UTRECHT UNIVERSITY
DEPARTMENT OF INFORMATION AND COMPUTING SCIENCES

MASTER THESIS IN COMPUTING SCIENCE

Parameter-less GOMEA

Applying Parameter-less Population Schemes to the Gene-pool Optimal Mixing Evolutionary Algorithm in a Black-box Optimization Setting.

Author
Willem DEN BESTEN, BSc

Student Number
ICA-3685632

Supervisor
Dr. ir. Dirk THIERENS

June, 2015

Abstract

The Gene-pool Optimal Mixing Evolutionary Algorithm (GOMEA) is an optimization framework with a single tunable parameter: the population size. In this thesis we explore the use of parameter-less population schemes to remove this parameter. It allows users to skip parameter tuning and makes scalability analysis easier to perform for researchers. The Exponential Population Scheme (EPS) is tested, which simply restarts the population upon convergence with double the population size. According to traditional scalability analysis there is only minor constant overhead, but EPS inevitably throws away several populations worth of fitness evaluations. The Parameter-less GA would reduce this wasteful behavior using population racing, but it resulted in worse scalability instead. The population pyramid scheme is extracted from the novel P3 algorithm and is applied to EPS in order to reuse old populations. This makes EPS more efficient in the number of fitness evaluations and allows for a slower growth of the populations. The memory usage increases dramatically, but this is remedied by adding a selection procedure. As a result of these experiments, the run-time of P3 is significantly decreased by operating on populations (as EPS does) instead of single solutions.

Contents

Chapter 1

Introduction

1.1 Problem Description

Research Goal: *Removing the population size parameter from the Linkage Tree Genetic Algorithm by applying parameter-less population schemes, and ensuring the incurred overhead is minimal in terms of fitness evaluations and run-time.*

The problem in black-box optimization is finding good solutions using only a fitness evaluation function and no a priori knowledge of the problem domain. A single evaluation of the fitness function is an expensive operation to perform, and no problem-specific heuristics can be applied. Genetic Algorithms (GAs) are suited to this type of optimization due to their ability to optimize using problem structure found in a population of solutions.

But the user must specify a size for the population beforehand, and this parameter has a large impact on the ability of the GA to solve the problem. Setting the right population size is difficult even for users with a thorough understanding of the GA. Often they will resort to using either a default value or finding a seemingly decent value through trial-and-error. The same behavior occurs for the other parameters, such as the crossover probability or mutation rates found in simple GAs. In either case it would be better if the GA were to adaptively decide values for these parameters itself.

Several theories exist for the removal of the mutation rate and crossover probability by setting them to a robust value. However, the population size single-handedly determines whether the GA solve the problem (to optimality) based on the other parameter settings. Underestimating the value of the population size will result in a prematurely converged population with non-optimal solutions. Overestimating it will result in optimal solutions, but at the cost of optimizing more solutions than was actually necessary. Hence the population size cannot be set in a problem-independent fashion, so it is the most non-robust parameter for any GA. Removing the population size requires a change to how the GA manages its population, hence its population scheme must change.

For the user to easily use a GA to its full potential, it must run without setting a single parameter. Hence a parameter-less population scheme must be applied to the GA, and the overhead in performance it incurs on the GA should be minimal. We focus on applying parameter-less population schemes to the competent Linkage Tree Genetic Algorithm [14] from the family of Gene-pool Optimal Mixing Algorithms[15]. LTGA comes with no existing parameters except for the population size, and it performs well on a variety of benchmarks.

1.2 Contributions

In this thesis the first results on applying parameter-less population schemes to the GOMEA framework are introduced. Three parameter-less population schemes are compared to the original GOMEA configured through bisection analysis. The contributions to the literature are explained in the remainder of the section.

The exponential population scheme (EPS) wraps GOMEA into a parameter-less population scheme by forcibly restarting the population upon premature convergence, but with double the population size. Because EPS partially incurs the cost of bisection, it would be expected to perform significantly worse than the original. As it turns out EPS performs only slightly worse if not on par with the original, thus showing some of the flaws inherent in how scalability analysis is most often performed.

Several state-of-the-art genetic algorithms (GAs) adopted the parameter-less GA to varying degrees of success, which extends the scheme of EPS with population racing. Though the parameter-less GA is supposed to reduce the overhead of EPS through early termination of the smaller populations, the results show quite the opposite behavior for GOMEA. Few populations are terminated early on and more over-sized populations are started than is necessary. It shows that simply adopting a well-known population scheme is not sufficient for developing a good parameter-less version of an algorithm.

The population pyramid at the core of the parameter-less population pyramid (P3) was analyzed for applicability to the GOMEA framework. We show that the population pyramid is a viable population scheme even after removing the local searcher and changing several parameters to conform with GOMEA. However, without the local searcher the run-time overhead at least quadruples in comparison to P3.

We introduce a combination of the EPS and P3 schemes called multiple insertion, which applies the population pyramid scheme to EPS in order to reuse the previously converged populations. A binary tournament selection mechanism is added so the user can fine-tune the growth of the pyramid as it can be very memory-intensive to store the previous populations for many generations. The new population scheme improves upon the number of fitness evaluations required by EPS, while drastically reducing the run-time overhead caused by maintaining a population pyramid. Reintroducing the local searcher used by P3 yields a robust GOMEA that performs better in the number of fitness evaluations and on par with the run-time of the original GOMEA on the majority of the test problems.

1.3 Outline

The remainder of this thesis is organized as follows. First a review is given in chapter 2 on the current state of GOMEA with an emphasis on linkage trees. Chapter 3 explains the process of analyzing a GA and some of the shortcomings when doing so. Chapter 4 reviews the three parameter-less population schemes tested in this thesis: the Exponential Population Scheme (EPS), the Parameter-less GA, and the Parameter-less Population Pyramid (P3). In chapter 5 a scalability analysis is done for GOMEA and the aforementioned population schemes with the exception of P3, which is analyzed in chapter 6 to show the extraction of the population pyramid scheme. Lastly, chapter 7 explains a generic representation of the population schemes and the Multiple Insertion scheme derived from it. The experiments and their results on the Multiple Insertion scheme are detailed in chapter 8. A discussion of the related work on parameter-less population schemes is given in chapter 9 with a reflection on the previous results. The thesis is summarized in the chapter 10 and some ideas are given for future work.

Chapter 2

GOMEA

This chapter reviews the literature on the family of Gene-Pool Optimal Mixing Evolutionary Algorithms (GOMEAs) [15] with a more specific focus on one of its members. In the first section GOMEA and the generic linkage model called the family of subsets are explained. The second section describes the linkage tree model and how it is constructed through a hierarchical clustering algorithm. The remaining sections summarize the changes made to the GOMEA framework in recent years, namely the application of child node filtering, the use of forced improvement, and a change in the ordering of the family of subsets.

2.1 Background

The Linkage Tree Genetic Algorithm (LTGA) is a competent GA for solving optimization problems using linkage learning introduced by D. Thierens [14]. GAs rely on their ability to mix partial solutions into new fit solutions to be efficient optimizers. This works on the assumption that solutions in a fit population share common substructures that strongly contribute to the fitness of the population. Traditional GAs take a randomized approach to selecting which problem variables should be mixed between solutions, hence they do not take the dependencies between the problem variables into account. Not only does the randomized approach lead to nonsensical combinations of the partial solutions, it is disruptive towards good partial solutions as well. Preferably the problem variables representing the partial solutions are identified so the GA can efficiently find and copy these partial solutions into new solutions. Linkage learning allows the GA to identify those sets of problem variables with strong probabilistic dependencies. LTGA is based on these two concepts: a model for representing the linkage between problem variables called the linkage tree, and a GA for optimally mixing solutions using the linkage tree. The framework for GAs called the Gene-pool Optimal Mixing Evolutionary Algorithm is the result of generalizing LTGA to use a more generic linkage model. Amongst the linkage models compatible with GOMEA, the linkage tree has the best performance in general. Further research on the subject of linkage trees has been incorporated into the research on GOMEA itself. No other models than the linkage tree are considered in this thesis, hence the terms LTGA and GOMEA can and will be used interchangeably. The latest implementation of LTGA used by P. Bosman and D. Thierens is publicly available from the CWI website [1] and was last described in the paper on LTGA and linkage measures[19].

[1]http://homepages.cwi.nl/~bosman/

2.2 GOMEA

GOMEA is a GA designed for optimally mixing partial solutions into new solutions. To do so it requires a generic model for representing problem variables whose values should be copied together during crossover. This model is called the family of subsets (FOS). Each subset in the FOS is a set of problem variable indices. For example, a 4-bit problem could have a FOS of $\{\{1,2\},\{1,2,3,4\},\{3\}\}$. Each subset identifies a group of problem variables whose bits are copied together during crossover. GOMEA maintains a population of n initially random solutions. How the FOS model is learned from the population is irrelevant to GOMEA, it only requires a function taking as input the current population and giving as output a FOS. It is assumed that the fitness evaluation function operates on solutions encoded as bit strings of length l.

Each generation begins by learning a FOS from the population using the model learning function. Then each solution is subjected to the Gene-pool Optimal Mixing (GOM) crossover operator. First the solution is cloned into an offspring population, so the building blocks in this solution are preserved for GOM with the remaining unimproved solutions. For each subset in the FOS, GOM will donate the bits indicated by the subset from a donor randomly selected from the population into the current solution. For example, a donation of bits $\{2,3\}$ from donor solution 110011 to parent solution 001100 would result in an offspring solution 000000. If the donor and the parent solution have the same bits for the problem variables, then the offspring solution is equal to the parent solution and no fitness evaluation is performed. For example, a donation of bits $\{1,4\}$ from donor solution 101101 to parent solution 100001 would result in an offspring solution 100001, whose fitness is already known and does not need to be recomputed. If the donation resulted in an offspring solution with a greater or equal fitness to its parent solution, then the changes are kept, otherwise the changes are undone. The donor changes between each donation to ensure there is no bias towards a certain solution (which most likely consists of a specific combination of partial solutions). Listing 1 shows the pseudo-code for the GOM operator. The offspring population replaces the current population after each solution has undergone GOM crossover. GOMEA continues evaluating generations until the termination criterion has been met. Listing 2 shows the pseudo-code for the GOMEA framework.

Several termination criteria exist for when the GA should end. The first two criteria are based on computational limitations set by the user. Either the maximum number of fitness evaluations that are allowed to be performed has been reached, or some time limit has been exceeded. Assuming the global optimum is known in advance, the GA can be terminated upon finding a solution with the optimal fitness value. The other termination is premature convergence of the population. Premature convergence happens when the population cannot be improved any further, either because a few solutions have taken over the population or there is not enough diversity to escape the local optima. More specifically, the population has converged when the population has not changed between two generations. If any of these criteria is met, then the GA is terminated after the current generation has been completed.

2.3 Hierarchical Clustering

At the start of each generation, LTGA constructs a linkage tree whose nodes form the family of subsets given to the GOM crossover operator. A linkage tree is a binary tree over all problem variables that gives a concise hierarchical model of the strongest linkage in the population. The leafs of a linkage tree are the singleton subsets of the problem variables. Every other node is the union of the problem variables stored in its two child nodes. The root node is the set of all problem variables. From definition of a binary tree

```
for $C_i \in FOS$ do
    $x'$ = RandomlySelectedFrom(Population);
    if $x_{C_i} \neq x'_{C_i}$ then
        $prev$ := fitness($x$);
        swap($x_{C_i}, x'_{C_i}$) ;                    /* Donate $x'_{C_i}$ to $x_{C_i}$.   */
        if $fitness(x) \geq prev$ then
            copy($x_{C_i}, x'_{C_i}$) ;                /* Accept the donation..  */
        else
            swap($x_{C_i}, x'_{C_i}$) ;                /* Undo the donation.  */
return $x$
```

Algorithm 1: GOM($x, FOS, Population$)

```
$Population$ := RandomPopulation($size$);
while $\neg$ TerminationCriterionMet() do
    $FOS$ := LearnModel($Population$);
    for $x \in Population$ do
        $x$ := GOM($x, FOS, Population$)
```

Algorithm 2: GOMEA($size$)

it follows that there are l leaf nodes, $l-2$ intermediate nodes, and 1 root node.

Linkage information measures the mutual dependence between subsets of problem variables, also called clusters. The motivation for measuring linkage is to efficiently find clusters that represent building blocks. A building block is an assignment to a subset of problem variables that is frequently present in the population and contributes strongly to the fitness of a solution. This frequency of the building block is captured as a probabilistic dependency between the problem variables, which is measurable using a linkage measure based on entropy. GOMEA is able to exploit this linkage structure by copying and preserving building blocks efficiently using the linkage tree model.

There are several applicable measures from information science, such as mutual information (MI) and variation of information (VI). The mutual information is shown in equation 2.2. High values of mutual information indicate a strong dependency between the variables and the reverse holds for the variation of information. These measures are entropy-based computations of the linkage between clusters of variables. Let equation 2.2 denote the entropy of a cluster C_i, where Ω_{C_i} is the set of all possible assignments of bits to the variables in C_i. Entropy scales with the number of variables, which biases the measure towards merging large clusters, so the measure is often divided by the joint entropy $H(C_i \cup C_j)$. As a result, the normalized distance D is a metric in the range $[0,1] \in \mathbb{R}$. The normalized variants of MI and VI are denoted by MNI and NVI, respectively. Mutual information and the normalized distance measures tend to perform equally well, as shown by P. Bosman and D. Thierens [17].

$$H(C_i) = \sum_{x \in \Omega_{C_i}} p(x) \log p(x) \tag{2.1}$$

$$MI(C_i, C_j) = H(C_i) + H(C_j) - H(C_i, C_j) \tag{2.2}$$

The linkage tree is constructed by a hierarchical agglomerative clustering algorithm based on a bottom-up approach of building the tree from the leaves up. The mutual information is used as the similarity measure in this summary. The set of clusters is initialized with the singleton sets of the variables. These are the leaves of the linkage tree. Then the algorithm continues to merge the clusters with the highest average pair-wise distance between two clusters until only the cluster containing all variables remains. Each

```
$merged := \{\{0\}, \{1\}, \ldots, \{l-1\}\}$;
$unmerged := merged$;
while $|unmerged| > 1$ do
    $(C_i, C_j) := \min_{(C_i,C_j) \in unmerged} D(C_i, C_j)$;
    $unmerged := unmerged + \{C_i \cup C_j\} - \{C_i, C_j\}$;
    $merged := merged + \{C_i \cup C_j\}$;
    if $ChildNodeFilter(C_i \cup C_j, C_i, C_j)$ then
        $merged := merged - \{C_i, C_j\}$;                    /* Filter 2 */
    for $C_k \in unmerged$ do
        Update($D(C_i \cup C_j, C_k)$);                       /* Update Distance */
$merged := merged - \{0, \ldots, l-1\}$;                      /* Filter 1 */
$merged :=$ FisherYatesShuffle($merged$);
return $merged$
```

Algorithm 3: Clustering(P)

cluster forms a node in the linkage tree with its merged clusters as the children. Listing 3 shows the pseudo-code for the hierarchical clustering algorithm.

$$MI_{\text{UPGMA}}(C_i, C_j) = \frac{1}{|C_i||C_j|} \sum_{x \in C_i} \sum_{y \in C_j} MI(x, y) \tag{2.3}$$

Computing the exact entropy of large clusters is expensive because every possible assignment of the variables must be enumerated. The UPGMA (unweighted pair group method of arithmetic mean) has been adapted in recent literature on LTGA[17] to speed up the hierarchical clustering algorithm. Equation 2.3 shows the new distance measure. Clustering starts by computing a matrix of MI values between the problem variables in $O(nl^2)$ time. The distance between a merged cluster and the remaining clusters can be computed in constant time, in contrast to the exponential time needed for the exact computation. Computing the hierarchical clustering can be done in $O(^2)$ time as demonstrated by the authors of UPGMA [12]. Both of the currently available LTGA implementations by B. Goldman and P. Bosman use the reciprocal neighborhood chain algorithm outlined in the UPGMA paper. The calculation of entropy for the initial distances depends on the gene-frequencies between each pair of problem variables, which can be stored in a frequency matrix. Each entry in the frequency matrix contains the frequencies in the population for the four possible assignments to the pair of problem variables. Constructing this frequency matrix requires $O(nl^2)$ time which is the upper bound in run-time for the construction of a linkage tree.

2.4 Filtering the Family of Subsets

Not all subsets in the FOS formed by the linkage tree are useful during crossover. Preferably the bad subsets are removed before any fitness evaluations are wasted on them during crossover. The most generic rule for filtering a FOS is removing the subset containing all problem variables. Using this subset during crossover is equivalent to performing binary tournament selection on the two parent solutions, which will speed up premature convergence due to the loss of diversity in the population. Hence this subset is always removed from the FOS.

Further filtering can be done using the parent-child relations formed by the subsets in the linkage tree[19]. Each non-leaf cluster forms a parent-child relation with its left and right child clusters. The exact rule for filtering a subset depends on the distance measure used to build the linkage tree. Examples are provided for the UPGMA variant of mutual information. Child node filtering is the process of filtering the children of a parent-child relation based on whether the children are fully dependent subsets. The linkage between the children is 'perfect', if the combined subset of variables forms a building block in the population. The parent subset preserves the linkage between the two subsets, when copying one of its child subsets would disturb the linkage. Hence the child subsets are removed from the FOS during creation. The criterion for removing the children for the mutual information measure is checking whether the distance between the parent and both of its children is equal. Because of the limited precision available for floating-point computations, the distance between a parent and child is called equal when their ratio lies within $[0.99, 1.01] \in \mathbb{R}$. Child node filtering is most effective on populations composed of building blocks, because the strict subsets of a building block are more likely to be filtered out by this rule. A rule has been proposed for filtering out parent subsets when the child subsets are independent, but this rule was not implemented as part of the implementation used in this thesis. An example of child node filtering using the normalized variation of information can be found in the paper by B. Goldman and W. Punch[21].

2.5 Forced Improvement

Forced improvement is an addition to GOMEA introduced in order to improve its performance on the MaxCut optimization problem[16] (see appendix A.5). If the fitness of a solution has not improved during crossover, then forced improvement is performed on the solution. Forced improvement applies the GOM crossover operator again, but the donor solution is now set the best individual in the population. The difference is that crossover will stop immediately after a fitness improvement has been found. If there again was no improvement in the fitness of the solutions, then the solution is replaced entirely by the best individual in the population. This alters the linkage between variables in the population in favor of the best individual, but only if the solution could no longer be improved upon. A population has converged when the fitness variance of its solutions has reached zero, which is guaranteed to occur due to the final forced improvement step.

Since the addition of forced improvement to GOMEA, another adjustment was made in order to perform better on combinatorial optimization problems. In the original LTGA implementation[14] the acceptance criteria of the GOM operator was to only accept solutions of better fitness, but the acceptance criteria was changed to accept solutions of equal fitness as well [20]. The reasoning behind this change was the explosive number of fitness plateaus present in complex instances of the Maximum Satisfiability problem. Since then the forced improvement step has been changed to occur on two conditions[19]: either the solution has not changed at all, or there has been no improvement in the entire population for $1 + \lfloor 10 \log(n) \rfloor$ generations. The latter value was empirically tuned and is a robust parameter to GOMEA.

2.6 FOS Ordering

The order in which the subsets of the FOS are stored determines the order in which GOMEA applies them as crossover masks. Hence the cluster ordering has an influence on how fitness landscape is traversed by crossover. When LTGA was first introduced[14], the clusters were ordered in least linkage first order, which is the reverse order in which the clusters were merged. From the perspective of LTGA, clusters with high linkage are more likely to identify strong building blocks for crossover. Hence the clusters are applied in order of least confidence in success first. The influence of the cluster ordering has been explored in part by B. Goldman in his paper on variants of LTGA[18]. Goldman compared the least linkage first order to the smallest cluster first order, and his results have shown neither ordering is optimal. Hence the FOS ordering is a tunable parameter for performance, but to ensure there is no bias towards certain problems the ordering is randomized in the latest literature on GOMEA.

Chapter 3

Population Sizing

GAs differ from other heuristics because they maintain a population of solutions. The key difference lies in the fact that the GA may not solve the problem to optimality depending on the size of the population. For simple problems, such as Counting Ones, there exists the Population Sizing theory for setting a theoretically robust population size. However, the theory does not apply to most real-world problems, nor do similar theories exist for LTGA. Dealing with and analyzing the population size parameter is integral to performing experiments on GAs. This chapter is devoted to explaining how LTGA and GAs in general are frequently analyzed in research, including the use of bisection for population sizing, and the disadvantages that come with this in practice.

3.1 Experimental Studies

At the core of an experiment is the definition of a measure for the performance of a GA. Whether one is more suitable than the other depends on the knowledge of the test problems at hand. More specifically, knowing the global optimum beforehand gives a different perspective on the efficiency of the GA. Suppose the global optimum is unknown, then the performance measure can only report the best fitness the GA has encountered at any point. This is often the case for case studies on real-world optimization problems. Suppose the global optimum is known beforehand, then the performance measure can be two-fold: can the GA find the global optimum before premature convergence occurs, and in how many fitness evaluations it does so. The former measure is meaningful when compared to the results of other algorithms, but is less informative on its own. The latter measure is useful for both comparing algorithms against each other, and determining the efficiency of the GA on a test problem. Researcher prefer test problems with known global optima because the available performance measures are more expressive. The test problems chosen for the experiments in this thesis have known global optima.

First the performance measure is defined more formally. The fraction of runs that convergence on the global optimum is the success rate. Each successful run used a number of fitness evaluations before it reached the global optimum. The average number of fitness evaluations before reaching the global optimum is the measure for the efficiency of the GA. Unsuccessful runs do not contribute to the average. Unless specified otherwise, the 'before reaching the global optimum' part is dropped for conciseness. In general there is a strong correlation between the success-rate and the population size, because the latter determines the likelihood of premature convergence. Figure 3.1 plots the expected behavior of the success-rate against the population size in an idealized setting. On its own the success-rate is not an interesting performance measure, because any value significantly lower than 100% indicates that the GA could not solve the problem reliably. An 'optimal' population size guarantees that the GA reliably converges onto the global optimum, but is also minimal

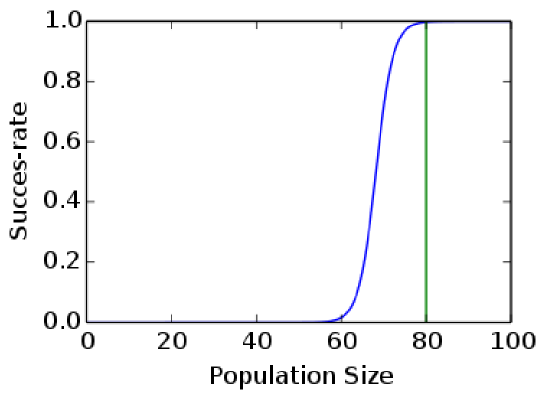


Figure 3.1: A plot of the expected behavior of the success-rate against the population size. The blue line indicates the success-rate. The green line indicates the 'optimal' population size.

in the number of fitness evaluations. Computing the exact optimal population size is computationally infeasible, so a statistical approximation is used instead. Researchers often use the bisection method to sample a reliable population size for the GA[10], [16], [21].

Performing an experiment is equivalent to sampling a GA for a finite number of parameter configurations. A fair experiment measures the changes of adjusting one parameter while setting the remaining parameters to a constant value. First the domain of each parameter is reduced to a discrete finite range of values. Fully capturing the interdependencies would require the experiment to evaluate the Cartesian product of all parameters. This leads to an explosion in the number of parameter configurations that must be run. As shown before, the population size can be factored out by performing bisection search, but it adds the computational burden of running bisection for the remaining parameter configurations. Only so much data can be presented in a research paper due to page limitations. Thus researchers are encouraged to remove parameters from experiments by choosing robust values. The conclusion remains that designing algorithms with non-robust parameters leads to additional complexity for both researchers and user alike. This is a strong motivation for developing parameter-less algorithms from a scientific perspective, especially to avoid the cost of bisection.

3.2 Bisection

Bisection is a method for finding a robust population size using a binary-search. A population size is accepted if the success-rate of r runs exceeds some threshold. It is assumed that the success-rate is a non-decreasing function of the population size, so the threshold separates the domain of population sizes into two disjoint continuous ranges. If the assumption holds, a binary search on the population sizes will return the first population size for which the success-rate exceeds the threshold. Binary search requires an upper bound on the interval, which is found by doubling the population size until the success-rate exceeds the threshold. The odds of failing x out of r runs follows the binomial distribution, hence the number of trials r increases the minimal population size accordingly. The r parameter determines the expected success-rate of the GA, which can be estimated using the Rule of Three [6]. Using a threshold of 100 with 100 runs gives an expected success-rate of about 97%.

Bisection comes with a set of flaws one should be aware of. The assumption of the

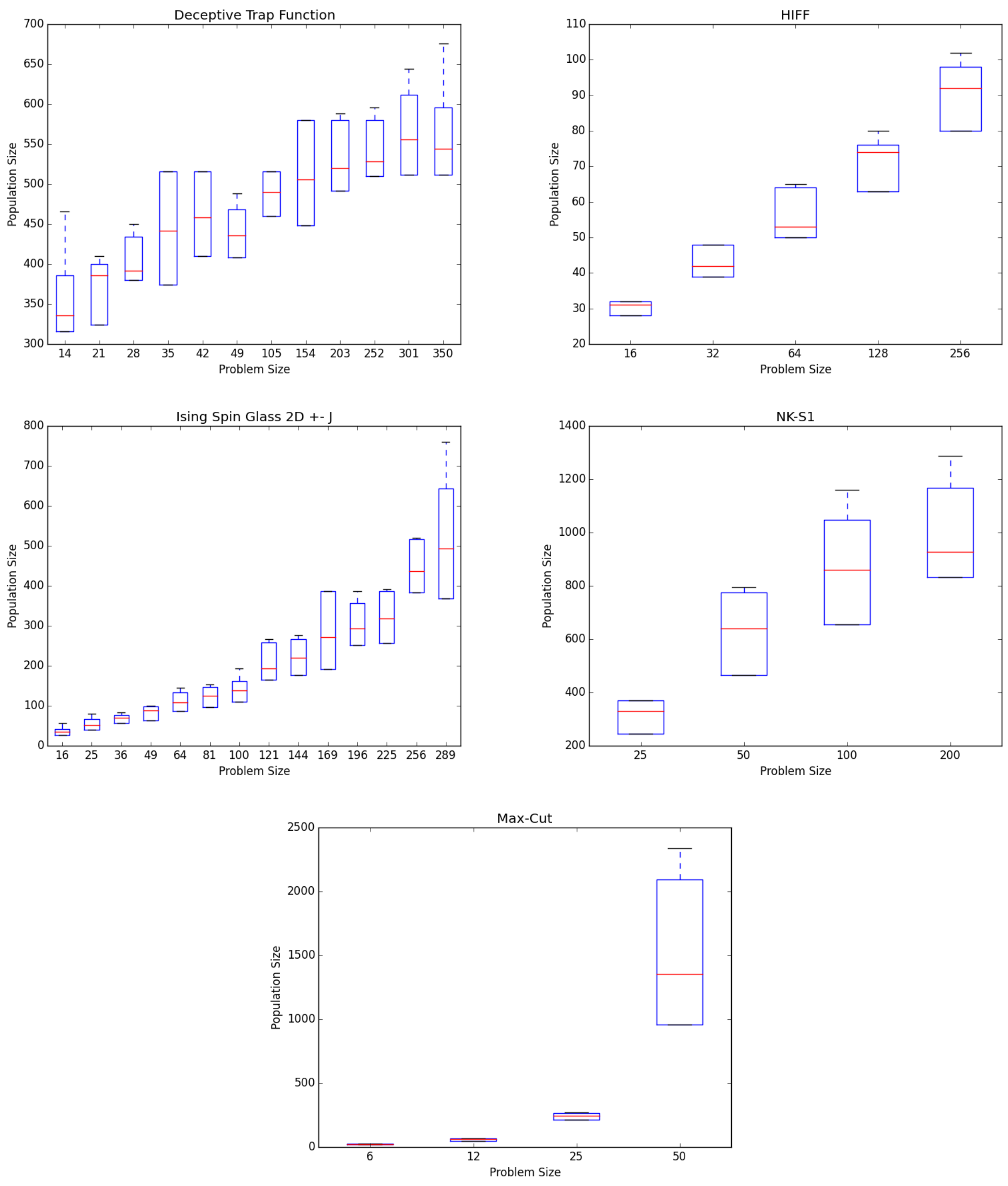


Figure 3.2: Box-plots of the bisected population sizes for the five test problems. The statistics were gathered from 10 bisections per configuration of length and problem. The box shows the data between the 10th percentile and the 90th percentile, and the whiskers indicate the minimum and maximum of the data. The red line indicates the median of the data.

success-rate being a non-decreasing function of the population size does not hold in practice due to the randomness factor in sampling only r runs. The results of performing multiple bisections on the five test problems are shown in figure 3.2. Each box-plot was generated from 10 bisections using 100 runs per parameter configuration. As one can see there a large variance in the population size sampled by bisection, which can differ up to a factor two between the minimum and maximum of the data. Not only does this significantly affect the performance of GOMEA, but performing a single bisection hides this variance in the population size. Hence it becomes obvious that the assumption of monotonicity does not hold at all on any of the test problems. However, the bisected population sizes do give a strong indication of how well GOMEA scales on the test problems. None of the research papers referred to throughout this paper have used a bisection size larger than 100 in their experiments. Increasing r reduces the variance in the bisected population size, were it not that performing 100 runs is already a prohibitively expensive operation. The same reasoning applies to running multiple bisections and taking the median population size.

Long time of execution has a negative impact on the length of the feedback loop of research, when the researcher must wait for the results of the bisection before continuing with his research. Performing bisection on small problem sizes is certainly doable, but this constraints the researcher to assuming his results scale up to larger problem sizes. For this reason the bisection on large instances is often postponed till the last part of the research project.

3.2.1 Multiple Instances, One Bisection

It is not uncommon for experiments to perform a single bisection on multiple problem instances by using a different instance per run[10], [16], [21]. For mathematical models the instances are often randomly generated using a configurable random generator. But the randomness has a negative effect on the quality of a bisection analysis. The complexity of the problem instances will vary, and so will the minimally required population size for solving the specific problem instance. But bisection will reject any population size that cannot reliably solve every problem instance, so whether or not a population size is accepted is determined by the most complex instances. The bisected population size will be an overestimation of the minimum population size for most instances. This does not bring into question the validity of bisection as a tool for scalability analysis, but it does give a skewed perspective on the absolute performance of the algorithm. This may lead conclusions where a parameter-less algorithm outperforms the bisected algorithm because it can better exploit the less complex problem instances.

3.2.2 Single Instance, Multiple Bisections

Ideally the median population size of multiple runs of bisection is taken as the final population size, because the median is a robust statistic against outliers. Additionally, results of an earlier experiment are easier to reproduce (and thus verify) because the median bisected population size is more likely to be reproduced than a single sample of a bisection on the population size.

An example: the usefulness of forced improvement on the performance of GOMEA is put to the test in an experiment. There are two options for comparing the two configurations of parameters, namely sharing one bisected population size or performing separate bisection runs for each configuration. The first case makes for a seemingly sound experiment, because the only variable of the experiment is the use of forced improvement. However, the assumption that the population size is constant may be flawed due to forced improvement better exploiting non-improving solutions. This could intuitively lead to a decrease in the population size that is required for solving the problem to optimality. Hence the bisection could also be performed separately, but how many bisection runs should be performed? Even on single instance problems the variance in population size could result in statistically significant differences when only one bisection is sampled, as seen in figure 3.2. Hence bisection is not that reliable for detecting shifts in the distribution of 'optimal' population sizes, and so neither for the difference in the number of fitness evaluations. The researcher is tuning two bisection parameters to ensure no statistical errors are made, while also worrying whether the experiment can be run in a reasonable time.

3.3 Conclusions

Creating and running experiments for GAs is a costly procedure when there are several non-robust parameters involved. It is a necessity to simplify the experiments by setting (robust) fixed parameter values or reducing the coverage of parameter values. However, removing the population size parameter requires either an expensive bisection or careful configuration by the researcher. There is no clear definition of what an optimal population size is in either scenario. Hence there is good motivation for adopting parameter-less population schemes that do not suffer from these complications.

Chapter 4

Parameter-less Population Schemes

4.1 Population Schemes

A population scheme is a scheme for maintaining one or more populations as an evolutionary algorithm progresses. Simple genetic algorithms (GAs) use a fixed size for the population, which forces solutions to compete for a place in the population. Hence the simple GA has at least one parameter: the population size. But selection and recombination methods come with their own parameters. This chapter is devoted to population schemes without the population size parameter.

The population size parameter is an example of a non-robust parameter, meaning there is no parameter setting that performs optimally regardless of the optimization problem. This statement is observable by the difference in population sizes returned by the bisection search on GOMEA for the test problems (shown in figure 3.2). From the plot it can be inferred that the difference in optimal population sizes can be larger than an order of magnitude. Hence no population size can be set without significantly under- or overestimating the population size for a different problem, most notable seen for the HIFF function and the NK-S1 problem instances.

Premature convergence is the problem of a population converging towards a set of solutions that do not contain the information necessary for finding the global optimum. Whether the selective pressure was too high or the population size was too small, premature convergence is the reason a run of the GA can no longer solve the problem. Simple GAs require a sufficiently large population to ensure enough diversity is present initially and can be preserved over time. Each population scheme deals with premature convergence in its own way, be it with population racing or constantly introducing new solutions.

A population scheme is considered *parameter-less* if and only if all parameters it requires are robust. This includes parameters with theoretically robust values such as a recombination probability based on the Schema theory. Clearly the simple GA is not parameter-less. The remainder of the chapter is devoted to parameter-less population schemes taken from known literature.

4.2 Exponential Population Scheme

One of the simplest parameter-less population schemes is the scheme of restarting a single population with an increased population size upon premature convergence until the solution quality has become acceptable. The growth function determines the rate at which the population size grows in the number of converged runs. The scheme for doubling the population size upon convergence is called the Exponential Population Scheme (EPS), named for the exponential growth of the populations. This technique was first discussed by George Harik and Fernando Lobo[8] in the context of GAs. EPS is comparable to the behavior of a user of a GA, who will often keep doubling the population size until they are satisfied with the result. Though users often start guessing at some arbitrary value such as 100, EPS will start at population size 1 to ensure there is no overestimation of the population size on smaller problem instances. Optionally, the user can provide some lower bound as the starting population size, if he knows for certain that any smaller population will prematurely convergence. An advantage of this population scheme is its simplicity, because it can be applied to most algorithms without the need to modify the algorithm.

Determining whether a population has converged can be difficult, if not impossible, for some problems and algorithms. Convergence can be extremely slow due to the genetic drift that occurs when a population contains many building blocks of equal fitness. The point at which one group of building blocks overtakes the population is determined mostly by a random walk when there is no selective pressure between the groups of building blocks. Genetic drift is a difficult problem with selecto-recombinative genetic algorithms, which is why G. Harik & F. Lobo did not experiment upon the exponential population scheme in their works. Alternatively, convergence can be slow by the design of the algorithm in question. Algorithms using a niching technique, for example restricted tournament replacement[10], maintain a diverse population meaning it is unlikely to converge quickly. Hence algorithms using niching cannot be used in conjunction with this population scheme.

A convergence-based population scheme such as EPS is very effective when convergence is easy to measure and happens at a consistent rate. The forced improvement step of the GOMEA framework increases the rate of convergence towards the best individual in the population once solutions are no longer improving. This eliminates the problem of genetic drift on basis of which G. Harik and F. Lobo rejected this population scheme. In addition to the fast convergence rates, the GOM operator evaluates $2l - 1$ changes to the solution during crossover, in contrast to the one or two solutions generated by a traditional crossover operator such as two-point crossover. This means GOMEA converges in less generations than a regular GA, because it performs more changes to the solution per generation. For example, solving the HIFF function for $l = 128$ requires an average of 8 generations per run. GOM is applied to the entire population, meaning $n(2l - 1)$ changes are tested per generation of the run. Hence one run of GOMEA has a not-so-varying cost in fitness evaluations, because the number of generations does not vary much as well. As a result the number of fitness evaluations for multiple runs of EPS can be divided into clusters with little variance. In which cluster the run falls depends entirely on how many times GOMEA was run and thus what the final population size was. Hence the samples of EPS can be divided into two sets, the runs with a number of fitness evaluations lower than that of GOMEA configured through bisection, and the runs with a higher number of fitness evaluations. The ratio between the two sets determines whether EPS will outperform the bisected GOMEA. This behavior holds true even when child node filtering (section 2.4) is used which reduces the average number of changes tested by GOM. Read section 5.3 for the experimental results involving this population scheme.

4.3 Parameter-less Genetic Algorithm

The Parameter-less Genetic Algorithm by George Harik and Fernandez Lobo was the first work on a parameter-less population scheme. In their work, the authors argued about the troubles practitioners face when confronted with non-robust parameters. Instead of performing studies on decent parameter settings for specific problems they argued that the genetic algorithm should estimate the parameters itself. Using the schema theorem, the selection pressure and recombination rate of the selecto-recombinative genetic algorithm

0	run generation 1 of population 1
1	run generation 2 of population 1
2	run generation 3 of population 1
00	run generation 1 of population 2
10	run generation 4 of population 1
11	run generation 5 of population 1
12	run generation 6 of population 1
10	run generation 2 of population 2
20	run generation 7 of population 1
21	run generation 8 of population 1
22	run generation 9 of population 1
20	run generation 3 of population 2
000	run generation 1 of population 3
100	run generation 10 of population 1
...	...

Table 4.1: Counter (base 3)

can be set to a theoretically robust value. The population size parameter of the GA is removed by evaluating multiple populations in a racing setting. A counter mechanism was introduced to schedule which population is evaluated next, while also ensuring smaller populations are evaluated k times more than any larger population. Larger populations require more fitness evaluations to evaluate one generation, hence this counter ensures smaller populations get an equal share of fitness evaluations in order to develop. When the largest population has been evaluated k times, a new population of double the highest population size is spawned. Whenever the average fitness of a large population becomes higher than that of a smaller population, the latter one is removed and the counter is reset to zero. This mechanism helps terminate populations stuck in genetic drift early on, thus wasting less fitness evaluations on the smaller populations. When the last population is removed due to premature convergence, a new population with double the population size is spawned. An example of the counter mechanism is shown in table 4.1.

GOMEA has not been combined with the Parameter-less Genetic Algorithm in research up till now. The two GAs conflicts in their design intentions. GOMEA converges quickly in terms of generations and the population racing scheme was designed to deal with slow convergence due to genetic drift. Given a growth parameter of four, the set of populations would mostly consist of one or two populations because of the fast convergence rate. A growth parameter of two would have several populations racing against each other, but performing an iteration of GOMEA becomes increasingly more expensive for higher population sizes. Hence performance is lost in evaluating over-sized populations because these are started faster. Because the Parameter-less GA uses an exponential growth for its populations, it is basically an extension of the exponential population scheme (EPS). An experimental study comparing EPS to the Parameter-less GA for three different values of k is included in chapter 5.

4.4 Parameter-less Population Pyramid

The Parameter-less Population Pyramid (P3) is a parameter-less evolutionary algorithm that was introduced by B. Goldman and W. Punch [21]. Both P3 and LTGA use GOM crossover (chapter 2) with linkage trees to explore the fitness landscape through a population. They differ in how they exploit the solutions in the population, in other words P3 uses a different population scheme than the fixed-size population scheme. It has been

```
$x$ := Hillclimb(RandomSolution($l$));
AddIfUnique($P_0, x$);
for $P_i \in P$ do
    $x'$ := GOM($P_i, FOS_i, x$);
    if $Fitness(x') > Fitness(x)$ then
        AddIfUnique($P_{i+1}, x'$);
```

Algorithm 4: Iterate()

```
if $x \notin Seen$ then
    $P_i := P_i \cup x$;
    $Seen := Seen \cup x$;
    UpdateFrequencies($P_i, x$);
    $FOS_i$ := Clustering($P_i$);
```

Algorithm 5: AddIfUnique(P_i, x)

shown that P3 outperforms LTGA configured through bisection in the number of fitness evaluations required to solve the problem, though no run-time analysis has been published as of yet.

Our interest in P3 comes from its novel population scheme: the population pyramid. It is a set of populations with two specific rules for inserting new solutions and performing crossover with the existing solutions. The set of populations expands slowly and forms a pyramid-like structure where the populations are arranged in increasing fitness and decreasing size. Hence each population can be referred to as a layer of the 'pyramid', so these two terms will be used interchangeably in the context of P3.

P3 operates on the population pyramid as follows. Initially there is only one empty population P_0. Every iteration, a new random solution undergoes local search by a first-improvement hill climber and is then added to the bottom layer of the pyramid P_0. Population P_0 has changed due to the insertion of the solution, so the linkage tree of P_0 is reconstructed. Then the solution undergoes crossover with the layers of the population pyramid, starting with the bottom layer and working its way up. The crossover operator is the GOM operator (see listing 1) used by LTGA and uses a linkage tree learned from the pyramid layer. If crossover at layer P_i results in a fitness improvement, then the solution is added to the next population P_{i+1}. If the population P_{i+1} does not exist, then it is initialized empty and the solution is inserted. The population of layer P_{i+1} changes after inserting a solution, so the corresponding linkage tree is rebuilt. Then the solution is GOM crossed over with population P_{i+1}, and this process is repeated until the top of the pyramid has been reached. This is how the population pyramid grows over time. Any change to the solution resulting in an equal or improved fitness during GOM crossover is kept throughout the entire iteration. Even if the solution is not inserted into the next population, it will undergo GOM crossover with the next population, unless the top has been reached. To ensure diversity is preserved in the pyramid, a set data-structure is used to check whether a solution is already present in any of the populations of the pyramid. If it was already present in the pyramid, then the solution is not added to the next population and the iteration simply continues. This means the linkage tree of the population is not reconstructed as the solution is rejected as a duplicate. Note that when a solution has reached the top of the pyramid and there was a fitness improvement, then a new layer is allocated and the solution is crossed over with that layer. However, GOM crossover cannot change a solution if the only available donor is the solution itself, so the iteration ends because the new top has been reached and there can be no further fitness improvement. Listing 4 shows the pseudo-code for one iteration of P3.

The crossover operator used by P3 differs from GOMEA in two aspects: the cluster ordering and selection of donors. Clusters are sorted in smallest-first order instead of the randomized order used by the most recent version of GOMEA. GOMEA rejects a cluster during crossover when the solution and donor have equal values for the problem variables. P3 alters this procedure by searching the entire population in a random order until a donor with non-equal values has been found. The upper-bound on the run-time of the crossover operator increases by a factor linear in the population size, however the trade-off between searching more donors and not rejecting an amount of clusters can be either positive or negative. The authors of P3 opted to use this 'exhaustive' donor searching because an empirical experiment shows it decreased the run-time of P3 significantly.

P3 applies the child node filtering rules discussed in section 2 to reduce the FOS with a focus on saving fitness evaluations on the smaller pyramid layers. Filtering has a stronger effect than usual on the number of fitness evaluations because P3 regularly introduces new empty layers. On small populations the child nodes are more likely to be filtered because the entropy is computed from such a small sample. There is another difference, namely the linkage measure used by P3 is the normalized variation of information, while GOMEA uses the mutual information instead. Previous literature has shown that the two measures agree on at least 92% of the clusters, and the difference in performance is negligible[17].

The run-time of P3 is significantly slower than that of GOMEA as a result of the population pyramid. Instead of learning one linkage tree per generation, the linkage tree in a layer is rebuilt every time a new solution is added to the layer. To speed up the rebuilding process, the pair-wise gene-frequencies used to compute the initial distances for the clustering algorithm are stored in a frequency matrix. Instead of using $O(nl^2)$ time to recompute the frequency matrix every time the linkage tree is reconstructed, the frequencies are incrementally updated whenever a solution is added to the population. Hence P3 requires only $O(l^2)$ time instead of $O(nl^2)$ time for rebuilding the linkage tree, because updating the matrix is cheaper than recounting the frequencies in the entire population. During crossover at most $O(|C|) = O(l)$ evaluations are performed, assuming a suitable donor can be found. Amortizing the cost of building the linkage tree gives a fair run-time analysis. Under the assumption of a lower bound of $O(l)$ on the fitness evaluation function, the cost of performing crossover is the upper bound on the run-time of P3. Traversing the population can potentially take $O(nl)$ time per fitness evaluation for each of the $O(l)$ evaluations per GOM crossover. Hence the cost of improving a solution per layer is roughly $O(nl^2)$. An analysis of run-time per iteration is more difficult, because the number of layer in the pyramid cannot be effectively estimated.

P3 solves the problem of premature convergence by introducing a new solution at the start of each iteration. The solution's fitness gradually increases as it traverses the pyramid. This is an important property of the population pyramid, because bits donated by highly fit solutions are more likely to be accepted by solutions of lower fitness. If the solution is mixed with the higher layers of the pyramid without undergoing sufficient optimization, the new information introduced by the solution is more likely to be destroyed. Hence the population pyramid protects the diversity performing GOM crossover in order of least-fit populations first. No solution is entered twice into the population, which ensures that layers cannot be dominated by a small fraction of the unique solutions. Using these two properties, P3 can continue exploring new parts of the fitness landscape until the user decides it is done.

The first improvement hill climber (FIHC) is a hill climber designed to find good local optima in a minimal number of fitness evaluations. Listing 6 shows the pseudo-code for the first improvement hill climber. FIHC traverses the solution in a random order, evaluating bit flips until a fitness improvement has been found. If no fitness improvement was found after testing each index in the random order, then the algorithm terminates, otherwise the

```
$indices := [1, \text{Length}(solution)]$;
$tested := \emptyset$;
do
    $improved :=$ False;
    Shuffle($indices$);
    for $i \in indices$ do
        Flip($solution[i]$);
        $newFitness :=$ Evaluate($solution$);
        if $BetterThan(newFitness, fitness)$ then
            $fitness := newFitness$;
            $tested := \emptyset$;
            $improved :=$ True;
        else
            Flip($solution[i]$);
        $tested := tested \cup \{i\}$;
while $improved$;
return $(solution, fitness)$
```

Algorithm 6: FIHC($solution, fitness$)

bit is flipped and the algorithm continues with the same order. FIHC assumes no problem-specific knowledge; every tested bit flip is counted as one fitness evaluation. The random order ensures there is no positional bias to specific bits. When a bit flip is evaluated twice without the solution having changed, the next bit flip is tested instead to prevent redundant evaluations. FIHC is used to optimize the solution at the start of each iteration before it had undergone crossover with the pyramid.

Algorithms based on layers of populations have been researched in the past by G. S. Hornby [11], but the focus was only on postponing premature convergence. Only fixed population-size layers were considered in Hornby's work. Previous algorithms have shown that layering on the number of changes by the GA is more effective than distributing solutions into layers based on average fitness.

Chapter 5

EPS and Parameter-less GA: Experimental Study

The goal of this chapter is to first show the performance of GOMEA on the test problems alone, and then measure the overhead introduced by the exponential population scheme and the parameter-less GOMEA. The first section shows the results of LTGA on the test problems using bisected population sizes. Section two compares the results of running the exponential population scheme against the results of GOMEA. Section three does the same for the Parameter-less GA.

5.1 Configuration

The implementation of GOMEA was provided by P. Bosman and was last used in a paper on MLNGA [19]. Further optimizations and a custom implementation of the parameter-less GA were necessary, so the published code by Bosman is not entirely representative of the results reported here. This pertains only to the experiments on the run-time of the algorithms, because the number of fitness evaluations is an implementation-agnostic performance measure.

The population sizes for GOMEA were determined using the bisection method (section 3.2) with an accepted success-rate of 99 out of 100 runs. In case of multiple instances, the bisection performed each run on a separate instance. Running a bisection for each problem instance separately was beyond the computational scope of this paper. The median population size returned by 10 bisection searches is used as the final population size. This is to ensure that the population size used by GOMEA is not a terrible overestimation and that it is more easily reproducible. The parameter-less population schemes were sampled for 300 runs each to accurately capture the distribution of fitness evaluations and the outliers. With the population size removed from sight, there are no other parameters left for configuration besides the problem-specific parameters and the parameter tested in the experiment. Appendix A explains the five test problems chosen for this paper and their parameter configuration.

GOMEA has the following termination criterion: either premature convergence or finding the global optimum. When the fitness variance of a population has reached zero, then the population is said to have converged. Alternative, if the entire population has failed to improve in fitness, then the population has also converged. None of the algorithms explored in this chapter failed to find the global optimum within a reasonable time. Premature convergence is not a termination criterion when using a parameter-less population scheme.

When it is unclear whether the results of an experiment are statistically significant, a two-tailed Mann-Whitney U test will be performed on the data. The threshold for statistically significance is set to $\alpha = 0.01$.

5.2 GOMEA

This section covers the experimental results of running GOMEA using the population sizes determined by bisection. Here follows a quick summary of the configuration of GOMEA. The FOS ordering is randomized before crossover. A new solution is accepted by GOM crossover if and only if the fitness is greater than or equal to the fitness of the original solution. The UPGMA adaptation of the mutual information is used. Forced improvement is applied only if a solution has not changed after crossover or the population has not improved in $\lfloor 1 + \log n \rfloor$ generations. Forced improvement ends immediately after an improvement to the fitness has been found. Lastly, the fitness evaluations performed during bisection do not contribute to the results at all.

The results in figure 5.1 show the mean number of fitness evaluations for two variants of LTGA configured through bisection. The plot labeled with *-NONE* does not use child node filtering, nor does it forced improvement. Together the plots illustrate the effect of child node filtering and forced improvement on GOMEA for reference in the next experiment on EPS. Drawing conclusions from these runs alone is difficult due to the variance in the population size returned by the bisection search, as shown in figure 5.2. The error-bars in figure 5.2 show the large standard deviation when sampling multiple bisection searches. Whether it is forced improvement or child node filtering which significantly impacts the population size on the MaxCut problem is unknown. The conclusions drawn in section 5.3 on the effects of forced improvement and child node filtering also apply to this experiment.

Lastly, note the low standard deviation indicated by the error bars in figure 5.1, which are only noticeable on the tiniest of problems. A low standard deviation in the number of fitness evaluations implies a low standard deviation in the average number of generations as the population size is a constant. These results reinforce the view that the number of fitness evaluations used by GOMEA is a function of the population size. This observation was already made in section 4.2 on the influence of GOMEA on the performance of EPS, and it is now verified.

5.3 Exponential Population Scheme

This section covers the experimental results of running the exponential population scheme on the Exponential Population Scheme (section 4.2). Figure 5.3 shows a comparison of the exponential population scheme and GOMEA.

The results show a constant overhead in added performance to GOMEA on the deceptive trap functions and the HIFF function. This behavior falls within the expectations set beforehand, because the two problems consist of only one problem instance each. Hence the bisection gives a reasonable estimation of the optimal population size.

However, the performance on the randomly generated problem instances is apparently equivalent, even though the exponential population scheme incorporates the overhead of partial bisection whereas GOMEA does not incorporate the cost of bisection at all. The results are best explained as the bisection giving an overestimated population size for the majority of the problem instances.

Figure 5.4 shows a comparison of the median bisected population size against the average population size of the last population in use by the exponential population scheme. It is clear from the plot that the bisection gives an upper bound on the required population while the exponential population scheme shows GOMEA could often solve the problem with a smaller population size. This holds for both the single-instance and the multiple-instance problems. Performing the bisection is significantly more expensive than simply

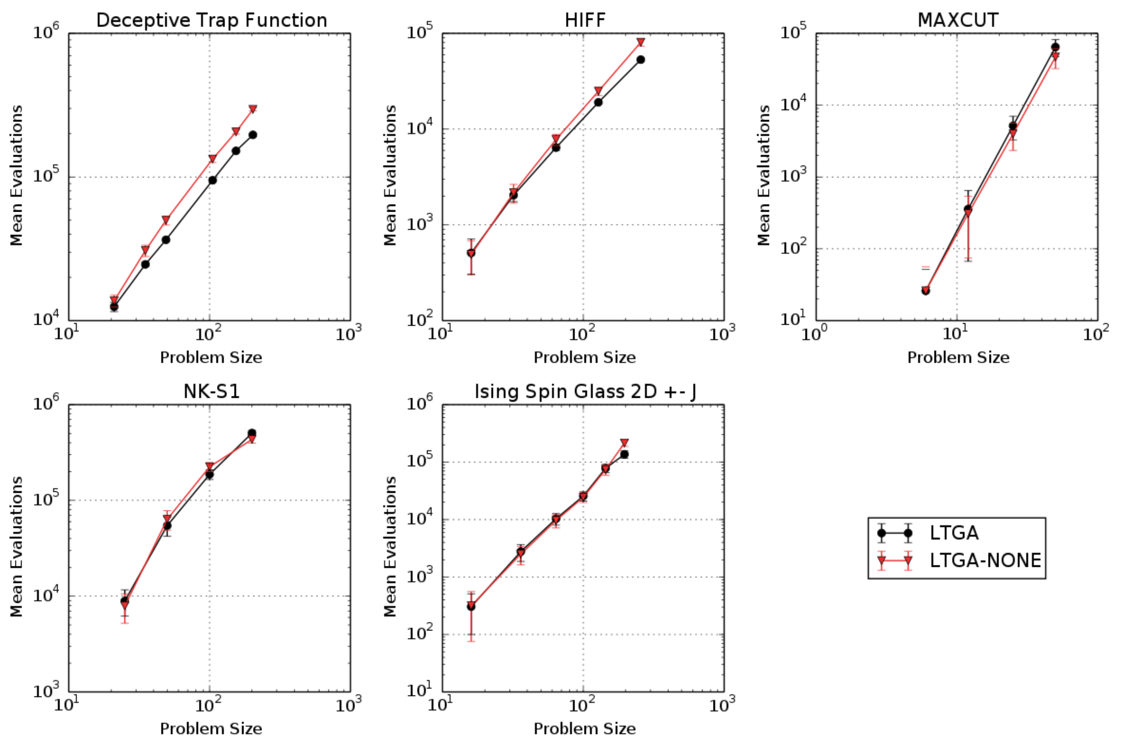


Figure 5.1: Comparison of the average number of fitness evaluations for the two LTGA variants. The error-bars indicate the standard deviation in the number of fitness evaluations.

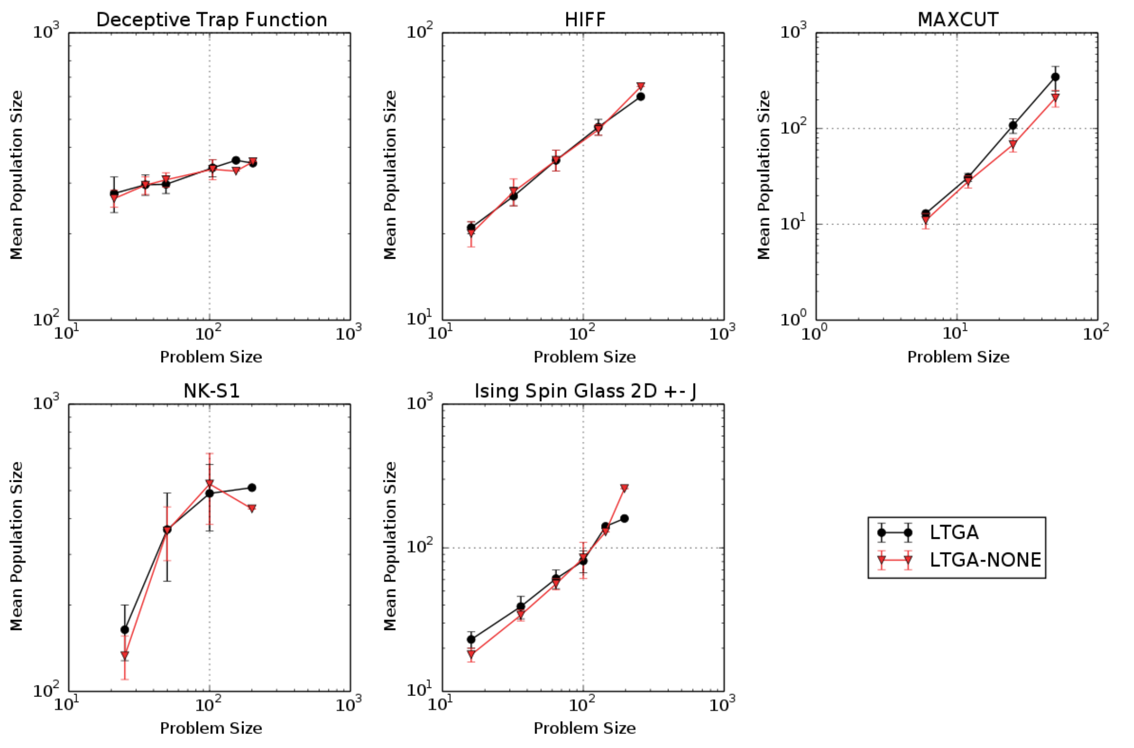


Figure 5.2: Comparison of the median population size of 10 bisection searches for the two LTGA variants. The error-bars indicate the standard deviation in the population size.

running the exponential population scheme, so there is ample motivation for using the exponential population scheme instead.

5.3.1 Forced Improvement and Child Node Filtering

Forced improvement was added to GOMEA to ensure the population converges properly even when the GOM operator accepted solutions of equal fitness. However, Goldman has shown that the population maintained by GOMEA still converges reliably without forced improvement. To be exact, GOMEA reliably terminates when population convergence is measured as not accepting a new solution for an entire generation. This seems counter-intuitive because GOMEA accepts solutions of equal fitness as well. The influence of the forced improvement and the adjusted termination criterion are put to test as part of this small experiment.

Child node filtering is a mechanism that mostly enhances the performance of GOMEA on problems such as the deceptive trap functions and the HIFF function. However, child node filtering has the potential of removing large amounts of clusters when used on small populations. Because the exponential population scheme begins with a population of size one, the potential gain in fitness evaluation might be significant. The influence of child node filtering is put to the test as part of this small experiment.

Figure 5.5 shows the results for the four variants of the exponential population scheme. The abbreviations FI and CNF denote Forced Improvement and Child Node Filtering, respectively. The observable influence of forced improvement is minimal on the deceptive trap function, HIFF, and the NK-S1 instances. On the Ising Spin Glasses and MaxCut instances there is a small observable difference in the plots. According to the Mann-Whitney U test the use of forced improvement is statistically significantly better on the Ising Spin Glasses and the largest MaxCut instances. In none of the samples does the use of forced improvement negatively impact the performance of the exponential population scheme. The difference in performance caused by child node filtering can be mostly addressed to the problem characteristics instead of the population scheme. In conclusion, the exponential population scheme performs best when using forced improvement and child node filtering, though the difference in performance is not that significant.

5.4 Parameter-less GA

Figure 5.6 shows the average number of fitness evaluations until reaching the global optimum for the parameter-less GA improved with GOMEA. The behavior of the parameter-less Genetic Algorithm using LTGA is similar to the exponential scheme, as was predictable due to LTGA's convergence rate. The trade-off between converging populations and starting the correct population early is balanced out in the total number of fitness evaluations per run. The one virtue of the Parameter-less Genetic Algorithm for saving evaluations on small population by early termination does not apply when using LTGA as the genetic algorithm. Performance may be gained by early termination, but the trade-off in starting more expensive populations is significantly higher than the aforementioned gain. This can be seen by the poor scalability of parameter-less GOMEA when using a base growth of 2. There are no conclusive advantage to using the more complicated Parameter-less Genetic Algorithm instead of the exponential population scheme on single-objective bit-string optimization problems.

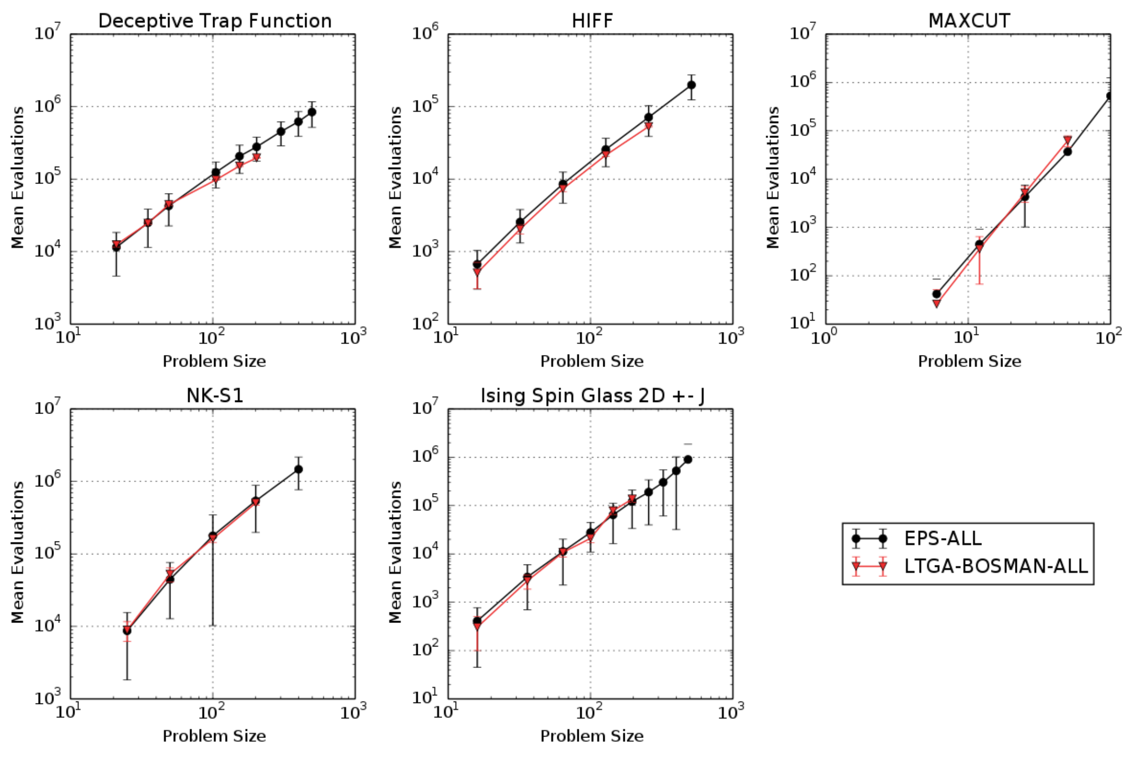


Figure 5.3: Comparison of the average number of fitness evaluations for EPS and GOMEA configured through bisection.

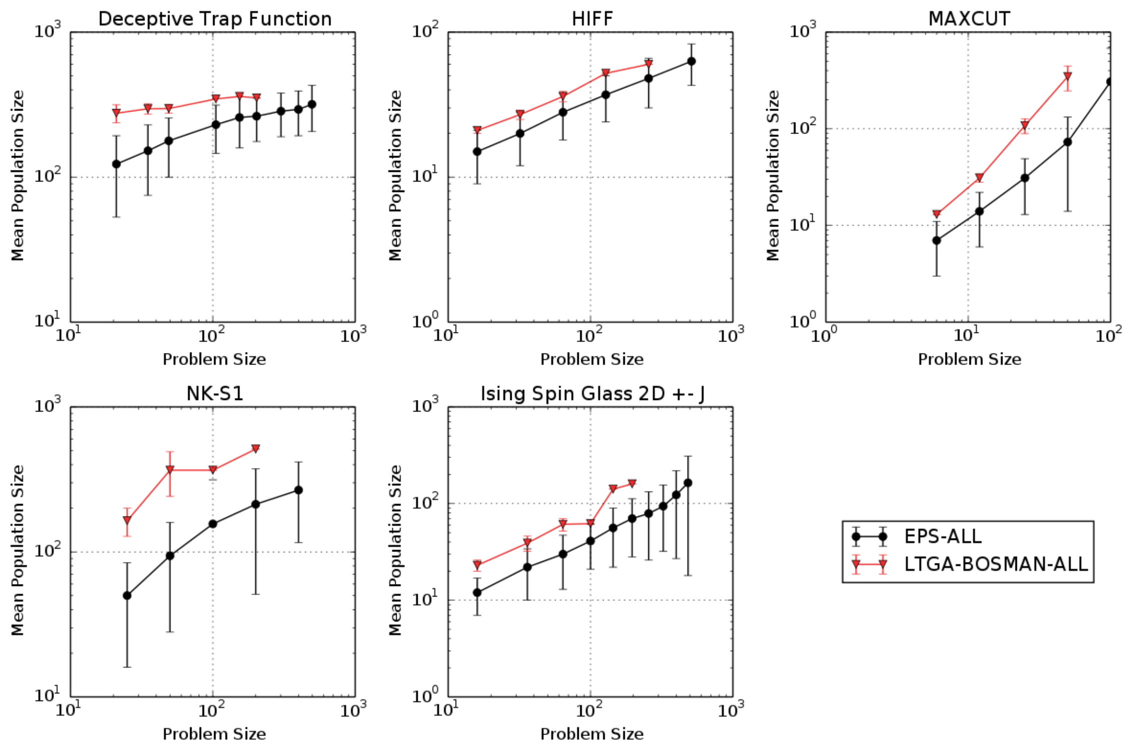


Figure 5.4: Comparison of the average final population sizes of EPS and the bisected population sizes of GOMEA.

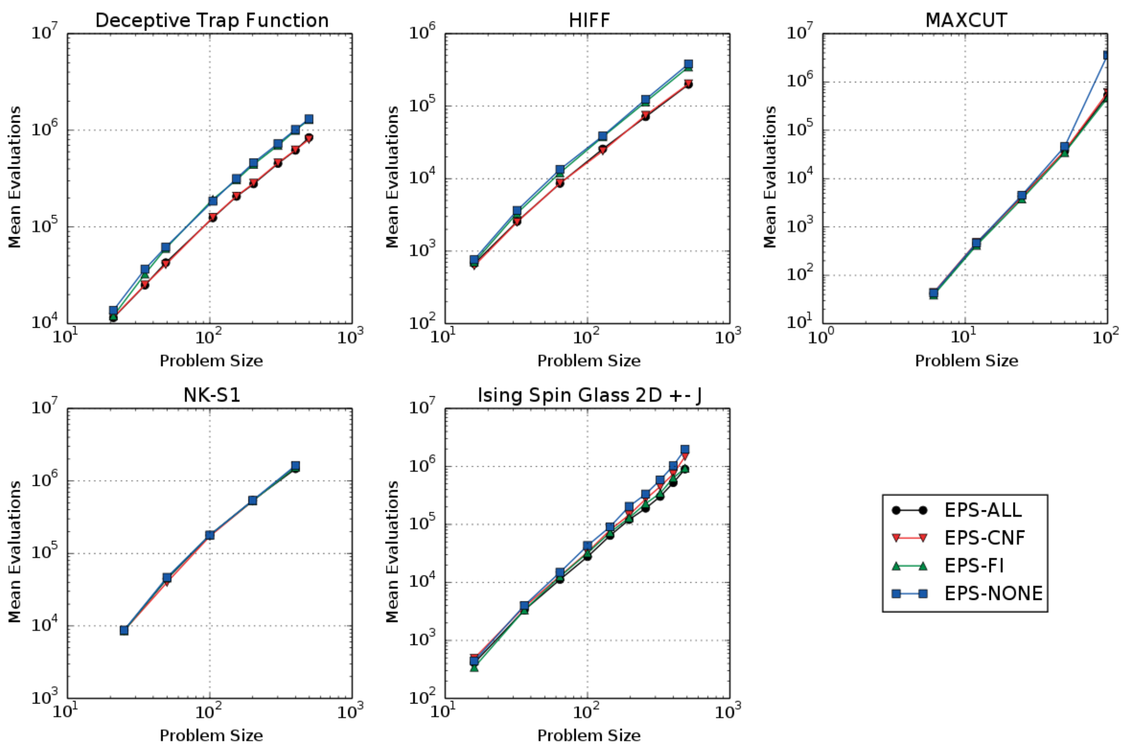


Figure 5.5: Comparison of the average number of fitness evaluations for the four variants of EPS. EPS-NONE uses no forced improvement or child node filtering. EPS-FI uses no child node filtering. EPS-CNF uses no forced improvement. EPS-ALL uses forced improvement and child node filtering.

5.5 Conclusions

GOMEA and two of its parameter-less variants were compared in an experiment to determine the overhead of using the parameter-less population schemes. First the 'optimal' population sizes for GOMEA were determined using the bisection search (section 3.2) for each of the five test problems. Note that the cost in fitness evaluations of performing bisection does not contribute to the final results of GOMEA, hence these fitness evaluations are considered *free*. In contrast, EPS performs a similar procedure for finding its final population size as bisection does, but these fitness evaluations *are* counted towards the final results. Given this observation, it seems counterintuitive that EPS actually performs on par with GOMEA even though the latter gets an order of magnitude more fitness evaluations for free in the preprocessing step. It reveals how the bisection search is likely to overestimate the population size required to solve the problem instance, as EPS will often solve the problem instance with a significantly lower population size. This average difference in population sizes between EPS and GOMEA is large enough to compensate for the overhead caused by EPS. Using a less strict threshold for the bisection search would likely reduce the overestimation of the population size, but the expected success-rate would drop as well. Given that EPS is required to solve the problem to optimality, this would lead to a skewed comparison where GOMEA is allowed to fail but EPS is not. From this point forward EPS will represent GOMEA in the benchmarks, as the performance of the two is akin and comparing parameter-less population schemes against each other is easier because the success-rate of the runs is no longer a factor. The adaptation of the parameter-less GA proved to be less fruitful. Though it was originally designed as a more efficient alternative to EPS, the concurrent racing of populations has a negative

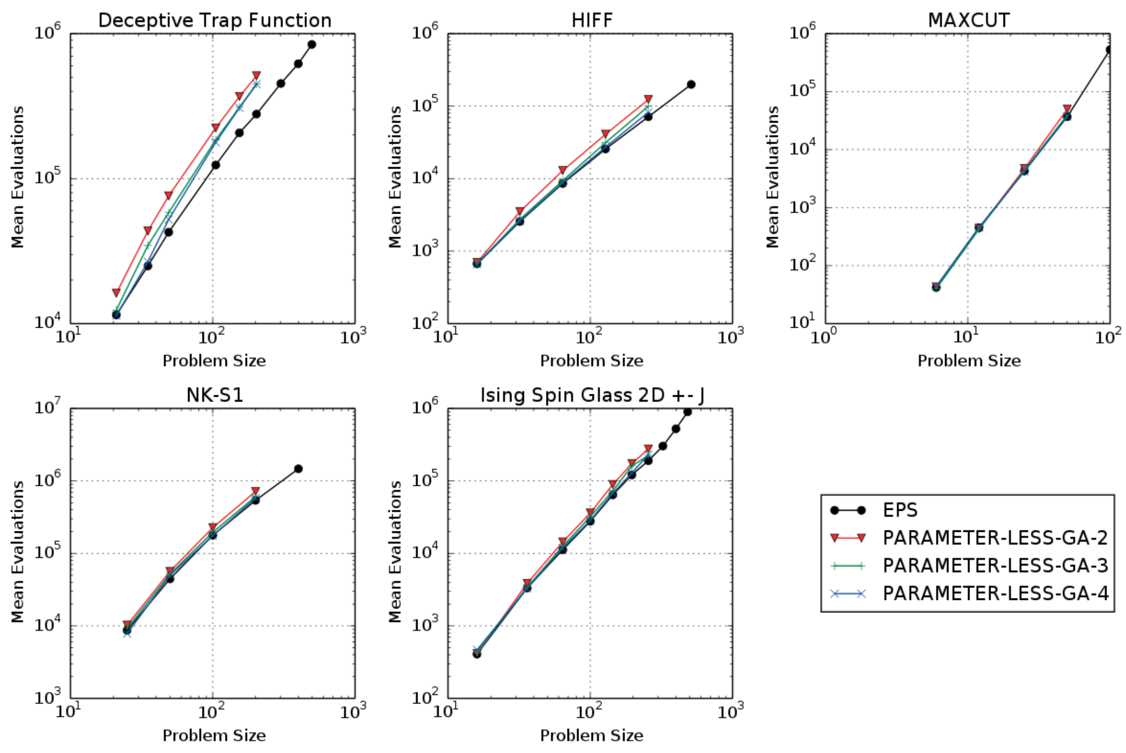


Figure 5.6: Comparison of the average number of fitness evaluations for the Parameter-less GA and EPS. The integer suffix in a label denotes the growth factor k for the Parameter-less GA.

impact on EPS due to the low number of generations in GOMEA versus the high number of generations used by a traditional GA. From the experiments we conclude there is no appropriate reason for exploring the parameter-less GA any further in this thesis. The simplicity and effectiveness of EPS make it a suitable reference point for the performance of other parameter-less population schemes.

Chapter 6

P3: Experimental Study

6.1 Overview

In the original paper [21] Goldman presented P3 as a carefully crafted combination of a hill climber, the pyramid, and the crossover method. The pieces complement each other so optimal performance is obtained. Though the combination of pieces was shown to perform admirably on all test problems, Goldman gave only a motivation for his choices, but no actual results of running P3 with differently configured pieces. His algorithmic design choices for the pieces form a set of tunable parameters that are worthy of exploration.

The goal of this chapter is to focus on P3 and figure out how its components contribute to its performance. The first two parameters to explore are relate to the crossover method used by P3. Crossover has two tunable parameters set differently from LTGA, namely the FOS ordering and the donor searching. The third parameter is whether or not the initial solutions are hill climbed by the first improvement hill climber. How these three parameters influence the performance of P3 is the subject matter of this chapter. An experiment is set up to explore those parameter configurations that are deemed worthy of reporting. Table 6.1 shows the code names for each parameter configuration used when reporting the results.

The influence of the FOS ordering is tested by comparing the random FOS ordering against the smallest-first FOS ordering. Goldman has previously shown that the FOS ordering does significantly alter the performance of LTGA [18] depending on the problem characteristics. However, the random FOS ordering had not been introduced back then. Sorting the FOS ordering is an $O(l \log l)$ operation, in contrast to the $O(l)$ Fisher-Yates shuffle for generating a random ordering. The null-hypothesis is assumed for the influence of the FOS ordering on the performance of P3.

Searching for a new donor until the donated bits are different from the original bits is a potentially expensive procedure where the entire population could be traversed. The motivation provided Goldman was purely based on empirical results, and the usefulness in copying converged deceptive traps. The odds of triggering an exhaustive donor search increases as a building block becomes more frequent in the population. Without exhaustive donor searching, the frequent building blocks introduce a bias towards solutions with the same building block, because the probability of selecting a random donor with the same building block increases with frequency. For LTGA this bias helps towards convergence of the population, but for P3 this bias potentially reduces the diversity of the existing populations. Hence the hypothesis for exhaustive donor searching is that the influence is significant, but whether the difference in performance is positive or negative differs is expected to differ per problem.

The hill climber provides decent quality solutions to the pyramid for crossover. Previous results have shown that performing crossover on populations of local optima works for

	PT	PTD	PTLS	PTLSD	PThree	P3
Exhaustive Donor Searching	No	Yes	No	Yes	Yes	Yes
First-Improvement Hill Climber	No	No	Yes	Yes	Yes	Yes
Smallest-first Cluster Order	No	No	No	No	Yes	Yes

Table 6.1: The names of the P3 variants and which components they include.

smaller populations. However, the cost of hill climbing solutions does not always outweigh the gain in fitness evaluations during crossover. The height of the pyramid is dependent on the fitness of the initial solutions and the number of fitness improving steps during crossover. Thus using the hill climber should reduce the height of the pyramid significantly. The hypothesis is stated as such: the efficiency of P3 degrades heavily without the use of the first improvement hill climber.

6.2 Configuration

The experiments are performed under the same conditions as in chapter 5 unless specified otherwise. Part of the initial work on P3 was verifying the results in the original paper[21] using an independent implementation of the P3 algorithm. The results by Goldman were verified and the resulting implementation was used to perform this experimental study. This experiment was then cross-verified using the implementation by Goldman, though those results are not reported here. Some of the test problems used by Goldman were not used as part of the experimental study, mostly because the added computational burden outweighed the only slightly larger coverage of problem characteristics.

6.3 Results

Figure shows the median number of evaluations until reaching the global optimum for the five variants of P3 on the test problems. The data for the variants of P3 was divided by results of P3 as Goldman originally designed it. This is to emphasize the differences from the original implementation, instead of independently showing the scalability of the variants.

Exhaustive donor searching shows the same erratic behavior regardless of whether the hill climber is used. Forcing donations of building blocks on the HIFF problem helps significantly with optimizing smaller building blocks and mixing the larger building blocks. Forcing the copying of deceptive traps has a severe negative impact on the performance. Forcibly copying non-trap clusters in a population of reasonably converged deceptive traps leads to wasted fitness evaluations because non-optimal traps are created and then rejected. The effect of exhaustive donor searching is largely positive on the Ising spin glasses, yet it has a negatively impact on the NK-S1 instances. This identifies exhaustive donor searching as a tunable parameter with a robust setting, i.e. not doing so, with significant impact on the performance.

Local searching has the expected positive effect on the performance of P3, but the hypothesis that hill climbing is quintessential for P3 to work is rejected. On the NK-S1 instances the use of a hill climber has a significant negative impact. The trade-off between hill climbing solutions for better linkage is not exactly worthwhile for the nearest-neighbor NK-landscapes. It is worth mentioning that local searching is part of the crossover operator because the singleton problem variable clusters are included in the FOS. Each singleton cluster is equivalent to attempting a bit flip, which is guaranteed happen when exhaustive donor searching is used. Goldman put a large emphasis on the FOS ordering because this

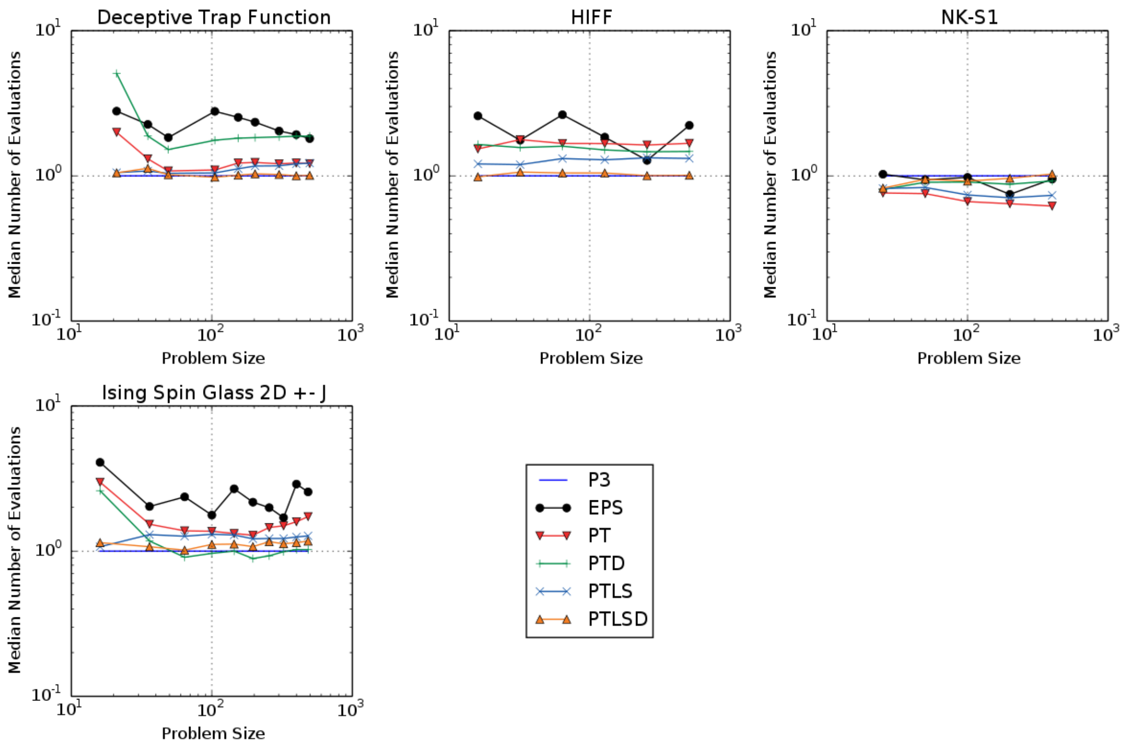


Figure 6.1: Comparison of the average number of fitness evaluations for the five variants of P3. Table 6.1 explains the parameter configuration per label. See appendix C for a magnified version of this plot.

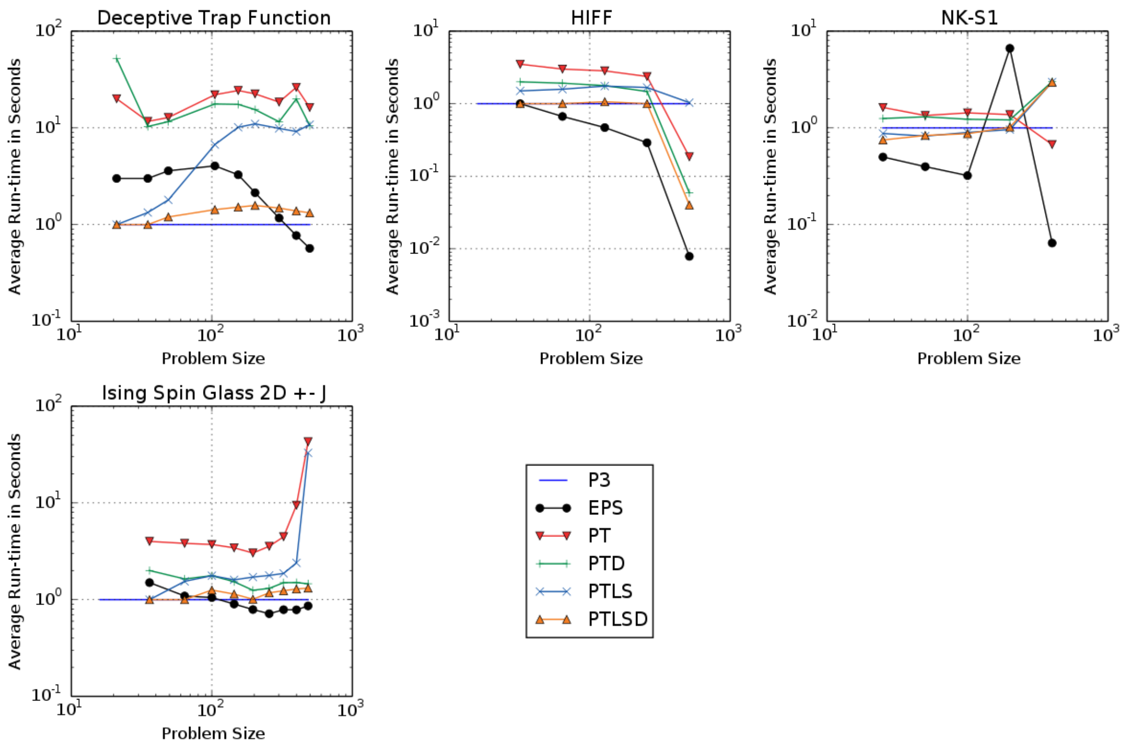


Figure 6.2: Comparison of the average run-time for the five variants of P3. Table 6.1 explains the parameter configuration per label. See appendix C for a magnified version of this plot.

includes the order in which local search is performed by the crossover operator. Usage of the hill climber with P3 is strongly recommended, but it is not a mandatory component for decent performance.

Whether or not the random FOS ordering was used did not have a overly significant impact on the performance. This identifies the FOS ordering as a robust parameter for either variant of the ordering. Some variants using the smallest first ordering were culled from the plots because they represented only redundant information.

Figure 6.2 shows the average runtime of P3 in milliseconds. The run-times of P3 and GOMEA give better insight on when to use one over the other. Evaluating a new solution is costly in a black-box setting because the fitness is recalculated from scratch. Additionally, fitness evaluations performed by the hill climber do not incur the cost of learning a linkage tree. Lastly, the cost of learning linkage trees is a dominant factor in the run-time of P3. These factors together determine the run-time of GOMEA, and the variants of P3. GOMEA outperforms P3 by a large constant on the HIFF function and NK-S1 instances. The original P3 exhibits exceptionally fast run-times on the deceptive traps function due to the behavior of the population pyramid when using exhaustive donor searching and the local searcher. This is so because the population pyramid will not grow beyond two layers as the crossover is almost guaranteed to copy over all deceptive traps in the first layer.

Goldman's observation of using exhaustive donor searching to improve the run-time of P3 were confirmed. Though exhaustive donor searching is not necessarily an improvement in performance, the results confirm it better exploits each evaluation per linkage tree resulting in a measurable decrease in run-time. The exception to the rule are the NK-S1 instances for reasons outlined earlier.

One observation remains true for all test problems: using the population pyramid without the hill climber adds another large constant overhead to the run-time.

6.4 Conclusions

The original P3 algorithm was decomposed into a population scheme with three optional components. The three components are the hill climber, exhaustive donor searching, and the smallest-first FOS ordering. Each of these components is optional when using GOMEA, so their influence on P3 was put to the test in an experiment. Two components, namely the hill climber and exhaustive donor searching, had a significant yet non-robust impact on the performance and run-time of P3. The FOS ordering was not as significant on the performance nor the run-time of the algorithm. It has been shown that P3 is not reliant on the hill climber to compete with GOMEA in performance, though GOMEA still outperforms P3 in the average run-time. Further work must be done to reduce the run-time and memory foot-print of the P3 algorithm to compete effectively with GOMEA and the exponential population scheme.

Chapter 7

Generalizing The Population Schemes

The previous chapters applied already existing parameter-less population schemes to the GOMEA framework and have shown their relative strengths and weaknesses. On one hand there is the exponential population scheme known for efficiency in model-building, and on the other hand there is the parameter-less population pyramid known for efficiency in fitness evaluations. From a pseudo-code perspective the two algorithms look nothing alike and share only the GOM operator and model learning from LTGA. The goal of this chapter is to improve upon the existing population schemes by combining them into a new population scheme. To do so, a visual interpretation of the population schemes is given to more easily explain how they are similar as evolutionary algorithms.

7.1 Visualizing Population Schemes

The focus in this section is showing the similarities between the exponential population scheme (EPS) and the parameter-less population pyramid (P3). Deriving a generic representation for either population scheme requires a more careful look at how GOMEA works. Each of the schemes relies on the GOM operator for performing crossover between a solution and a population. Hence the natural building block for deriving a generic representation is the GOM operator. There two key properties to describing a call to GOM: the set of solutions that are available for donation, and what happens with the solution after crossover. Figure 7.1 shows the visualization of the four population schemes used in this thesis. A breakdown on how to read and interpret each visualization is given in the following sections.

7.1.1 GOMEA

First the visualization of GOMEA is discussed, as shown in sub-figure 7.1a. The grid represents one run of the algorithm that had 5 generations (vertical axis) and a population size of 7 (horizontal axis). Each cell of the grid represents the state of solution x at generation y. Then each row must represent the state of the population at generation y, and each column is the progression of solution x over the generations. The solid arrows show whether the solution changed by GOM was accepted into the next generation. Because GOMEA always accepts the offspring solution into the new population, the progression is a chain of arrows in the visualization. GOMEA traverses the grid in left-to-right first (applying GOM to the population), bottom-to-top second order (initial generation until termination). Lastly, the dashed lines show which solutions from the previous generation have successfully donated bits to a solution.

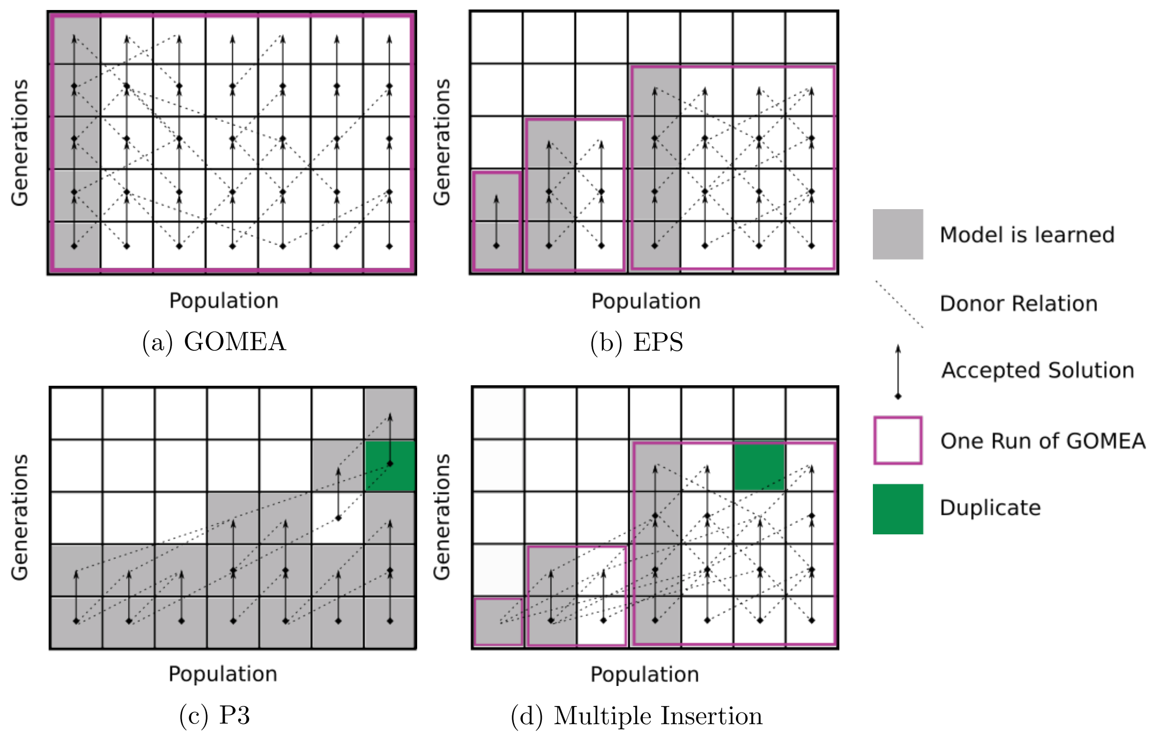


Figure 7.1: Visual representation of the population schemes: GOMEA, EPS, P3, and Multiple Insertion.

The probability of a successful donation decreases over time, and this is seen in the uppermost row where the donations are more sparse. If a cell has no dashed line with the previous row, then there was no successful donation during crossover, hence the solution indicated by the cell was not even improved. As seen by the mess of dashed lines in the grid, each solution has the remainder of the population at its disposal for mixing.

Let the age of a solution x be the number generations y it has existed, in other words the row index y of column x in the visualization. Two solutions 'live' in the same population if they are improved under the same circumstances, meaning they are improved using the same linkage tree and have access to the same donor solutions. If two solutions live in the same population and are of the same age, then applying GOM crossover to them should be optimal as their fitness should be similar due the competition with each other over the span of multiple generations. This automatically true for GOMEA, and together with the high availability of donors, the GOM operator is able to optimally mix solutions.

GOMEA learns the model, namely the linkage tree, from the entire population (i.e. row) of generation y. Whenever a model is learned before GOM crossover is applied, the corresponding cell is colored gray. For GOMEA this is the start of the generation, i.e. the first column. Clearly GOMEA is minimal in the number of times a model is learned, because otherwise a model would have to be used over multiple generations. No parameter-less population scheme can learn only one model on a row, because then it would have perfectly predicted the population size. Only oracles can tell beforehand what the population size necessary to solve the problem instance is.

The grid will be of similar size for most runs of GOMEA, assuming the average number of generations is similar and the population size is constant. Because GOM will attempt about $2l - 1$ fitness evaluations during crossover, the total number of fitness evaluations used by GOMEA will not vary much. Each run of GOMEA will have a grid of similar size with an almost equal number of attempted donations per cell. The experiments in chapter 5 on LTGA have shown this behavior to be true.

The visualization highlights the strengths and weaknesses of a population scheme. Given that the global optimum resides amongst solutions of similar fitness in the final generation, GOMEA has the weakness that it must fully evaluate every generation until the final generation has been reached. However, the sparsity of gray squares shows that relatively few linkage trees are constructed during a single run. The dashed arrows show how optimal mixing allows every solution to donate to every other solution over the entirety of the run. These observations are more meaningful when compared to the other population schemes.

7.1.2 EPS

Sub-figure 7.1b shows the visualization of the exponential population scheme (EPS). EPS starts with a run of GOMEA of size one, and continuously restarts the population on convergence with a doubled population size. Each of the pink squares denotes one run of GOMEA and is traversed exactly as explained in the previous section. The pink squares illustrate how EPS is a rather simplistic wrapper around GOMEA for doubling the population size. An iteration of EPS is the complete evaluation of one run of GOMEA, traversed from left-to-right in the grid. The remainder of the visualization builds on the previously established rules.

One thing of note is that the horizontal axis no longer corresponds to a single population. Instead each column represents one solution from the moment it was drawn from the random generator. The first population of EPS draws one solution from the population generator. The second population will draw two solutions from the population generator. The third population will draw four solutions from the population generator, and so on. New solutions are drawn from the generator after the previous solutions have been used up. This interpretation of the horizontal axis as a sequence of solutions drawn from a generator also applies retroactively to the GOMEA visualization. Learning a linkage tree only uses the solutions of generation y inside the pink square instead of the entire row.

The visualization shows the two major weaknesses of the EPS scheme. First, there is no optimal mixing between the runs of GOMEA as indicated by the lack of dashed arrows between the runs. Second, the gray squares are more abundant and show the wasted effort in building linkage trees for the previous generations. But it has already been shown that no parameter-less population scheme can get away with learning only one model per row, hence there being an additional cost of model learning is inevitable. Another useful observation is how the final population is a smaller grid than before, illustrating that the final population of EPS *can* be smaller in population size and the number of generations than a run of GOMEA. This is the primary reason for how EPS performs on par in fitness evaluations with GOMEA configured through bisection, because the latter will overestimate the population size (and thus the grid) by a fair bit.

7.1.3 P3

Sub-figure 7.1c shows the visualization of the parameter-less population pyramid (P3). The pseudo-code for P3 can be found for reference in listing 4 in section 4.4. With P3 comes a slightly different interpretation of the visualization, which mostly changes how the grid traversed. Green squares indicate that a solution was already present in the population pyramid, hence it was not accepted into the next layer of the pyramid. There is no solid arrow leading into a green square for this very reason. There is a direct correspondence between a population and the layer of a pyramid. Both contain a set of solutions, but the latter cannot contain duplicates by definition. The words for population and pyramid layer will be used interchangeably from this point forward.

Almost every cell is colored gray to show just how often P3 learns a new model, with the only exception being when a solution is not accepted into the next population. That only occurs when the solution is a duplicate already present in the pyramid, or the solution did not improve in fitness after GOM crossover with the population. The latter happens more often for the higher layers of the pyramid, because improving the fitness of an already optimized solution is more difficult. The linkage tree must be updated every time a new solution is inserted into the corresponding pyramid layer.

Each iteration of P3 a solution is drawn from the population generator, and is then improved until GOM crossover has been performed with each layer of the pyramid. A column x is then equivalent to x'th iteration of P3. Whereas GOMEA traverses the grid from the left-to-right first, P3 traverses the grid from bottom-to-top first. This behavior is the exact opposite of each other, as is illustrated further in the next subsection.

Looking at the dashed lines in the visualization, the solutions receive only donations from the previously evaluated solutions. The mixing that occurs in P3 is unidirectional over the horizontal axis, in contrast to the bi-directional mixing in GOMEA and EPS. Not only that, a solution x at generation y only has access to other solutions that also reached generation y. Hence the number of donors available to a solution (x, y) is lower than the number of donors available to a similarly placed solution (x, y) in GOMEA.

Age becomes an even more important factor when GOM is applied to solutions that were improved using different populations. How many the layers the solution has passed (the row) is equivalent to the age of the solution in GOMEA. The linkage trees learned by P3 become more precise as more solutions are added to the pyramid. However, a solution x was improved with a different linkage tree and population from any other solution. Thus when a solution is donated to, the age of the donor and the receiving solution will be the same, but they were raised in different populations. Hence there is a possible discrepancy in fitness and structure between the solution and the donor, which may lead to lesser performance when performing GOM crossover. The entropy in a population grows slower as more solutions are added, hence the linkage trees will change less over time. As a result, the discrepancy between solutions and donors is reduced over time, but is never truly eliminated.

Again, the visualization helps in understanding the strengths and weaknesses of P3. The scheme is over-saturated with model learning at every step of the algorithm, even though the entropy in the populations changes increasingly slower over time. Crossover uses donors of equal age to the receiving solution, but the donors were constructed under different circumstances. This may lead to fitness discrepancies in the early stages of the algorithm, because the model and populations are changing significantly with the addition of a single solution. In exchange for these weaknesses, the population pyramid requires no population size parameter and it gains access to the older generations earlier on than GOMEA.

7.1.4 Anology to Graph Searching

An analogy can be made between exploring a fitness landscape and searching a graph through either dept-first or breadth-first search. The goal of this analogy is to relate the population schemes in a manner that is more familiar to most computer scientists. The graph is defined by its vertices, namely the initial random solutions, and its edges which go to solutions in the local neighborhoods of the solutions. GOM crossover tests exactly $2l - 1$ different solutions through donations, so a solution can be interpreted as a vertex connected to $2l - 1$ other vertices. The result of GOM crossover is the offspring solution, so only one out of the $2l - 1$ edges is actually traversed by the search. Together the solutions in a population form a massive graph that explores the regions of the fitness landscape. First we show how P3 and GOMEA have exactly the opposite approach to exploring the 'graph' of solutions.

GOMEA's behavior is similar to that of a breadth-first search. First it greedily evaluates the neighboring solutions of the current population through the use of GOM crossover. Only after the entire neighborhood has been checked for each solution will it continue the search from the new population. The inherent drawback to this approach is that the breadth of the search (random solutions) is infinite, and so a limit (the population size) is required to continue the search.

P3's behavior is similar to that of a depth-first search. It greedily explores the neighborhood of one solution by repeatedly applying GOM crossover. Only after the solution has been fully exploited it continues the search from a new solution. The inherent drawback to this approach is that the linkage tree must be rebuilt for almost every solution. If P3 accepted solutions of equal fitness into the next pyramid layer, the depth-first search might never terminate. P3 can slowly expand the breadth of its search without setting a population size only because its depth-first search will terminate as there is a finite number of fitness values.

EPS combines this behavior of a breadth-first and a depth-first search. Each population is explored with a breadth-first search on the neighborhood by GOMEA, but the current population is fully exploited before the new population is started (hence the depth-first search). EPS provides the best compromise between the high cost of searching depth-first and requiring no fixed population size when searching breadth-first.

The parameter-less GA (section 5.4) was invented to cut the dept-first search of EPS short if the populations drifted, but it is too eager in evaluating larger populations. One might say it increases the breadth of the search too greedily based on the its growth parameter, just to shorten the depth-first search it maintains in the form of populations.

7.2 Multiple Insertion

Multiple Insertion is a new population scheme for GOMEA designed around the previous population schemes. The goal is to enhance EPS by using the population pyramid as a way to reuse the solutions from the previously converged populations. Similarly, the goal can be stated as using the strictly growing populations of EPS with P3 instead of one solution per iteration. In either case the result is a combination of both population schemes. The next subsection defines the multiple insertion (MI) scheme, followed by a detailed walkthrough, and the section concludes with the visualization of the scheme.

7.2.1 Definition

Listing 7 and 8 show the pseudo-code for the multiple insertion scheme. Note how it is similar to the pseudo-code for P3 (listing 4, section 4.4). The notion of iteration is taken over from EPS, where each iteration a new population is generated. The *Growth(i)* function determines the size of the population generated at the start of iteration i. This could for example be the exponential growth function $e(i) = 2^{i-1}$, or the linear growth function $l(i) = i$. It is assumed the iterations start counting at 1, as the minimum population size is also 1.

Let P_i be the i'th layer (or population) of the pyramid at the start of an iteration. First a population X is randomly generated with the population size determined by the growth function. Optionally the hill climber is invoked on each of the solutions in population X. The entire population X is checked for duplicates already in the pyramid, and the unique solutions are inserted into the bottom layer of the pyramid P_0 using the *UpdatePopulation* function. Then GOM crossover is applied to the population X using the pyramid layer P_0. The linkage tree for P_0 was already reconstructed when the unique solutions were inserted into the pyramid. Note that the unique solutions are now available as donors to

```
$X := \{x_0, x_1, \ldots, x_{Growth(i)-1}\}$;
$accepted := \{\}$;
for $x_j \in X$ do
    $x_j :=$ RandomSolution($l$);
    if $useLocalSearch$ then
        $x_j :=$ Hillclimb($x_j$);
    if $x_j \notin Seen$ then
        $Seen := Seen \cup \{x_j\}$;
        $accepted := accepted \cup \{x_j\}$;
UpdatePopulation($P_0, accepted$);
for $P_i \in P$ do
    $accepted := \{\}$;
    for $x_j \in X$ do
        $x'_j :=$ GOM($P_i, FOS_i, x_j$);
        if $Fitness(x'_j) > Fitness(x_j)$ and $x'_j \notin Seen$ then
            $Seen := Seen \cup \{x'_j\}$;
            $accepted := accepted \cup \{x'_j\}$;
        $X := X \cup x'_j \setminus x_j$;
    UpdatePopulation($P_{i+1}, accepted$);
```

Algorithm 7: Iterate(i)

```
if $|X| > 0$ then
    $P_i := P_i \cup X$;
    for $x_j \in X$ do
        UpdateFrequencies($P_i, x_j$);
    $FOS_i :=$ Clustering($P_i$);
```

Algorithm 8: UpdatePopulation(P_i, X)

the solutions in X during GOM crossover with P_0. Those solutions whose fitnesses were improved by the GOM crossover are inserted into the set of accepted solutions. Solutions that are already present in the pyramid are filtered from the set of accepted solutions. Finally, the set of accepted solutions is inserted into the next pyramid layer P_{i+1} using the *UpdatePopulation* function. This process repeats itself with X and the next pyramid layer P_{i+1} until the top of the pyramid has been reached. Note that the entire population X transitions to the next pyramid layer for GOM crossover, not just the solutions accepted into the next pyramid layer. As with P3, GOM crossover with the top layer may result in a new layer P_t. The new top layer P_t contains every solution from the set of accepted solutions at layer P_{t-1}. Unlike P3, the new layer may contain enough information for GOM crossover to improve the solutions, which would lead to the construction of a new top layer P_{t+1}. This process repeats itself until no fitness improvement is found in the top layer of the pyramid.

The function *UpdatePopulation* is used to insert the solutions whose fitnesses were improved into the next pyramid layer. It efficiently inserts every solution into the population, updates the frequency matrix accordingly, and then reconstructs the linkage tree for said pyramid layer. If none of the solutions were improved by the GOM crossover, then nothing happens to the pyramid.

7.2.2 Walkthrough

A walkthrough of the multiple insertion scheme is provided in the form three detailed iterations of the algorithm. The notation from the previous subsection is used, in addition to following notation: let x_j denote the i'th solution of X, and let x'_j denote solution x_j after applying GOM crossover once. The linear growth function $f(i) = i$ is assumed.

1. X : $\{x_0\}$
   Pyramid : $\{P_0 = \{\}\}$

   At iteration 1, the growth function is 1, so only solution x_0 is generated. As the pyramid is empty, solution x_0 must be unique and is added to the bottom layer of the pyramid P_0. GOM crossover cannot produce new solutions from a population of only one solution, so it is not necessary to construct a linkage tree for P_0 as it would have no effect. The iteration ends because there are no more pyramid layers to perform crossover on X with.

2. X : $X = \{x_1, x_2\}$
   Pyramid : $\{P_0 = \{x_0\}\}$

   At iteration 2, the growth function is 2, hence two solutions x_1 and x_2 are randomly generated. These solutions are entirely random, so it is unlikely x_1 and x_2 are duplicates of each other or x_0. Assuming the solutions are not equal, they are added to the population P_0. GOM crossover is applied to the new solutions in X with the updated population P_0. Before this can happen, the linkage tree must be rebuilt because the population P_0 was altered by inserting the two solutions. Note that the solutions x_1 and x_2 can be donors to each other because both were accepted into P_0. Let x'_1 and x'_2 denote the solutions after the GOM operator has been applied. Assume the fitness of x'_1 is strictly higher than that of x_1, and similarly for x'_2 and x_2. Then both x'_1 and x'_2 are inserted into the next population P_1, which is allocated at this point in time. Then the GOM operator at population P_1 is applied to the new solutions, before which the linkage tree of $P1 \cup \{x'_1, x'_2\}$ must be constructed. Under the assumption there were no fitness improvements, the iteration ends as there no more populations to perform crossover with.

3. X : $X = \{x_3, x_4, x_5\}$
   Pyramid : $\{P_0 = \{x_0, x_1, x_2\}, P_1 = \{x'_1, x'_2\}\}$

   At iteration 3, the growth function is 3, hence three solutions x_3, x_4 and x_5 are randomly generated. All of them are unique, so they are added to P_0, and the linkage tree for P_0 is rebuilt. Applying GOM on X with population P_0 results in the solutions $X = \{x'_3, x'_4, x'_5\}$. These solutions are added to P_1, and subsequently the GOM operator is applied to X with the updated population P_1. The linkage tree must be rebuilt first because the population P_1 has changed since it was built last. The resulting solutions are $X = \{x''_3, x''_4, x''_5\}$. Suppose $x''_4 = x'_2$, then the solution x''_4 was already in the pyramid as solution x'_2 and it is not accepted into population P_2. Similarly assume that $fitness(x''_5) = fitness(x'_5)$, then x''_5 is not accepted into population P_2 as the fitness of the solution has not improved after GOM crossover. Only x''_3 is accepted into P_2, and subsequently crossover occurs with $P_2 = \{x''_3\}$. Constructing a linkage tree for $P_2 \cup \{x''_3\}$ is not necessary, because the population is only of size one. Child node filtering would remove every cluster from the linkage tree, because the entropy between every cluster is equal (namely zero). The iteration ends because there are no more pyramid layers to perform crossover on X with.

4. X : $\{x_6, x_7, x_8, x_9\}$
   Pyramid : $\{P_0 = \{x_0, x_1, x_2, x_3, x_4, x_5\}, P_1 = \{x'_1, x'_2, x'_3, x'_4, x'_5\}, P_2 = \{x''_3, \}\}$

Repeat until the termination criterion has been met.

7.2.3 Visualization

Sub-figure 7.1d shows the visualization of the multiple insertion population scheme. The density of the gray squares has been significantly reduced in contrast to P3. Assuming the number of fitness evaluations remains similar to P3 for this scheme, then there should be a drastic gain in run-time because less linkage trees are learned. Of course there is no guarantee that the assumption holds, hence an experiment is set up in next chapter for verification. We can however give a reasoning for why this assumption should hold.

Adding a population pyramid to archive every generation so far is rather expensive in memory usage. This leads to worse caching behavior in an implementation, hence there should be a run-time penalty compared to EPS. Because multiple solutions are being improved at the same time, the likelihood of one solution being accepted into the next population increases. Hence it is highly probable that the model in the next generation is relearned after crossover. Storing and accessing the hash-map that ensures no duplicates are entered into the pyramid is a costly task as well. However, the addition of the population pyramid should result in less fitness evaluations, so this may compensate for the run-time cost of maintaining the population pyramid.

Only the mixing of the GOM crossover operator is left to be discussed. The dashed lines show that solutions can receive donations from either the previous populations or the current population under construction. There is the constraint that a solution is only a donor if it was accepted in the previous generation, because otherwise it would not be in the population. The receiving solution has more available donors during GOM crossover than it would have in either EPS or P3. More importantly, the solutions in the current population are improved using the same model, hence they are of the same age and live in the same population. As mentioned for GOMEA, these conditions allow for optimal mixing between the solutions. When mixing occurs with a donor from a previous population, there may still be a fitness discrepancy as shown for P3.

The linkage tree is no longer relearned for every solution inserted into the pyramid, so the population pyramid loses some of its immediate adaptiveness to a change in the pyramid. As more solutions are inserted into the pyramid layer at once, namely a subset of the population currently being improved, the entropy in the pyramid layer changes more significantly than it would have if one solution were inserted. This means that even though the model is relearned less frequently, the changes to the model are more substantial.

The observations made in the previous two paragraphs reinforce the idea that multiple insertion will not result in more fitness evaluations. We hypothesize that learning the model less frequently has little effect, or is at least compensated for by the better mixing conditions due to the reuse of the previous populations and optimal mixing.

Forced improvement is not used during crossover because the final step duplicates solutions and this is not allowed by the population pyramid. It is also partially present in the population pyramid, because the solution receives crossover with higher populations even if it was not improved previously. The termination criterion for population convergence is a mix between EPS and P3, namely when the population has not been improved during an entire generation *and* when there are no more layers of the pyramid to perform crossover with.

Using the exponential growth function for EPS is a necessity because otherwise too many fitness evaluations are wasted on prematurely converged populations. However, multiple insertion is able to reuse the previous populations, so it is important to test whether slower growth functions would be effective as well. It is likely that decreasing the growth rate will result in less fitness evaluations because the population size of the final iteration is less likely to be an overestimation.

```
if $|P_i| < 16$ then
    return
$numPairs := \lfloor \mathrm{Min}(c * n_{improved}, \frac{|P_i|}{2}) \rfloor$;
$tournament := \mathrm{Randomize}(P_i)$;
for $(x, y) \in Pairs(tournament, numPairs)$ do
    $deleted :=$ nil;
    if $fitness(x) > fitness(y)$ then
        $deleted := y$;
    if $fitness(x) < fitness(y)$ then
        $deleted := x$;
    else
        $deleted := \mathrm{Random}(x, y)$;
    $P_i := P_i \setminus deleted$;
    $\mathrm{UpdateFrequencies}(P_i, -deleted)$;
    $Seen := Seen \setminus deleted$;
```

Algorithm 9: BinaryTournament($P_i, c, n_{improved}$)

A linkage tree is lazily rebuilt from the population at the moment it is required by GOM crossover. The reason for this lazy behavior is twofold: rebuilding the linkage tree for each inserted solution is redundant, and reconstructing the linkage tree before it has been used by GOM crossover is a waste. The latter property lets other procedures alter the population without requiring another reconstruction of the linkage tree.

7.3 Memory Management

Maintaining a population pyramid comes at the cost of storing every unique solution in both the pyramid and the set data-structure of known solutions. By design the set data-structure should have an expected $O(l)$ lookup time and $O(nl)$ memory requirement, but there are no requirements for a specific implementation such as a hash-table. On large problem instances the population pyramid may explode when local optima are not guaranteed to have a high solution quality. In the worst-case each local optimum is stored in some layer of the population pyramid. This worst-case bound explodes when no hill climber is used and thus each of the 2^l solutions can be inserted into the population pyramid.

Even the number of local optima is potentially exponential, take for instance the deceptive trap function (see appendix A). If any one trap has not converged to a local optimum of all ones or all zeros, then the fitness can be improved by converging said trap, hence the bit string is not a local optimum. The local optima of a deceptive trap function can thus be encoded in a bit string of length $\frac{l}{k}$, where each bit encodes a trap set to all ones or all zeros. The number of local optima on the deceptive trap function is exactly by $2^{\frac{l}{k}}$. Storing this many solutions can be difficult even on modern computers with tens of gigabytes of memory. Ignoring the memory constraint, operating on data of such scale is bound to slow down the implementation. For the population pyramid to scale on large problem instances, some changes must be made to the population pyramid scheme.

Memorizing each solution encountered so far is expensive in memory and could be unnecessary. For LTGA to mix solutions the building blocks must be present in the population, but not every solution containing them needs to be stored. Using another niching technique (next to the removal duplicates) to select solutions for removal is inappropriate, because this is harmful to building blocks that are propagated more often through the population pyramid. Instead the solutions can be filtered using selection procedures from the literature on traditional GAs. The population pyramid uses no procedures dependent on fitness proportions, and this useful property would be gone if fitness proportional selection is used. Hence our the choice fell on binary tournament selection where solutions compete against each other in pairs. Setting the tournament size to two helps avoid adding another parameter to the population scheme.

Applying the selection procedure changes the population, so the associated linkage tree must be relearned. There is one moment during crossover when the population can be changed without needing to reconstruct the corresponding linkage tree, namely between the moment of GOM crossover and insertion of the solutions into the population. Let n_{improved} be the number of solutions inserted into a pyramid layer, assuming that multiple insertion is used. The selection procedure proposed here performs a binary tournament selection on $s = 2c * n_{\text{improved}}$ randomly selected solutions from pyramid layer. Note that this excludes the newly introduced solutions from being selected, as they have not been inserted yet. Ties in fitness are broken randomly. Let c be a parameter for tuning the fraction of solutions to be removed from the pyramid layer. At the end of selection exactly $c * n_{\text{improved}}$ solutions have been removed from the population. Selection is only applied on populations containing at least 16 solutions to ensure small populations are given a chance to grow. Listing 9 shows the pseudo-code for binary tournament selection. The function is invoked during the *UpdatePopulation* function (listing 8) before the population is modified on line 2.

This selection procedure culls more solutions after crossover with the lower pyramid layers because the expected number of improvements during GOM crossover is higher when the solutions have not been optimized yet. The motivation for this behavior is twofold. First, the upper pyramid layers contain stronger building blocks that should be protected from selection. Second, using this definition c scales robustly with the growth function used by multiple insertion.

The selection procedure can be entirely avoided by setting c to zero, so there is a robust parameter setting. However, no automatic adjustments are given for the value of c, so it is a non-robust parameter in that sense. The parameter c scales with the number of improvements during crossover, which is more robust than scaling with the non-robust population size as occurs often for traditional selection.

If the selection procedure negatively impacts the efficiency of the population pyramid, then there are two possible explanations as to why that happens. Either the selection procedure introduces a negative bias towards fit solutions, or we have underestimated the impact of maintaining a diverse population. Layers in the bottom of the population pyramid are expected to be larger and contain more random solutions, so the selection procedure should be more aggressively applied there.

Removal of solutions from the population pyramid has another potentially useful side effect. When a solution has entered the population pyramid, it can no longer be inserted into any other layer. This static behavior is relevant because the populations and thus the average fitnesses change over time. After a solution has been removed by the selection procedure, it can be reinserted into another part of the pyramid. Hence selection allows the pyramid to change over time into populations that are more coherent in fitness.

Chapter 8

Multiple Insertion: Experimental Study

The goal of this chapter is to test whether the changes proposed in the previous chapter are effective additions to P3. The first section describes the changes made to the experimental setup for this particular experiment. The second section shows the results of adding multiple insertion to the population pyramid scheme (PT) (section 6) and the original parameter-less population pyramid (P3). The third section evaluates the effect of adding binary tournament selection to the population pyramid scheme (PT). Lastly, a summary of the results is provided along with a robust parameter setting to ensure that multiple insertion remains a parameter-less population scheme.

8.1 Configuration

The experimental setup is the same as chapter 6 unless specified otherwise. P3 uses the parameter configuration of the original P3 algorithm, as explained in section 4.4. The population pyramid scheme (PT) removes from P3 the hill climber, exhaustive donor searching, and it randomizes the FOS ordering. Multiple insertion is added to both population schemes. Multiple insertion requires the specification of a growth function to determine how fast the population increase in size per iteration of the algorithm. Let $f(i)$ be the size of the population inserted into the population pyramid at iteration i. Three growth functions are tested: the linear function $f(i) = i$, the quadratic function $q(i) = i^2$, and the exponential function $e(i) = 2^(i - 1)$. One can interpret both PT and P3 as a form of multiple insertion if the growth function is a constant $c(i) = 1$. It is assumed the number of iterations starts counting at 1.

Measuring the memory overhead of a population scheme is required to evaluate the usefulness of the binary tournament selection. The memory overhead of a population scheme is measured as the size of the pyramid at the moment of termination. This is a robust measure that translates well between different implementations, which is exactly the same reasoning behind measuring the number of fitness evaluations. The amount of allocated memory allocations is easily optimized by a compiler or programmer and thus is a poor indicator for the overall memory overhead of the algorithm. Learning a linkage tree and performing crossover require only a static allocation of $O(l^2)$ memory, hence the memory usage by P3 is bounded by the size of the population pyramid. Each solution is replicated exactly twice when it is inserted into the pyramid.

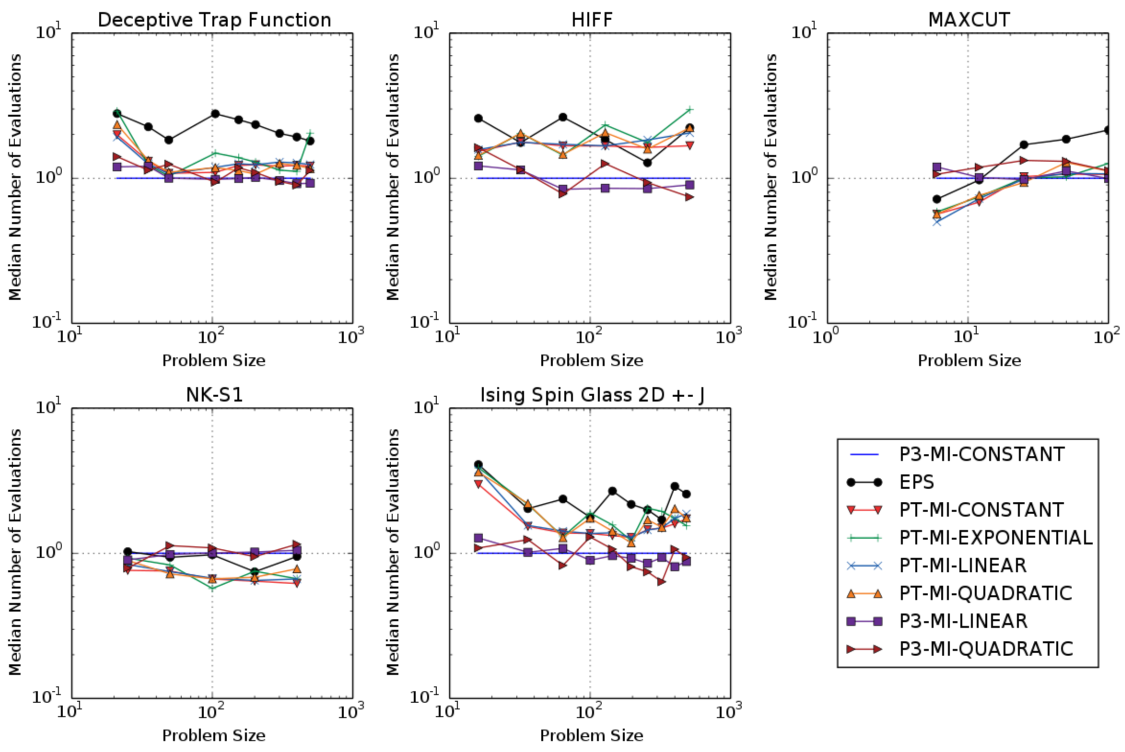


Figure 8.1: Comparison of the median number of fitness evaluations for the multiple insertion variants of PT and P3.

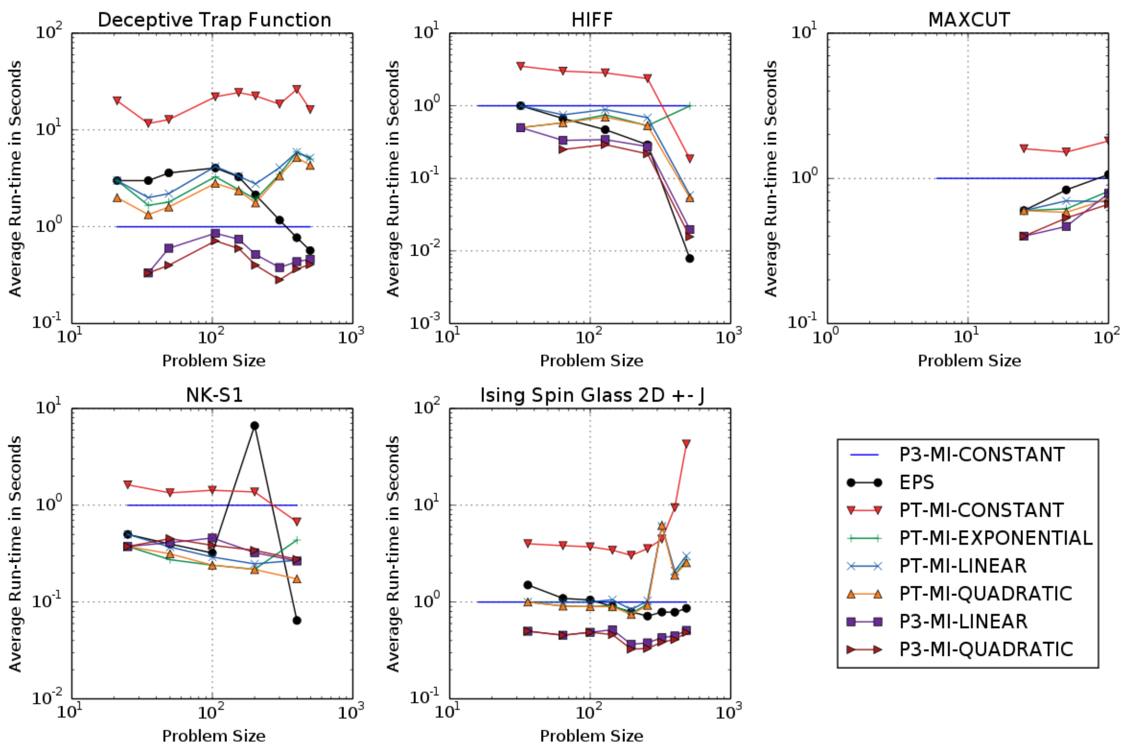


Figure 8.2: Comparison of the average running time for the multiple insertion variants of PT and P3.

			Fitness Evaluations			Pyramid Size		
Problem	l	c	10p	med	90p	10p	med	90p
DT (k=7)	497	0	$3.6 \cdot 10^5$	$4.6 \cdot 10^5$	$6.4 \cdot 10^5$	$2 \cdot 10^3$	$2.6 \cdot 10^3$	$3.4 \cdot 10^3$
DT (k=7)	497	0.25	$3.2 \cdot 10^5$	$4.2 \cdot 10^5$	$5.7 \cdot 10^5$	$1.5 \cdot 10^3$	$1.9 \cdot 10^3$	$2.4 \cdot 10^3$
DT (k=7)	497	0.5	$3 \cdot 10^5$	$4.1 \cdot 10^5$	$5.5 \cdot 10^5$	$9.7 \cdot 10^2$	$1.2 \cdot 10^3$	$1.6 \cdot 10^3$
HIFF	512	0	$1.3 \cdot 10^5$	$1.9 \cdot 10^5$	$2.9 \cdot 10^5$	$4.6 \cdot 10^2$	$6.3 \cdot 10^2$	$9 \cdot 10^2$
HIFF	512	0.25	$1.3 \cdot 10^5$	$1.7 \cdot 10^5$	$2.7 \cdot 10^5$	$4.3 \cdot 10^2$	$5.3 \cdot 10^2$	$7.4 \cdot 10^2$
HIFF	512	0.5	$1.2 \cdot 10^5$	$1.7 \cdot 10^5$	$2.5 \cdot 10^5$	$3.2 \cdot 10^2$	$4 \cdot 10^2$	$5.3 \cdot 10^2$
MaxCut	100	0	$5.3 \cdot 10^4$	$1.9 \cdot 10^5$	$6.8 \cdot 10^5$	$5.3 \cdot 10^2$	$1.5 \cdot 10^3$	$4.4 \cdot 10^3$
MaxCut	100	0.25	$5.3 \cdot 10^4$	$1.7 \cdot 10^5$	$5.4 \cdot 10^5$	$4.6 \cdot 10^2$	$1.1 \cdot 10^3$	$3 \cdot 10^3$
MaxCut	100	0.5	$5.1 \cdot 10^4$	$1.8 \cdot 10^5$	$6.4 \cdot 10^5$	$3.5 \cdot 10^2$	$8.5 \cdot 10^2$	$2.5 \cdot 10^3$
NK-S1	400	0	$7.1 \cdot 10^5$	$9.8 \cdot 10^5$	$1.8 \cdot 10^6$	$1.8 \cdot 10^3$	$2.5 \cdot 10^3$	$4.3 \cdot 10^3$
NK-S1	400	0.25	$7.6 \cdot 10^5$	$1.1 \cdot 10^6$	$1.8 \cdot 10^6$	$1.6 \cdot 10^3$	$2.2 \cdot 10^3$	$3.6 \cdot 10^3$
NK-S1	400	0.5	$9 \cdot 10^5$	$1.2 \cdot 10^6$	$1.8 \cdot 10^6$	$1.3 \cdot 10^3$	$1.7 \cdot 10^3$	$2.5 \cdot 10^3$
Spin Glasses	484	0	$2.1 \cdot 10^5$	$4.7 \cdot 10^5$	$1.9 \cdot 10^6$	$5.7 \cdot 10^2$	$1.2 \cdot 10^3$	$4.4 \cdot 10^3$
Spin Glasses	484	0.25	$2 \cdot 10^5$	$4.3 \cdot 10^5$	$1.5 \cdot 10^6$	$5 \cdot 10^2$	$9.3 \cdot 10^2$	$2.8 \cdot 10^3$
Spin Glasses	484	0.5	$2 \cdot 10^5$	$4.3 \cdot 10^5$	$1.5 \cdot 10^6$	$3.9 \cdot 10^2$	$6.8 \cdot 10^2$	$1.9 \cdot 10^3$

Table 8.1: Results for binary tournament selection used on PT with multiple insertion and the linear growth function.

8.2 Growth Functions for Multiple Insertion

Figure 8.1 shows the average number of fitness evaluations for the P3, PT, and its multiple insertion variants. The labels in the legend follow a specific format: {Scheme} − *MI* − {Growth}, where Scheme is either PT or P3 and Growth is either the linear, quadratic, or exponential growth function. See appendix C for an enlarged version of the plot. Similarly to results in chapter 6, the results are normalized using the results of the original P3 to better highlight the differences in the performance. As seen in figure 8.1, the quadratic and exponential growth functions diverge significantly from PT at several occasions. In contrast, the linear growth function is less prone to this erratic behavior. Especially the exponential growth function tends to have outliers in the number of fitness evaluations, to the point of exceeding EPS. For EPS it makes sense to scale exponentially, as a slower growth leads to more population being converged and rejected, which is terribly expensive. PT with multiple insertion is an extension of EPS that reuses every solution whereas EPS cannot do that. Because PT with multiple insertion is less prone to outliers with a slower growth function, there is little reason to use an exponential growth function.

Figure 8.2 shows the average run-time in milliseconds for P3, PT, and the multiple insertion variants. Using multiple insertion with PT reduces the run-time to the level of the original P3 which uses the hill climber. However, increasing the growth function beyond linear has no significant effect on the run-time, as seen by the tightly grouped lines per population scheme. Applying multiple insertion to the original P3 algorithm greatly speeds up the run-time. On any of the test problems the run-time of P3 matches or outperforms EPS, which is something it rarely achieved without multiple insertion. The exception to the rule are the NK-S1 problem instances, but that is due to use the poor performance caused by the hill climber. The effectiveness of multiple insertion shows that P3 and the population pyramid as designed originally waste a liberal amount of processing time on rebuilding linkage trees to little effect.

The addition of multiple insertion has introduced a new parameter in the form of the growth function. Ensuring the multiple insertion scheme remains parameter-less requires

			Fitness Evaluations			Pyramid Size		
Problem	l	c	10p	med	90p	10p	med	90p
DT (k=7)	497	0	$3 \cdot 10^5$	$4.3 \cdot 10^5$	$6 \cdot 10^5$	$1.7 \cdot 10^3$	$2.3 \cdot 10^3$	$3.1 \cdot 10^3$
DT (k=7)	497	0.25	$2.8 \cdot 10^5$	$4.1 \cdot 10^5$	$5.7 \cdot 10^5$	$1.3 \cdot 10^3$	$1.7 \cdot 10^3$	$2.3 \cdot 10^3$
DT (k=7)	497	0.5	$2.7 \cdot 10^5$	$3.8 \cdot 10^5$	$5.3 \cdot 10^5$	$8.8 \cdot 10^2$	$1.2 \cdot 10^3$	$1.5 \cdot 10^3$
HIFF	512	0	$1.2 \cdot 10^5$	$2 \cdot 10^5$	$3.2 \cdot 10^5$	$4 \cdot 10^2$	$6.5 \cdot 10^2$	$9.5 \cdot 10^2$
HIFF	512	0.25	$1.1 \cdot 10^5$	$2 \cdot 10^5$	$3.1 \cdot 10^5$	$4 \cdot 10^2$	$5.8 \cdot 10^2$	$8.1 \cdot 10^2$
HIFF	512	0.5	$1 \cdot 10^5$	$1.7 \cdot 10^5$	$2.7 \cdot 10^5$	$3 \cdot 10^2$	$4.2 \cdot 10^2$	$5.7 \cdot 10^2$
MaxCut	100	0	$5.9 \cdot 10^4$	$1.9 \cdot 10^5$	$6.2 \cdot 10^5$	$5.8 \cdot 10^2$	$1.5 \cdot 10^3$	$4.2 \cdot 10^3$
MaxCut	100	0.25	$6.3 \cdot 10^4$	$1.9 \cdot 10^5$	$6.6 \cdot 10^5$	$5.3 \cdot 10^2$	$1.2 \cdot 10^3$	$3.6 \cdot 10^3$
MaxCut	100	0.5	$5.3 \cdot 10^4$	$1.8 \cdot 10^5$	$5.6 \cdot 10^5$	$3.9 \cdot 10^2$	$8.9 \cdot 10^2$	$2.1 \cdot 10^3$
NK-S1	400	0	$7.9 \cdot 10^5$	$1.2 \cdot 10^6$	$2 \cdot 10^6$	$2 \cdot 10^3$	$2.9 \cdot 10^3$	$5.1 \cdot 10^3$
NK-S1	400	0.25	$7.8 \cdot 10^5$	$1.1 \cdot 10^6$	$1.8 \cdot 10^6$	$1.7 \cdot 10^3$	$2.3 \cdot 10^3$	$3.5 \cdot 10^3$
NK-S1	400	0.5	$8 \cdot 10^5$	$1.2 \cdot 10^6$	$2.1 \cdot 10^6$	$1.3 \cdot 10^3$	$1.7 \cdot 10^3$	$2.8 \cdot 10^3$
Spin Glasses	484	0	$2.5 \cdot 10^5$	$4.4 \cdot 10^5$	$1.8 \cdot 10^6$	$6.6 \cdot 10^2$	$1.1 \cdot 10^3$	$4 \cdot 10^3$
Spin Glasses	484	0.25	$2.5 \cdot 10^5$	$4.3 \cdot 10^5$	$1.3 \cdot 10^6$	$6 \cdot 10^2$	$9.5 \cdot 10^2$	$2.4 \cdot 10^3$
Spin Glasses	484	0.5	$2.3 \cdot 10^5$	$4.1 \cdot 10^5$	$1.2 \cdot 10^6$	$4.5 \cdot 10^2$	$6.8 \cdot 10^2$	$1.6 \cdot 10^3$

Table 8.2: Results for binary tournament selection used on PT with multiple insertion and the quadratic growth function.

us to set a robust, or even recommended, value for the growth function parameter. The performance of the linear growth function is on par with the quadratic function and more reliable than the exponential function, hence we recommend the use of the linear growth function.

8.3 Selection Procedure

This section shows the results of performing the selection procedure in combination with multiple insertion. Tables 8.1, 8.2, and 8.3 show the results of running PT using binary tournament selection per configuration of the linear, quadratic, and exponential growth function, respectively. Only the largest problem instances from the experiments done so far are shown in the tables. The columns 10p, med, and 90p indicate the 10th percentile, the median, and the 90-th percentile, respectively. The selection parameter c was tested for values $\frac{1}{4}$ and $\frac{1}{2}$, and the values for $c = 0$ were reused from the previous experiment.

We expect the selection procedure to reduce the population pyramid size to some fraction of the previous results. As a fraction c of the solutions is removed through selection, we would expect the population pyramid to be reduced by a similar factor. On several of the problem instances the reduction in size did not match our expectations. On the deceptive trap function applying selection has a positive effect on the number of fitness evaluation and the pyramid actually shrinks as much as expected. Most likely the solutions containing fewer converged (optimal) traps are removed more often, hence crossover is more likely to select from the fit solutions of useful traps. HIFF responds well to selection, but the reduction in solutions never comes close to the fraction c. This behavior holds for each the growth functions as shown in the tables.

The behavior of selection on the NK-S1 instances stands out the most amongst the results for the linear growth function. Performing selection on the NK-S1 instances has a measurable negative impact in contrast to the other problem instances. A similar behavior was observed when the influence of the hill climber on P3 was tested in chapter 6. It also shows the importance of covering a broad range of problems in order to find these

			Fitness Evaluations			Pyramid Size		
Problem	l	c	10p	med	90p	10p	med	90p
DT (k=7)	497	0	$3.6 \cdot 10^5$	$7.5 \cdot 10^5$	$8 \cdot 10^5$	$1.9 \cdot 10^3$	$3.6 \cdot 10^3$	$3.8 \cdot 10^3$
DT (k=7)	497	0.25	$3.5 \cdot 10^5$	$3.6 \cdot 10^5$	$7.3 \cdot 10^5$	$1.4 \cdot 10^3$	$1.5 \cdot 10^3$	$2.8 \cdot 10^3$
DT (k=7)	497	0.5	$3.3 \cdot 10^5$	$3.4 \cdot 10^5$	$7.1 \cdot 10^5$	$1.2 \cdot 10^3$	$1.3 \cdot 10^3$	$2.3 \cdot 10^3$
HIFF	512	0	$1.2 \cdot 10^5$	$2.7 \cdot 10^5$	$3 \cdot 10^5$	$4.2 \cdot 10^2$	$8.1 \cdot 10^2$	$8.6 \cdot 10^2$
HIFF	512	0.25	$1.2 \cdot 10^5$	$2.6 \cdot 10^5$	$2.8 \cdot 10^5$	$4.1 \cdot 10^2$	$7.1 \cdot 10^2$	$7.5 \cdot 10^2$
HIFF	512	0.5	$1.1 \cdot 10^5$	$2.4 \cdot 10^5$	$2.5 \cdot 10^5$	$3.3 \cdot 10^2$	$5.8 \cdot 10^2$	$6.1 \cdot 10^2$
MaxCut	100	0	$6.7 \cdot 10^4$	$2.2 \cdot 10^5$	$6.2 \cdot 10^5$	$6.7 \cdot 10^2$	$1.8 \cdot 10^3$	$4 \cdot 10^3$
MaxCut	100	0.25	$8.3 \cdot 10^4$	$2.1 \cdot 10^5$	$6.1 \cdot 10^5$	$6.9 \cdot 10^2$	$1.5 \cdot 10^3$	$3.1 \cdot 10^3$
MaxCut	100	0.5	$7.8 \cdot 10^4$	$2.1 \cdot 10^5$	$5.5 \cdot 10^5$	$5.6 \cdot 10^2$	$1.3 \cdot 10^3$	$2.7 \cdot 10^3$
NK-S1	400	0	$8.4 \cdot 10^5$	$9.9 \cdot 10^5$	$1.8 \cdot 10^6$	$2.2 \cdot 10^3$	$2.6 \cdot 10^3$	$4.7 \cdot 10^3$
NK-S1	400	0.25	$8.1 \cdot 10^5$	$9.7 \cdot 10^5$	$1.7 \cdot 10^6$	$1.8 \cdot 10^3$	$2 \cdot 10^3$	$3.4 \cdot 10^3$
NK-S1	400	0.5	$8 \cdot 10^5$	$1 \cdot 10^6$	$1.7 \cdot 10^6$	$1.6 \cdot 10^3$	$1.9 \cdot 10^3$	$3 \cdot 10^3$
Spin Glasses	484	0	$1.7 \cdot 10^5$	$3.9 \cdot 10^5$	$1.7 \cdot 10^6$	$4.7 \cdot 10^2$	$9.9 \cdot 10^2$	$3.9 \cdot 10^3$
Spin Glasses	484	0.25	$1.8 \cdot 10^5$	$3.8 \cdot 10^5$	$1.6 \cdot 10^6$	$4.8 \cdot 10^2$	$8.5 \cdot 10^2$	$3 \cdot 10^3$
Spin Glasses	484	0.5	$1.6 \cdot 10^5$	$3.7 \cdot 10^5$	$1.6 \cdot 10^6$	$3.9 \cdot 10^2$	$7.2 \cdot 10^2$	$2.5 \cdot 10^3$

Table 8.3: Results for binary tournament selection used on PT with multiple insertion and the exponential growth function.

anomalies. Ising Spin Glasses show exactly the opposite behavior, because high selection pressure significantly reduces the pyramid size and the distribution of fitness evaluations. Previous experiments on the Ising Spin Glasses have shown there is a clear advantage to using a hill climber. This problem-specific behavior must be taken into account when comparing results between algorithms with and without a local searcher or other techniques biased towards high fitness.

Using selection is as much as a robust parameter as using hill-climber is a robust parameter. It can be safely used without hurting the number fitness evaluations significantly, but it can be tuned (or turned off) if the problem at hand reacts poorly to elitist search behavior.

8.4 Conclusions

The results of applying multiple insertion on the population pyramid scheme (PT) and the parameter-less population pyramid (P3) were compared to those of the exponential population scheme (EPS). Using multiple insertion results in drastically reduced run-times compared to their original counterparts. The effect of the growth function on the number of fitness evaluations varies per problem. The results indicate that using a quadratic- or exponential growth function makes the algorithm more susceptible to outliers in performance. Hence we recommended the use of multiple insertion in conjunction with either EPS or P3, and more specifically using multiple insertion with the linear growth function for reliable performance. Binary tournament selection was effective at reducing the size of the population pyramid while also reducing the number of fitness evaluations required on some of the problems. The usage of the selection procedure is entirely optional and meant solely for reducing the memory requirements of the population pyramid. However, setting the selection parameter to a value of 0.5 is practical on all but one of the problem instances.

Chapter 9

Discussion

This chapter is devoted to those insights on parameter-less population schemes that have no proper place in the experimental studies. The first section summarizes the related work done so far on other similar parameter-less GAs from known literature. The second section focuses on adaptive population sizing schemes and particularly on the review of them by F. Lobo and C. F. Lima [13]. The remainder of the section summarizes the pitfalls one will encounter when designing an adaptive population sizing scheme (for GOMEA).

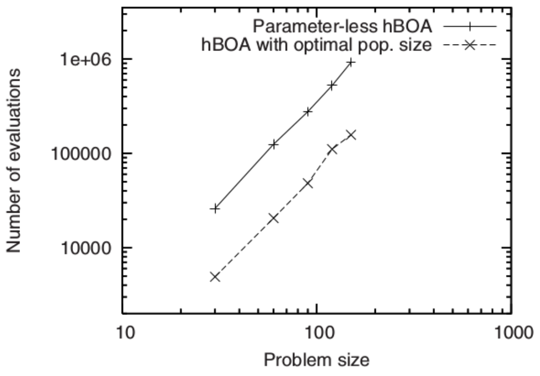
9.1 Other Parameter-less Genetic Algorithms

Existing GAs are rarely adapted to a parameter-less population scheme, but preliminary research has been performed for two probabilistic model building GAs: the hierarchical Bayesian Optimization Algorithm and the Extended Compact GA. Both algorithms were adapted to the parameter-less GA, which was and still is the only comprehensive work on parameter-less population schemes so far. The following sections illustrate the difficulties of applying a parameter-less population scheme to other GAs, and how their results relate to those in this thesis.

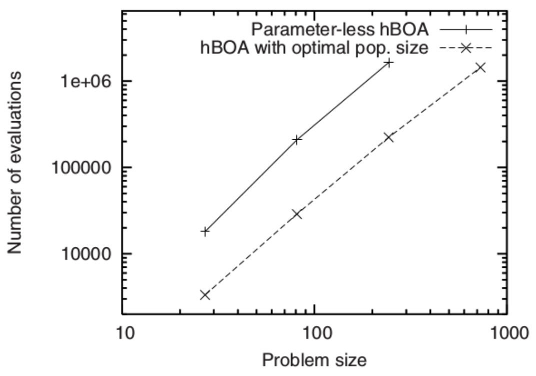
9.1.1 Parameter-less hBOA

The Hierarchical Bayesian Optimization Algorithm is a probabilistic building model genetic algorithm that learns a Bayesian network from the population and samples new solutions from the model. Learning the Bayesian network is a relatively expensive step in comparison to learning a linkage tree. A niching technique called restricted tournament selection is used in which a solution must compete for its place in the population against the closest amongst w solutions in terms of Hamming distance. Niching ensures the population remains diverse in order to prolong premature convergence, though it will still occur in practice. Parameter-less hBOA[10] uses the parameter-less GA by G. Harik and F. Lobo[8] with a different counter. The alternative counter schedules the evaluation of larger populations earlier than the original scheme would. The growth parameter was set to 4, which means the window of concurrent populations tends to be small.

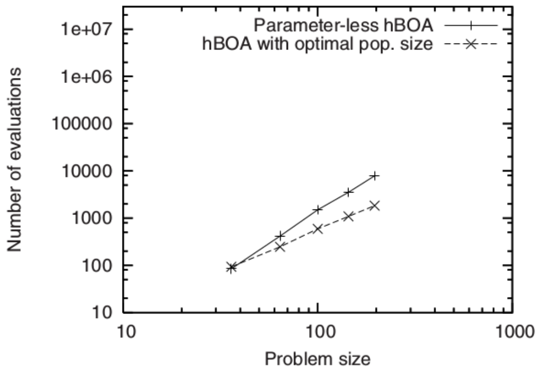
Figure 9.1 shows the results of the parameter-less hBOA solver on three optimization problem: the deceptive trap function (size 3), the hierarchical trap function (a variant of HIFF), and a set of randomly generated of Ising Spin Glass 2D $\pm$ J problem instances. On the first two problems only a constant overhead was introduced by using the population scheme, but the results do differ almost an order of magnitude in number of evaluations. A non-constant overhead in number of evaluations was found on the Ising Spin Glass instances. However, a best improvement hill climber was applied in conjunction with the Parameter-less hBOA algorithm. It has yet to be verified whether the worse scaling of



(a) Deceptive Trap Function



(b) Hierarchical Trap Function



(c) Ising Spin Glass

Figure 9.1: Figure 2, 3, and 4 from the paper on Parameter-less hBOA

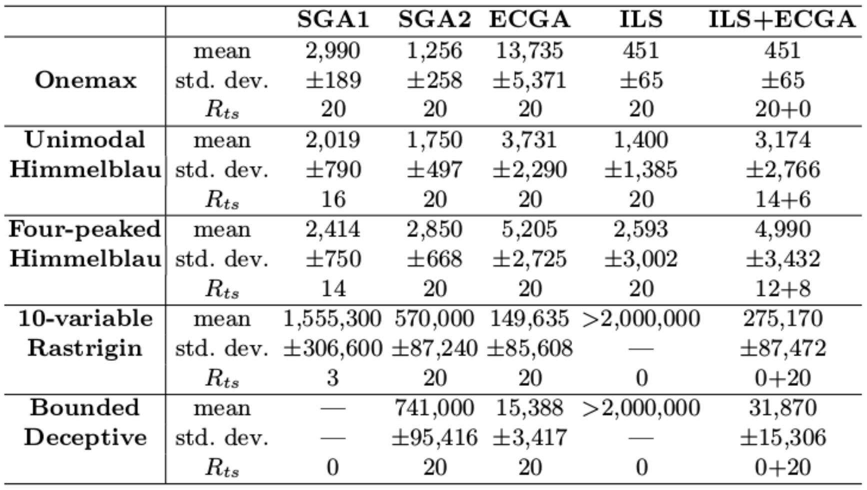
hBOA can be attributed to the use of the local searcher or the use of restricted tournament selection.

One property of the results here stand out though: using the parameter-less GA can reduce the scalability of the algorithm on certain problems. A similar behavior was seen when the parameter-less GA was combined with GOMEA (section 5.4). This happened for the lowest values of the growth parameter, which causes the creation of too many populations at the same time. Perhaps the worse results of hBOA can be addressed to a poorly chosen growth parameter for their harder test problems. This has a stark contrast with the parameter-less variants of GOMEA, which only suffer a constant overhead or even improve on the performance of the original algorithm. With this said, there is yet a lot of work that can be done towards designing a parameter-less population scheme for hBOA that better reuses previous computations as was done with P3 and LTGA.

Goldman independently implemented the hBOA algorithm and the parameter-less variant for a comparison with his P3 algorithm. At the moment of writing the paper has not been published yet, but the publicly available results show similar behavior for the parameter-less hBOA. On the Nearest Neighbor NK-Landscapes and Ising Spin Glasses the results showed a smaller constant overhead of using the parameter-less population scheme on hBOA. Because this thesis was produced before the official publication of the data, the results are not included here.

9.1.2 Parameter-less ECGA

The Extended Compact Genetic Algorithm is another probabilistic model building GA that was extended with the parameter-less GA. ECGA learns a partitioning of the problem



		SGA1	SGA2	ECGA	ILS	ILS+ECGA
Onemax	mean	2,990	1,256	13,735	451	451
	std. dev.	±189	±258	±5,371	±65	±65
	R_{ts}	20	20	20	20	20+0
Unimodal Himmelblau	mean	2,019	1,750	3,731	1,400	3,174
	std. dev.	±790	±497	±2,290	±1,385	±2,766
	R_{ts}	16	20	20	20	14+6
Four-peaked Himmelblau	mean	2,414	2,850	5,205	2,593	4,990
	std. dev.	±750	±668	±2,725	±3,002	±3,432
	R_{ts}	14	20	20	20	12+8
10-variable Rastrigin	mean	1,555,300	570,000	149,635	>2,000,000	275,170
	std. dev.	±306,600	±87,240	±85,608	—	±87,472
	R_{ts}	3	20	20	0	0+20
Bounded Deceptive	mean	—	741,000	15,388	>2,000,000	31,870
	std. dev.	—	±95,416	±3,417	—	±15,306
	R_{ts}	0	20	20	0	0+20

Figure 9.2: Table 1 from the paper on the Parameter-less ECGA

variables that efficiently compresses the population. When a building block occurs frequently in the population, then replacing its occurrences with a unique symbol is likely to compress the population. Hence the partitioning identifies the frequent building blocks in the population, so the model captures the non-overlapping linkage between variables. The authors recommend the use of an iterated local searcher (ILS) alongside the parameter-less ECGA to ensure maximum performance. ILS uses a simple first-improvement hill climber until it reaches a local optimum, and then applies increasingly larger mutations until the local optimum has been escaped by the hill climber. The two algorithms are given an approximately equal number of evaluations, which ensures the combination performs well on a broad class of problems.

Amongst the test problems were the deceptive trap function and the OneMax problem. The other three test problems were real-valued mathematical functions encoded in bit-strings. These problems contain peaks that serve as strongly attracting local optima, making them difficult to solve using only a local searcher. Particularly the Rastrigin function is a well-known performance test problem for both real-valued and bit-string optimizers. The authors did not include experiments of ECGA with bisected population sizes, so the actual overhead introduced by the parameter-less GA is still unknown.

9.2 Adaptive Population Sizing Schemes

Over the course of the last decade there have been several attempts at designing GAs using adaptive population sizing, as shown in the review of these algorithms by F. Lobo and C. F. Lima[13]. Adaptive population sizing schemes are a subset of population schemes that dynamically change the size of one or more populations between generations of the GA. None of the adaptive population sizing schemes introduced in the literature so far were parameter-less, with the only exception being the Parameter-less Genetic Algorithm. Often the population size is replaced by some parameter for sizing the population that is only slightly more robust. The first section discusses some of the recommendations made in the review for authors of these types of population schemes. The second section reviews our personal experience in trying to design a adaptive population sizing scheme that is also truly parameter-less and failing at that.

9.2.1 Recommendations

The authors of the aforementioned review made a few recommendations for other authors to follow when researching adaptive population sizing schemes. We consider these recommendations here and reflect our view on how these recommendations apply to our research.

1. Do not enforce a bound for the maximum population size.
2. Test on problems with known population sizing requirements.
3. Do fair comparisons.
4. Do scalability analysis.
5. Make life easier for users.

Recommendation 1 and 5 can be merged into one requirement of ensuring that the scheme is parameter-less. Any parameter-less algorithm cannot introduce a non-robust parameter based on population size, nor can it make life harder for the user by introducing other non-robust parameters. All population schemes discussed in this thesis fulfill this one requirement.

Recommendation 2 requires that the test problems represents a diverse set problem characteristics and have known population sizing requirements. Each of the test problems mentioned in the appendix has at least one known global optimum, which in turn allows for an analysis of the population sizing requirements using bisection. The test problems cover several problem characteristics such as deceptive functions, hierarchical overlapping functions, strongly overlapping functions, and a NP-Hard optimization problem. Figure 3.2 illustrates the diverseness of the population sizing requirements for each of the test problems. Given the choice, we recommend using the deceptive trap function, HIFF, and at least one NP-complete optimization problem. For example, HIFF and nearest neighbor NK-landscapes have slow or fast growing population sizing requirements, respectively. So any adaptive population sizing GA that reliably solves both problems is unlikely to overestimate or underestimate the growth of its populations towards specific problems.

Recommendation 3 requires that experiments are performed to analyze the relation between the run-time and the solution quality. Preferably an analysis is performed on both the solution quality over time and the time until reaching a certain solution quality. Both the number of fitness evaluations and run-time in seconds were analyzed in this thesis. Solution quality over time is biased towards local searchers and can only be reported for specific parameter configurations, hence this performance measure was not used in this thesis.

Recommendation 4 requires that experiments are performed to analyze the scalability of the algorithms for the set of test problems. In the context of applying a population scheme to an algorithm it is important to analyze whether the scalability of the algorithm changes. The scalability analysis was easier to perform for the parameter-less population schemes and showed that the incurred overhead was at most constant.

9.2.2 Realizing a scheme.

Designing a parameter-less population scheme for GOMEA based on adaptive population sizing turned out to be a lost cause without introducing extremely non-robust parameters. The existing literature on adaptive population sizing depends either on the Population Sizing theory or introduces replacement parameters. The behavior of GOMEA is different from the simple GA in terms of crossovers performed per generation and thus also the speed of convergence per generation. An adaptive scheme is constructed on basis of two rules: there are one or more populations and these may be dynamically resized. The first attempt at designing an adaptive population sizing scheme was to dynamically grow the population according to some function of the number of generations so far. Three functions were tested $f(g) = g$, $f(g) = g^2$, and $f(g) = 2^g$ before concluding this scheme was infeasible. The gap created by resizing the population was filled with random solutions and the model was learned from the previous population to avoid noise added by the random solutions. After inspection of the populations the reasons for non-termination were determined: either the population converged to a solution, or the population grew too fast to converge properly. The latter case occurred only for the exponential function. The first case occurred because the mixing of GOMEA depends on the donors being of similar fitness to the receiving solution. Hence if a set of strong donors is present in the population, then their building blocks are more likely to take over the population. This is particularly the case when merging a population of random solutions against the previously improved population. Techniques such as selection and niching can help solve this problem, but they would introduce new non-robust parameters into GOMEA. A similar result was obtained when trying to preserve solutions from smaller populations in the parameter-less GA.

An analogy to the loops of population schemes can be made as done in chapter 7 on the population pyramid and GOMEA. For example, the exponential scheme removes the population size parameter by wrapping the GOMEA algorithm into an outer loop over population sizes in the integer domain of powers of two. But the adaptive population sizing schemes attempt to modify the inner-loop of GOMEA by traversing over (multiple) (dynamic) populations per generation. However mechanisms such generational growth (and shrinking), or age-based mechanisms are dependent on the number of generations, which is itself a non-robust property dependent on the populations and the optimization problem. More specifically, the problem determines the average number of generations GOMEA would take to improve a solution to the global optimum or in other words its age. The population consists of multiple solutions of different ages, and experiments have shown that GOMEA does not work efficiently when learning linkage and mixing on solutions of varying ages. Hence the traditional approaches to population sizing are inevitably based on non-robust parameters and thus are not easily generalized to a parameter-less population scheme. As side note, P3 does not suffer as much from adaptively sizing the population pyramid because the old solutions are no longer operated on, whereas the other schemes continue operating on the same populations.

Chapter 10

Conclusions

When research on this thesis began there was one major question: how can P3 be a parameter-less version of LTGA, and still improve on the results of the optimally configured LTGA. The question was translated into the research goal of improving GOMEA and P3 by analyzing the core differences between the two algorithms. This chapter gives an evaluation of the research goal and some examples for future work.

10.1 Summary

The original Gene-pool Optimal Mixing Evolutionary Algorithm framework requires the user to set a population size that has a large impact on its performance. In this thesis we implemented and tested several parameter-less population schemes for removing this parameter from the framework. A more specific focus was taken on the Linkage Tree Genetic Algorithm, which is the best-performing instance of the GOMEA framework. We provide results for three known parameter-less population schemes compared against the original framework configured to optimality with bisection. Beneficial additions were made to the schemes by taking a closer look at the design choices behind the original schemes and the behavior of GOMEA. This thesis contains the first results of its kind and provides a foundation for future work on the subject.

First the exponential population scheme (EPS) was implemented, which simply restarts GOMEA on population convergence with the population size doubled. This behavior of trial-and-error is similar to that of a user figuring out a good population size. It turns out that evaluating the prematurely converged populations doubles the total number of fitness evaluations in the worst case. In fact, EPS performs on par with GOMEA configured through bisection on the complex problem instances. Because of its simplicity and efficiency it can directly substitute for GOMEA in any other experiment.

Secondly, the parameter-less GA was implemented with the selecto-recombinative GA replaced by GOMEA. It extends EPS by having multiple populations racing against each other concurrently. The trade-off between starting new populations before converging old populations turns out to only worsen the scalability of EPS. Other state-of-the-art GAs have adapted the parameter-less GA to varying degrees of success, but the reasons for their adaptation do not apply to GOMEA.

Last but not least is the Parameter-less Population Pyramid (P3), an evolutionary algorithm based on the works of LTGA before the framework of GOMEA was introduced. P3 introduces a novel population scheme to LTGA called the population pyramid that outperforms the original LTGA in overall fitness evaluations at the cost of a significantly increased run-time. An experiment was conducted to test how P3 behaves when some of its design choices are aligned with those made for GOMEA. This included changing the FOS ordering, removing the hill climber, and removing exhaustive donor searching. The resulting population pyramid (PT) is a parameter-less population scheme for GOMEA that reduces the overall number of fitness evaluations, but at least quadruples the run-time.

However, by extending PT with multiple insertion and binary tournament selection, the run-time and memory overhead are significantly reduced. Multiple insertion combines EPS with the population pyramid, which allows solutions from previously converged populations to contribute to the current population. Because previous populations are retained for donation, the growth rate is reduced from exponential to a linear function. Multiple insertion can be applied to P3 to greatly reduce the time spent on relearning models. Binary tournament selection introduces a robust tunable mechanism for reducing the size of the population pyramid without disrupting the performance significantly.

Some of the population schemes turned out to be less effective or even disruptive, such as the parameter-less GA. Regardless of that, the research goal has been met. Enhancing GOMEA with the population pyramid and multiple insertion led to an improved parameter-less version of GOMEA, in addition to providing a faster variant of P3.

10.2 Practical Recommendations

One question lingers on after summarizing the work: when is it appropriate to use which parameter-less population scheme. Tuning the original GOMEA framework by hand is never deemed necessary, so we give some practical recommendations for different scenarios in which a parameter-less GOMEA can be used.

First we consider the scenario in which the user simply needs an optimizer with a short run-time, or wants to evaluate a change to the GOMEA framework. In this case we recommend the exponential population scheme for its similarity to the original framework and its fast run-times. If an evaluation of the fitness function is a cheap operation, then using any other scheme is most likely premature optimization. Changing the underlying GOMEA implementation affects each population of EPS independently, which makes it easier to reason about the effect of a modification without the change propagating through the populations.

If every ounce of performance in fitness evaluations and run-time must be exploited, then the population pyramid is the way to go. Several choices can be made to tune the population pyramid in addition to the choices possible for GOMEA, such as exhaustive donor searching, hill climbing, multiple insertion, etc. The performance can be tuned to an extent in which the scheme outperforms a reasonably configured GOMEA. A valid scenario would be a competition in which a problem instance must be solved to optimality.

The other scenario to consider is one closer to the principles of black-box optimization. When a black-box optimizer is required for solving the problem, the fitness function tends to be an expensive operation aside from being horribly complex. In this case we recommend the population pyramid scheme with multiple insertion for its efficiency in fitness evaluations. The linear growth function gives the best compromise between fitness evaluations and the run-time cost of model learning.

What remains to be discussed is whether a hill climber should be used, and otherwise how the selection procedure of multiple insertion should be tuned. Whether a hill climber is useful depends on the cost of evaluating a bit-flip and the problem characteristics. Use the hill climber when the effect of a bit-flip can be computed incrementally, which is possible on many combinatorial optimization problems. Otherwise, start by using the hill climber as they are applicable to a broad range of problems, and remove it afterwards if it proves to be inefficient on the problem. In either case the selection procedure should not be necessary when using the hill climber. When the algorithm responds poorly to the local optima provided by the hill climber, or fitness evaluations (even bit-flips) are prohibitively expensive, the use of a hill climber is discouraged. An example of the former are the NK-S1 problem instances known for their rugged fitness landscape. The GOM operator performs bit-flips itself, which are often wasted when the hill climber has already exhausted the local neighborhood of bit-flips. In this scenario the setting of the selection procedure depends entirely on the memory constraints set by the user. A non-aggressive setting ($c \leq 0.5$) is preferred to maximally reuse the solutions (and thus fitness evaluations) done so far.

See appendix B for an unspoiled plot of the recommended algorithms.

10.3 Future Work

Modern CPUs allow for embarrassingly parallel runs of GOMEA on the same problem in order to speed up computations. Because parameter-less algorithms do not terminate due to premature convergence, each run will expand its search until the termination criterion is met. In practice this reduces the variance in run-time of GOMEA because every run is terminated when the requested solution quality has been hit. However, the state is not shared between runs when run in parallel. Sharing solutions between runs could lead to higher efficiency, especially if they are mixed with populations of similar fitness.

Another possible method of threading would be to have a shared population pyramid between multiple runs of P3. Multiple insertion has shown that solutions can be inserted in batches without losing efficiency, which in turn can be divided over multiple threads. Additionally, the crossover procedure can be evaluated concurrently for each layer of the pyramid. There would be more emphasis on less redundant computations and less wasted memory by focusing the search on a single growing population pyramid. Concurrently building the linkage trees without blocking other threads would be the largest challenge in achieving true parallelization.

Because the lowest pyramid layer of P3 is constantly changing, the linkage tree there is constantly being rebuild. Without the hill climber, the linkage information contained in the lowest pyramid layer of P3 is mostly random. Instead of storing those random solutions, the bottom layer can be replaced by a random linkage tree. Less memory is used because an entire population is no longer stored in memory, and learning a random linkage tree is trivial in run-time. When using an aggressive setting for the selection procedure, this assumption of random linkage may no longer hold. Whether this bias towards fit random solutions is helpful depends on the problem at hand.

No adaptive scheme has been given yet for dynamically changing the c parameter of the selection procedure while P3 is running. A simple scheme for scaling c depending on the distance to some memory limit may be sufficient for removing this parameter. There is also more work to be done on testing other selection procedures, or merging small layers in the top of the population pyramid. These procedures help reshape the population pyramid over time, which may lead to better performance as well.

Appendix A

Test Problems

The test problems used throughout this paper are listed in this appendix. The optimization problems are encoded into solutions of bit strings of length l. Each section lists the problem definition and explains the problem characteristic for which it was chosen.

A.1 Deceptive Trap Function

The deceptive trap function is a benchmark problem designed to be difficult to solve for local search heuristics and GAs[4]. A function is deceptive if the fitness landscape it induces misleads heuristics towards a non-optimal configuration. Equation A.1 shows the deceptive trap function for k bits, where t out of k bits are set to one. The benchmark problem considered here is the concatenation of $\frac{l}{k}$ deceptive trap functions into a bit string. Hence the fitness function is the sum of the deceptive trap functions.

$$DeceptiveTrap(t) = \begin{cases} k - 1 - t & \text{if } t < k \\ k & \text{if } t = k \end{cases} \tag{A.1}$$

Each trap is optimal when all bits are set to one, but every other configuration leads to the strong local optimum of all bits set to zero. Escaping this local optimum requires up to k non-improving bit flips, something that many local searchers are not capable of. A trap set to all zeros or all ones would be a bad or good building block, respectively. A population containing every good building block at least once is required in order to effectively the solve deceptive trap function with a GA. This problem tests the ability of a GA to preserve and combine good building blocks. The linkage between the bits in a trap functions is easily detected by entropy-based linkage measures because the trap functions do not overlap at all.

A.2 Hierarchical If and Only If

Hierarchical If and only If (HIFF) is a benchmark problem based on a hierarchy of overlapping trap functions[7]. The hierarchy is a complete binary tree defined over the problem variables. Each leaf is a single problem variable, and each node is the union of all its two child nodes. Let t be the number of bits set to one for a node of size s. The trap function is defined by equation A.2. A node contributes to the total fitness *if and only if* every bit is set to the same value. The fitness function is simply the sum of the trap function for every node in the tree. The global optimum is either the solution of all zeros or all ones. HIFF is only defined for problems sizes that are powers of two.

$$HIFF(s,t) = \begin{cases} s & \text{if } t = s \vee t = 0 \\ 0 & \text{otherwise} \end{cases} \tag{A.2}$$

The bits represented by a node form a building block. Local searchers face the problem of improving towards mismatching optimal building blocks, meaning the parent node of the mismatching blocks cannot contribute to the fitness. Overcoming this obstacle requires at least s bit flips, where s is the size in bits of the child nodes. The likelihood of a local searcher producing no mismatching building blocks becomes less likely as the tree grows. HIFF tests the ability of a GA to recombine building blocks without converging towards mismatching building blocks, in other words preserving diversity in the population. Linkage learning allows the algorithm to detect the trap bits and use these when copying solutions. The Linkage Tree learned by LTGA can perfectly match each node one-to-one with a crossover mask given a population containing multiples of each building block.

A.3 Ising Spin Glass

Ising Spin Glasses are a set of optimization problems [1] based on the Ising Model, a mathematical model for magnetization. The Ising Model represents (anti-)ferromagnetic interactions between neighboring vertices in a lattice graph. The Hamiltonian (equation A.3) measures the energy of the system when vertices of positive- or negative sign ($x_i \in \{-1, 1\}$) interact with each other.

$$IsingModel(x) = -\sum_{c_{ij} \in C} x_i c_{ij} x_j \tag{A.3}$$

Ising Spin Glasses are specific models where the interactions c_{ij} between the vertices are drawn from a random distribution. The goal is finding a ground state, which is a state in which the system has minimal energy. Spin Glasses are represented as two- or three-dimensional toroidal grids, for example in two-dimensions a flattened torus. The problem instances used in this thesis were taken from the public repository hosting the source of the P3 algorithm. As such they were used in the latest publication on P3 by B. Goldman and W. Punch [21]. The spin glasses are denoted by $2D \pm J$, which indicates the use of a two-dimensional toroidal grid and interactions of either 1 or -1 with probability p $(= 0.5)$. Ground states for these instances can be found in polynomial time[3].

A.4 Nearest Neighbor NK Landscapes

NK landscapes are a mathematical model of an NP-complete optimization problem with a tunable parameter k for the ruggedness of the fitness landscape[5]. The fitness contribution of a gene is a real function f_i over itself and k other genes. Hence the sub-function depends on $k+1$ bits, according to the original definition by Goldstein. The goal is to maximize the sum of all sub-functions, as defined in equation A.4. In a Nearest Neighbor NK landscape each gene depends on the k subsequent genes in the bit-string.

$$NearestNeighorNK(x) = \sum_{i=0}^{l} f_i(x_i, \ldots, x_{(i+k)\%l}) \tag{A.4}$$

Nearest Neighbor NK landscapes are solvable in polynomial time using a dynamic program[9]. Sub functions f_i map each configuration of $k + 1$ bits to a value drawn from the uniform random distribution in the range of $[0, 1)$. Depending on the definition there may be a wrapping neighborhood in which genes at the end of the solution depend on genes at the start of the solution. The NK-S1 instances have $k = 4$ and a non-wrapping neighborhood.

A.5 MaxCut

MaxCut is the NP-HARD optimization problem of finding a maximum cut on a given graph $G(V, E)$ [2]. A cut of graph G partitions the vertices V into two disjoint subsets X and Y. The cut-set is the number of edges with an end-point in both X and Y. A maximum cut is the largest possible cut-set of the input graph $G(V, E)$. The optimization variant used as a test problem takes as input a complete graph with integer-weighted edges. There is a bijective mapping between bit-strings and partitions of two subsets, hence bit-strings are a possible encoding of solutions for the MaxCut problem. The instances used in this thesis were last used in this paper[19], and they were solved to optimality on the BIQMAC server.

$$MaxCut(G(V,E),X,Y) = \sum_{\{(v_i,v_j)\in E | v_i \in X \wedge v_j \in Y\}} c(v_i,v_j) \tag{A.5}$$

Appendix B

Practical Recommendations

This appendix contains the plots of the number of fitness evaluations and the run-times of the algorithms recommended in the practical recommendations section of the conclusions (chapter 10).

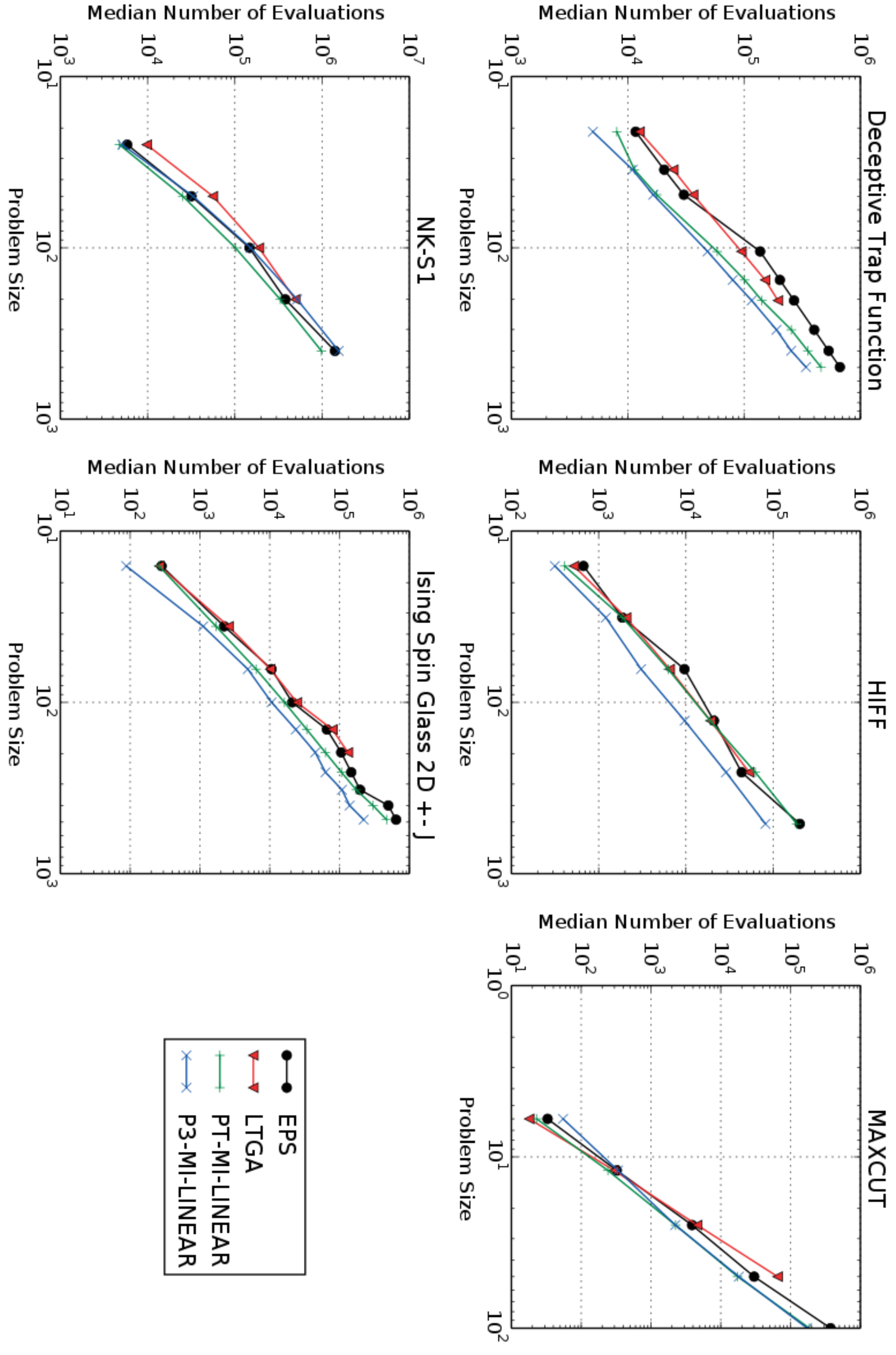


Figure B.1: Comparison of the median number of fitness evaluations between the parameter-less population schemes recommended in the practical recommendations. Shows the Exponential Population Scheme (EPS), the Population Pyramid (PT) with Multiple Insertion, the P3 with Multiple Insertion, and LTGA configured through bisection.

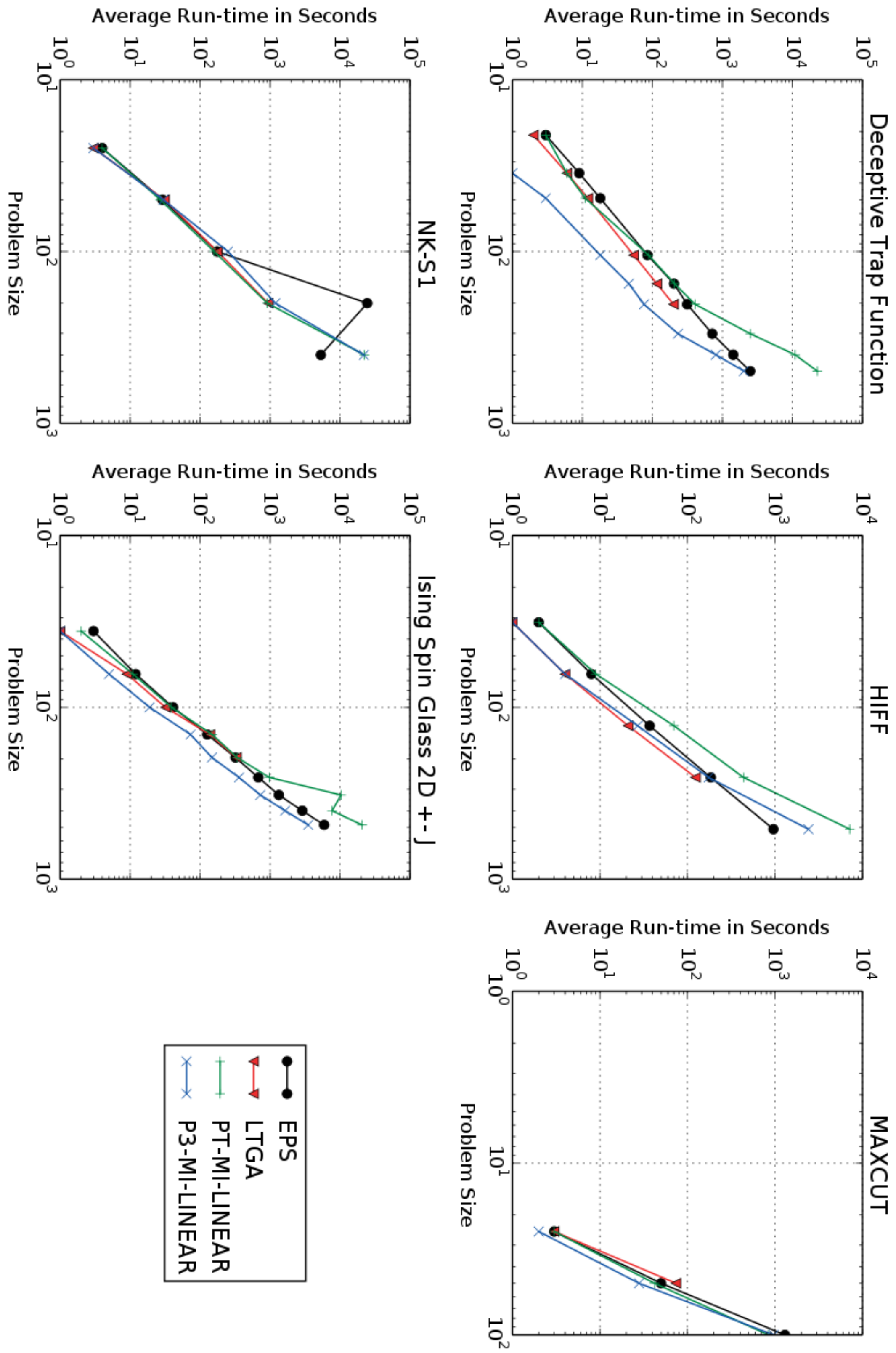


Figure B.2: Comparison of the average run-times between the parameter-less population schemes recommended in the practical recommendations. Shows the Exponential Population Scheme (EPS), the Population Pyramid (PT) with Multiple Insertion, the P3 with Multiple Insertion, and LTGA configured through bisection.

Appendix C

Experimental Results

This appendix contains enlarged versions of the plots found in the experimental studies in chapters 6 and 8. The table below is a copy of table 6.1 and explains the labels found in the plots.

	PT	PTD	PTLS	PTLSD	PThree	P3
Exhaustive Donor Searching	No	Yes	No	Yes	Yes	Yes
First-Improvement Hill Climber	No	No	Yes	Yes	Yes	Yes
Smallest-first Cluster Order	No	No	No	No	Yes	Yes

Table C.1: The names of the P3 variants and which components they include.

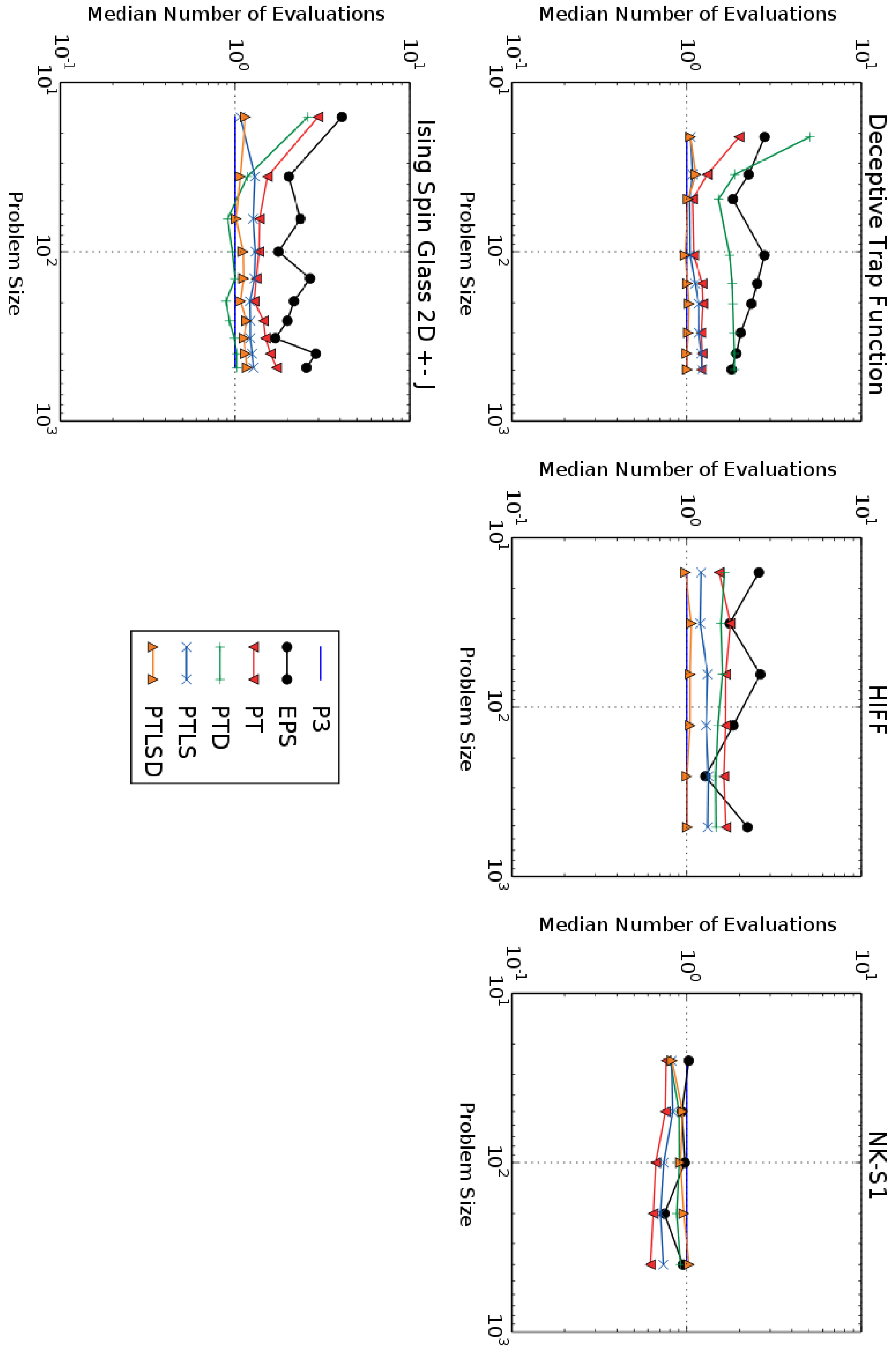


Figure C.1: Magnified version of figure 6.1. Shows a comparison of the median number of fitness evaluations for the five variants of P3. See the start of this appendix for a table explaining the labels.

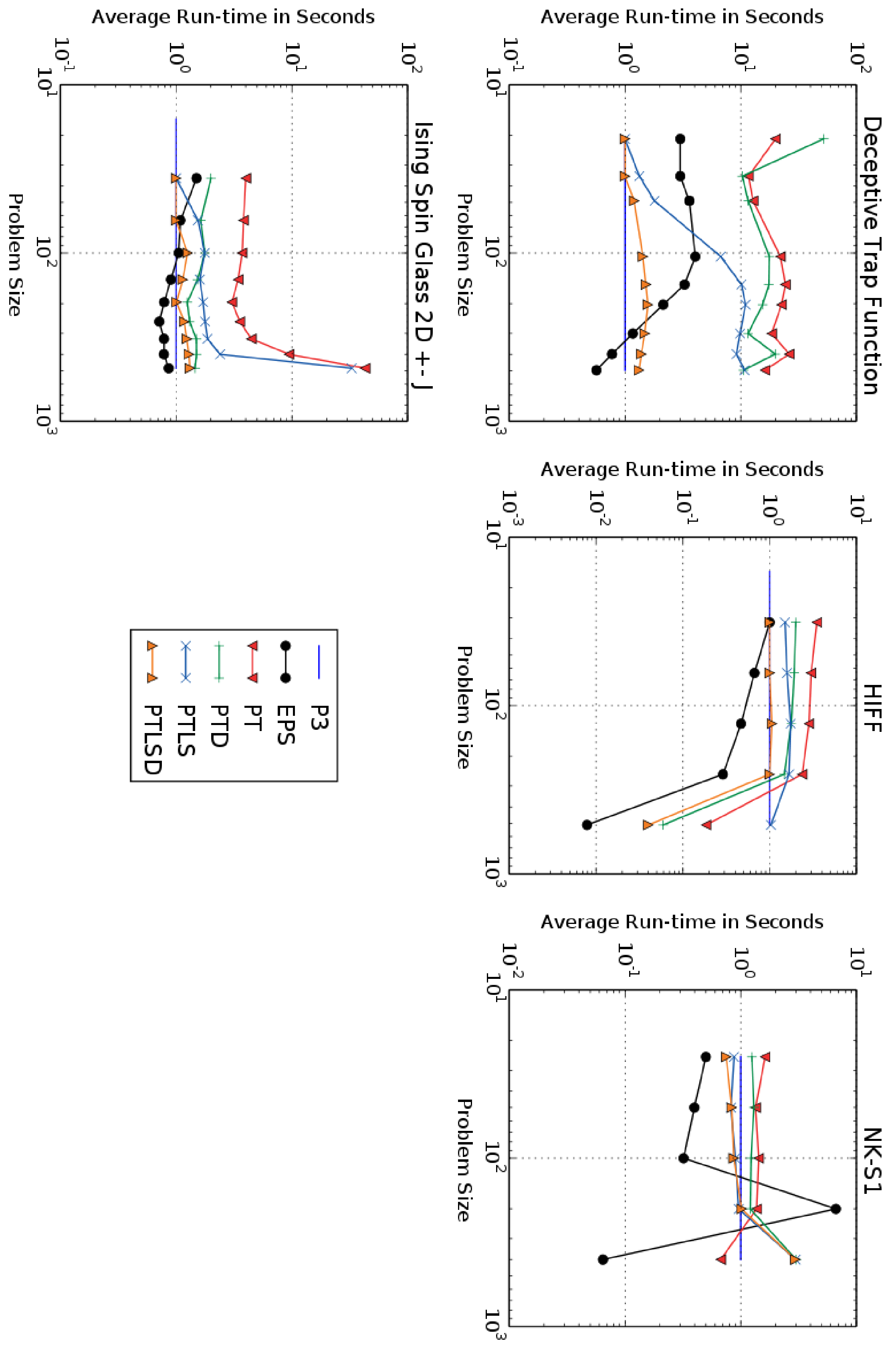


Figure C.2: Magnified version of figure 6.2. Shows a comparison of the average run-times for the five variants of P3. See the start of this appendix for a table explaining the labels.

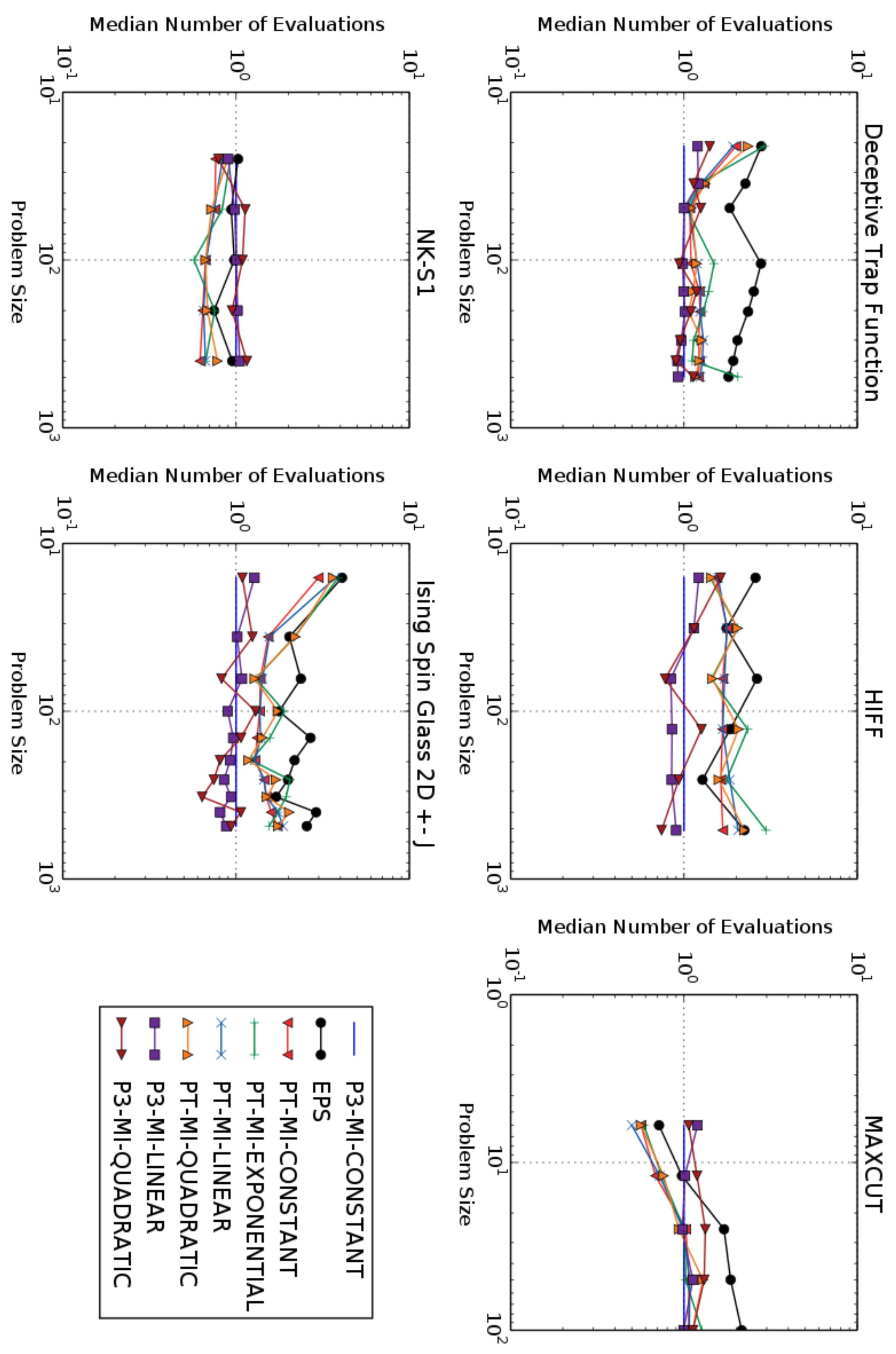


Figure C.3: Magnified version of figure 8.1. Shows a comparison of the median number of fitness evaluations for the exponential population scheme (EPS), the population pyramid scheme (PT) with multiple insertion, and the parameter-less population pyramid (P3) with multiple insertion. The labels are formatted as such: {Scheme}-MI-{Growth}, where Scheme is the population scheme and Growth is the growth function used for multiple insertion.

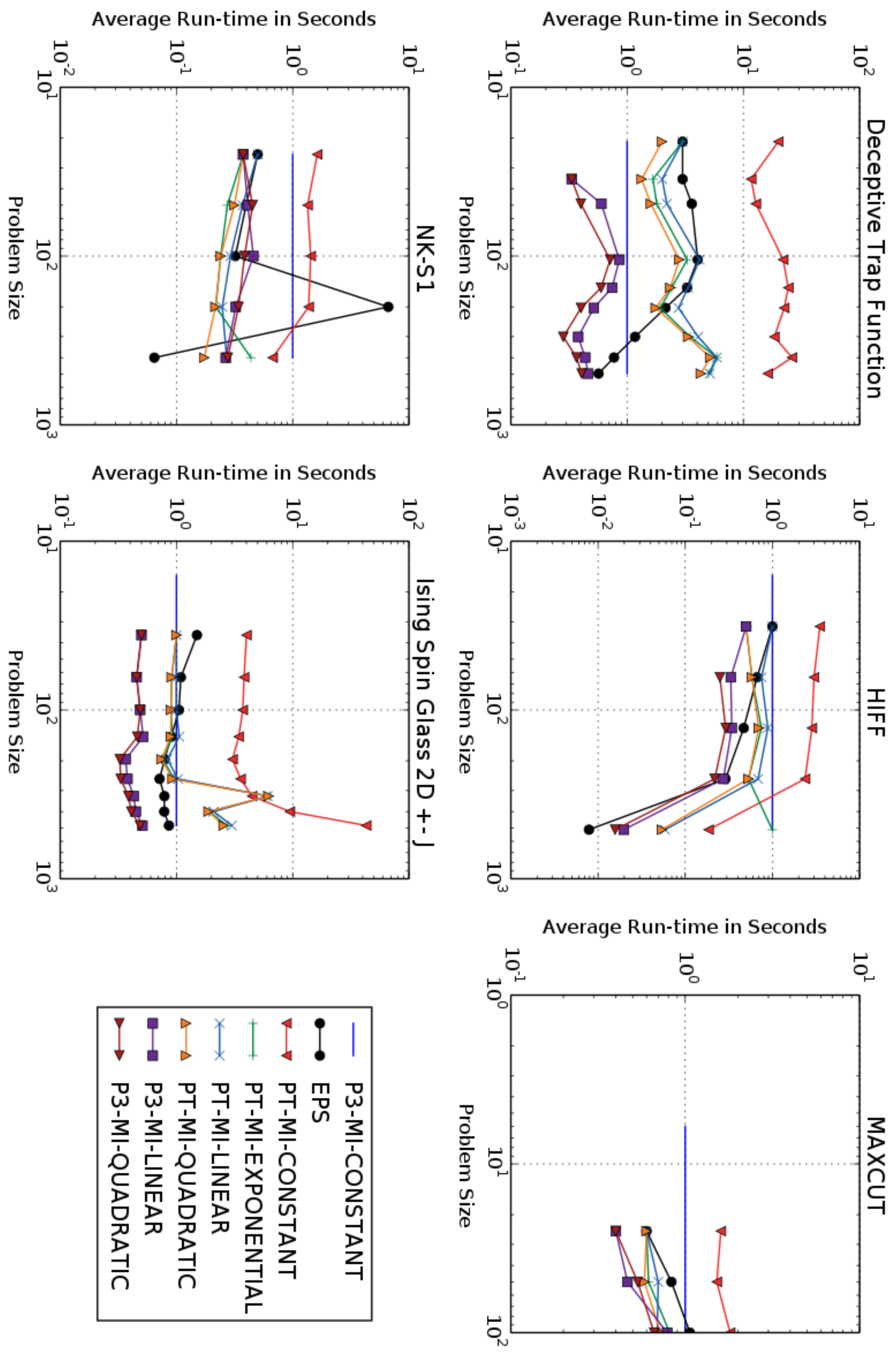


Figure C.4: Magnified version of figure 8.1. Shows a comparison of the average run-times of the exponential population scheme (EPS), the population pyramid scheme (PT) with multiple insertion, and the parameter-less population pyramid (P3) with multiple insertion. The labels are formatted as such: {Scheme}-MI-{Growth}, where Scheme is the population scheme and Growth is the growth function used for multiple insertion.

Bibliography

[1] D. Sherrington and S. Kirkpatrick, "Solvable model of a spin-glass," *Physical review letters*, vol. 35, no. 26, p. 1792, 1975.

[2] M. R. Garey and D. S. Johnson, "Computers and intractability: a guide to the theory of NP-completeness," *San Francisco, LA: Freeman*, 1979.

[3] F. Barahona, "On the computational complexity of ising spin glass models," *Journal of Physics A: Mathematical and General*, vol. 15, no. 10, p. 3241, 1982.

[4] D. E. Goldberg, "Simple genetic algorithms and the minimal, deceptive problem," *Genetic algorithms and simulated annealing*, vol. 74, p. 88, 1987.

[5] S. Kauffman and S. Levin, "Towards a general theory of adaptive walks on rugged landscapes," *Journal of theoretical Biology*, vol. 128, no. 1, pp. 11–45, 1987.

[6] B. D. Jovanovic and P. S. Levy, "A look at the rule of three," *The American Statistician*, vol. 51, no. 2, pp. 137–139, 1997.

[7] R. A. Watson, G. S. Hornby, and J. B. Pollack, "Modeling building-block interdependency," in *Parallel Problem Solving from Nature—PPSN V*, Springer, 1998, pp. 97–106.

[8] G. R. Harik and F. G. Lobo, "A parameter-less genetic algorithm," in *Proceedings of the Genetic and Evolutionary Computation Conference*, ser. GECCO '99, 1999, pp. 258–267.

[9] A. H. Wright, R. K. Thompson, and J. Zhang, "The computational complexity of nk fitness functions," *Evolutionary Computation, IEEE Transactions on*, vol. 4, no. 4, pp. 373–379, 2000.

[10] M. Pelikan and T.-K. Lin, "Parameter-less hierarchical BOA," in *Genetic and Evolutionary Computation – GECCO 2004*, ser. Lecture Notes in Computer Science, vol. 3103, Springer Berlin Heidelberg, 2004, pp. 24–35.

[11] G. S. Hornby, "ALPS: the age-layered population structure for reducing the problem of premature convergence," in *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '06, New York, NY, USA: ACM, 2006, pp. 815–822.

[12] I. Gronau and S. Moran, "Optimal implementations of UPGMA and other common clustering algorithms," *Inf. Process. Lett.*, vol. 104, no. 6, pp. 205–210, Dec. 2007.

[13] F. G. Lobo and C. F. Lima, "Adaptive population sizing schemes in genetic algorithms," in *Parameter Setting in Evolutionary Algorithms*, ser. Studies in Computational Intelligence, vol. 54, Springer Berlin Heidelberg, 2007, pp. 185–204.

[14] D. Thierens, "The linkage tree genetic algorithm," in *Parallel Problem Solving from Nature, PPSN XI*, ser. Lecture Notes in Computer Science, vol. 6238, Springer Berlin Heidelberg, 2010, pp. 264–273.

[15] D. Thierens and P. A. Bosman, “Optimal mixing evolutionary algorithms,” in *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, ACM, 2011, pp. 617–624.

[16] P. A. Bosman and D. Thierens, “Linkage neighbors, optimal mixing and forced improvements in genetic algorithms,” in *Proceedings of the 14th annual conference on Genetic and evolutionary computation*, ACM, 2012, pp. 585–592.

[17] ——, “On measures to build linkage trees in LTGA,” in *Parallel Problem Solving from Nature-PPSN XII*, Springer, 2012, pp. 276–285.

[18] B. W. Goldman and D. R. Tauritz, “Linkage tree genetic algorithms: variants and analysis,” in *Proceedings of the 14th annual conference on Genetic and evolutionary computation*, ACM, 2012, pp. 625–632.

[19] P. A. Bosman and D. Thierens, “More concise and robust linkage learning by filtering and combining linkage hierarchies,” in *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO ’13, New York, NY, USA: ACM, 2013, pp. 359–366.

[20] K. L. Sadowski, P. A. Bosman, and D. Thierens, “On the usefulness of linkage processing for solving max-sat,” in *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, ACM, 2013, pp. 853–860.

[21] B. W. Goldman and W. F. Punch, “Parameter-less population pyramid,” in *Proceedings of the 2014 Conference on Genetic and Evolutionary Computation*, ser. GECCO ’14, ACM, 2014, pp. 785–792.