariate terms in the model

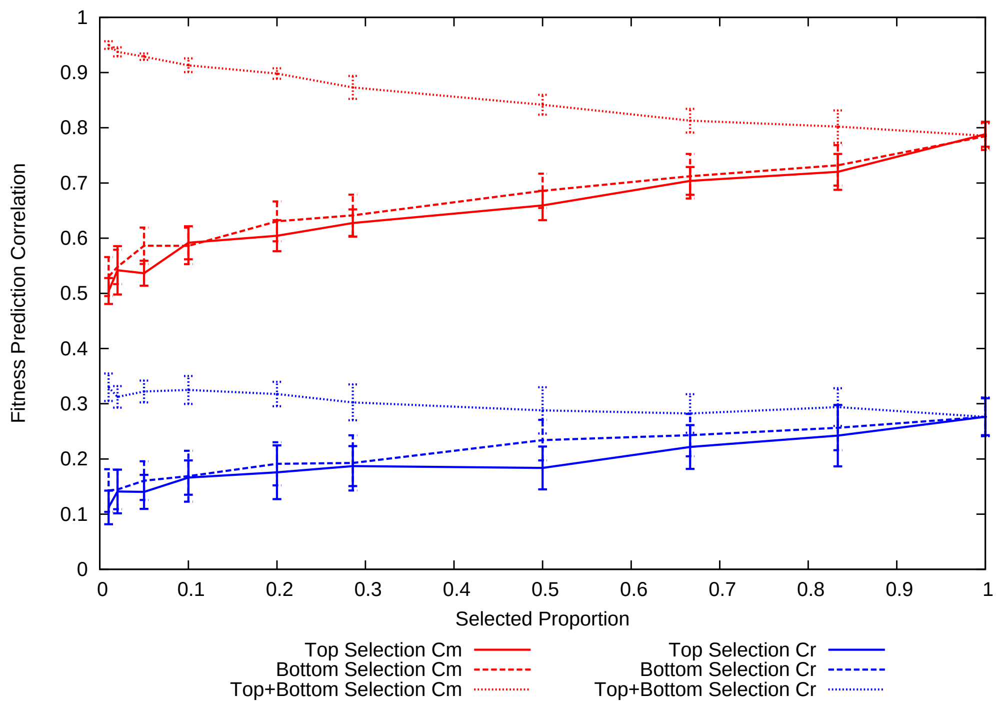


Figure C.93: FPC against selection proportion for fully specified 200 bit MaxSAT with trivariate and univariate terms in the model

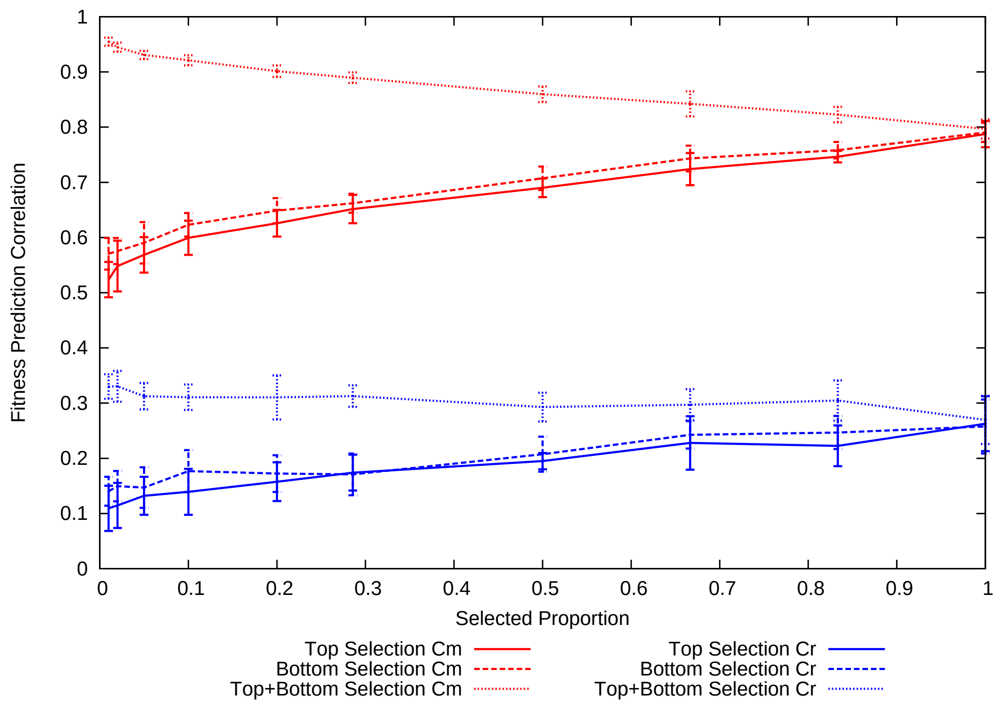
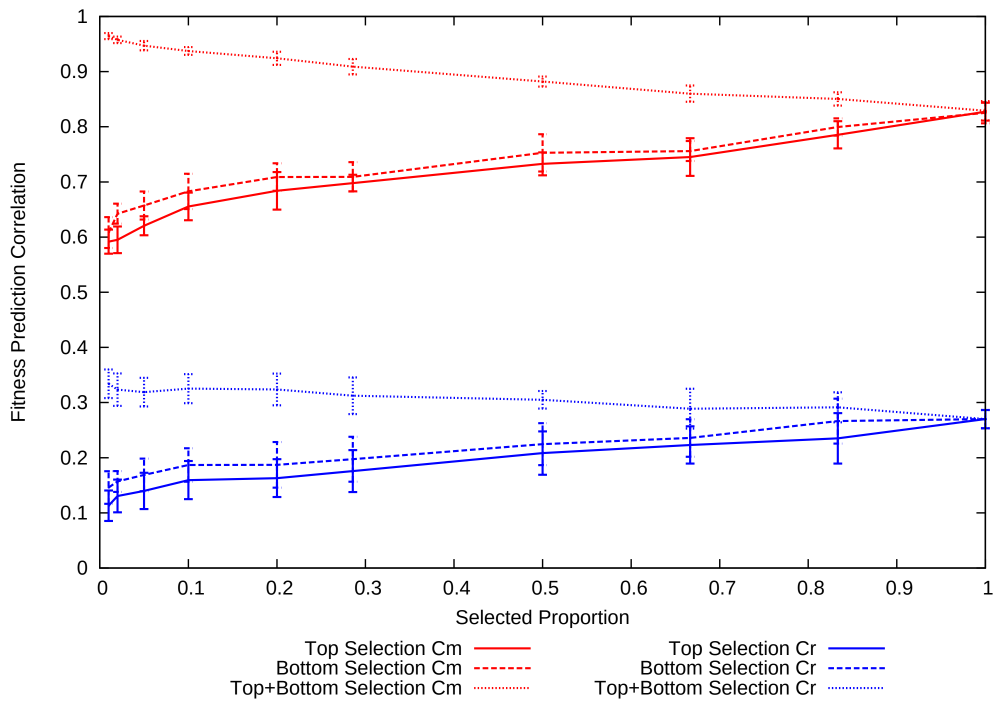


Figure C.94: FPC against selection proportion for fully specified 250 bit MaxSAT with trivariate and univariate terms in the model



Appendix D

Figures - Structure Learning Experiments

The figures in this chapter relate to the experiments discussed in Chapter 7.

Figure D.1: Number of interactions found vs selection proportion for 10 bit Onemax

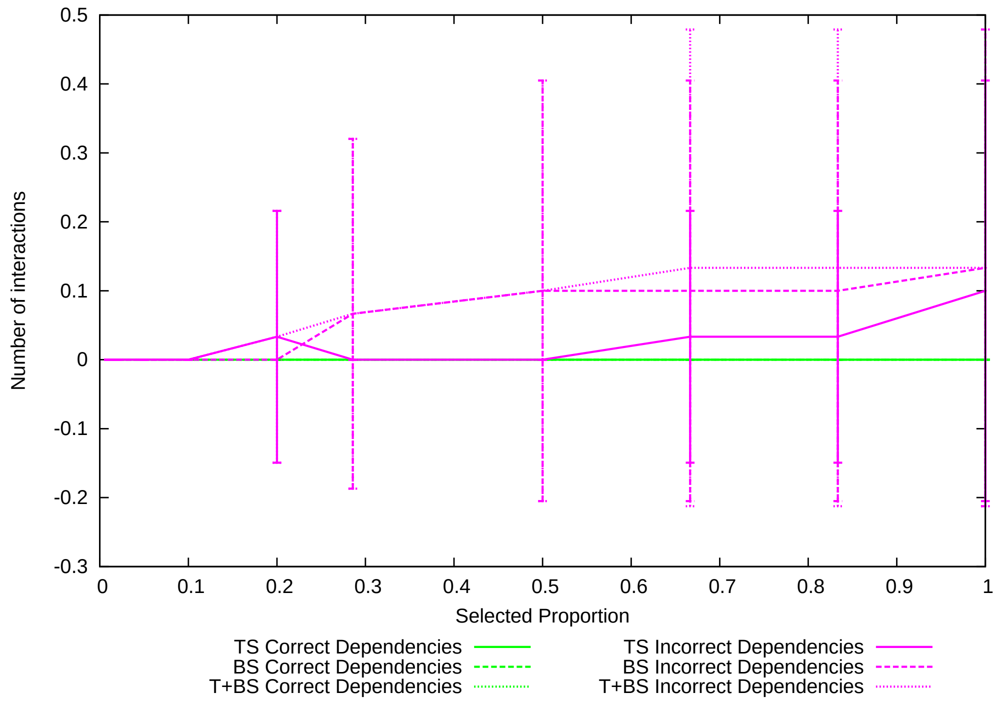


Figure D.2: Number of interactions found vs selection proportion for 20 bit Onemax

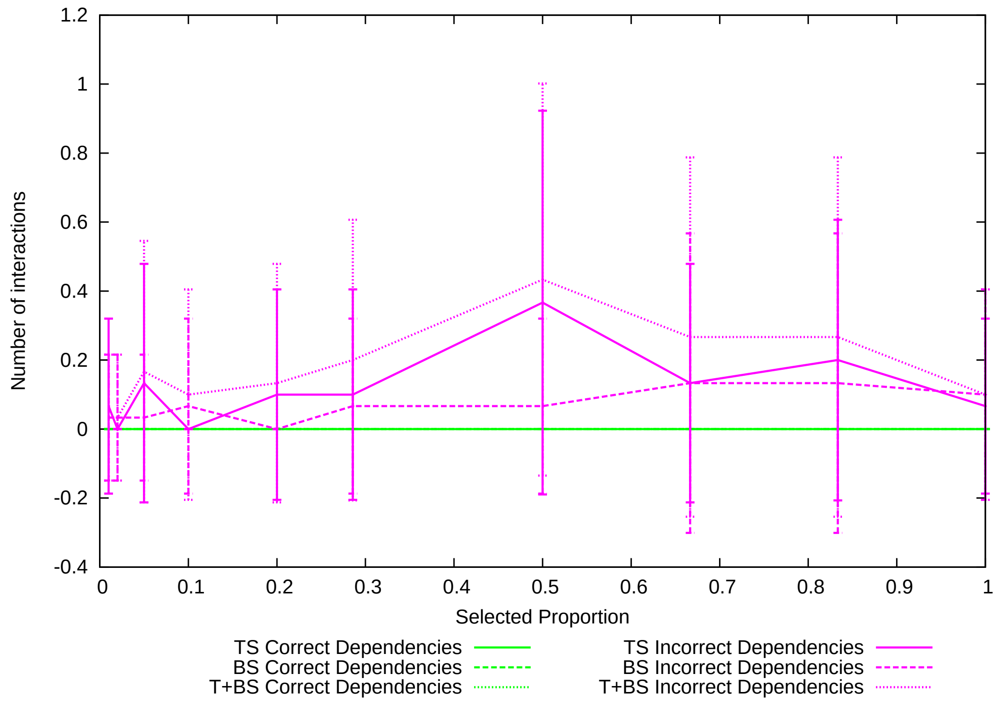


Figure D.3: Number of interactions found vs selection proportion for 50 bit Onemax

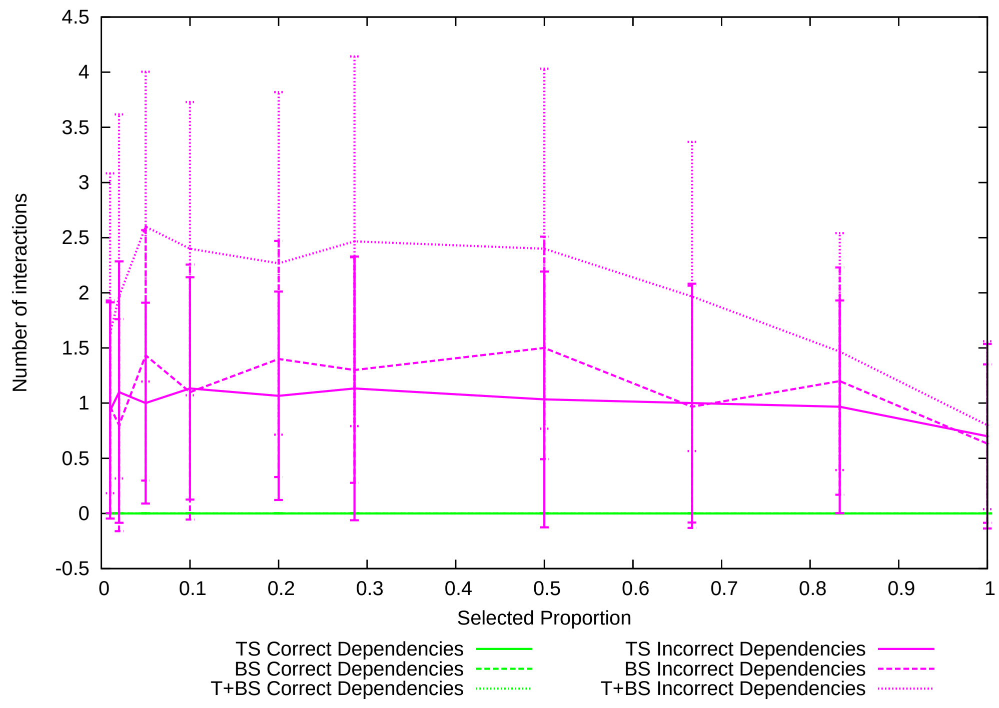


Figure D.4: Number of interactions found vs selection proportion for 75 bit Onemax

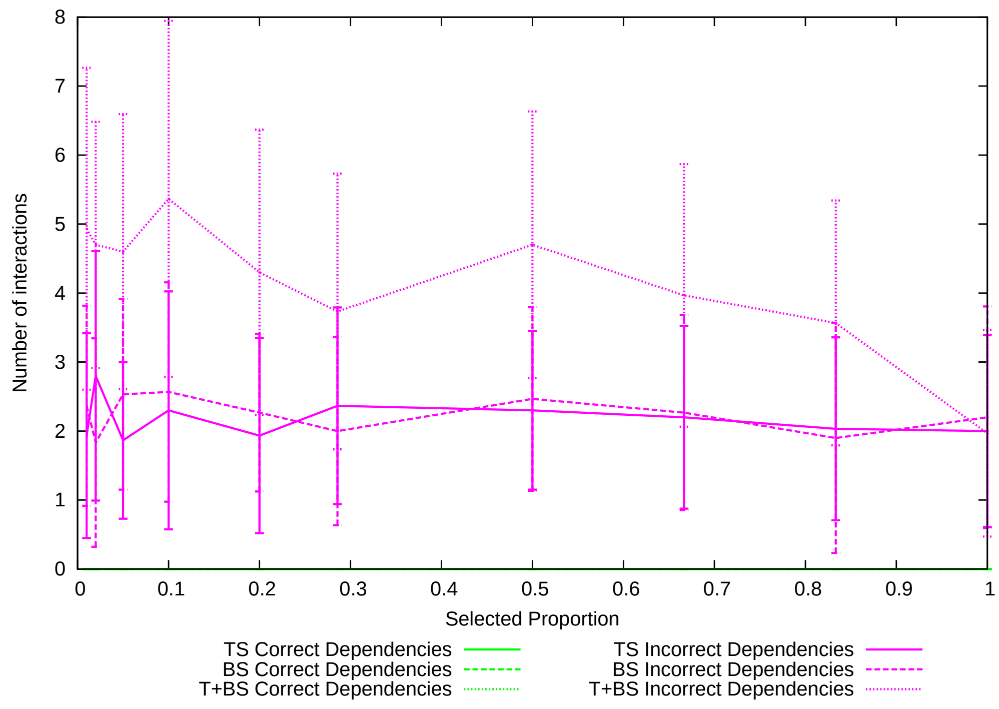


Figure D.5: Number of interactions found vs selection proportion for 100 bit Onemax

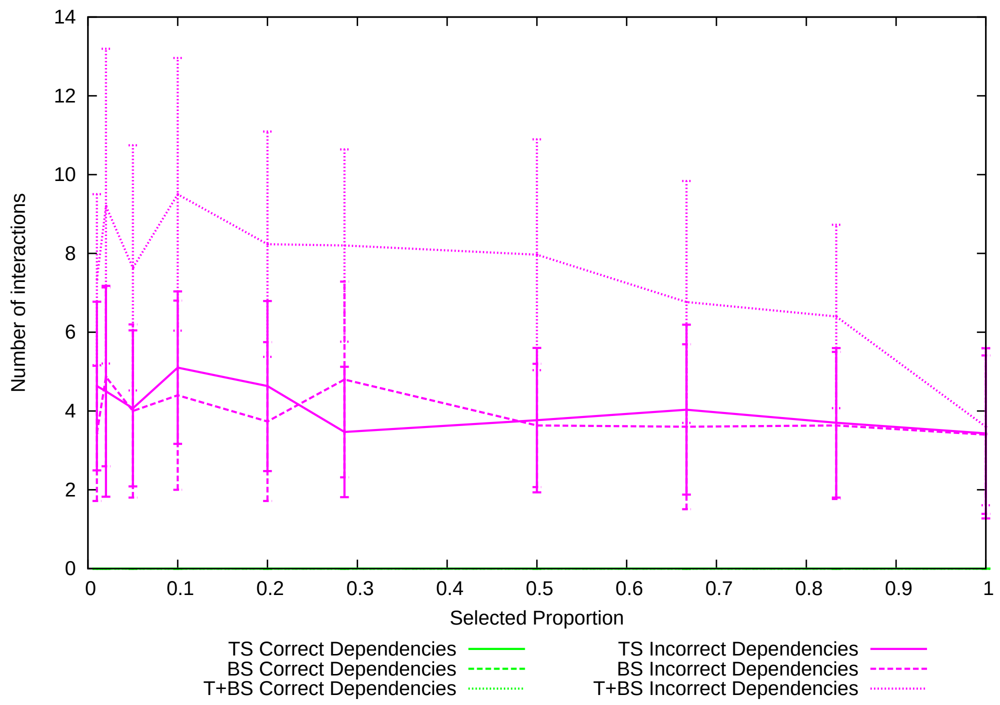


Figure D.6: Number of interactions found vs selection proportion for 200 bit Onemax

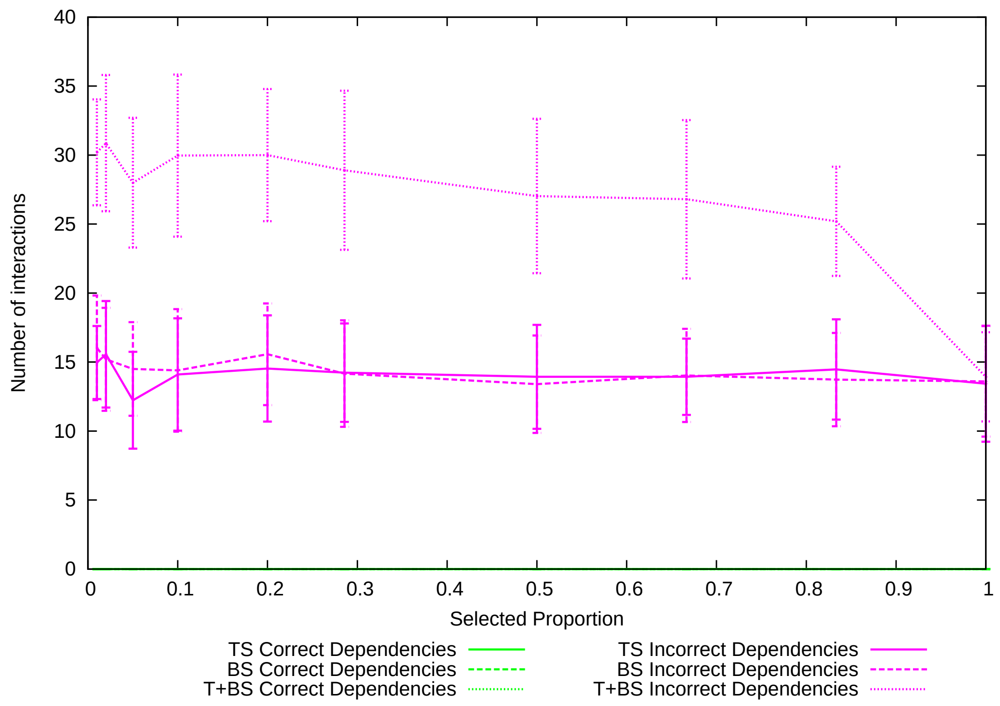


Figure D.7: Number of interactions found vs selection proportion for 500 bit Onemax

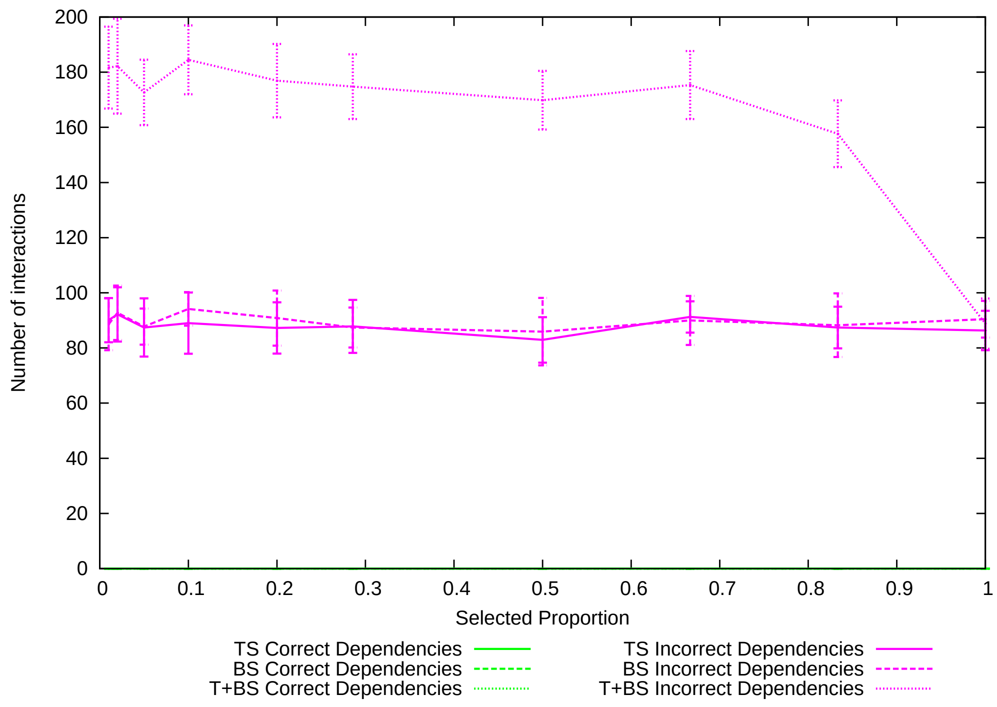


Figure D.8: Number of interactions found vs selection proportion for 750 bit Onemax

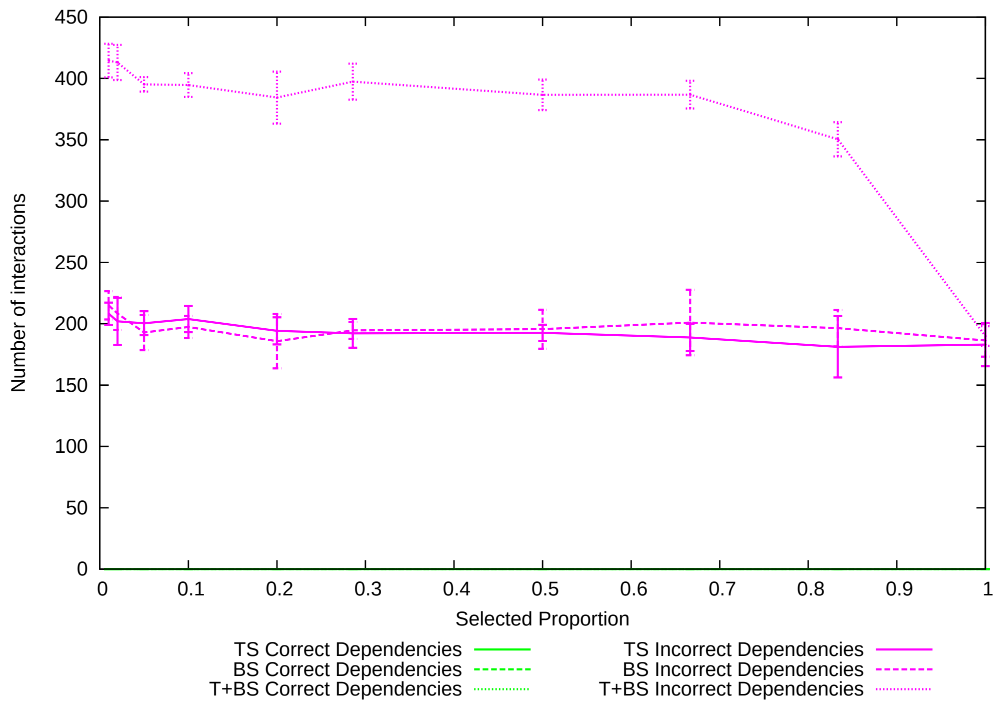


Figure D.9: Number of interactions found vs selection proportion for 1000 bit Onemax

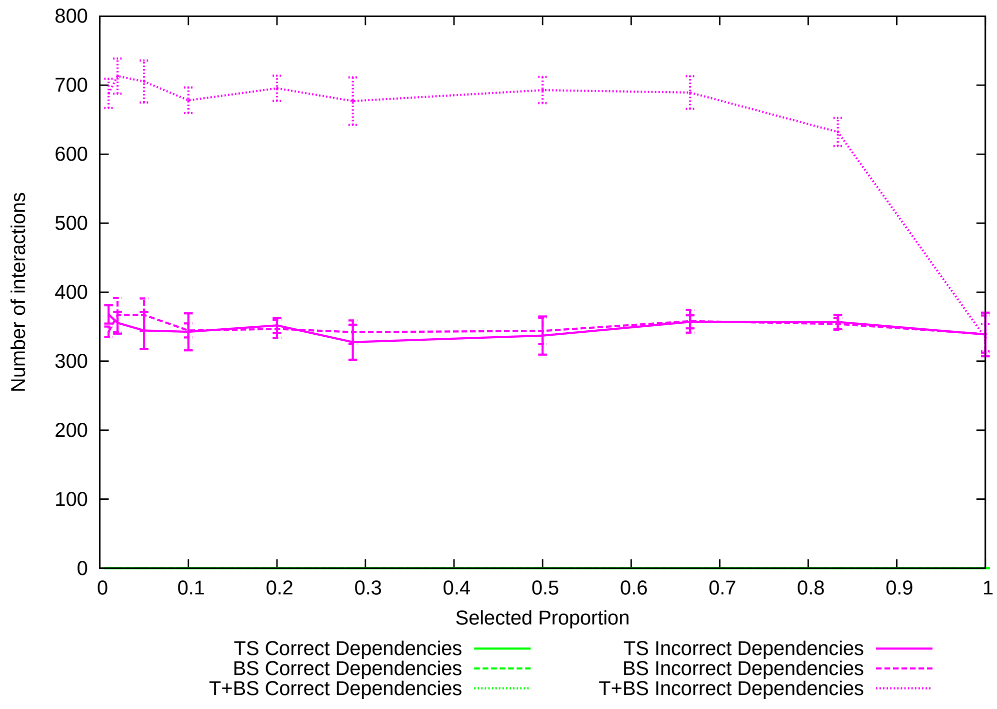


Figure D.10: P and R vs selection proportion for 16 bit 2D Ising

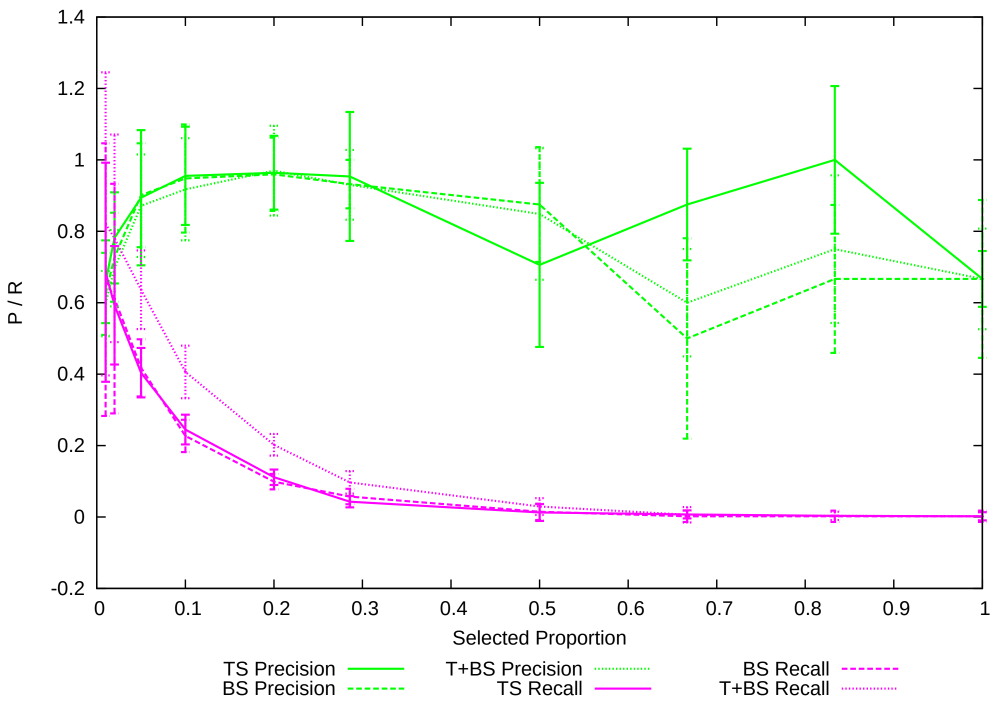


Figure D.11: P and R vs selection proportion for 25 bit 2D Ising

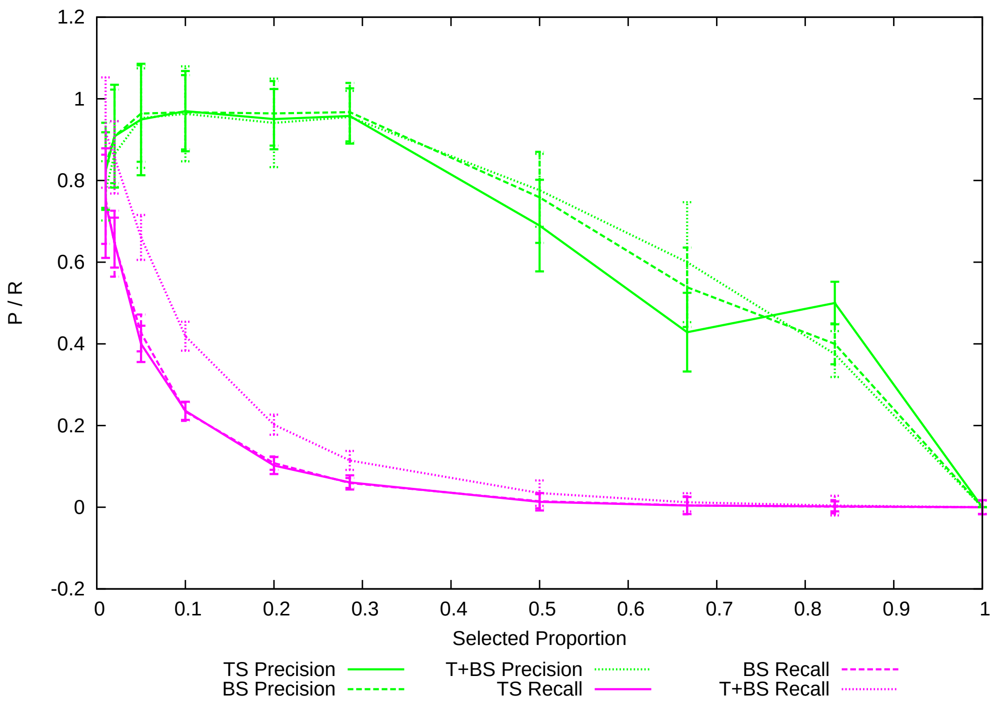


Figure D.12: P and R vs selection proportion for 36 bit 2D Ising

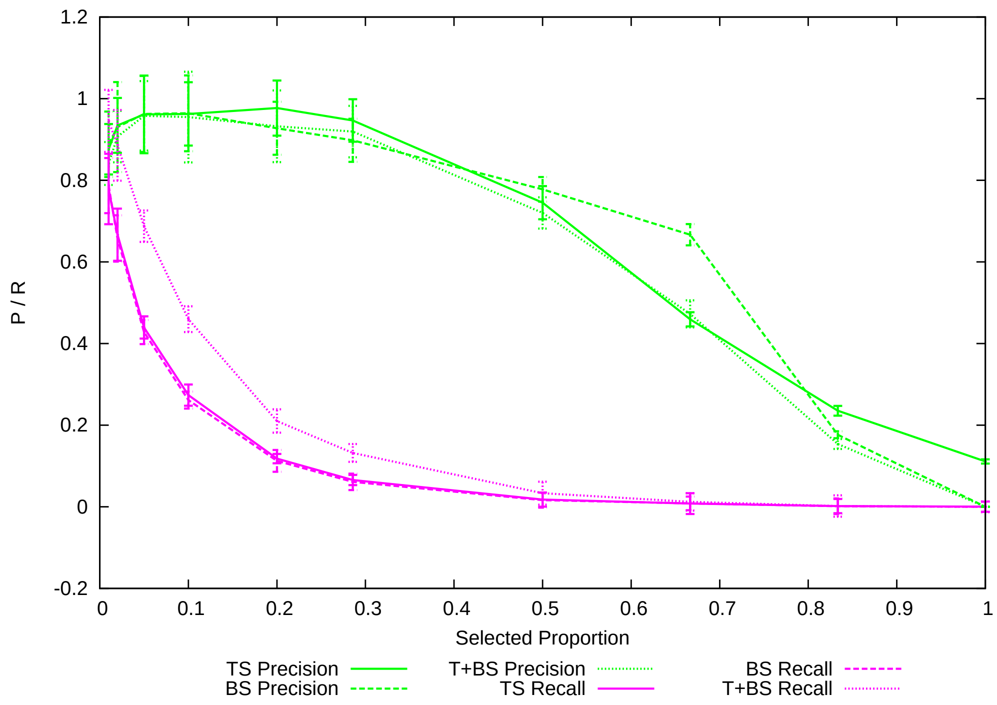


Figure D.13: P and R vs selection proportion for 49 bit 2D Ising

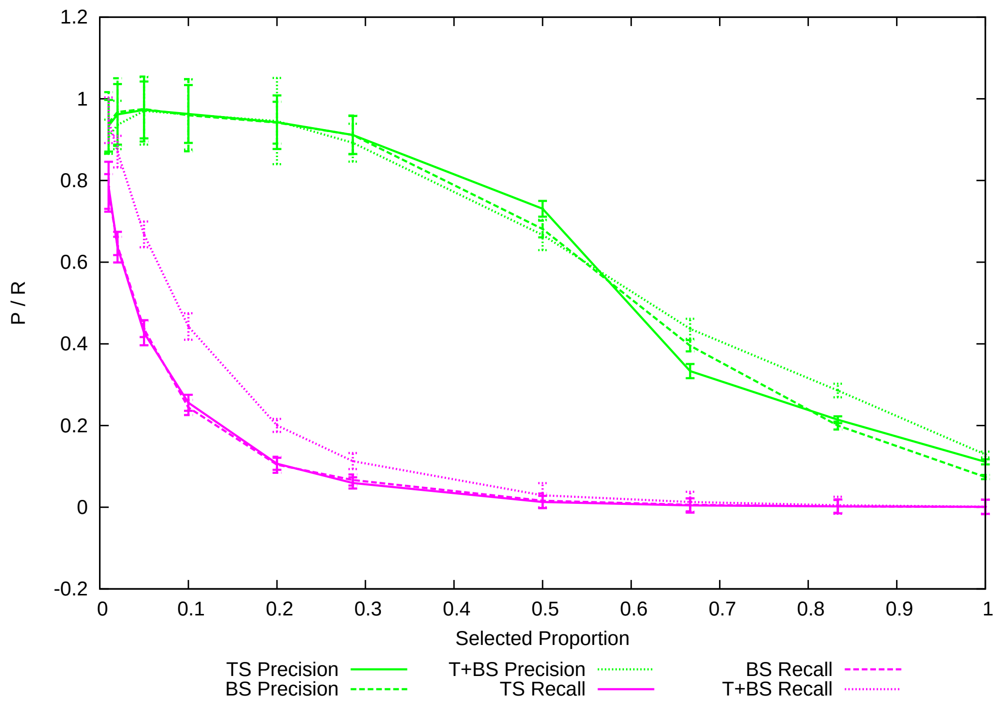


Figure D.14: P and R vs selection proportion for 64 bit 2D Ising

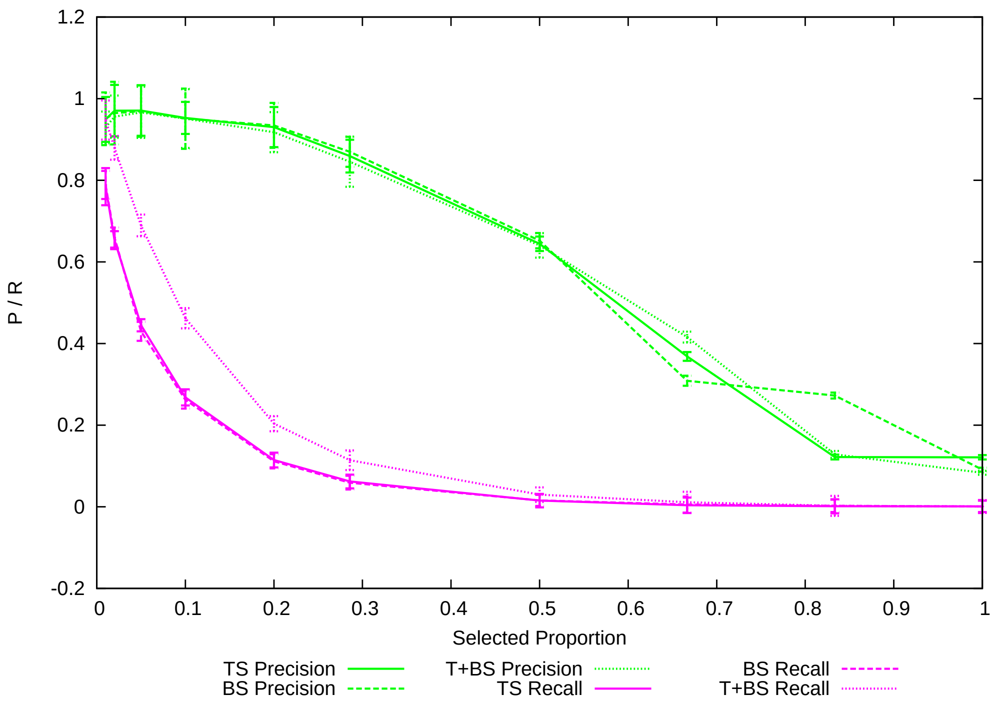


Figure D.15: P and R vs selection proportion for 100 bit 2D Ising

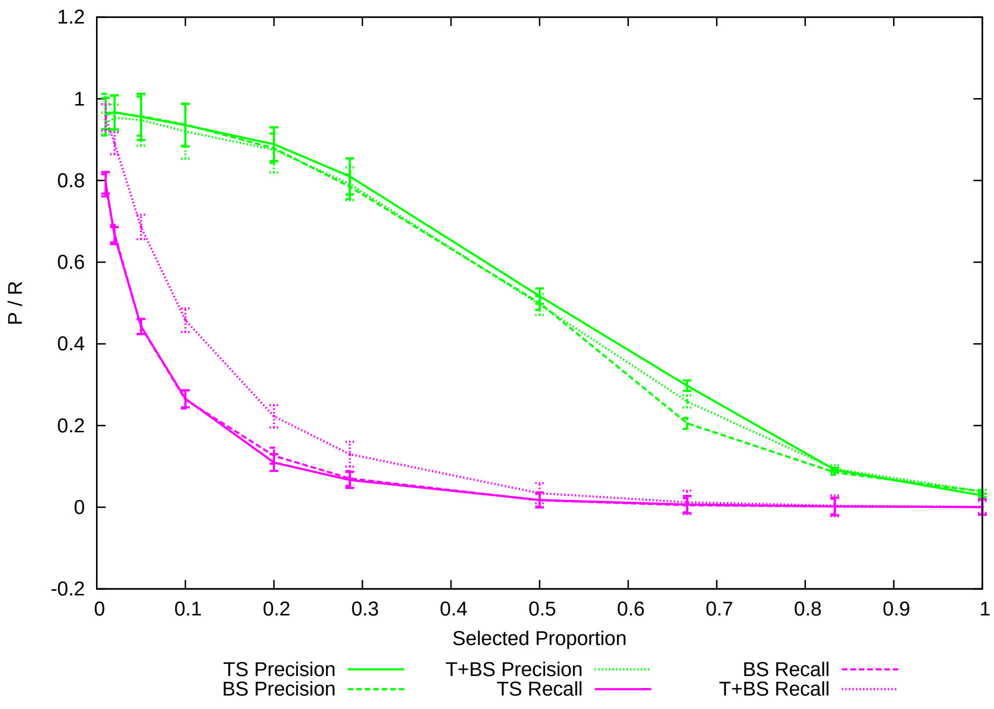


Figure D.16: P and R vs selection proportion for 256 bit 2D Ising

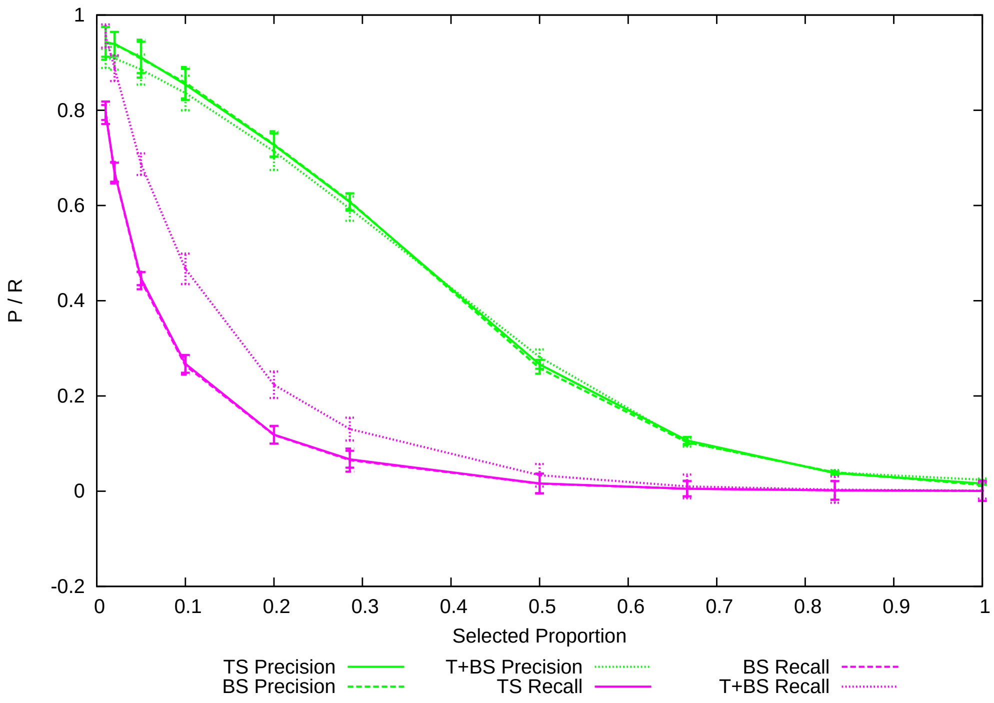


Figure D.17: P and R vs selection proportion for 324 bit 2D Ising

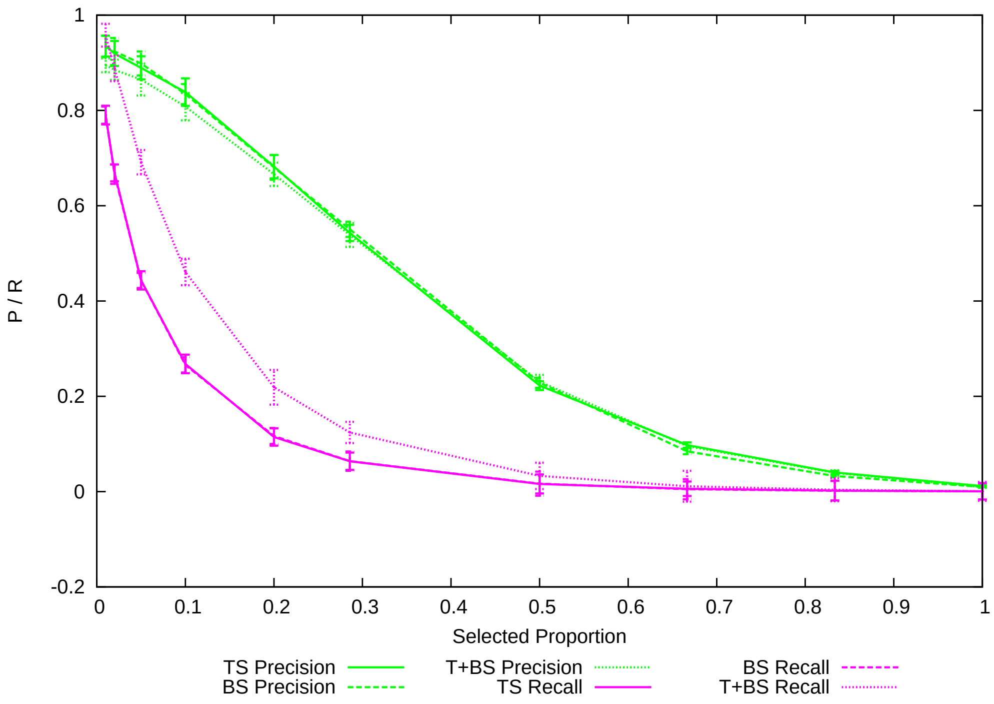


Figure D.18: P and R vs selection proportion for 400 bit 2D Ising

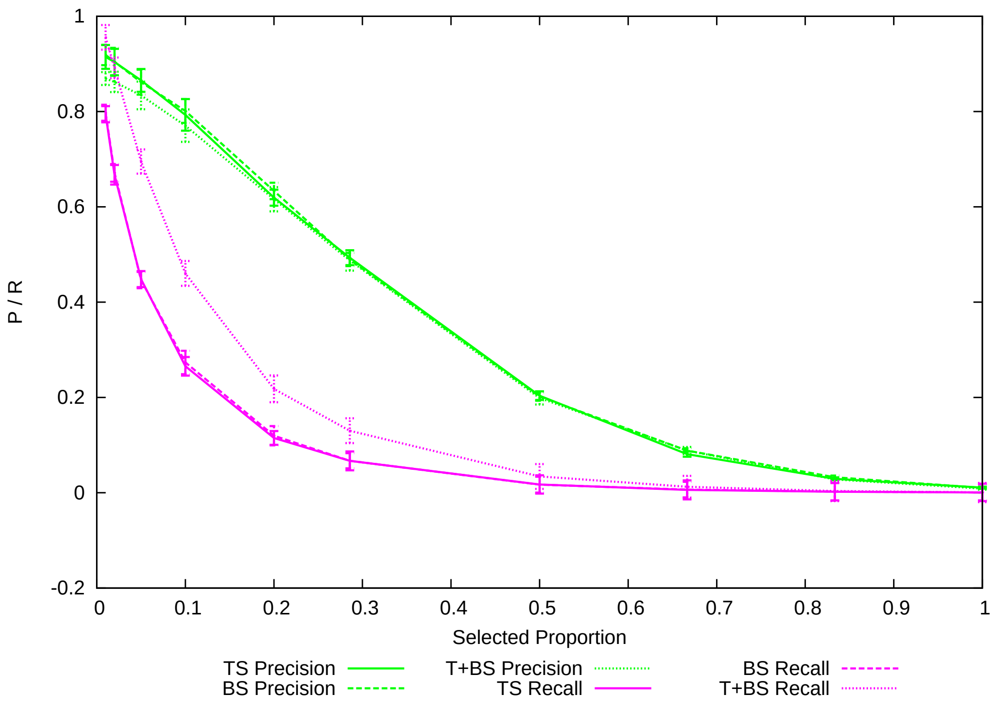


Figure D.19: Precision and recall of interactions learned for 100 bit Ising, selecting top 1%

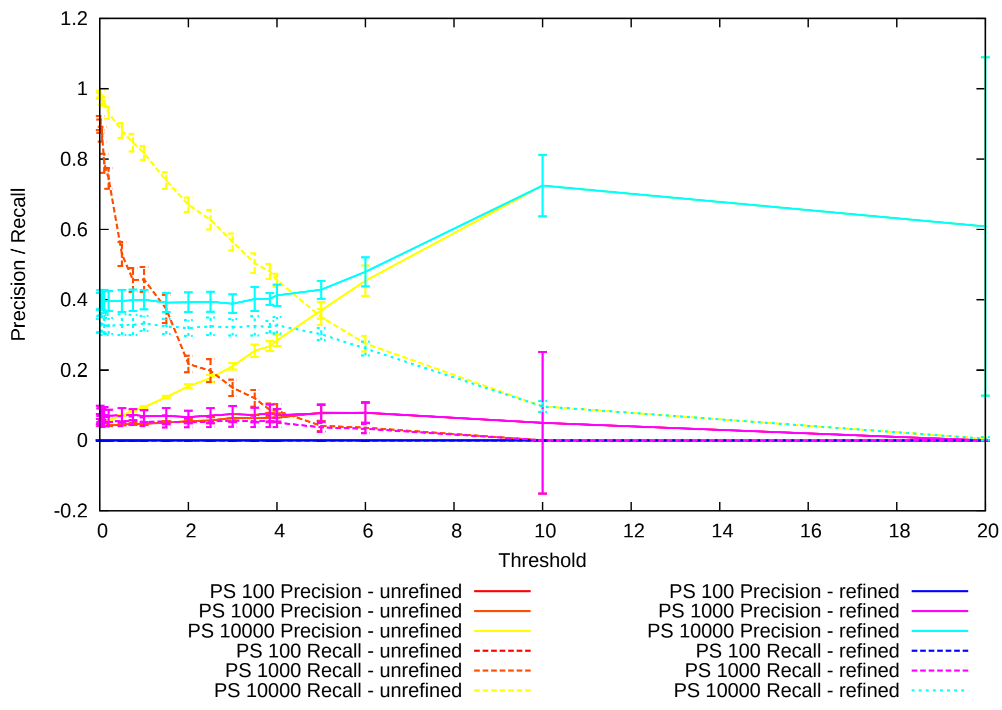


Figure D.20: Precision and recall of interactions learned for 100 bit Ising, selecting top 2%

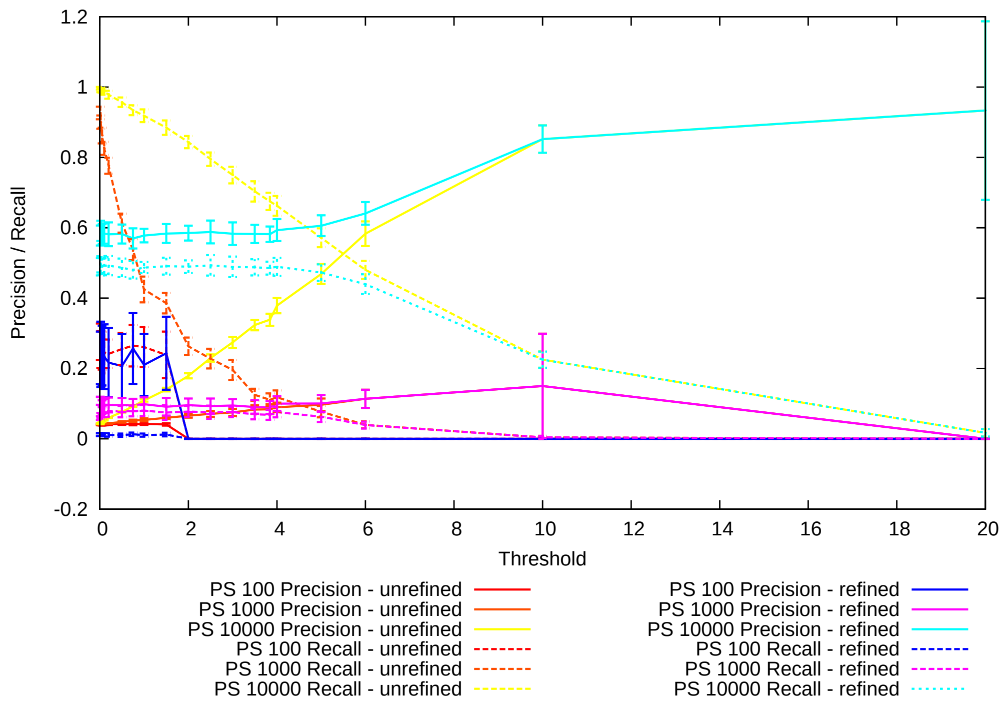


Figure D.21: Precision and recall of interactions learned for 100 bit Ising, selecting top 5%

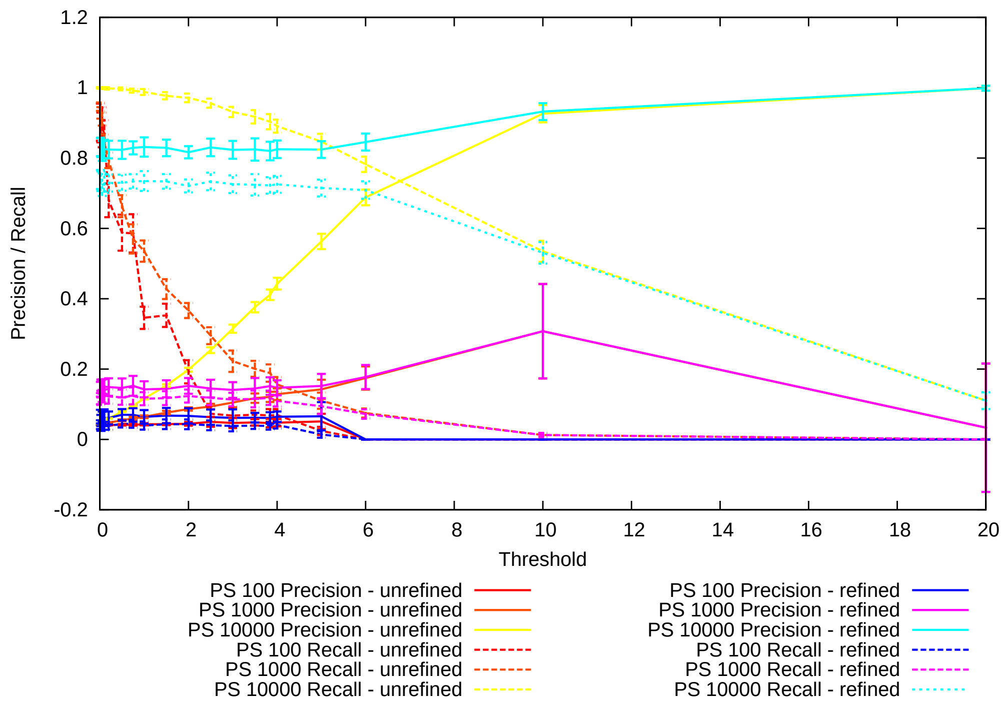


Figure D.22: Precision and recall of interactions learned for 100 bit Ising, selecting top 10%

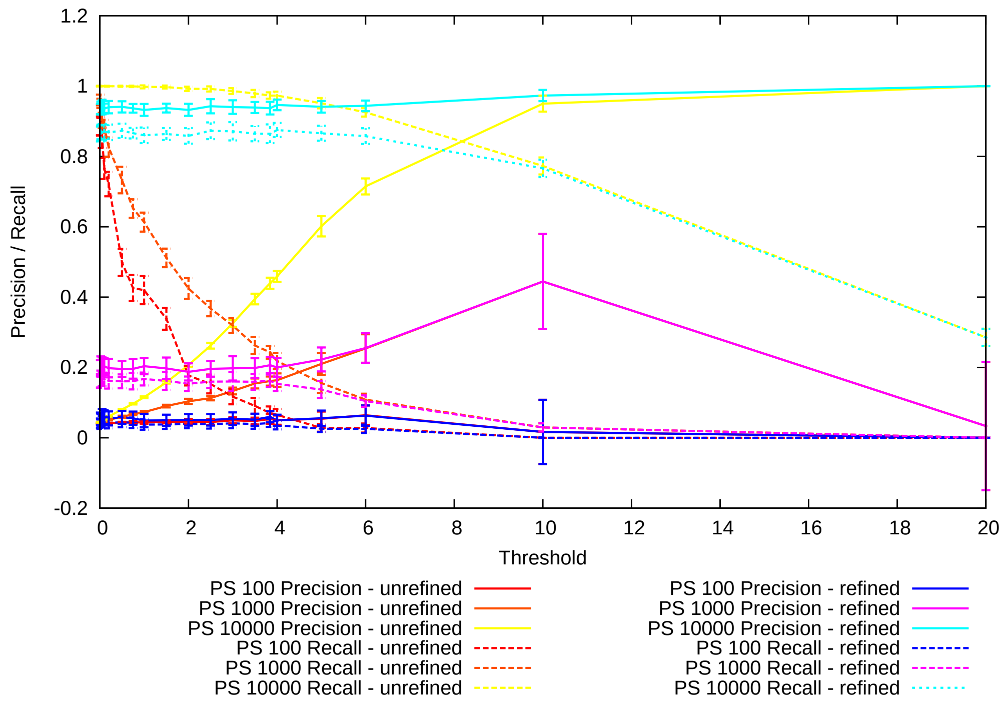


Figure D.23: Precision and recall of interactions learned for 100 bit Ising, selecting top 20%

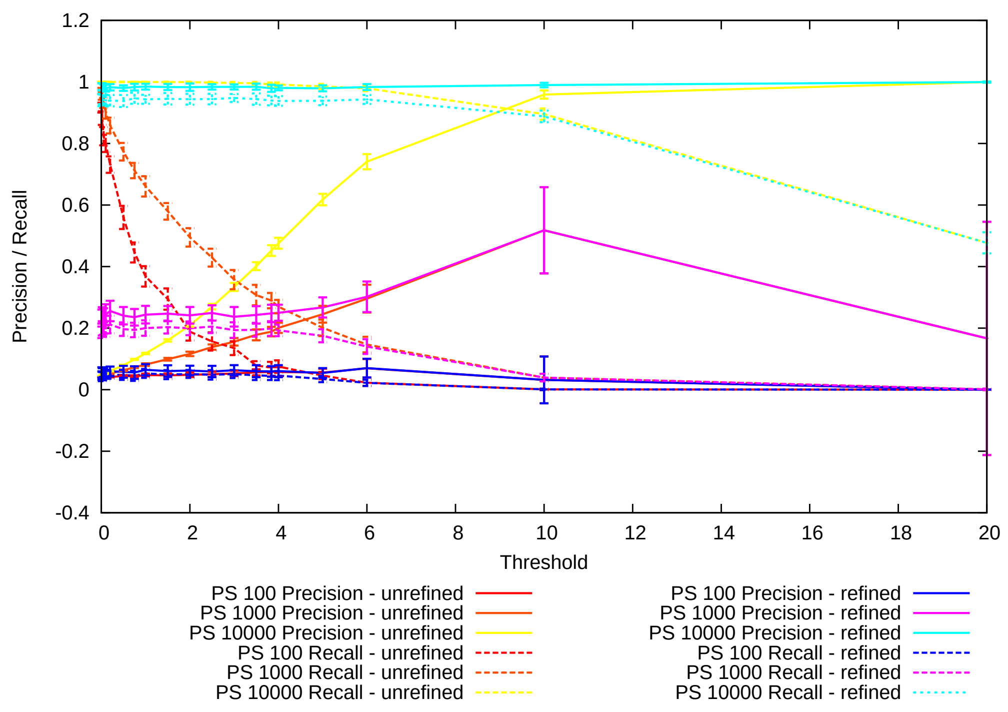


Figure D.24: Precision and recall of interactions learned for 100 bit Ising, selecting top 50%

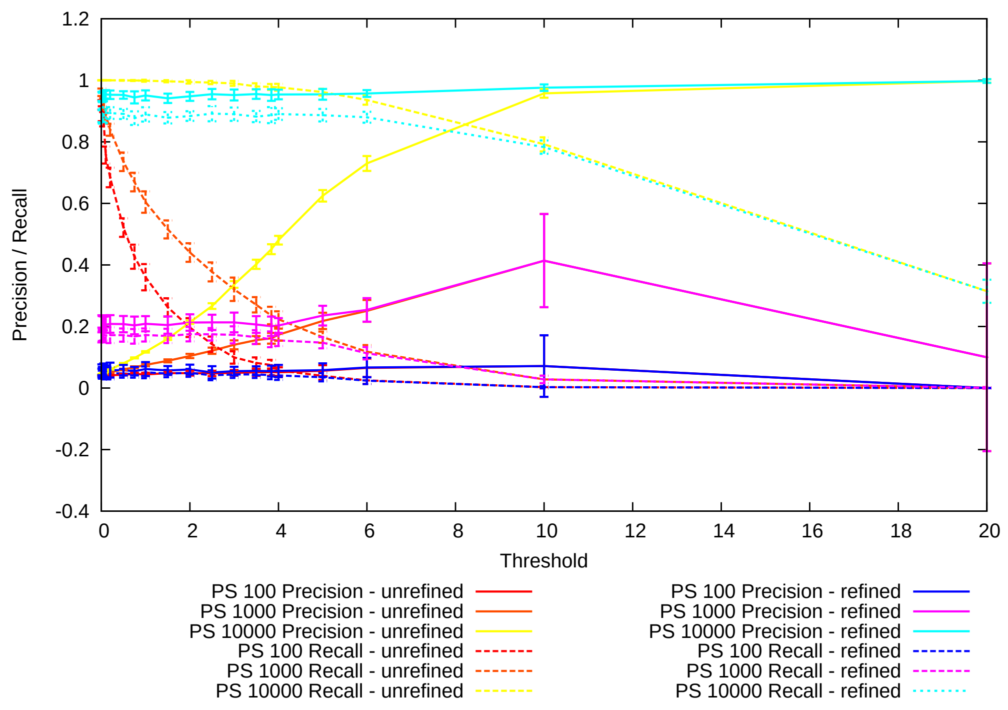


Figure D.25: Precision and recall of interactions learned for 100 bit Checkerboard problem, selecting top 1%

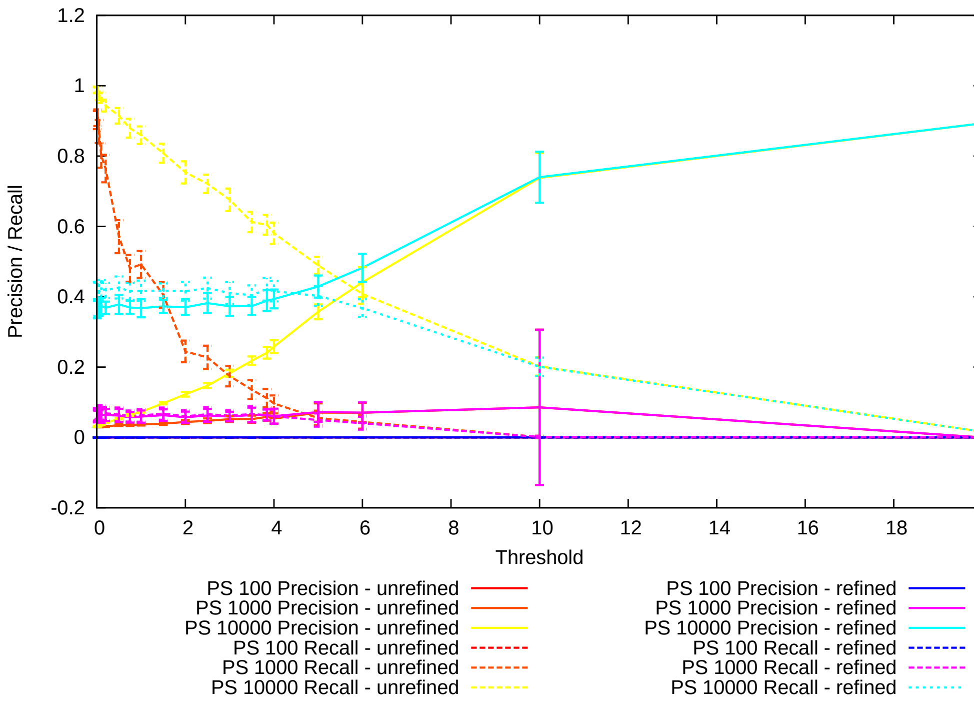


Figure D.26: Precision and recall of interactions learned for 100 bit Checkerboard problem, selecting top 2%

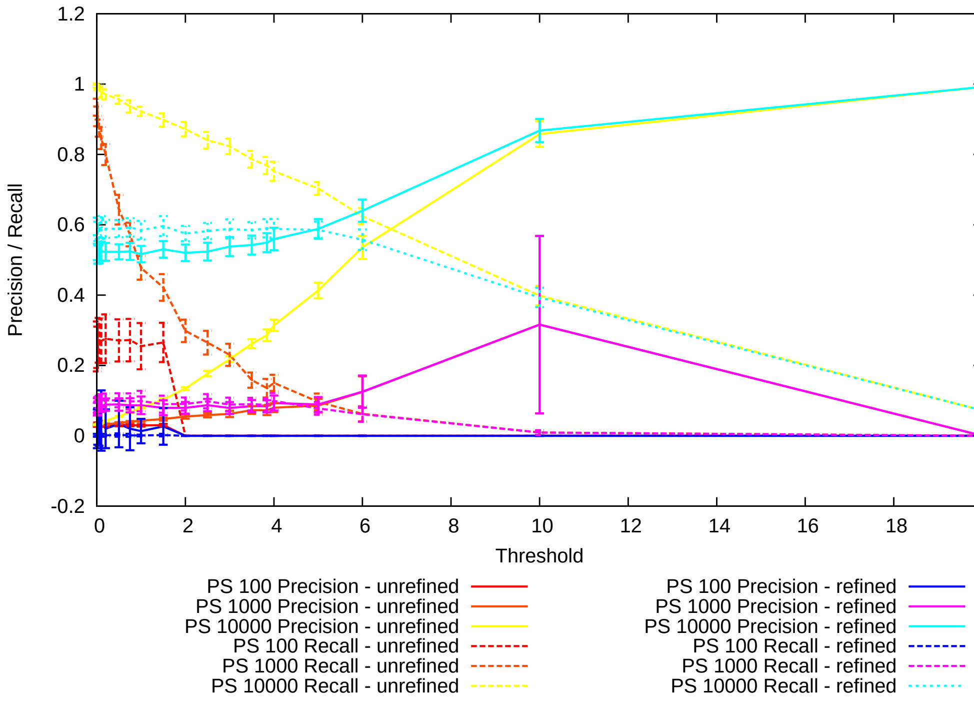


Figure D.27: Precision and recall of interactions learned for 100 bit Checkerboard problem, selecting top 5%

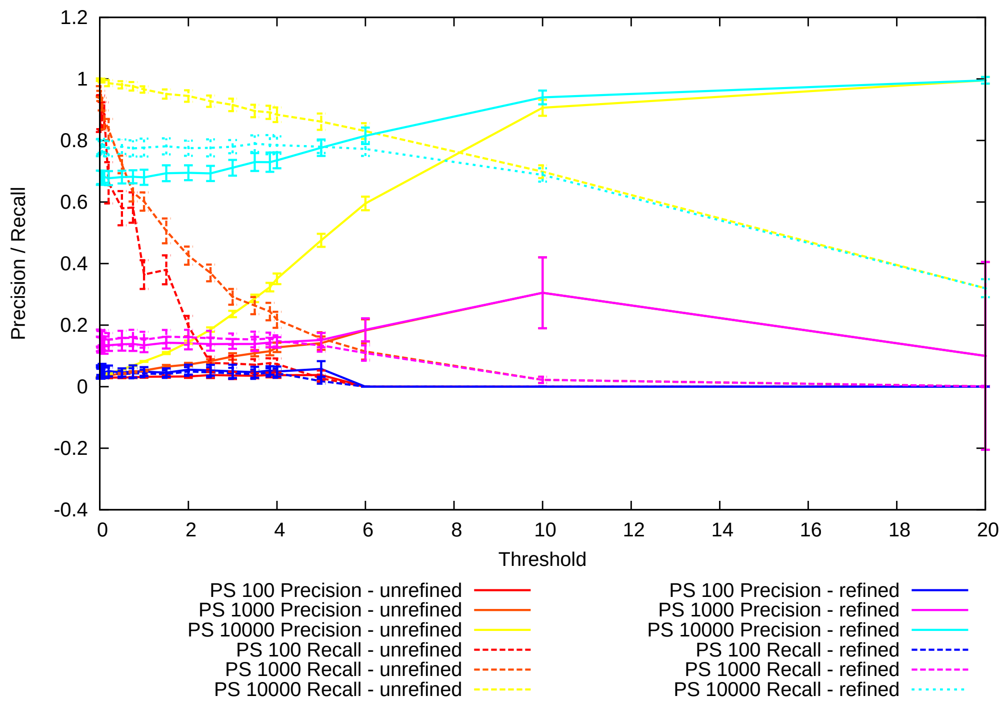


Figure D.28: Precision and recall of interactions learned for 100 bit Checkerboard problem, selecting top 10%

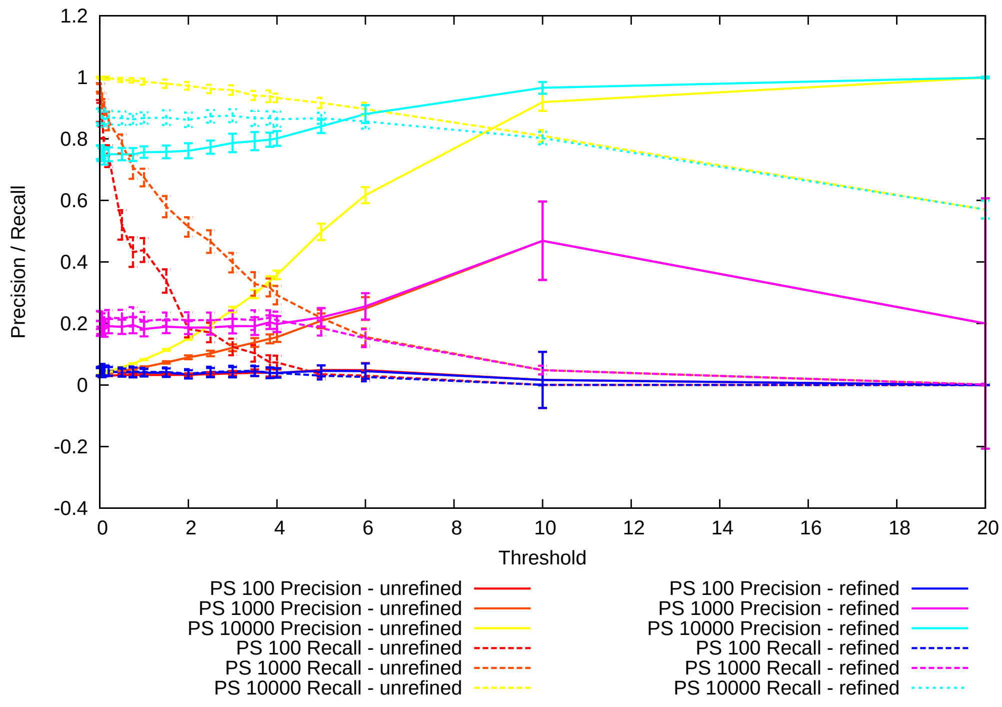


Figure D.29: Precision and recall of interactions learned for 100 bit Checkerboard problem, selecting top 20%

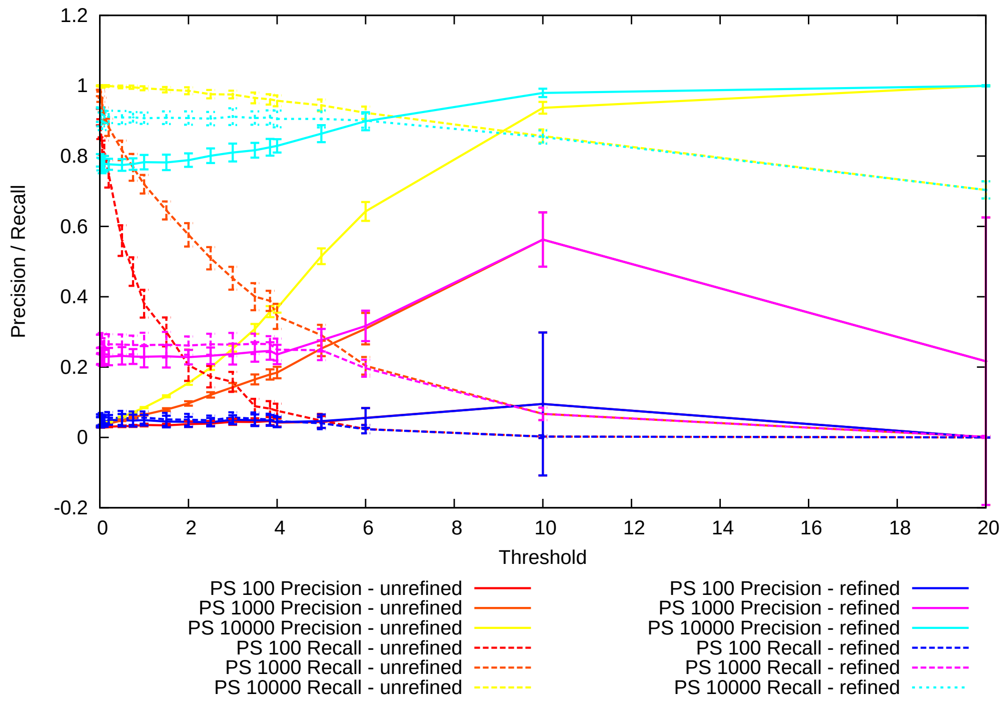


Figure D.30: Precision and recall of interactions learned for 100 bit Checkerboard problem, selecting top 50%

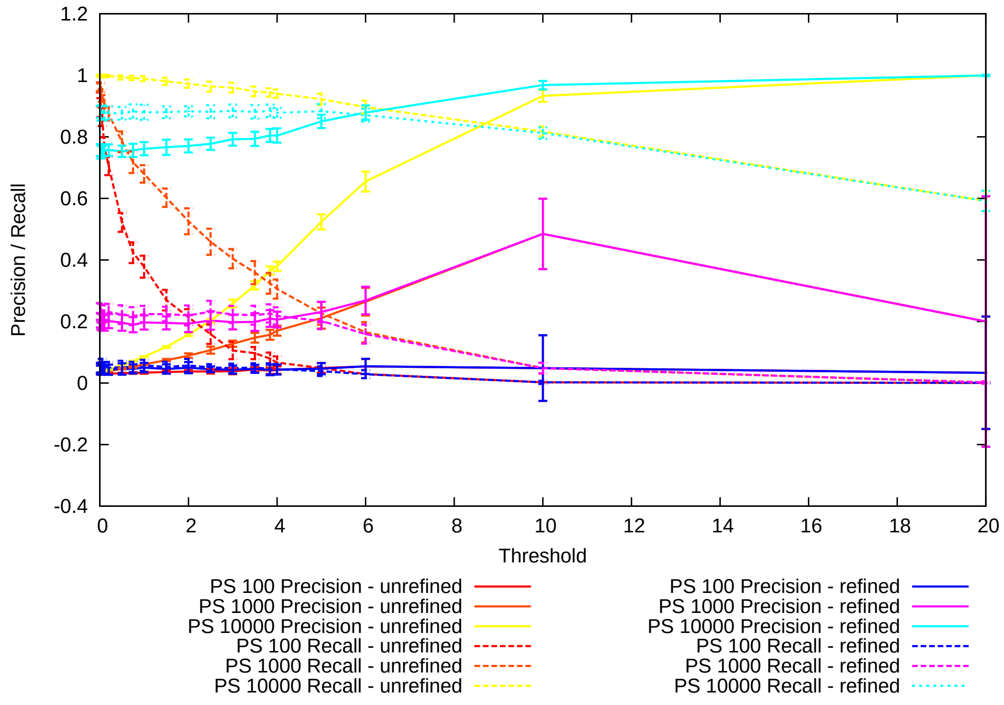


Figure D.31: Precision and recall of interactions learned for 100 bit MaxSAT problem, selecting top 1%

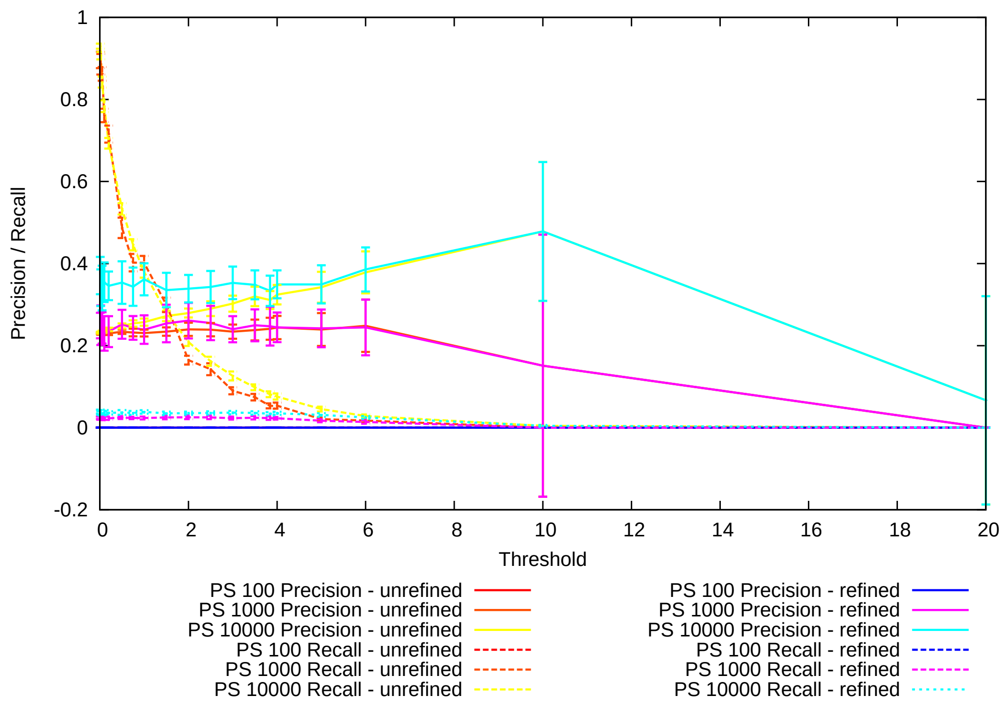


Figure D.32: Precision and recall of interactions learned for 100 bit MaxSAT problem, selecting top 2%

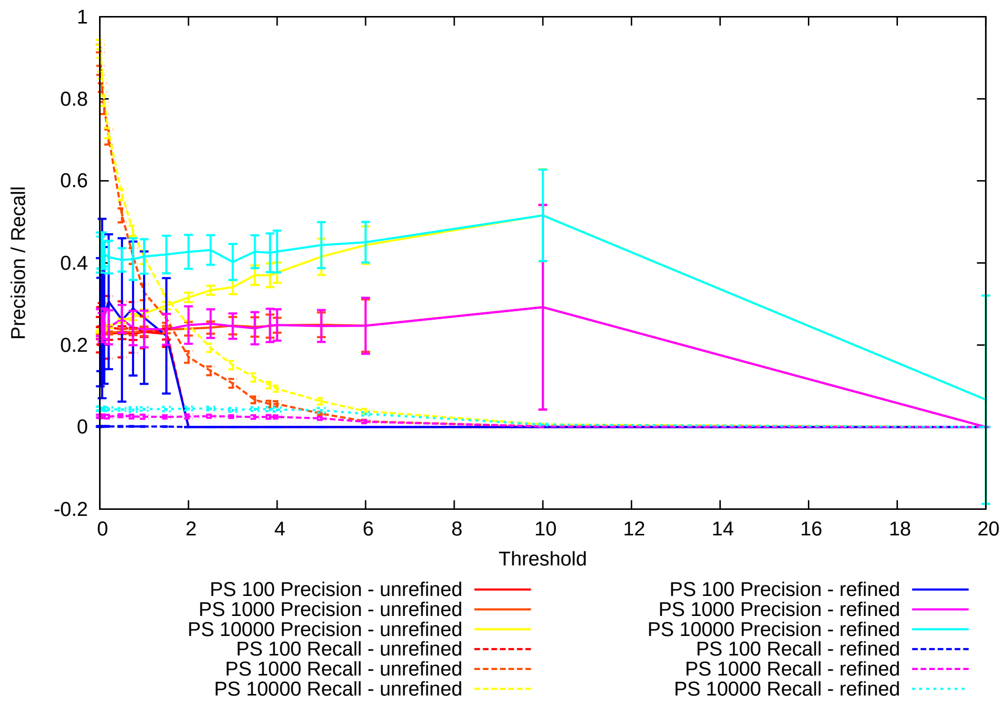


Figure D.33: Precision and recall of interactions learned for 100 bit MaxSAT problem, selecting top 5%

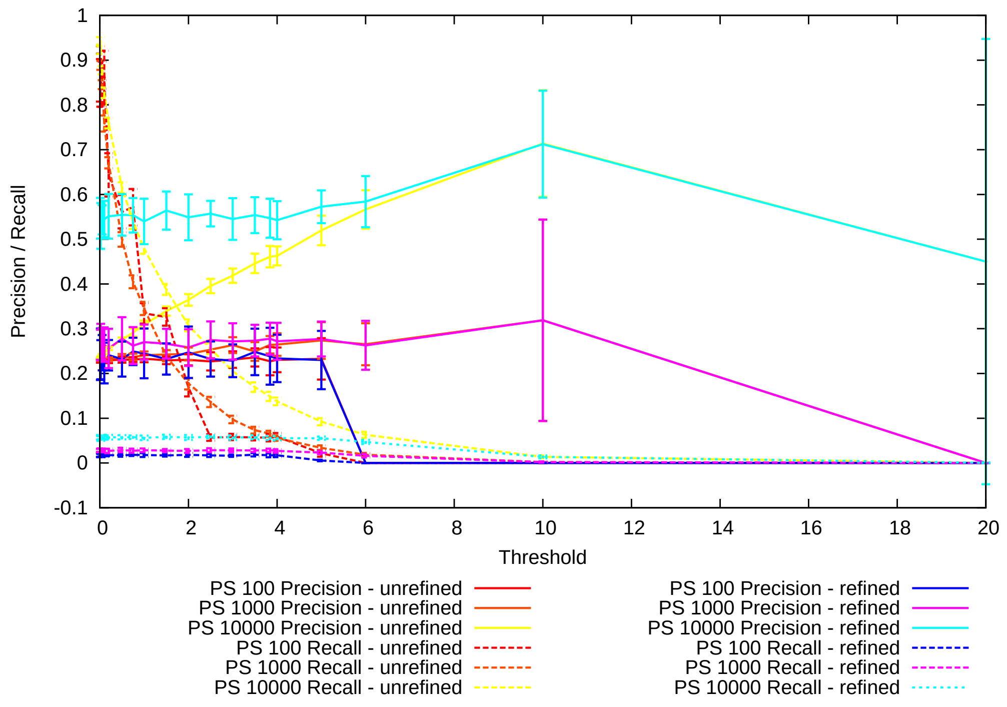


Figure D.34: Precision and recall of interactions learned for 100 bit MaxSAT problem, selecting top 10%

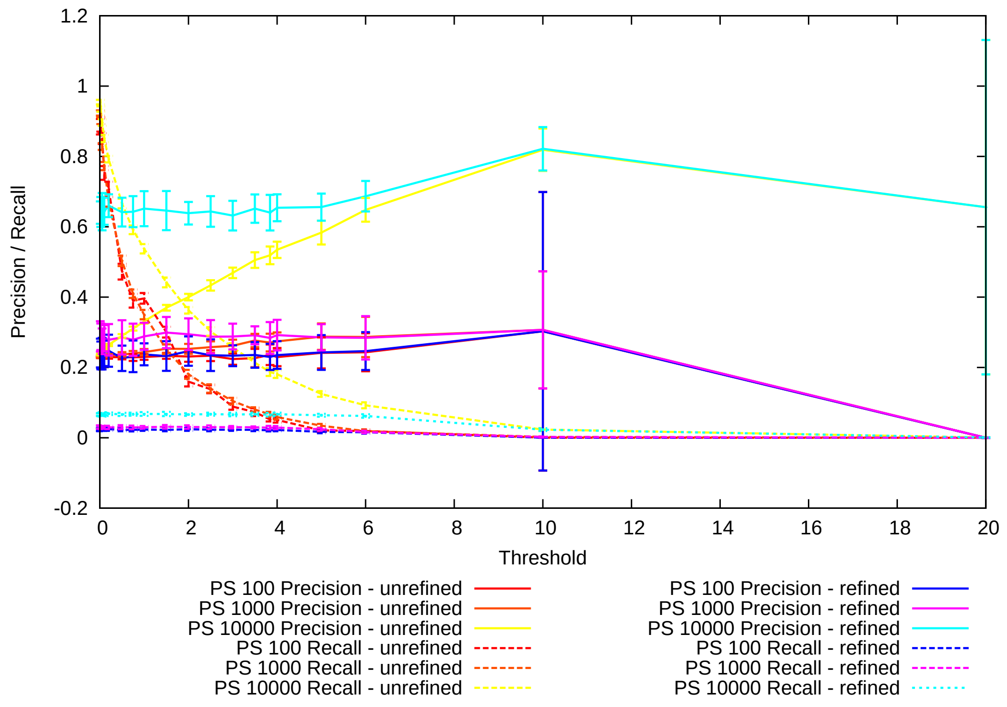


Figure D.35: Precision and recall of interactions learned for 100 bit MaxSAT problem, selecting top 20%

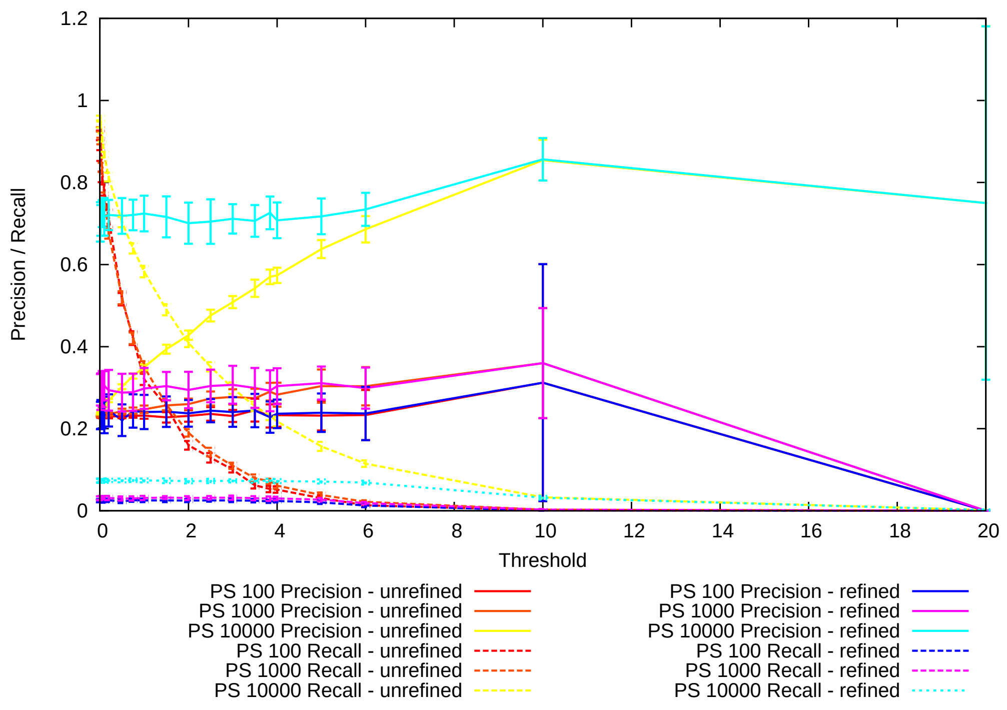
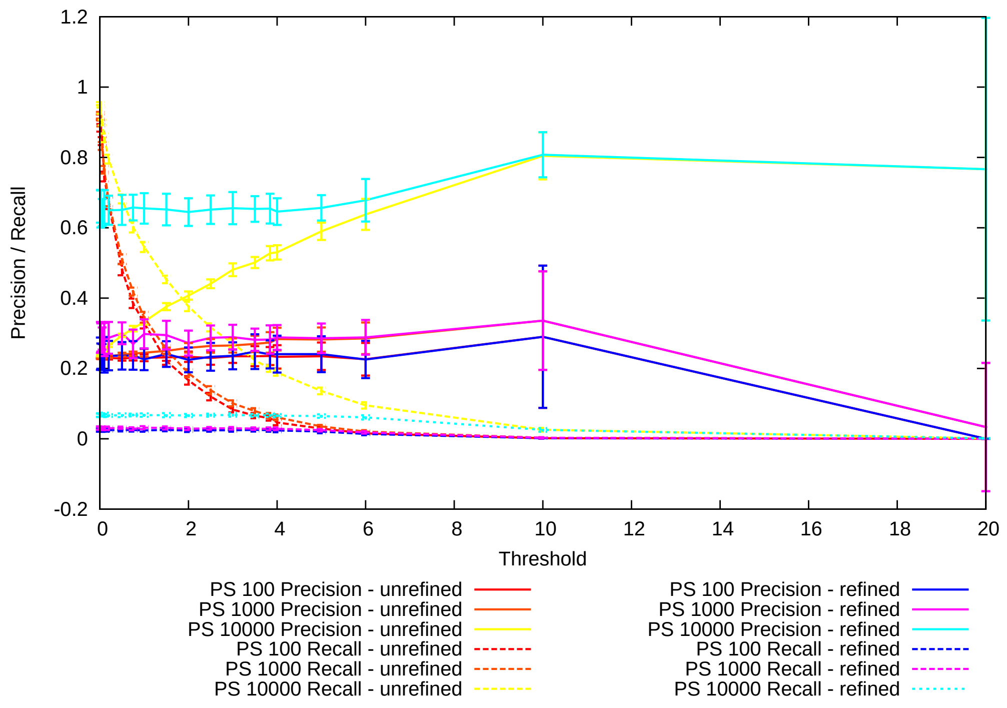


Figure D.36: Precision and recall of interactions learned for 100 bit MaxSAT problem, selecting top 50%


Precision / Recall
Threshold
PS 100 Precision - unrefined
PS 1000 Precision - unrefined
PS 10000 Precision - unrefined
PS 100 Recall - unrefined
PS 1000 Recall - unrefined
PS 10000 Recall - unrefined
PS 100 Precision - refined
PS 1000 Precision - refined
PS 10000 Precision - refined
PS 100 Recall - refined
PS 1000 Recall - refined
PS 10000 Recall - refined