Universiteit Leiden
The Netherlands

Opleiding Informatica

Empirical performance evaluation of the linkage tree genetic algorithm

Guido Wassenaar

Supervisors:
Thomas Bäck & Furong Ye

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

08/08/2020

Abstract

The linkage tree genetic algorithm (LTGA) is a variation of a genetic algorithm which seeks to employ linkage information between variables to prevent disruption of good subsolutions with crossover. This thesis compares the performance of this algorithm to twelve other algorithms on certain pseudo-boolean problems. Empirical evaluation using the software IOHprofiler shows that the LTGA can perform very good on higher dimensional problems, but is easily outperformed by others when the dimension is low. Furthermore, a minimum population size for the LTGA of $\sim$16 is suggested as the population sizes lower than 16 proved little use. Concluded lastly is that certain problem instances tested, Low Autocorrelation Binary Sequences (LABS), two instances of the Ising model (Torus and Triangular), and the Maximum Independent Vertex Set benefit from linkage information as the LTGA performed very good compared to other algorithms on these.

Contents

1 Introduction **1**
- 1.1 Thesis overview . . . 1

2 Definitions **1**
- 2.1 The problem domain . . . 1
- 2.2 Genetic algorithms . . . 2
  - 2.2.1 Fitness and selection . . . 3
  - 2.2.2 The algorithm . . . 3
- 2.3 The Linkage Tree Genetic Algorithm . . . 3
  - 2.3.1 Linkage and dependency . . . 4
  - 2.3.2 Clustering . . . 4
  - 2.3.3 Crossover . . . 5
  - 2.3.4 The algorithm . . . 5

3 Related Work **5**

4 IOHprofiler **6**

5 Experiments **6**
- 5.1 Setup of the LTGA . . . 6
  - 5.1.1 Population . . . 6
  - 5.1.2 Problem dimension . . . 7
  - 5.1.3 Implementation . . . 7
  - 5.1.4 Budget . . . 7
- 5.2 Problem instances . . . 7
- 5.3 Compared algorithms . . . 8

6 Results **9**
- 6.1 LTGA vs Other algorithms . . . 10
- 6.2 Population size . . . 14

7 Conclusions **16**

8 Further research **17**

References **18**

1 Introduction

Many problems in science can be translated to *pseudo-boolean problems*. These are functions which accept bit-strings – sequences of 0 and 1 with length n– which output a real value representing the fitness value of a solution to the problem.
There are many algorithms proposed to search the gigantic space of these problems for the optimal solution. One of those algorithms is the Linkage Tree Genetic Algorithm proposed by D. Thierens in 2010 [1]. This variation of a genetic algorithm seeks to employ linkage information between solution candidates to improve the crossover operation. This will be explained in more detail in section 2.3.
There are many other algorithms with the same goal of finding the optimal solution $f(x)$ for pseudo-boolean problems. However it is not yet known how the LTGA compares to these. In this thesis the following research question is addressed: *How does a linkage learning genetic algorithm perform compared to other algorithms on pseudo-boolean problems?*. Answering this question would provide more insight in the applications of the algorithm.
To answer this question, we are using the software suite IOHprofiler, which allows us to compare any algorithm to a set of other algorithms already tested in the suite. It also allows for visualizations of the generated data.

1.1 Thesis overview

Section 2 seeks to provide more background information on the research conducted. In section 3 discusses related work on this topic. Section 4 gives more insight on the software used to answer our research question. Section 5 shows the setup of the experiments, including the setup of the LTGA, the compared algorithms and tested problem instances. The results are also presented in this section. Section 7 concludes the thesis, and section 8 discusses future research.
This bachelor thesis is supervised by Prof.dr. T.H.W. Bäck and F. Ye MSc at the Leiden Institute for Advanced Compute Science (LIACS).

2 Definitions

The Linkage Tree Genetic Algorithm (LTGA) is a genetic algorithm. This section will provide information on the problem domain and describes genetic algorithms in general, where they fall short and why the LTGA might be a solution to these limitations.

2.1 The problem domain

In this thesis we focus on *pseudo-boolean problems* [2], which are functions of the form

$$f : \{ 0, 1 \}^n \to \mathbb{R}$$

where n is a nonnegative integer. In words, we are looking at functions that take a bit-string as an input, and return a real number as output. n, the number of bits in the string, is called the *dimension* of the problem. A common, although trivial, but theoretically well understood test

problem is the *OneMax* function for example, defined as:

$$OneMax : \{0,1\}^n \to [0..n], x \mapsto \sum_{i=1}^{n} x_i$$

OneMax counts the number of ones in a bit-string. For example, consider the bit-strings $A = 00110010$, $B = 10000001$ and $C = 11111111$. Using OneMax on dimension $n = 8$ results in $OneMax(A) = 3$, $OneMax(B) = 2$ and $OneMax(C) = 8$. OneMax is the best-studied benchmark problem [3]. We can easily see that the optimal solution is the bit-string where all bits equal to 1, like the bit-string C. The algorithms tested do not know this however. Furthermore, there are many pseudo-boolean problems that are not this simple with huge search spaces. Genetic algorithms can be used to search the domain for good solutions.

2.2 Genetic algorithms

Genetic algorithms (GAs) are a type of evolutionary algorithms which are inpsired by Charles Darwin's theory on evolution in nature. It is based around the idea of a population, in which solution candidates (or chromosomes) reproduce offspring. Generally speaking there is a form of selection, based around the fitness of the candidate, to pick which candidates create the offspring. GAs also include a method of crossover which combines the genes of the parents, of which an example can be seen in 1. GAs can also include random mutation of offspring, which randomly modifies genes in offspring. This can be seen in figure 2.
GAs are broadly used in optimization problems, as they can provide good solutions fairly quickly despite large search spaces. They can be used in many different sectors, a few examples being engineering [4], planning problems like time-tables [5] and efficient fitting of objects [6].
Genetic algorithms, like any algorithm [7], are not the solution to every problem. While able to find global optima, GAs tend to get attracted to local optima [8].
Another limitation is that genetic algorithms do nothing with linkage - they never learn which variables form important partial solutions. If genes are related in some way, genetic algorithms can throw this potentially useful information away with crossover or mutation. It could be useful to protect these links in some way to yield better results, which is why the Linkage Tree Genetic Algorithm was proposed.

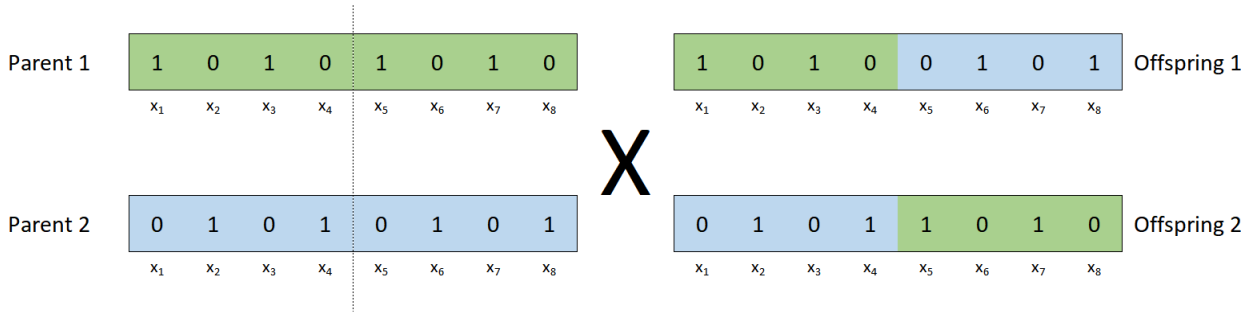

Figure 1: This image shows an example of single-point crossover, where the crossover point is chosen to be in the middle.

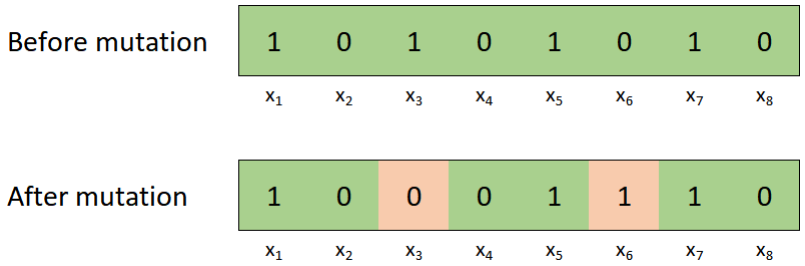

Figure 2: In this image genes x_3 and x_6 in the offspring are randomly mutated.

2.2.1 Fitness and selection

Whether a candidate is a good one for selection is based on its *fitness*. This is a measure that scores the quality of a solution candidate. In this thesis we use 23 pseudo-Boolean function as fitness function and maximize the fitness.
There are many creative ways to select parents for recombination. Generally speaking, parents that have a good fitness score are more likely to be selected.

2.2.2 The algorithm

The structure of a genetic algorithm is of the following form, based on [9]:

Algorithm 1: Genetic Algorithm

1. Initialize a randomly generated population of N n-bit chromosomes
2. Calculate fitness for every chromosome
3a. Select pair of parent chromosomes
3b. Perform crossover on selected parents with chance p_c. If no crossover is done, the children will be copies of the parents
3c. Mutate the two offspring on each gene with probability p_N per bit (i.i.d.) and place them in the new population
3d. Go to 3a until the new population contains N solutions
4. Replace old population with new population
5. If no exit condition is met, go to 2.
6. Return best found fitness

2.3 The Linkage Tree Genetic Algorithm

The Linkage Tree Genetic Algorithm [1] (LTGA) was introduced in 2010. The aim of this algorithm is to learn which variables form important subsolutions and protect these from being seperated from eachother by crossover. Mutation is absent from the LTGA to prevent subsolutions from being modified.
The LTGA uses hierarchical clustering to build a linkage tree bottom up, where every node is a cluster of problem variables. The hierarchical clustering algorithm uses a distance measure D to perform the clustering. Next, two solution candidates in the population of size N are chosen as parents. Then the linkage tree is traversed starting from the root so the traversal begins at the least dependent cluster. The clusters in the linkage tree are used as crossover masks to generate a new

offspring pair, which is described in section 2.3.3. The fitness of the parent pair and the offspring pair are compared, and when one of the children is better than both parents, the offspring pair replaces the parents. The traversal continues with this most fit pair. Once the traversal is complete, the single best solution found during the traversal is copied to the new generation. This process is repeated until a new generation of size N is built. Then the process starts again.

2.3.1 Linkage and dependency

As mentioned earlier, the LTGA uses linkage information to determine which problem variables are used in crossover. Thierens describes linkage learning as learning which variables should be treated as a dependent set of variables. The degree of dependency between variables is measured using distance measure D which is described in section 2.3.2. Higher distance between clusters means a lower dependency, and therefore also lower *linkage*.

2.3.2 Clustering

The clustering algorithm used in the LTGA uses a distance measure D to calculate the distance between clusters. As we wanted to stick to Thierens' implementation of the LTGA in [1], the same distance measure is used, defined as

$$D(X_1, X_2) = 2 - \frac{H(X_1) + H(X_2)}{H(X_1, X_2)}$$

where H calculates the (joint) entropy. This is a normalized distance measure based on mutual information. Entropy H is defined as:

$$H(X_k) = -\sum_i p_i(X_k) \log p_i(X_k)$$

where X_k is a discrete, random variable with probability mass function $p(X_k)$.
The implementation incorporates a stack. Initially all n single variable clusters are pushed onto the stack. Then the clustering algorithm iteratively joins the two nearest clusters, and also pushes these onto the stack. This results in a stack filled with $2n - 1$ clusters, where the top of the stack is the least dependent cluster. The traversal is done by simply popping the stack.
The algorithm to generate the hierarchical cluster tree of the problem variables is defined as follows in pseudocode [1]:

Algorithm 2: Hierarchical cluster tree

1. Compute the proximity matrix using distance metric D.
2. Assign each variable to a single cluster.
3. Repeat until one cluster is left:
4. Join two nearest clusters c_i and c_j into c_{ij}.
5. Remove c_i and c_j from the proximity matrix.
6. Compute distance between c_{ij} and all clusters.
7. Add cluster c_{ij} to the proximity matrix.

2.3.3 Crossover

Crossover in the LTGA is done with the cluster's crossover mask. An example of this crossover can be seen in Figure 3. In this figure, the top two parent chromosomes are recombined. The bits at the places corresponding to the genes in the cluster (x_2, x_5, x_7) are swapped between the offspring chromosomes.

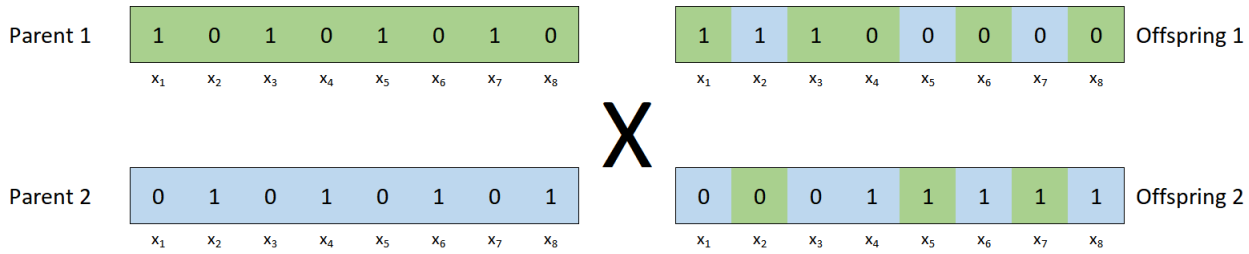

Figure 3: Example of crossover with a crossover mask from cluster (x_2, x_5, x_7).

2.3.4 The algorithm

The LTGA is described in pseudocode [1] as follows:

Algorithm 3: Linkage Tree Genetic Algorithm

```
1. Create initial population of size N.
2. Repeat until stop criteria met:
3.     Build the linkage tree using Algorithm 2.
4.     Do for N solution pairs:
5.         Traverse one step in the linkage tree.
6.         Set crossover mask to the current clustering.
7.         Cross solution pair using the crossover mask.
8.         When one offspring is better than both parents:
9.             Replace parent pair with the offspring pair.
10.        If the tree is fully traversed:
11.            Copy best solution to next population.
12.            Else, go to step 5.
```

The stop criteria that are used are:

- All individuals are identical
- The budget of function evaluations has run out.

3 Related Work

This thesis evaluates the perfomance of the LTGA proposed in *The linkage tree genetic algorithm* [1] by Dirk Thierens. Since then the LTGA has been used in various other works. This section discusses some other articles that relate to this thesis.

For example in [10] the usefulness of linkage learning is considered to solve MAX-SAT. A GOMEA (Gene-pool Optimal Mixing evolutionary algorithm) is a type of evolutionary algorithm that learns interdependencies between variables. Like in this thesis, the GOMEA instance used is the LTGA. In this paper, the performance is also compared to non-GOMEA based algorithms with positive results for the LTGA. It also shows that the performance of the LTGA in both a black-box and white-box setting.
Furthermore, in [11], enhancement techniques are proposed based on two GOMEAs, the LTGA and the MLNGA (Multi-scale Linkage Neighbors Genetic Algorithm). Combining their strengths, a new model is proposed; the Linkage Trees and Neighbors Genetic Algorithm. This new model is then compared to the two algorithms it is based upon with comparable or better results.
In this [12] paper, linkage learning is explored as a tool of solving multidimensional knapsack problems. Referring to the *No free lunch* theorom, linkage learning can not be the perfect solution for every problem. To evaluate linkage learning for the problem, the Linkage Learning Tree Algorithm was used. The article concludes that the Multidimensional Knapsack Problem does not have evident linkages.

4 IOHprofiler

The experiments that were performed were done using a specialized piece of software called IOHprofiler [13]. This software can be used to analyze and compare iterative optimization heuristics. IOHprofiler consists of two parts, *IOHexperimenter* and *IOHanalyzer*. IOHexperimenter can be used to generate data on the performance of a given algorithm written in C++, Python or R. This data can then be imported in IOHanalyzer, which can visualize the data in various interesting ways. The most used features from IOHanalyzer in this thesis are the *Expected runtime* and *Probability density function* features, which provided useful insights in the runtime and progression of the algorithm. IOHanalyzer is also able to load datasets from various repositories.
IOHanalyzer can be accessed at http://iohprofiler.liacs.nl/

5 Experiments

To answer our research question, a number of experiments were performed. This section describes how these were set up, and the results of these experiments.

5.1 Setup of the LTGA

5.1.1 Population

Multiple population sizes were tested in the LTGA. The population size will be referred to as λ. The sizes that were tested are of size $\lambda = 2^n, 2 \leq n \leq 8$, resulting in population sizes of 4 to 256. In the experiments, population sizes are compared to each other, as well as various other algorithms. In the results the LTGA instances with different population sizes are referred to as `ltga_<populationsize>`, ltga_128 for example for the LTGA with a population size of 128.
The population is initialized randomly on every run.

5.1.2 Problem dimension

The dimensions that were tested are 16, 64 and 100. These dimensions were chosen as these are also tested on the other algorithms in the PBO-suite, so diverting from these dimensions would give issues with the comparison. The other algorithms are also run on a dimension of 625, but this was not done with the LTGA as the lower dimensions already gave a good idea of the performance of the algorithm.

5.1.3 Implementation

The LTGA was implemented in Python. The implementation used in this thesis was based around the source code which can be found at [14], and modified to be used with IOHexperimenter. Furthermore, the implementation was checked for errors and related to the original implementation in [1].

5.1.4 Budget

The algorithms run off of a budget, which is the number of function evaluations that can be done. Once the budget runs out, or any other exit condition is met, the algorithm stops. In the experiments a budget was chosen of $10000n$ where n equals the problem dimension. This value was chosen as this matches the results in the PBO dataset closely.
On each problem instance, the algorithm was run 11 times indepently like the compared algorithms.

5.2 Problem instances

Every problem instance available (23 in total) has been evaluated by the LTGA. This section shows the problems briefly, of which the results are documented in section 6. For more information on the problem instances see [3].

- **F1, OneMax**: the problem described earlier in section 2.1. The problem OneMax asks to optimize the amount of 1's in the bit-string.
- **F2, LeadingOnes**: the problem that seeks to maximize the number of initial ones at the start of the bit-string. The problem is formally defined in [3] as

$$LO : \{0,1\}^n \to [0..n], x \mapsto max\{\, i \in [0..n] \mid \forall j \leq i : x_j = 1 \,\} = \sum_{i=1}^{n} \prod_{j=1}^{i} x_j$$

- **F18, Low Autocorrelation Binary Sequences**: LABS is a notoriously hard problem that seeks to maximize the merit factor. This merit factor is proportional to the reciprocal of the secquence's autocorrelation, and when written as a pseudo-boolean function over $\{\, 0,1 \,\}^n$, gives the following function [3]:

$$F_{LABS}(\vec{x}) = \frac{n^2}{2\sum_{k=1}^{n-1}(\sum_{i=1}^{n-k} x'_i \cdot x'_{i+k})^2} \text{ where } x'_i = 2x_i - 1$$

- **F19-21, The Ising model**: The Ising model is an NP-hard problem. The problem can be described as a graph, a lattice specifically, where each vertex represents an atom and edges represent bonds between other adjacent atoms. Each vertex can be in one of 2 states, -1 or 1. Each edge $\langle i, j \rangle$ has a strength $J_{i,j}$. Neighbouring vertices i and j, with states σ_i and σ_j respectively, contribute an amount of energy to the total amount of energy H, where H is defined as

$$H(\sigma) = -\sum_{(i,j)} J_{i,j}\sigma_i\sigma_j$$

  when ignoring external magnetic fields. The optimization problem is described as finding so-called *ground states*, which are minimal energy configurations. This is a hard combinatorial problem. IOHexperimenter has implementations of a 1D (Ring) problem, a 2D (Taurus) problem, and a triangular problem – defined on a 2D lattice. These are referred to as **F19**, **F20** and **F21** respectively.

- **F22, Maximum Independent Vertex Set (MIVS)**: An independent vertex set is defined as a subset S of vertices of some graph G where none of the vertices in S are direct neighbours of eachother. A maximum independent vertex set of G is defined as the largest independent vertex set for that graph.

- **F23, The N-Queens problem**: The N-Queens problem is based on chess. Consider an $N \times N$ chessboard. The aim is to place N queens on the board, so that none of the queens can attack any other queen. In IOHprofiler, this problem is implemented as a maximization problem.

5.3 Compared algorithms

The LTGA was compared with twelve other algorithms available in IOHexperimenter, as the results of these algorithms are available in the PBO dataset in IOHanalyzer. The algorithms are briefly summarized in this thesis. For more information on these algorithms, including pseudocode and background information, see [3]. These are the algorithms:

1. **gHC**: A greedy hillclimber algorithm, which flips a single bit each iteration, going through the string from left to right until any optimum is reached.

2. **RLS**: Randomized Local Search flips a random bit in each iteration.

3. $(1 + 1)\mathbf{EA}_{>0}$ an Evolutionary algorithm implementation with $\lambda = 1$. The algorithm is defined in [3].

4. $(1 + 10)\mathbf{EA}_{>0}$ Same as previous, but with $\lambda = 10$.

5. **fGA**: the Fast Genetic Algorithm, as proposed in [15].

6. $(1 + 10)\mathbf{EA}_{r/2,2r}$: The two-rate EA uses two mutation rates every iteration, both with a parameter r. The mutation rates used are $r/2n$ for one half of the offspring, and $2r/n$ for the other. r is updated after every iteration randomly, favoring the value used for the best offspring in λ.

7. $(1+10)\mathbf{EA}_{norm}$: an evolutionary algorithm where the mutation strength is sampled from a normal distribution.

8. $(1+10)\mathbf{EA}_{var}$: extension of the previous algorithm, where variance is also adapted.

9. $(1+10)\mathbf{EA}_{log-n}$: an EA which uses a self-adaptive choice.

10. $(1+(\lambda,\lambda))\mathbf{GA}$: the self-adjusting $(1+(\lambda,\lambda))$ genetic algorithm, where the population size λ is updated every iteration, including the mutation rate and crossover bias as these depend on population size.

11. **(30 + 30) vGA**: the "Vanilla" genetic algorithm, using roulette-wheel-selection. There is a $p_c = 0.37$ chance of 1-point crossover in this implementation, and the mutation operator flips every bit in an individual with probability $p_m = 2/n$.

12. **UMDA**: The univariate marginal distribution algorithm, which represents the family of estimation of distribution algorithms (EDAs). This algorithm was introduced in [16]. In the tables and figures in the experiments, this algorithm is referred to as *EDA*.

6 Results

This section shows the results of the conducted experiments using tables and figures. Here are the definitions of the table headers:

- ERT: the expected runtime of the algorithm to reach the target value v, taking failed runs into account that did not reach the target value. The lower the ERT, the faster the algorithm reaches the target value, so lower equals better. The formula for the ERT is:

$$ERT(r, s, B, AHT) = \frac{r-s}{s}B + AHT$$

  where

  - r is the number of independent runs of an algorithm.
  - s is the number of succesful runs that reached target value v. $s \leq r$ is always true.
  - B is the maximum budget of function evaluations.
  - AHT is the average first hitting time of the successful runs s.

- Mean: the mean runtime of the successful runs.
- Median: the median runtime of the successful runs.
- sd: Standard deviation of the runtime of the successful runs.
- ps: The percentage of runs that reached the target value.

As a lot of data has been generated, we have decided to only show results of functions where the performance of the LTGA was notably different from other algorithms. Also, only figures of dimension 100 are shown as these are the hardest tested problems.

6.1 LTGA vs Other algorithms

This section shows the comparisons made between the LTGA and other algorithms. The tables are ordered alphabetically by the algorithm names, with the LTGAs on the top. The best performing LTGA is highlighted, as is the best performing non-LTGA. The figures show a subset of the tested algorithms to reduce visual clutter. At least the best-performing LTGA and the best-performing non-LTGA are shown. Of the (1+10)EAs the best performing algorithm is shown if relevant. Furthermore, if other non-LTGAs perform similar to each other, only the best performing is shown. The figures are the results of dimension 100, as this is the most challenging dimension tested.

F1: OneMax	Dimension = 16, Target = 16					Dimension = 64, Target = 64					Dimension = 100, Target = 100				
Algorithm	mean	median	sd	ERT	ps	mean	median	sd	ERT	ps	mean	median	sd	ERT	ps
ltga_4	173	147	69	1320	36%	Inf	Inf	Inf	Inf	0%	Inf	Inf	Inf	Inf	0%
ltga_8	511	517	208	**511**	100%	3925	4187	750	**4733**	91%	6136	6567	1035	18287	55%
ltga_16	965	1026	298	965	100%	5331	5751	1339	5331	100%	9132	9905	1985	**9132**	100%
ltga_32	1456	1457	393	1456	100%	8998	9091	2614	8998	100%	18251	18924	2813	18251	100%
ltga_64	2193	2157	698	2193	100%	17136	17189	4107	17136	100%	30609	29731	7092	30609	100%
ltga_128	3933	4549	1982	3933	100%	31265	33360	5672	31265	100%	53477	53585	5294	53477	100%
ltga_256	4771	4935	3173	4771	100%	61598	65524	9397	61598	100%	100458	104784	15253	100458	100%
(1+(λ,λ)) GA	61	62	21	61	100%	406	359	177	406	100%	582	587	84	582	100%
(1+1) EA_>0	49	43	16	49	100%	331	341	64	331	100%	629	560	241	629	100%
(1+1) fGA	69	65	23	69	100%	484	469	138	484	100%	758	759	168	758	100%
(30,30) vGA	5632	3609	4365	5632	100%	53769	47265	17277	53769	100%	142666	127240	43085	142666	100%
EDA	195	205	38	195	100%	661	657	76	661	100%	1062	1017	214	1062	100%
gHC	16	17	2	**16**	100%	65	65	0	**65**	100%	100	101	2	**100**	100%
RLS	53	48	29	53	100%	249	233	71	249	100%	391	397	94	391	100%

Table 1: Results of F1: OneMax on all dimensions.

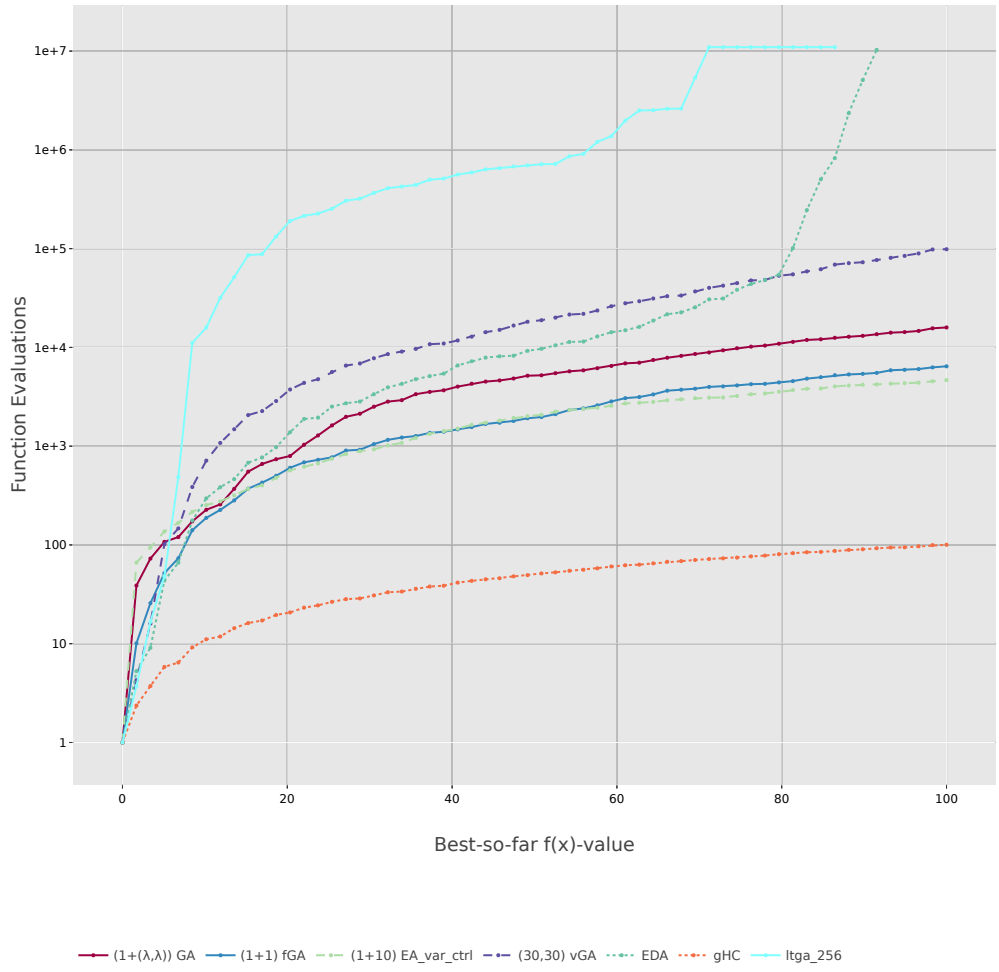

Figure 4: Mean evaluations on F2: LeadingOnes, dimension=100.

As can be seen in table 1 and figure 4, the LTGA's runtime is much higher than the compared algorithms on problems F1 and F2. While most LTGAs manage to reach the optimum at some point on F1, this was not the case for F2. On F2, apart from the the EDA, the LTGA was the only algorithm to not reach $f(x) = 100$.
On F18: LABS however, the LTGA performs much better as can be seen in figure 5. While the LTGA takes a bit longer to increase $f(x)$ than others at the start, it manages to steadily increase for much longer than any other algorithm in the PBO suite, reaching a much higher $f(x)$ than the others in less function evaluations. Though ltga_32 reached a mean $f(x) = 4.47$ and a best $f(x) = 5.24$ and ltga_128 a mean of $f(x) = 4.7$ and a best of $f(x) = 5.26$, these values do not come close to the theoretical maximum. On dimension 64 the maximum found by the LTGAs is $f(x) = 5.33$ by ltga_128. However this is still a lot lower than the optimum of 9.846 on $n = 64$ given in [17]. While this maximum is not known exactly for dimension 100, it seems logical that $f(x)$ does not come close on this dimension as well.

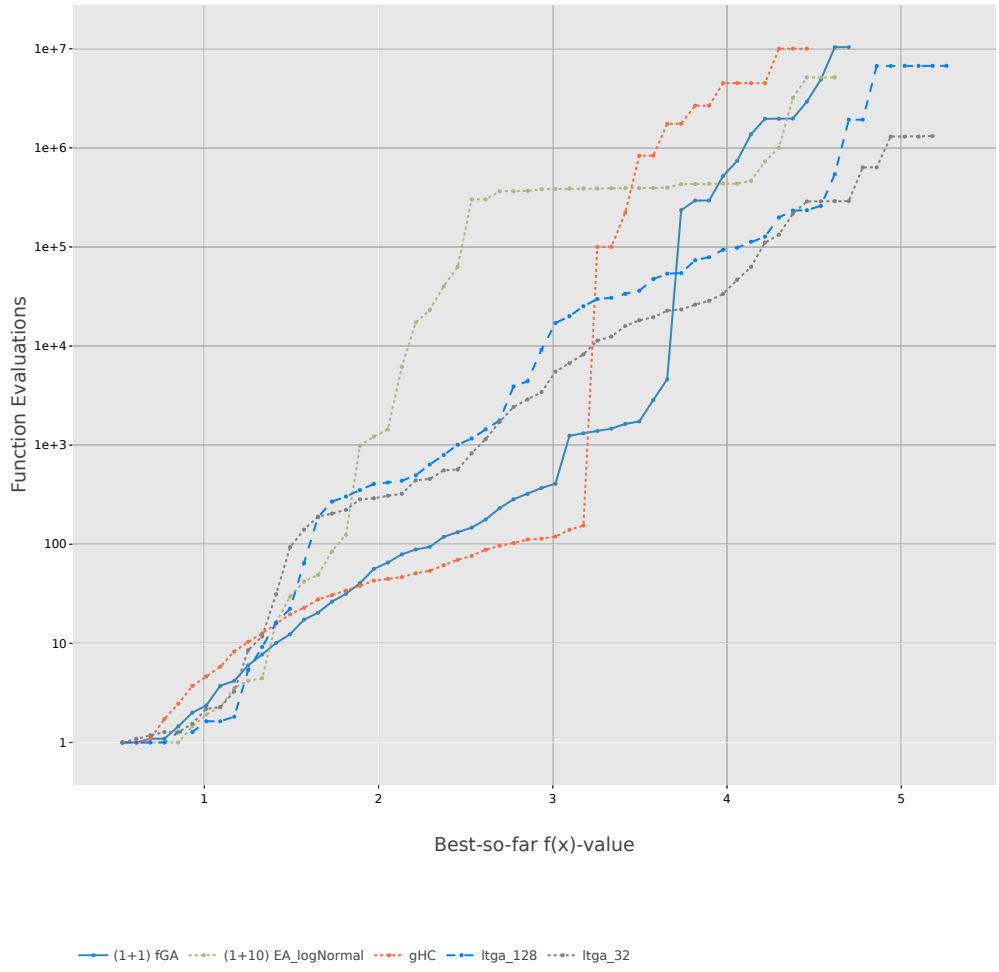

Figure 5: Mean evaluations on F18: LABS, dimension=100.

The next algorithms that were tested are F19-F21, the Ising model. Table 2 shows that the performance of the LTGA was not good on F19. Algorithms that did not reach the target value are excluded from the table, and only the best performing (1+10)EA is shown to keep the table small.

F19: Ising model, ring	Dimension = 100, Target = 200				
Algorithm	mean	median	sd	ERT	ps
ltga_128	420261	425050	88477	789632	64%
ltga_256	772457	799090	60418	**772457**	100%
(1+(λ, λ)) GA	582270	630690	270162	682270	91%
(1+1) fGA	84326	77324	81741	84326	100%
(1+10) EA_normalized	50299	35368	32997	50299	100%
RLS	43529	31561	39701	**43529**	100%

Table 2: Mean evaluations on F19: The Ising Model, Ring, dimension=100.

On the other Ising model instances contrary to F19, the LTGA performed well. In table 3 can be seen how ltga_32 outperforms every algorithm on dimension 100. This is almost true on dimension 64 as well, as here the LTGA is only surpassed by (1+10)EA_normalized. This algorithm manages

to reach an extremely low $ERT = 1006$ on this dimension, which is remarkable as the performance of this algorithm is not as mind-blowing on dimensions 16 and 100. Whether the result of this EA is a perfect case or simply erronous, is out of scope of this thesis unfortunately.
Table 4 shows the results on the final Ising model. Similar to F20, the LTGA again is the fastest by a big margin, except on dimension 16.

F20: Ising model, Torus	Dimension = 16, Target = 64					Dimension = 64, Target = 256					Dimension = 100, Target = 400				
Algorithm	mean	median	sd	ERT	ps	mean	median	sd	ERT	ps	mean	median	sd	ERT	ps
ltga_16	899	854	203	**899**	100%	15309	15163	2194	50003	45%	41018	35267	14355	180452	27%
ltga_32	1244	1098	541	1244	100%	26709	29079	7204	**26709**	100%	52312	51483	14376	**79071**	82%
ltga_64	1819	2196	1126	1819	100%	42258	42238	5801	42258	100%	102437	102824	15452	132856	91%
ltga_128	3333	3896	1858	3333	100%	80474	80719	16412	80474	100%	171924	175634	27318	171924	100%
ltga_256	3747	3972	3051	3747	100%	135169	134402	33472	135169	100%	335992	322478	48204	335992	100%
(1+(λ,λ)) GA	72	73	36	72	100%	2177	1343	1748	93199	82%	4353	2670	4138	226576	82%
(1+1) fGA	69	63	32	69	100%	45615	883	134191	136637	82%	2486	2063	1080	**102486**	91%
(1+10) EA_{r/2,2r}	155	155	51	155	100%	6175	1643	13714	47135	91%	4225	4206	1800	575653	64%
(1+10) EA_logNormal	582	549	352	582	100%	1940	1875	831	92962	82%	5435	4374	5615	227657	82%
(1+10) EA_normalized	106	96	31	2666	91%	1006	942	371	**1006**	100%	1948	1879	778	376948	73%
(30,30) vGA	3088	2552	1915	3088	100%	78366	79304	27110	169388	82%	263725	254348	105840	363725	91%
EDA	626	632	261	10226	73%	14230	12964	5710	248287	64%	43260	43101	27636	876594	55%
gHC	27	28	7	**27**	100%	221	198	89	41181	91%	447	435	125	222669	82%
RLS	49	41	22	2609	91%	652	584	204	154252	73%	1323	1384	173	834657	55%

Table 3: Mean evaluations on F20: The Ising Model, Torus, dimensions 16, 64, 100.

F21: Ising model, Triangular	Dimension = 16, Target = 96					Dimension = 64, Target = 384					Dimension = 100, Target = 600				
Algorithm	mean	median	sd	ERT	ps	mean	median	sd	ERT	ps	mean	median	sd	ERT	ps
ltga_4	230	191	101	834	55%	NA	NA	NA	Inf	0%	NA	NA	NA	Inf	0%
ltga_8	348	308	236	**541**	91%	5168	5175	1717	14246	55%	14215	14215	NA	236055	9%
ltga_16	757	643	522	757	100%	9379	9366	2779	**9379**	100%	21300	17776	8619	**33272**	82%
ltga_32	1148	1222	485	1148	100%	15823	16067	2346	15823	100%	34420	33405	7147	34420	100%
ltga_64	1518	1621	867	1518	100%	28256	26236	6714	28256	100%	60318	57864	8739	60318	100%
ltga_128	3094	4170	1656	3094	100%	51172	50485	6055	51172	100%	103116	103740	22616	103116	100%
ltga_256	3986	2955	2617	3986	100%	89886	95246	14023	89886	100%	185401	183008	27294	185401	100%
(1+(λ,λ)) GA	78	62	50	78	100%	6109	786	10047	97131	82%	30371	1673	63471	601800	64%
(1+1) fGA	67	52	42	67	100%	1332	1174	636	235389	64%	3594	2022	4951	**103594**	91%
(1+10) EA_normalized	109	82	68	109	100%	948	685	818	**41908**	91%	7064	1571	14569	107064	91%
(30,30) vGA	2552	2458	2141	2552	100%	97934	77271	57338	138894	91%	171823	130379	106721	394045	82%
EDA	570	412	336	3130	91%	55779	6421	111194	146801	82%	24421	6364	37121	124421	91%
gHC	23	23	7	**23**	100%	104	96	38	91126	82%	205	202	35	375205	73%
RLS	48	41	19	5737	82%	445	394	164	91467	82%	813	663	418	1200813	45%

Table 4: Mean evaluations on F21: The Ising Model, Triangular (Isometric 2D Grid), dimension 16, 64, 100.

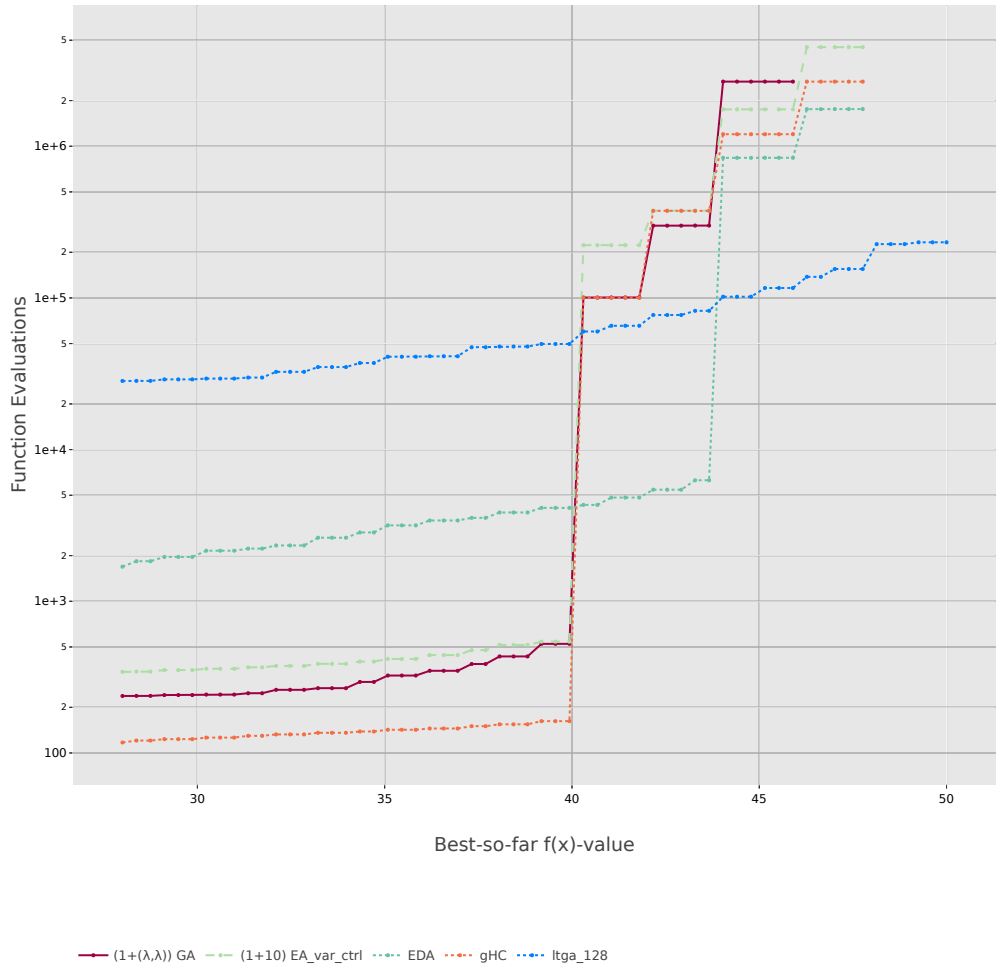

Figure 6: Mean evaluations on F22, dimension=100.

The final result shown in this section is the result of F22, the maximum independent vertex set. The LTGA held up very well on this problem, as it manages to reach a much higher $f(x)$ than the other algorithms in less function evaluations. As a matter of fact, the LTGA is the only algorithm that manages to reach the target value $f(x) = 50$ within the tested budget.

6.2 Population size

Another topic of interest was the comparison of performance of different population sizes. Section 6.1 already gave some insight in the performance of different population sizes through its tables. The purpose of this section is to provide more insight in the variation of population size.
First of all, in F2:LeadingOnes shows clear differences between population sizes as can be seen in figure 7. Looking at the figure, it can be seen that higher population sizes manage to achieve a higher $f(x)$. ltga_8 appears to not follow this trend though, as it performs better than ltga_16 and ltga_32. Whether this is luck because of a beneficial initial population or not has yet to be researched.

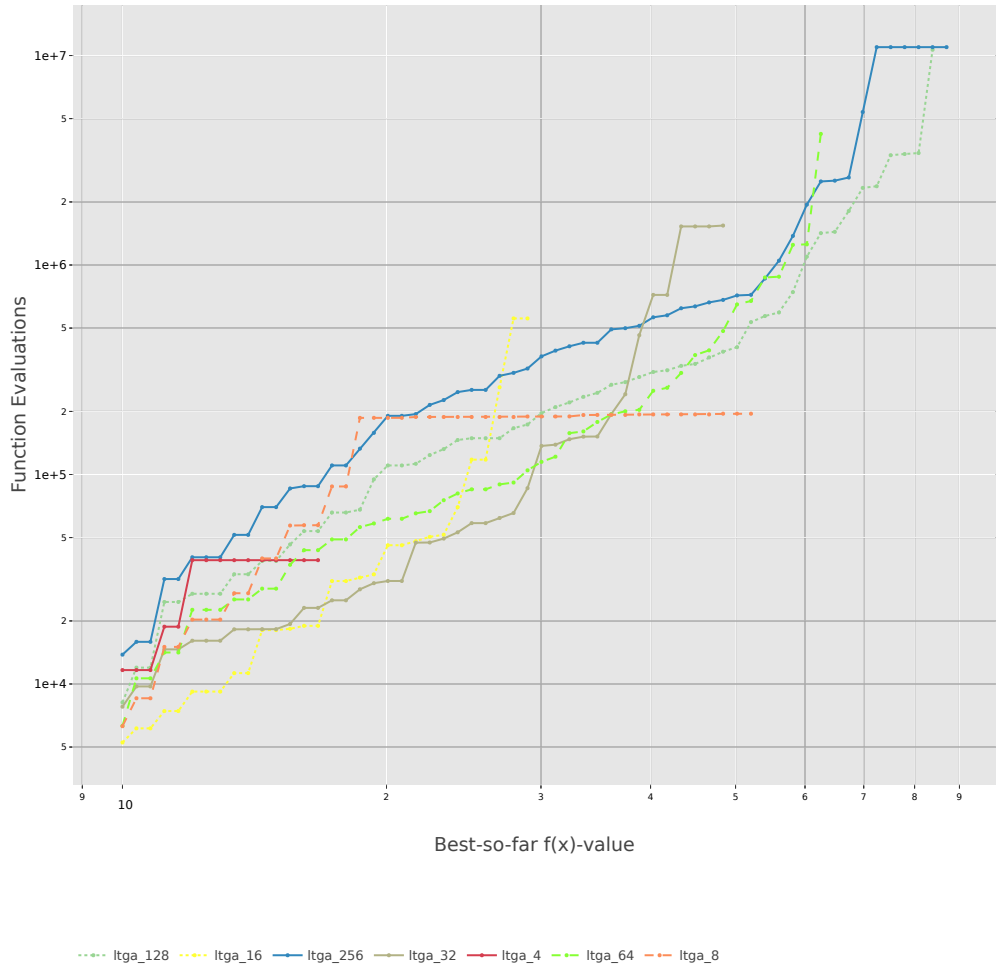

Figure 7: Mean evaluations of LTGAs on F2, dimension=100.

F18: LABS show different results as shown in figure 8. Here ltga_32 performs much better than the variants of higher population sizes. This effect is also visible on F20, which can be seen in figure 3 in the previous section.

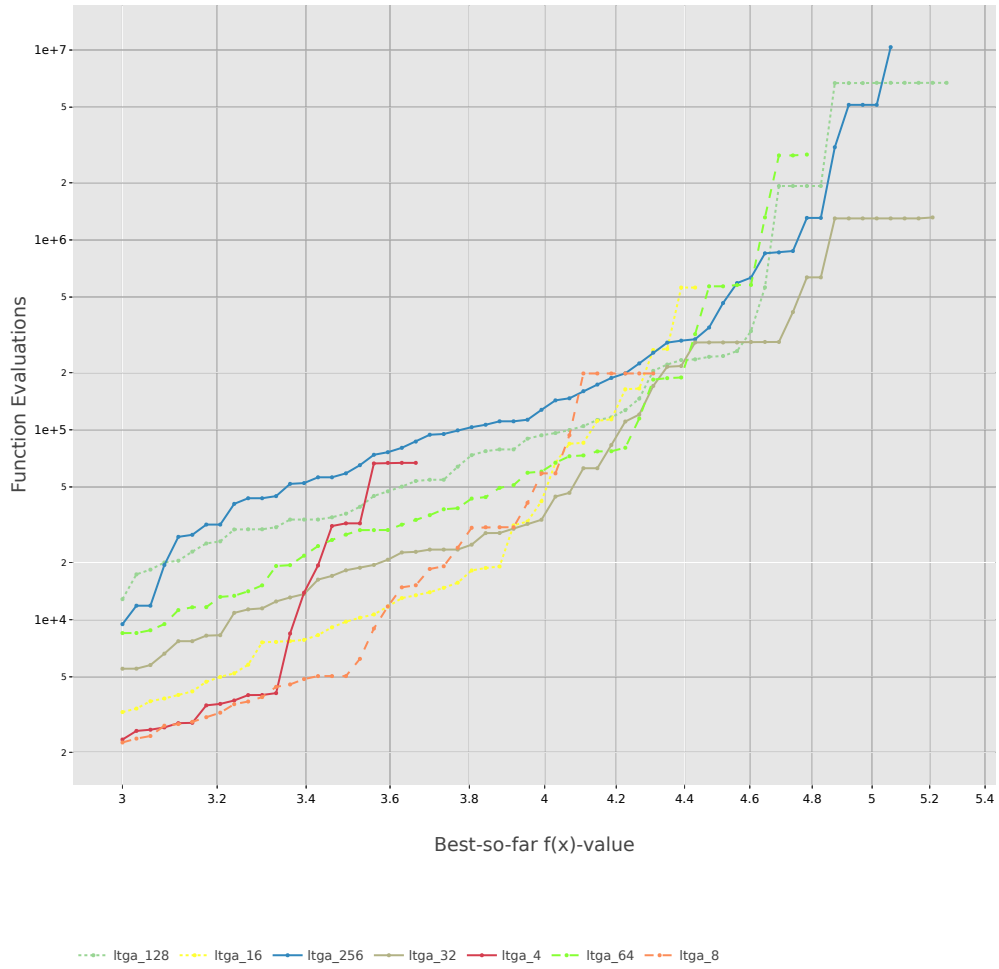

Figure 8: Mean evaluations of LTGAs on F18: LABS, dimension=100.

7 Conclusions

From the conducted experiments a number of conclusions can be made. First of all, the LTGA does not perform fast on the tested low dimensional problems compared to other algorithms. There are instances however where the LTGA is much faster when the problem dimension increases, for instance in 6. The LTGA seems to mainly be an interesting choice on higher dimensional problems, as lower dimensional problems are more easily solved by algorithms that are less reliant on learning linkage.

Secondly, the size of the population has a non-linear effect on the expected runtime. Generally speaking a larger population increases the ERT (F1: OneMax for example), although a larger population sizes might be required to find the optimum; F2: LeadingOnes on dimension 64 can only be solved by ltga_128 and ltga_256. Sometimes a high dimensional problem can be effectively tackled with a low population size, for example in F21 where a population size of 16 is enough to solve both dimension 64 and 100 with the lowest ERT. It can therefore be recommended to

gradually increase the population count, starting from a value $\lambda \approx 16$, as the results of $\lambda = 4$ and $\lambda = 8$ rarely proved useful. These low population sizes seem to get trapped too fast as the algorithm stops when every solution candidate is the same, which happens quicker at lower dimensions.

Finally it can be concluded that problems F18, F20, F21 and F22 benefit from the use of linkage information. F2 however seems to mislead the algorithm, as it takes an immense amount of function evaluations and a high population count to reach the optimum.

8 Further research

An interesting topic for research could be the complexity of the algorithm. While the LTGA manages to keep function evaluations low on some complex problems, the CPU time taken by the algorithm is quite high because of the many entropy calculations which are computationally expensive. This is especially noticeable on higher population counts.
Furthermore, it would be interesting to see how the LTGA performs in combination with other algorithms, where the other algorithm provides a locally optimized population for the LTGA to use. Maby this could 'solve' the LTGA's slow startup time.

References

[1] Dirk Thierens. The linkage tree genetic algorithm. *Parallel Problem Solving from Nature*, pages 264–273, 2010.

[2] Endre Boros and Peter L. Hammer. Pseudo-boolean optimization. *Discrete Applied Mathematics*, 123(1):155 – 225, 2002.

[3] Carola Doerr, Furong Ye, Naama Horesh, Hao Wang, Ofer Shir, and Thomas Bäck. Benchmarking discrete optimization heuristics with iohprofiler. pages 1798–1806, 07 2019.

[4] J. M. Johnson and Y. Rahmat-samii. Genetic algorithm optimization and its application to antenna design. In *Proceedings of IEEE Antennas and Propagation Society International Symposium and URSI National Radio Science Meeting*, volume 1, pages 326–329 vol.1, 1994.

[5] S. Ghaemi, M. Taghi Vakili, and A. Aghagolzadeh. Using a genetic algorithm optimizer tool to solve university timetable scheduling problem. In *2007 9th International Symposium on Signal Processing and Its Applications*, pages 1–4, 2007.

[6] S. van Rijn, M. Emmerich, E. Reehuis, and T. Bäck. Optimizing highly constrained truck loadings using a self-adaptive genetic algorithm. In *2015 IEEE Congress on Evolutionary Computation (CEC)*, pages 227–234, 2015.

[7] David H Wolpert and William G Macready. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.

[8] Colin Reeves. *Genetic Algorithms*, volume 146, pages 109–139. 09 2010.

[9] Lawrence Davis. Handbook of genetic algorithms. 1991.

[10] Krzysztof Sadowski, Peter Bosman, and Dirk Thierens. On the usefulness of linkage processing for solving max-sat. pages 853–860, 07 2013.

[11] Peter A.N. Bosman and Dirk 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*, GECCO '13, page 359–366, New York, NY, USA, 2013. Association for Computing Machinery.

[12] Jean Paulo Martins and Alexandre Claudio Botazzo Delbem. The influence of linkage-learning in the linkage-tree ga when solving multidimensional knapsack problems. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, GECCO '13, page 821–828, New York, NY, USA, 2013. Association for Computing Machinery.

[13] Carola Doerr, Hao Wang, Furong Ye, Sander van Rijn, and Thomas Bäck. Iohprofiler: A benchmarking and profiling tool for iterative optimization heuristics. *arXiv preprint arXiv:1810.05281*, 2018.

[14] Brian W. Goldman and Daniel R. Tauritz. brianwgoldman/ltgavariantsandanalysis, 2012. https://github.com/brianwgoldman/LTGAVariantsAndAnalysis.

[15] Benjamin Doerr, Huu Phuoc Le, Régis Makhmara, and Ta Duy Nguyen. Fast genetic algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '17, page 777–784, New York, NY, USA, 2017. Association for Computing Machinery.

[16] H. Mühlenbein. The equation for response to selection and its use for prediction. volume 5, pages 303–346, 1997.

[17] Tom Packebusch and Stephan Mertens. Low autocorrelation binary sequences. *Journal of Physics A: Mathematical and Theoretical*, 49, 12 2015.