Utrecht University
Department of Information and Computing Sciences

Master Thesis in Computing Science

# Multi-objective LTGA

Linkage learning in multi-objective problems with the Linkage Tree Genetic Algorithm.

**Author**
Johannes Job Dorland, BSc

**Supervisor**
Dr. ir. Dirk Thierens

**Student Number**
ICA-3638952

July, 2019

**Abstract**

In recent years new types of evolutionary algorithms have been introduced that try to improve traditional crossover operators by learning the structure of the problem, most notably the Linkage Tree Genetic Algorithm. For single-objective problems linkage learning has proven to be a valuable improvement over the traditional crossover operators, since it can automatically learn and exploit the structure of a specific problem. Some initial research has been done to extend the use of LTGA into the domain of multi-objective problems, but no research has been done that looks at the effectiveness of linkage learning in a multi-objective environment. In order to do this some new multi-objective variants of the NK-landscape are introduced. For single-objective problems there are plenty of benchmark functions that model different types of problem variable correlations, but adaptations of these benchmark functions for multi-objective environments often leave the variable correlations between objectives unchanged.

# Contents

# Chapter 1

# Introduction

## 1.1 Problem Description

In recent years new types of evolutionary algorithms have been introduced that try to improve upon traditional crossover operators. Evolutionary algorithms are mainly used for black-box optimization problems where no information is present about the problem. Traditional crossover operators use problem specific information to combine parent solutions into offspring. The Linkage Tree Genetic Algorithm (LTGA) introduced by Thierens [1] takes a different approach and learns a model based on the population using a hierarchical clustering algorithm. This technique is called linkage learning, as it dynamically learns the interactions between problem variables. The effectiveness of this model has been proven for a variety of problems.

Some initial research has been done to extend the use of LTGA into the multi-objective domain by Luong et al. [2], but it remains unknown whether linkage learning is effective for multi-objective problems. It can be much harder to learn a structure for a multi-dimensional problem, as it is not as straightforward to define an ordering on the quality of the solutions. Furthermore parts of a solution might have interactions with different parts of the solution for each objective. This makes it harder to learn any structure, because it makes the entire problem appear more random.

Existing artificial multi-objective benchmark functions do not take this into account, which means that the structure of the problem is the same regardless of the objective. This has as an effect that it is hard to measure the performance of linkage learning on these types of problems, so some new benchmark problems are needed.

There are generally two approaches to tackle multi-objective problems. A Pareto based approach, using the Pareto dominance relation to define a partial order on the solutions, which tries to find the Pareto front in one run. The other approach is scalarization, where the multi-objective problem is turned back into a single objective problem by applying a transformation function that takes the vector from the multi-objective evaluation function and turns it into a scalar value.

## 1.2 Research Question

The goal of this thesis is to investigate the effectiveness of linkage learning for multi-objective problems. To guide this thesis there are three points of interest that will be investigated.

- How is a good benchmarking problem constructed where the structure between the different objectives can be altered?
- Is linkage learning effective for multi-objective problems with structure?

- Is linkage learning more effective with a scalarized or Pareto based approach for these type of problems?

## 1.3 Contributions

In order to do this a number of contributions are made in this thesis. To measure the effectiveness of linkage learning two things are needed: An algorithm that works for multi-objective problems and a set of multi-objective problems to start with. Two new benchmark function are introduced. First the $\rho$MNNK landscape is introduced, a simple multi-objective nearest neighbour NK-landscape based on the $\rho$MNK landscape. Secondly as current benchmark functions have either the same or a completely different structure in each objective a new benchmark function is introduced that has a configurable linkage structure, the $\lambda$MNK landscape. This benchmark function is a variant of the nearest neighbour NK-landscape, with a tuneable parameter $\lambda$ that introduces slight differences in the linkage structure of the NK-landscape.

A new variant of LTGA is introduced that is adapted for the multi-objective environment, called the Multi-objective Linkage Tree Genetic Algorithm (MO-LTGA). Linkage learning is very effective for the NK-landscape familiy of benchmark functions (both $\lambda$MNK and more traditional multi-objective variants). MO-LTGA is also compared to a scalarized variant of LTGA and iterated local search (ILS) to show the difference in performance of the different approaches.

## 1.4 Outline

In chapter 2 a background is given on multi-objective optimization and some baseline information is given. In chapter 3 an introduction is given of the LTGA algorithm. Within the framework of this thesis, I am referring to a specific revision of LTGA, published in Thierens and Bosman [3]. In chapter 4 the current benchmarking functions present in literature are introduced. An explanation of why these did not suit our purpose is covered in chapter 5, accompanied with a set of new benchmarking functions. The experimental setup is described in chapter 6. In chapter 7 MO-LTGA is introduced with an experimental analysis of the effectiveness of linkage learning. In chapter 8 a review and analysis of MO-GOMEA is shown. Some of the ideas of MO-GOMEA are used to improve MO-LTGA in chapter 9. To compare the Pareto based MO-LTGA to a scalarized algorithm a comparison is done with scalarized LTGA in chapter 10. In the discussion in chapter 11 a reflective look is taken at all the results of this thesis. This thesis concludes with a conclusion in chapter 12.

# Chapter 2

# Background Information

To start things of a little bit of background information is needed. First a formal definition of multi-objective optimization problems are given. Secondly we will look at a data structure to store a set of non-dominated points that can also insert new points efficiently, which is essential for good performance. Thirdly we will look at the hypervolume measure, a qualitative measure to compute the quality of a set of multi-dimensional points. Then we will look at two different approaches to solving multi-dimensional problems, how they work and how they differ.

## 2.1 Multi-objective Optimization Problems

There exist optimization problems where it is hard or impractical to capture the quality of a solution to the problem in a single value and it makes much more sense to have multiple measures that each measure a different aspect of the solution. Each measure is called an objective function. It calculates the quality, or fitness, of the solution for the given objective. This is why these kind of problems are referred to as multi-objective optimization problems.

Multi-objective optimization problems (MOPs) are optimization problems with $m > 1$ objective functions. This means that a fitness function will return a vector in $m$-dimensional space instead of a scalar.

$$F(S) = (F_1(S), F_2(S), ..., F_m(S))^T \tag{2.1}$$

Assume without loss of generality that we want to maximize all objectives. Since the objective function now returns a vector instead of a scalar it becomes much harder to compare two different solutions. A solution can have a better value in one objective, but be worse in all others. However a solution is still better than another if it has a better value for all objectives. This is called Pareto dominance. A solution $x$ is said to strictly dominate $y$, denoted $y \prec x$ if and only if $\forall i \in \{1, 2, .., m\}\ F_i(y) \leq F_i(x)$ and $\exists i \in \{1, 2, .., m\}\ F_i(y) < F_i(x)$. If two solutions do not dominate each other $x \not\prec y \wedge y \not\prec x$, they are mutually non-dominating. Since it is unlikely that there is a single solution that dominates all other solutions the goal is to find the set of solutions which are not dominated by any other existing solution. A solution $x$ is called Pareto optimal if and only if $\nexists z$ for which $x \prec z$. The set of all Pareto optimal solutions is the Pareto front, often also called the Pareto set.

## 2.2 Quad tree

Regular single-objective algorithms generally keep track of the best solution they ever encountered during the search and return this solution at the end. This is very simple

to do and it is a $\mathcal{O}(1)$ operation. For multi-objective problems this becomes a bit more cumbersome. Ideally we want to store all mutually non-dominating solutions found so far in a set which approximates the Pareto front, the local Pareto front. When new solutions are generated they are put in this local Pareto front if they are not dominated by any point in the set and any points they dominate are removed.

The simplest way to store this set is in a linked list. When inserting a new solution $x$ we traverse the entire linked list and for each solution $y$ we check if $x$ dominated $y$ or vice versa. If $x \prec y$, $x$ is dominated by $y$ and $x$ should not be included in our local Pareto front $\mathcal{P}$ and we can stop, since $x \prec y \rightarrow \nexists z \in \mathcal{P} \;\; z \prec x$. If $x$ is dominated, there cannot be a solution $z \in \mathcal{P}$ that is dominated by $x$, since that would also be dominated by $y$ and should not have been in $\mathcal{P}$ in the first place. The Pareto dominance relation is transitive. $z \prec x \wedge x \prec y \rightarrow z \prec y$. Similarly if $y \prec x$, we know $x$ should be included in $\mathcal{P}$ and $y$ must be removed. There cannot be a solution $z \in \mathcal{P}$ which dominates $x$, since that solution would also have dominated $y$, but $y$ was in our local Pareto set. This time however we need to traverse the entire linked list to search for more candidates that could be dominated by $x$. In the case that $x$ and $y$ are mutually non-dominated nothing happens and we go to the next element in the linked list. If $\forall y \in \mathcal{P} \; x \not\prec y \wedge y \not\prec x$, $x$ is also added to $\mathcal{P}$, and no solutions are removed.

The downside of this linked list approach is that an insertion generally takes $\mathcal{O}(n)$ where $n = |\mathcal{P}|$, the size of the local Pareto set. The ordering of this list has an impact. If good solutions that dominate a lot of the candidate solutions are in the front of the linked list the attempt for insertion can be stopped in an earlier stage. It is however quite hard to create and maintain a good ordering and the general linked list approach is not very effective.

Mostaghim et al. [4] introduced a data structure specifically designed for this kind of problem. This data structure is a heavily modified quadtree. Each node in this tree contains a solution and has $2^M - 2$ children. This structure can be queried more efficiently for retrieving solutions that dominate or are dominated by a certain solution, since at every level, only a subset of all the children have to be evaluated. While Mostaghim et al. [4] do not perform any analysis on the complexity of the insert operations and it can be argued that it is still $\mathcal{O}(n)$ worst case. Experimental analysis shows however that their data structure performs orders of magnitudes better than a linked list. Even though they introduce multiple variants of their quadtree with minor improvements, for this thesis the simplest variant Quad-Tree1 one is used. Although the other versions are slightly better for handling deletions the gains are negligible, but the implementation of Quad-Tree1 is easier. For full details of how this data structure works read the paper by Mostaghim et al. [4]. A comparison of the performance of the quadtree data and linked list is done in appendix A.

## 2.3 Hypervolume

Another problem that arises when moving from single-objective to multi-objective is that comparing results of different algorithms becomes harder too. Instead of having a single best solution the result now consists of a set of solutions, that all try to approximate the Pareto set. Comparing two of these sets and determining which set is better is a nontrivial task. Ideally what you want is a function that return a quantitative measure of how good a local Pareto front is. Luckily quite a lot of research has already been done is this area and the most widely used measure is the hypervolume indicator. The hypervolume indicator calculates the $m$-dimensional volume of the objective space that is dominated by a given set of points $\mathcal{P}$, given a reference point $r$. Every point that lies within this volume is dominated by one or more points in $\mathcal{P}$. This reference point is needed, since otherwise the

hypervolume would be infinitely large, as there is no lower limit on how "bad" a solution can be. To bound the hypervolume to a non-infinite number the reference point $r$ is used and only the volume of $\mathcal{P}$ with respect to $r$ is calculated. For this to work $r$ needs to be the worst possible solution, or be dominated by it. Usually it is quite straightforward to calculate some bound on the problem, since $r$ does not need to be a strict bound, but only a non-infinite one. Several algorithms have been developed to calculate the hypervolume. The used implementation was developed by Fonseca et al. [5]. Their implementation has a running time of $\mathcal{O}(n^{m-2} \log n)$ and has a publicly available implementation in C that is GPL-licensed [1]. The hypervolume indicator has a lot of nice properties. It is at a maximum for the optimal Pareto set, and when new non-dominated solutions are added to a local Pareto set the hypervolume will also always increase. Overall this indicator gives a good overall impression of the quality of a local Pareto front $\mathcal{P}$.

## 2.4 Pareto based versus Scalarized methods

Solving multi-objective problems poses some new challenges as most algorithms are designed to work with only a single objective function. That is a reason why a lot of multi-objective algorithms solve MOPs by transforming the MOP back to a single-objective problem in a process called scalarization.

$$\max \quad G(F_1(S), F_2(S), ..., F_m(S), \lambda) \tag{2.2}$$

where $G$ is an aggregation function $G : \mathbb{R}^m \mapsto \mathbb{R}$ and $\lambda$ is a tuneable parameter. The aggregation function $G$ must have at least one property and that is that an optimal solution to the problem $G$ is also a Pareto optimal solution to the problem $F$. Two popular aggregation functions are the linear weighting and the Tchebycheff weighting.

### 2.4.1 Linear Weighting

When using the weighted sum method the function $G$ just returns a weighted sum over all objective values, given a set of weights $\lambda$, where $\forall \lambda_i$, $\lambda_i \geq 0$ and $\sum_{i=1}^{m} \lambda_i^2 = 1$.

$$G(S, \lambda) = \sum_{i=1}^{m} \lambda_i F_i(S) \tag{2.3}$$

The weight vector $\lambda$ is an $m$-dimensional unit vector. This method works with any $m$-dimensional vector, as long as all weights are non-negative. Since $\lambda$ is a weight vector, only the relative weights matter, since that is what determines the direction of the vector. Scaling the entire vector has no influence on the behaviour. To avoid redundancy all the vectors are scaled to unit length, so that every direction is uniquely represented by a single vector.

It can be proven that a solution which is optimal for the aggregation function $G$, given a set of weights $\lambda$, is also a Pareto optimal point for the original MOP.

A major advantage this method has is that a linear weighting of a MOP in many cases results in a well-known single-objective variant of the same type of problem. Examples of such problems are multi-objective Max Cut and the multi-objective travelling salesman problem

The drawback of this approach is that not every Pareto optimal point can be found this way. There are Pareto optimal points for which there does not exist a weight vector $\lambda$ for which $G(S, \lambda)$ is optimal. This happens if the Pareto front is non-convex.

An example of this is shown in figure 2.1.

[1] Available at `http://lopez-ibanez.eu/hypervolume`

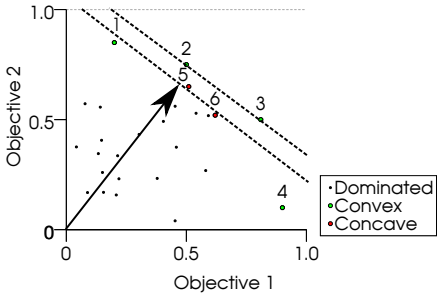


Figure 2.1: An example of a non-convex Pareto Front

Figure 2.1 shows an example of the objective space for some maximization problem. Black dots are points that are dominated by other points in the objective space. Red and green points are part of the Pareto front. The large arrow is an example of a weight vector $\lambda$. The perpendicular dotted lines are lines of equal fitness when using $\lambda$ for the scalarization function $G(S, \lambda)$. Points 1 to 6 are all Pareto optimal points. However, only points 1 to 4 are convex. Points 5 and 6 are non-convex. Since the perpendicular lines represent points of equal value, points that lie above this line are better than the points below a given line. Take a look at the second dotted line crossing points 2 and 3. Since they both lie on this line this means the scalarization would rate them equally good. There are no other points lying above the line, meaning they are optimal solutions for this given weight vector.

Now look at the first dotted perpendicular line. Point 5 lies on the line, but points 2 and 3 lie above it. We could rotate the vector clockwise giving more priority to objective 1, so point 2 moves under the line. However, point 3 will still stay above it. Similarly we can move the vector counterclockwise, giving more priority to objective 2, so point 3 moves under the line. This time around point 2 will stay above the line. The is no way to move the vector such that the perpendicular line has no other points above it. This means that for any weight vector $\lambda$, points 5 will never be an optimal solution. The same reasoning is valid for point 6.

### 2.4.2 Tchebycheff Weighting

A weighting that is capable of finding every point on the Pareto front is the Tchebycheff weighting. This weighting function also needs a direction vector $\lambda$, where $\forall\lambda_i$, $\lambda_i \geq 0$ and $\sum_{i=1}^{m} \lambda_i^2 = 1$, but in addition also needs an ideal point $z^*$. $z^*$ must be an artificial point in objective space that is not dominated by any solution, so it is sometimes also called a utopian solution as it is not actually a real solution. It acts as a reference point to which the distance of a solution is measured.

$$G(S, \lambda) = \max_{1 \leq i \leq m} \|\lambda_i(z_i^* - F_i(S))\| \tag{2.4}$$

Another difference with the linear weighting method is that with linear weighting the type of optimization, minimizing or maximizing does not change. If the MOP is a maximization problem, the linear weighting of that MOP problem must also be maximized, and vice versa for minimization. With the Tchebycheff weighting the function returns an artificial distance to an ideal point. We want to get as close as possible to this ideal point, so this distance must always be minimized. The position of the ideal point $z^*$ with respect to solution space determines whether it is minimizing or maximizing the problem. The distance metric calculates the distance for each objective to the ideal point $z*$ and multiplies it by the weight for that objective, specified by $\lambda_i$. Using this metric each Pareto optimal point has a corresponding vector $\lambda$ for which that particular point optimal for $G(S, \lambda)$. A graphical example of this can be seen in figure 2.2.

A thing to note with this method is that the created single-objective problem behaves quite differently from the the linear weighting method. The single-objective problems that the Tchebycheff weighting creates are minimizing a distance in objective space, while the linear weighting is often just optimizing a particular instance of a single objective problem.

Another problem is that the aggregation function returns the maximum over multiple distances any improvement in one of the values goes by unnoticed, unless it is the highest distance. This means that for some type of problems the Tchebycheff weighting contains flat regions. More on the topic of scalarization can be found in Jaszkiewicz et al. [6]

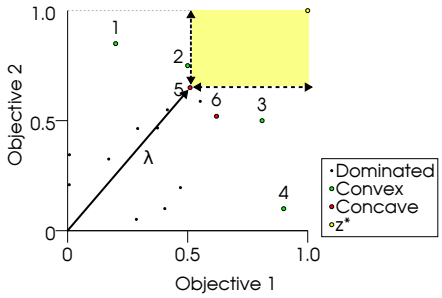


Figure 2.2: An example of a non-convex Pareto Front

The location of point 5 is $(0.51, 0.65)$. The distances in each objective to $z*$ are shown as the dotted lines. Suppose you have a weight vector $\lambda$, so that when applying the weighting these distances are the same. The area in which a point must lie such that it is better than point 5 is shown in yellow. Not coincidentally this area is equivalent to the area of points that dominate point 5. Since this area is empty point 5 is Pareto optimal, as there is no point in the objective space that dominates it. With a bit of imagination it is quite easy to see that for each point a different weight vector $\lambda$ exists and that the corresponding yellow region is empty.

### 2.4.3 Pareto Based

An entirely different approach to MOPs is by using Pareto based methods. Instead of turning the MOP back to a single objective problem, we look at it from a different perspective. In single objective optimization all solutions can be ranked according to their fitness values. Any two solutions can be compared by their fitness and either one of the two is better, or they are equally good. The same thing can be done for MOP using the dominance relation, one solution can dominate the other, or they are mutually non-dominating. Since the dominance relation is transitive a partial order can be defined over all solutions. This partial order can be used to guide a search towards the Pareto front. This is where we the similarity with single-objective problems ends though. While with single-objective problems we are usually only interested in a single solution which is the best we can find and if we have multiple solutions with the same fitness, we only keep one and lose interest in the others. With an MOP this does not work since our goal is not to find a single good solution, but to find, or at least approach, the Pareto front which may contain more than a single point. While if one solution dominates another it is trivial that we can throw away the dominated solution, the other solution is better in every way. However if two solutions are non-dominating we really want to keep more than one solution. As we are actually trying to find a set of solutions instead of a single solution a different approach has to be taken which involves sets of solutions.

Methods that try to find the Pareto front in a single run of the algorithm, without using aggregation functions, based on the dominance relation between two solutions are Pareto based.

# Chapter 3

# Linkage Tree Genetic Algorithm

## 3.1 Evolutionary Algorithms

Evolutionary algorithms (EAs) are algorithms that borrow certain ideas from evolutionary biology and apply them practically to solve black box optimization problems. More specifically they use mechanisms like mutation, reproduction and selection to evolve and improve upon a population of solutions. The general outline of an EA looks as follows. An EA starts by initialising a population. In general initial solutions are generated in a completely random fashion, but they can also be generated by a heuristic. This can however have a drawback. If the generated initial population contains solutions which are quite similar then the algorithm will converge very quickly. If the population converges too quickly the result is not optimal. This is called premature convergence. It is vital to start with a diverse enough population to prevent premature convergence, so generating initial solutions with heuristics should be done with care. After the initial population is created parents are selected from the population. These parents then create new offspring by means of reproduction. A recombination operator takes the information of the parents and creates an offspring solution from them. During this reproduction there is a small chance that the offspring gets a mutation which is not necessarily present in any of its parents. The newly created offspring competes with other offspring or its parents, depending on the type of EA, for a place in the population according to the "survival of the fittest" principle. Different EAs will do the steps of reproduction, mutation and selection each in a different way or may even omit some of these steps, but the general outline stays the same. Evolutionary algorithms only need a fitness evaluation function to determine how good a given solution is. Therefore, EAs are generally well suited to tackle black box optimization problems. They make no assumptions about the input problems and often still manage to get good results through the evolutionary process.

## 3.2 Estimation of Distribution Algorithms

Classic EAs generate offspring by recombining a select set of parents. This method does have some limitations however. Since the offspring is only created from a select set of parents only information present in these parents can be used. Optimization problems often contain some sort of structure. Good recombination operators try to exploit this structure. In practice good recombination operators are usually operators that are problem specific. They exploit knowledge from the fact that they have extra information about the problem and use that to recombine solutions. If we want a black box algorithm this is not an ideal approach. Instead we would like to have an algorithm that automatically finds this structure and exploit it. This is exactly what Estimation of Distribution Algorithms (EDA) introduced by Pelikan et al. [7] try to do. At each generation EDAs build a probabilistic model of the current population and new offspring is generated by sampling this model.

This is why EDAs are also referred to as Probabilistic Model Building Genetic Algorithms (PMBGA).

## 3.3 Gene-pool Optimal Mixing Evolutionary Algorithm

The Gene-pool Optimal Mixing Evolutionary Algorithm (GOMEA) introduced by Thierens and Bosman [3] is similar to an EDA in a way, but does not directly use a probabilistic model. Instead it uses a technique called optimal mixing where new solutions are created by taking a single parent solution from the population and iteratively mixing in parts of other donor solutions in a greedy fashion. As soon as an improvement is found the algorithm continues mixing with the improved solution.

The mixing is done using a crossover operator. The initial child solution $o$ is the same as the parent solution $p$. In each iteration of the mixing a subset of variables is taken and a random donor solution is selected. In the next step the bits of the selected variables from the donor solution are copied over to the child solution $o$ to create $o'$. If $o'$ is an improvement over $o$ than the solution $o'$ is accepted as our new child $o$ and otherwise it is rejected and the mixing continues with the old solution. This process is repeated until there are no more subsets of variables left to swap.

A vital part of this algorithm are the subsets of variables which are used in crossover. A good crossover operator will try to minimize the disruptive effect it has and instead tries to swap entire substructures. In practice good crossover operators are problem specific in order to do this, but ideally we want an operator who does this dynamically. In this case, the performance of the optimal mixing operator is determined by the subsets of variables which it uses.

## 3.4 Family of Subsets

This set of subsets is also referred to as a Family of Subsets (FOS). A FOS is a set of subsets of some set. In this case the main set is the set of problem variables $\{0, 1, ..., l-1\}$, so a FOS $\mathcal{F}$ is a set $\mathcal{F} = \{F^0, F^1, ..., F^{|\mathcal{F}|-1}\}$ where each set $F^i \subseteq \{0, 1, ..., l-1\}$. In each iteration of the GOMEA algorithm one subset $F^i$ is taken from $\mathcal{F}$ and the bits from the problem variables $F^i \subseteq \{0, 1, ..., l-1\}$ are copied over. An example of a very basic FOS is the univariate model $\mathcal{F} = \{\{0\}, \{1\}, ..., \{l-1\}\}$. In the univariate model no dependencies between problem variables are modelled and only single bits are copied over from donor solutions.

There are many more ways to create a FOS. The univariate structure is fairly limited since it does not model any dependencies between problem variables, so a good FOS will also group problem variables together which are highly correlated with each other.

**Algorithm 1:** Gene-pool Optimal Mixing Evolutionary Algorithm

**Input**: An individual $o$, a population $C$ and a FOS $\mathcal{F}$.
**Output**: An improved offspring solution $o$.
**for** $F^i \in \mathcal{F}$ **do**
  $o' \leftarrow o$
  $p \leftarrow Random(C)$ `/* Random donor from the population */`
  $o'_{F^i} \leftarrow p_{F^i}$ `/* Copy over the bits from donor to child */`
  **if** $o' \neq o$ ***and*** $fitness(o') \geq fitness(o)$ **then**
    $o \leftarrow o'$
  **end**
**end**
**return** $o$

## 3.5 Linkage Tree Genetic Algorithm

The Linkage Tree Genetic Algorithm (LTGA) was introduced by Thierens [1], even though several revisions of the algorithm exist. The most recent revision of LTGA is a GOMEA variant which uses a hierarchical cluster tree as FOS, also called a linkage tree. In this linkage tree each node represents a problem variable or a group of problem variables. In order to create a linkage tree you start off with a univariate structure and iteratively merge two sets of problem variables together until there are no sets left to merge. This merging is done using a similarity measure. In each iteration the set of variables which are most similar to each other are merged. To compute this similarity the mutual information is used in case of merging single problem variables, or using the unweighted pair group method with arithmetic mean (UPGMA) in case of merging sets of problem variables.

$$I(X,Y) = H(X) + H(Y) - H(X,Y) \tag{3.1}$$

H(X), H(Y) and H(X,Y) are the marginal and joint entropies.

$$I^{UPGMA}(F^i, F^j) = \frac{1}{|F^i||F^j|} \sum_{X \in F^i} \sum_{Y \in F^j} I(X,Y) \tag{3.2}$$

This merging of sets of problem variables creates a tree structure in a bottom-up fashion. All the individual problem variables create $n$ leaf nodes, where $n$ is the total number of problem variables. When two nodes are merged they create a parent which has the original two nodes as children. This means that at the end the root node will always contain the set of $n$ problem variables and the tree will have a total of $2n-1$ nodes. The advantage of using mutual information and the UPGMA clustering is that it can be done very efficiently. Computing the initial mutual information between all pairs of problem variables takes $\mathcal{O}(n^2)$. Once that is done merging clusters of problem variables can be done using the reciprocal nearest neighbour algorithm by Gronau and Moran [8], which is an optimal implementation of the UPGMA cluster algorithm. This takes another $\mathcal{O}(n^2)$ to complete, meaning the total running time of creating a linkage tree is $\mathcal{O}(n^2)$, which is very fast for a clustering algorithm.

Finally LTGA also has a termination criterion. There are two different criteria considered to check if the algorithm is done: The number of function evaluations and the fitness variance. Using the number of functions evaluations is straight forward. Beforehand a maximum number of evaluations is set and when this number is exceeded the search is halted. There is a caveat here however. LTGA uses an optimization that evaluations are not performed when the bits copied over from the donor are the same as in the recipient. It would be wasteful to spend a function evaluation on this example, as we know exactly what the outcome will be and that the new solution will be rejected anyway. As the population converges more and more the amount of times new potential solutions are skipped becomes more and more likely. If a population is fully converged it will end in an endless loop, as there is no possible crossover that will lead to a new function evaluation. To counter this the other criterion, the fitness variance was introduced. Fitness variance calculates the variance of all fitness values in the population. If the population is fully converged the fitness variation is also 0, but as the fitness variance only looks at fitness values, not at the bits of the solutions, it gives a more broad indication of how far the population is converged. Setting it to 0 is a robust parameter to prevent infinite loops, but it can also be set to a higher value, to quickly stop a search when the population is nearly converged.

**Algorithm 2:** Linkage Tree Genetic Algorithm

**Input**: A given population size $S$
**Output**: A population of $S$ that maximize the fitness
$population \leftarrow InitializeRandomPopulation(S)$
**while** $\neg$ *TerminationCriterion()* **do**
  $\mathcal{F} \leftarrow LearnLinkageTree(C)$
  **for** $individual \in population$ **do**
    $individual \leftarrow GOMEA(individual, C, \mathcal{F})$
  **end**
**end**
**return** *pop*

# Chapter 4

# NK-landscapes

## 4.1 Introduction to NK-landscapes

In 1993 Kauffman and Weinberger [9] introduced a new artificial problem called the NK-landscape to model certain fitness landscapes as found in problems in biology. An NK-landscape is a function $f : S \to \mathbb{R}$, where $S$ is a string of bits $\in \{0, 1\}$. NK-landscapes have two parameters, $n$, the amount of bits in $S$, and $k$, a parameter that controls with how many bits an individual bit interacts in the problem. Every bit in the string is connected to $k-1$ random bits, such that $n$ sets of $k$ bits are formed. $k$ can range from 1 to $n$, lower values for $k$ result in a smoother fitness landscape while higher values for $k$ result in a problem with a more rugged fitness landscape. Every set of $k$ bits has a fitness evaluation function which returns a value that is uniformly distributed on the interval $[0, 1]$. This value is stored in a lookup table of size $2^k$, one value for every possible combination of $k$ bits. The overall fitness is computed by taking the average over all the subfunctions. $k$ is often referred to as the degree of epistasis of the NK-landscape, a term borrowed from biology to describe interactions between different genes.

$$F(S) = \frac{1}{n} \sum_{i=1}^{n} f_i(S_i) \tag{4.1}$$

$$f_i(S_i) = f_i(S_i, S_{i,1}, S_{i,2}, ..., S_{i,k-1}) \in [0, 1] \tag{4.2}$$

$S_i$ returns the value of the $i$'th bit, and $S_{i,n}$ is the value of the $n$'th neighbour of the $i$'th bit in the string.

## 4.2 Nearest Neighbour NK-landscapes

NK-landscapes have traditionally been a popular artificial problem to compare the performance of evolutionary algorithms. It can however be argued that this type of problem is too hard compared to real life problems, since there is barely any structure. Pelikan et al. [10] introduced a new variant of NK-landscapes where the linkage of bits is determined by their position in the bitstring rather than being random. Whereas in Pelikan et al. [10] the size of the overlap of the subproblems was tuneable we will only focus on instances with maximal overlap in this thesis. These subproblems are the hardest in the class of nearest neighbour NK-landscapes and this ensures that the problems we are looking at are not becoming too easy, since this class of problems is already significantly easier than the classic NK-landscapes. Another advantage of nearest neighbour NK-landscapes is that they can be solved in $\mathcal{O}(2^k n)$ with dynamic programming. More on this in section 4.3.

In a nearest neighbour NK-landscape with maximal overlap a solution consists of $n$ bits which are either 0 or 1. Every $k$ consecutive bits in the bitstring are linked together in a group. In total there are $n - k + 1$ of such groups, meaning that every bit in the

bitstring is part of at least 1 group and at most $k$, depending on the location of the bit. Each group contributes to the final fitness of the entire bitstring and the contribution of the group is determined by the value of the $k$ bits it contains. Each of such group is also called a subproblem. This contribution is defined by a lookup table, which is unique for every subproblem. For every combination of values of $k$ bits the lookup table contains a random value between 0 and 1.

Note that with this definition the first and last bits occur in less groups and therefore their total impact on the fitness of the solution is smaller. The easiest way to counteract this is to make the bitstring cyclic. In such a cyclic problem there are $n$ subproblems of $k$ bits where subproblem near the edges of the bitstring wrap around and use bits of both the beginning and end of the string. The problem of doing this is that it becomes a lot harder to apply dynamic programming, whilst the gain is fairly small. After all this bias only occurs at the edges and does not change the structure of the problem in a significant way. Therefore we just use the simple definition with no cyclical subproblems and a total of $n - k + 1$ groups. This is the same definition that Thierens [1] uses.

The total amount of subproblems is $n - k + 1$ and for convenience the final result is scaled back to the interval $[0, 1]$ by multiplying the sum of all the subproblems by a factor $\frac{1}{n-k+1}$.

$$F(S) = \frac{1}{n-k+1} \sum_{i=1}^{n-k+1} f_i(S_i) \tag{4.3}$$

$$f_i(S_i) = f_i(S_i, S_{i+1}, ..., S_{i+k-2}, S_{i+k-1}) \in [0, 1] \tag{4.4}$$

## 4.3 Dynamic Programming

Nearest neighbour NK-landscapes can be solved efficiently using dynamic programming. The subproblems are more structured than in classic NK-landscapes and now contain an optimal substructure which can be exploited easily. The dynamic program solves a nearest neighbour NK-landscape by looking at the subproblems from left to right. Let $f_i$ denote the $i$'th subproblem and let $m = n - k + 1$ be the number of subproblems. The objective is to maximize the summation $\sum_{i=1}^{m} f_i$.

Some might assume that if $\sum_{i=1}^{m} f_i$.is optimal for a set of bits, this same set of bits is optimal for $\sum_{i=1}^{m-1} f_i$ as well. However this assumption is incorrect. An optimal solution to $\sum_{i=1}^{m-1} f_i$ is likely to have different bits at the end of the bitstring than the optimal solution to $\sum_{i=1}^{m} f_i$. The last $k - 1$ bits overlap with subproblem $f_m$ and so only fitness contributions of $f_m$ can be chosen that have the same $k - 1$ bits. The optimal value is likely to have a different set in place of these $k - 1$ bits, meaning it has a lower score for $\sum_{i=1}^{m-1} f_i$, but together with the fitness contribution of $f_m$ it has a better value for $\sum_{i=1}^{m} f_i$. This overlap needs to be taken into account when exploiting the substructure.

Since each subproblem $f_i$ only overlaps with $f_{i-k-1}, f_{i-k-2}, ..., f_{i-1}$, it is independent of the subproblems $f_1, ..., f_{i-k-2}$. This can be used by creating a matrix $A_{(i,j)}$ which stores the optimal value for the first $i$ subproblems, given that the solutions for the first $i$ subproblems ends in the bits with value $j$. For $j$ we use the integer representation of the $k - 1$ overlapping bits. The matrix has a size of $(m - 1) \times (2^{k-1})$.

The matrix is initialised by iterating over all combinations of $k$ bits at the start of the bitstring for the first subproblem and storing the maximum of $f_1$ for the last $k - 1$ bits in $A_{(1,j)}$. Note that this means, since the overlap is $k - 1$, there is only 1 bit not overlapping, so in all these cases the maximum is only a maximum over 2 possibilities. The non-overlapping bit is either 0 or 1. This is also true in the rest of the algorithm. For the consecutive subproblems $f_2, ..., f_{m-1}$ all $2^k$ fitness contributions of $f_i$ are considered.

The total fitness contribution is calculated by taking the fitness contribution of $A_{(i-1,j')}$ and adding the contribution of $f_i$ and then stored in $A_{(i,j)}$. Note that $j'$ only consists of the first $k-1$ bits and $j$ consists of the last $k-1$ bits and therefore this step involves a bit of bit shifting. For the last subproblem $f_m$ we are only interested in the global best regardless of the last bits. This is also the reason why the matrix only has $m-1$ subproblems. For the last subproblem we also iterate over all $2^k$ possible bits, but now we just keep track of the global best. In the end the running time of this algorithm is $\mathcal{O}(2^k n)$ with a memory requirement of $\mathcal{O}(2^{k-1}n)$, since $m$ is in the same order of magnitude as $n$ if $n >> k$. In the case that $k$ is in the same order of magnitude as $n$, or larger than $\sim 25-30$, then $2^k$ becomes the most dominant term and even this efficient dynamic program will become too slow or will hit memory limitations. In practice however nearest neighbour NK-landscapes are used in conjunction with much lower values for $k$.

## 4.4 Loose Linkage

When comparing the performance of certain algorithms using NK-landscapes you want to hide the underlying structure from the algorithm. For basic nearest neighbour NK-landscapes this can be a slight problem as adjacent bits are highly correlated, since they share one or more subproblems and if an algorithm has some sort of search bias this can influence the performance of the algorithm. To counter this potential bias the position of the bits in the bitstring can be shuffled randomly to hide this structure and prevent any search bias on influencing the performance of the algorithm. The underlying structure remains the same, but only the position of a bit is changed in the bitstring. This is called loose linkage. See figure 4.1 for an example. Each block represents a bit, and each subproblem is marked with a colour. Since each colour spans two bits $k = 2$. When the positions are not shuffled this is called tight linkage.

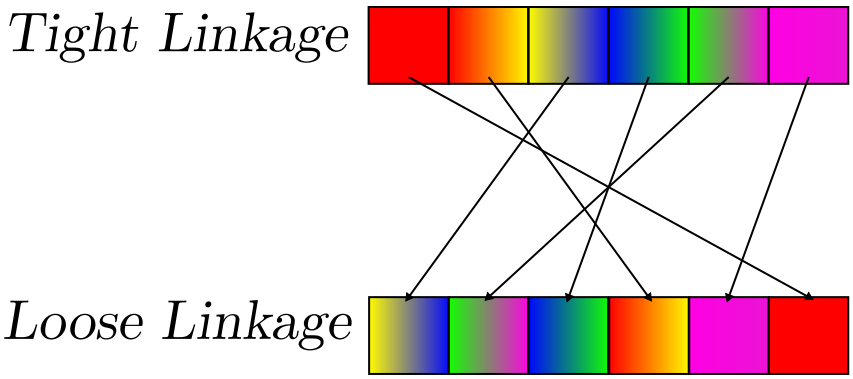


Figure 4.1: An example of loose linkage for $n = 6$ and $k = 2$.

For this thesis only the loose linkage variants are considered. Shuffling is done with the Fisher-Yates algorithm which permutes an array perfectly. Every possible permutation of an array has an equal probability of being generated. Pseudo-code for this algorithm can be found in algorithm 3.

$$F(S) = \frac{1}{n-k+1} \sum_{i=1}^{n-k+1} f_i(S_{x_i}) \tag{4.5}$$

$$f_i(S_{x_i}) = f_i(S_{x_i}, S_{x_{i+1}}, ..., S_{x_{i+k-2}}, S_{x_{i+k-1}}) \in [0,1] \tag{4.6}$$

where $x_i$ denotes the position of the $i$'th problem variable in the bitstring. For tight linkage $x_i = i$, but for problems that utilize loose linkage $x$ is a permutation of $[1, ..., N]$. The corresponding permututation for figure 4.1 would be $[6, 4, 1, 3, 2, 5]$.

**Algorithm 3:** The Fisher-Yates shuffle algorithm

**Output**: A random permutation of the array $[0, ..., N-1]$
x = [0,1,...,$N-1$]
**for** $i = 1$ **to** $N-1$ **do**
  $j \leftarrow \mathcal{U}(0, i)$  `/* Sample a random integer [0,`$i$`] (inclusive) */`
  swap x[i] and x[j]
**end**
**return** $x$

# Chapter 5

# Multi-objective benchmark functions

Commonly used multi-objective problems are Onemax - Zeromax and trap - inverse trap functions. The downside of these problems is that these type of artifical problems are often limited to only 2 dimensions and are generaly not suited to generalize into higher dimensions and secondly, their objective functions are often highly conflicting. In the case of Onemax - Zeromax every single possible solution is Pareto-optimal. While this can be interesting for investigating ways of finding a good spread of the Pareto front, it is less interesting from an optimization point of view. From an optimization point of view it might be more of interest to investigate multi-objective variants of the NK-landscape problem.

## 5.1 Trap - Inverse Trap Function

The trap - inverse trap problem is an extension to the single objective trap function into two objectives created by Pelikan et al. [11]. Trap functions are artificial problems with known optima, which can be useful for analysing and comparing the basic performance of an algorithm.

A trap function has two parameters: the problem length $n$ and the size of each "trap" $k$, where $n$ must be a multiple of $k$. Each group of $k$ consecutive bits contributes to the total fitness based on the number of 0s and 1s in each group. For single objective trap functions this sub function will return $k$ if the all the bits in the group are 1 and $k-u-1$ otherwise where $u$ is the number of 1s. Each group has 2 optima, a local optimum where all $k$ bits are 0 and a global one where all $k$ bits are 1. When we sum over all these $n/k$ groups the trap function as a whole has one global optimum where all bits are 1, and it has local optima where one or more groups contain $k$ 0s. Because of the created fitness landscape traditional algorithms are far more likely to converge to a local optimum than to the global optimum. This is due to the fact that the fitness landscape contains a slope towards the local optimum, which misguides most algorithms.

$$F(S) = \sum_{i=0}^{n/k-1} f_{trap}\left(\sum_{j=1}^{k} S_{i\times k+j}\right) \tag{5.1}$$

where

$$f_{trap}(u) = \begin{cases} k & \text{if } u = k \\ k-u-1 & \text{otherwise} \end{cases} \tag{5.2}$$

To create a two objective function a second objective is added which is the same as

the first objective, except that we substitute the function $f_{trap}$ in the following manner.

$$f_{inverse}(u) = \begin{cases} k & \text{if } u = 0 \\ u - 1 & \text{otherwise} \end{cases} \tag{5.3}$$

Notice that $f_{trap}(u) = f_{inverse}(k-u)$. The resulting two objective problem has a Pareto front which consists of ${}^{n}/_{k}+1$ points. Compared to the single objective variant every local optimum now has become Pareto optimal.

## 5.2 Multi-objective Max Cut

Max Cut is a classical NP-hard problem. In the standard single objective variant of Max Cut the goal is to partition a fully connected weighted graph $G = (V, E)$, into two disjoint subsets $X$ and $Y$, where $Y = V \setminus X$, in such a way that the sum of all edges which have one endpoint in $X$ and the other endpoint in $Y$ is maximal. $V$ contains $n$ vertices $\{v_1, ..., v_n\}$ and since it is a fully connected graph $E$ contains all edges $(v_i, v_j)$ with a corresponding weight $c_{ij}$. A solution to an instance of a Max Cut problem can be encoded as a binary string of length $n$. A 0 or 1 indicates whether a vertex is in set $X$ or $Y$ respectively. This means that each solution $S$ has an opposing solution $\neg S$, where the sets $X$ and $Y$ are effectively mirrored.

$$F(S) = \sum_{i=1}^{n} \sum_{j=i}^{n} \begin{cases} c_{ij} & \text{if } S_i \neq S_j \\ 0 & \text{otherwise} \end{cases} \tag{5.4}$$

Max Cut problems can be extended to multiple objectives by evaluating a different Max Cut instance for each objective.

## 5.3 $\rho$MNK-landscapes

Verel et al. [12] introduced multi-objective NK-landscapes with correlated objectives ($\rho$MNK). In $\rho$MNK-landscapes each entry in a lookup table now contains a vector of $m$ values, one for each dimension, instead of just being a scalar. The advantage of the $\rho$MNK problem is that with the parameter $\rho$ the correlation between the $m$ values can be determined. In most real life multi-objective problems there exists some correlation between the objectives and it is useful to be able to control the correlation with a parameter. The $m$ values are generated using a multivariate uniform distribution.

### 5.3.1 Multivariate Normal Distribution

Whilst the regular uniform distribution is a widely known probability distribution, it is not used as much as a multivariate distribution. Instead the multivariate normal distribution is often used as multivariate distribution. Hotelling and Pabst [13] show how to create a multivariate uniform distribution using a multivariate normal distribution.

A multivariate uniform distribution is generated in the following manner. First a correlation matrix $C$ is constructed of size $m \times m$ that models the correlation between each of the $m$ objectives. The correlation of an objective with itself is 1, so the diagonal of the matrix is 1, $\forall i \in \{1, 2, .., m\}\ C_{i,i} = 1$. For all other values in the matrix the correlation parameter $\rho$ is filled in. $C_{i,j} = \rho$ if $i \neq j$. There is only a limited range of values that $\rho$ can have since not every matrix is a valid correlation matrix. This can be seen in the following example. Imagine a multivariate uniform distribution on the range $[0, 1]$ with 3 dimensions where the correlation between all the different dimensions equals $-1$. If we

now sample from this distribution once we get 3 values $a_1$, $a_2$ and $a_3$. Since the correlation is $-1$ between $a_1$ and $a_2$, $a_2 = 1 - a_1$. The same can be argued about the relation between $a_1$ and $a_3$ and $a_2$ and $a_3$. $a_3 = 1 - a_1 \wedge a_3 = 1 - a_2$. However if we now substitute $a_2 = 1 - a_1$ we get the following contradiction: $a_3 = 1 - a_1 \wedge a_3 = a_1$. This can only be true if $a_1 = a_2 = a_3 = \frac{1}{2}$, but that also means that this is no longer a uniform distribution. Clearly it is impossible to have such a negative correlation for 3 objectives.

A matrix can only be a correlation matrix if it is positive semi-definite and in order to be positive semi-definite the value of $\rho$ must range between $\frac{-1}{M-1}$ and 1. It is also possible to manually create a positive semi-definite correlation matrix where the entries $C_{i,j}$ $i \neq j$ are not uniform, but for the sake of simplicity this is left out and only a single parameter $\rho$ is used. To create a multivariate uniform distribution a multivariate normal distribution $\mathcal{N}_m(0,1)$ is created first with a correlation matrix of $\mathcal{R} = 2\sin\frac{\pi}{6}C$. If we take a sample $(X_1, ..., X_m) \sim \mathcal{N}_m(0,1)$ then $\Phi(X_i)$ is multivariate uniformly distributed on the interval $[0,1]$ with correlation matrix $C$ where $\Phi$ is the cumulative distribution function of a regular univariate normal distribution $\mathcal{N}(0,1)$.

$$F^m(S) = \frac{1}{n}\sum_{i=1}^{n} f_i^m(S_i) \tag{5.5}$$

$$f_i^m(S_i) = f_i^m(S_i, S_{i,1}, S_{i,2}, ..., S_{i,k-1}) \in [0,1] \tag{5.6}$$

Where $\forall i$ $f_i^m(S_i)$ is multivariate uniformly distributed on the interval $[0,1]$. In this model the linkage structure remains the same in each dimension. Otherwise it would not be possible to generate a correlated $m$-dimensional vector. Each bit has the same neighbourhood in each dimension.

## 5.4 $\rho$MNNK-landscapes

For this thesis we are however more interested in the nearest neighbour NK-landscape family of problems, since these problems have more structure than the classic NK-landscape. Therefore I introduce the nearest neighbour variant, the $\rho$MNNK problem. In the same way the original $\rho$MNK is a generalisation of the NK-landscape, the $\rho$MNNK is a generalisation of the nearest neighbour NK-landscape. Instead of each value in the lookup table being a scalar, it is now an $m$-dimensional vector generated using a multivariate uniform distribution.

$$F^m(S) = \frac{1}{n-k+1}\sum_{i=1}^{n-k+1} f_i^m(S_i) \tag{5.7}$$

$$f_i^m(S_i) = f_i^m(S_i, S_{i+1}, ..., S_{i+k-2}, S_{i+k-1}) \in [0,1] \tag{5.8}$$

Where $\forall i$ $f_i^m(S_i)$ is multivariate uniformly distributed on the interval $[0,1]$.

## 5.5 $\lambda$MNK-landscapes

The drawback of $\rho$MNNK-landscapes is that their structure is identical in each part of the solution space. In 4.4 we saw the concept of loose linkage. A different approach to making a multi-objective variant of a nearest neighbour NK-landscape is not to change the lookup table, but rather use the concept of loose linkage to create a multi-objective variant. With loose linkage the position of the individual bits is shuffled, but the underlying structure remains untouched. Subsection 4.4 introduced a problem parameter $x$ that mapped problem variables to positions in the bitstring. A multi-objective problem can be created by using the same underlying problem for each objective function, but construct $x$ slightly different. If we randomly construct an $x^m$ for each objective the resulting problem will

have almost no correlation between the objectives. With the $\rho$MNNK-landscapes we have a control parameter $\rho$ with which we can control the correlation between the objectives. It would be nice if we could add a control parameter $\lambda$ to control how different the shufflings are across different objectives.

$$F^m(S) = \frac{1}{n-k+1} \sum_{i=1}^{n-k+1} f_i(S_{x_i^m}) \tag{5.9}$$

$$f_i(S_{x_i^m}) = f_i(S_{x_i^m}, S_{x_{i+1}^m}, ..., S_{x_{i+k-2}^m}, S_{x_{i+k-1}^m}) \in [0,1] \tag{5.10}$$

## 5.6 Changing the structure for each objective

To create the loose linkage structure for the single objective problem we used the Fisher-Yates algorithm as described in figure 3. We adapt this algorithm so it returns $m$ different permutations, one for each objective function. Instead of swapping element $i$ with element $j$, where $0 \leq j \leq i$, we first generate $j'$ using a continuous uniform distribution on the interval $[0, i+1)$. For each objective we add a small distortion $r_m$ to $j'$ which is randomly generated from a normal distribution $r_m \sim \mathcal{N}(0, \lambda)$. With the parameter $\lambda$ we can control how similar the resulting permutations will be. $r_m + j'$ is rounded downwards. It is now possible that $\lfloor r_m + j' \rfloor < 0$ or $\lfloor r_m + j' \rfloor > i$, so we need to wrap it around so the result is still in the range $[0, i]$. $j_m = \lfloor r_m + j' \rfloor \ \% \ i$. Wrapping around is preferred over clamping, since in this way the result is still discretely uniformly distributed in the range $[0, i]$, which would not be the case if we clamped the result to $[0, i]$. Furthermore if $\lambda$ was very large the clamping would create artifacts as all results would be clamped to either 0 or $i$. The $m$ created permutations individually are perfectly random, but if $\lambda$ is small the permutations still share a lot of structure with each other. In this manner all the objective functions will be correlated with each other since the underlying permutations are similar. Pseudocode for this algorithm can be found in algorithm 4.

**Algorithm 4:** Adapted Fisher-Yates shuffle algorithm

**Input**: Parameter $\lambda$ to control how similar the $m$ permutations are
**Output**: A $m$-dimensional array where each row contains a permutation of the array $[0, ..., N-1]$

```
for m = 0 to M do
    x[m] = [0,1,...,N − 1]            /* Initialise the permutations for each
    objective. */
end
for i = 1 to N − 1 do
    j' ← U(0, i + 1)        /* Sample a random float [0,i + 1] (exclusive) */
    for m = 0 to M do
        r ← N(0, λ²)
        j ← ⌊j' + r⌋ % i
        swap x[m][i] and x[m][j]
    end
end
return x
```

# Chapter 6

# Experimental Setup

For the experiments a variety of problems is used. As a reference problem multi-objective Max Cut is used and to study the influence of linkage learning the multi-objective nearest neighbour NK-landscape with correlated objectives ($\rho$MNNK) and the nearest neighbour NK-landscape with multidimensional linkage ($\lambda$MNK) are used.

Instances of problems are randomly generated with a predetermined seed. For each experiment if the problem specific parameters are the same, the generated instances will also be the same. This ensures that different algorithms are run on the exact same problem instances. Across different experiments this seed is not guaranteed to be the same and the problem instances might be different even though the problem specific parameters are the same.

Unless specified otherwise instances will be created in the following manner: Generated problems have a problem size $n$ of 100 and are 3-dimensional, i.e. they will have 3 objective functions. The only exception to this is are the instances of the trap - inverse trap function as the number of objectives is nonconfigurable and is 2-dimensional by design. The degree of epistasis $k$ for $\lambda$MNK and $\rho$MNNK instances is 5. $\rho$MNNK instances will be generated with $\rho = \{-0.4, 0, 0.4\}$. For $\lambda$MNK instances $\lambda$ is set to 0.3. The experiment in section 9.3.1 will prove this to be a reasonable value for $\lambda$.

LTGA and the various MO-LTGA variants which will be introduced further on have a large number of different parameter settings, each of which can have an effect on how other parameters affect performance. It is an optimization problem in its own to find the optimal set of algorithm parameters. The chosen set of parameters aims to provide decent performance. Even though the tests are presented in linear fashion, the entire process has been an iterative and results from one experiment influenced parameter settings in others. An example of this is that the cluster size is looked at in experiment 9.3.2, but the experiment that establishes the population size that it uses is done later in experiment 9.3.3.

Unless specified otherwise the default parameters are as follows: MO-LTGA uses a population size of 1600. MO-LTGA with clustering uses 5 clusters. All algorithms are allowed to use a maximum of $10^6$ function evaluations. Some algorithms like scalarized LTGA and MO-LTGA may terminate prematurely if the population has converged and therefore not necessarily use up the entire function evaluation budget.

For all the tested algorithms an elitist archive is maintained against which newly generated solutions are compared, and included if they are Pareto optimal in regards to the solutions in the archive. For this archive the quadtree structure by Mostaghim et al. [4] is used. In appendix A an experiment is done to make sure this has no significant impact on the algorithms.

The hypervolume indicator described in section 2.3 is used as the quality metric to measure how good a local Pareto front produced by an algorithm is. Technically a hypervolume is the $m$-dimensional volume if $m$ is greater than 3, for 3 dimensions this would

be called a regular volume and for 2 dimensions it is just a regular area. Even though all benchmark problems are 2 or 3-dimensional the y-axis is labeled as hypervolume to reflect the fact that the algorithms and methods used work for any number of dimensions. It would be confusing to having the y-axis labeled differently depending on the dimensionality of the problem.

## 6.1 Pooled Variance

For every instance the algorithm is run 5 times to average out the results. For every set of parameters there is also an additional problem since not all instances are created equally. Some instances might be slightly easier than others resulting in better results, or the other way around. To average these variations out 5 different runs are done for every set of parameters, which means that the total number of runs per set of parameters is 25. An average can be obtained by averaging the values of all 25 runs. This cannot be done directly for the standard deviation. If we took the standard deviation over the 25 values we would not only capture any variance in the individual runs of the algorithms, but also variance in difficulty of the instances. While we do want to obtain an average over the multiple instances we are not directly interested in the variance, we are more interested in the variance of the algorithms given an instance. When comparing algorithms they are tested on the same instances, meaning their measurements are paired. Since we want to know if algorithm A outperforms algorithm B, or if it does not, the variance in problem difficulty is irrelevant. To calculate the variance of the algorithm on a given instance we use pooled variance. Pooled variance estimates the variance of a set of populations and assumes that the variance within each population is the same.

$$s_p^2 = \frac{\sum_{i=0}^{n} s_i^2}{\sum_{i=0}^{n} n_i - 1} \tag{6.1}$$

$s_p^2$ is the pooled variance of all the populations, while $s_i^2$ and $n_i$ are the variance and size of the individual populations respectively.

# Chapter 7

# Multi-objective Linkage Tree Genetic Algorithm

## 7.1 Adapting LTGA

In order to adapt LTGA to the multi-objective environment several parts of the original algorithm need to be adjusted. The approach taken in this thesis is as follows, in LTGA the best solution found so far is guaranteed to be in the population as it can only be replaced by solutions that are better. For multi-objective problems there is no guarantee on the size of the Pareto front, yet having a variable population size is also complicated for EAs. Therefore many multi-objective EAs using a population also store the local Pareto front in an archive $\mathcal{A}$. This archive contains all local Pareto optimal points that are encountered during the search and by storing them in a separate archive it can be guaranteed that local Pareto optimal points which are encountered during the search are never thrown out.

There is no strict ordering of solutions anymore, instead there is a partial ordering based on the Pareto dominance relation. In single-objective LTGA the acceptance criterion is that during the optimal mixing step a mutation needs to be better, or equal to, than the original. For MO-LTGA this criterion is adapted to the following: During optimal mixing, a mutated offspring solution $o'$ is selected over the original $o$ if and only if $o' \succ o$ or $o'$ is a local Pareto point for which $\nexists x \in \mathcal{A},\ x \succ o'$, meaning $o'$ dominated $o$ or there is no point in the archive that dominates it respectively.

The third adaptation that is done is that the linkage tree is no longer learnt over the entire population, but only of a subset. The structure of the problem can be different for each objective. Therefore different parts of the objective space should be optimized differently. To do this only a subset of the population is optimized in each generation. This has as a downside that some solutions in the population may remain stale for one or more generations, but each generation is done more quickly as less solutions are involved. Each generation a subset of the population of size $t$ is created by selecting a random individual $i$ and adding the $t-1$ closest points using k-nearest-neighbour where the euclidean distance in objective space is used as the distance metric. Since $t$ itself is an inconvenient parameter, it is defined as a percentage of the population size. $t = c_{ratio} \times S$. $S$ is the population size and $0 < c_{ratio} \leq 1$ is the ratio of the population which is used in each generation for selection and recombination.

Lastly, the termination criterion is changed. In single-objective LTGA there are two criteria considered: The number of function evaluations and the fitness variance. The number of evaluations still works for MO-LTGA, but the fitness variance is different. Calculating any kind of variance over multiple dimensions is tricky. The main goal of fitness variance is to prevent an infinite loop. The easiest solution to do this for MO-LTGA is to look at the amount of function evaluations used in each generation. If no function evaluations are done in an entire generation MO-LTGA is terminated. When this

happens this only means that the subset from which that particular generation was chosen was fully converged, and this might not necessarily be the case for the entire population. It is however quite likely that if a part of the population is fully converged, that the rest of the population is also quite far in the converging process, even though this remains a heuristic.

The pseudocode for the multi-objective adaptation of optimal mixing can be found in algorithm 5 and the pseudocode for MO-LTGA can be found in algorithm 6.

**Algorithm 5:** Multi-objective Optimal Mixing

**Input**: An individual $o$, a subset of the population $C$, a FOS $\mathcal{F}$ and elitist archive $\mathcal{P}$.
**Output**: An improved offspring solution $o$.
$changed \leftarrow false$
**for** $F^i \in \mathcal{F}$ **do**
  $o' \leftarrow o$
  $p \leftarrow Random(C)$ `/* Random donor from the population */`
  $o'_{F^i} \leftarrow p_{F^i}$ `/* Copy over the bits from donor to child */`
  $addedToArchive \leftarrow updateArchive(\mathcal{P}, o)$
  **if** $o' \neq o$ ***and*** ( $o' \succ o$ ***or*** $addedToArchive$ ***or*** $f(o') = f(o)$) **then**
    $changed \leftarrow true$
    $o \leftarrow o'$
  **end**
**end**
**return** $o$

**Algorithm 6:** Multi-objective Linkage Tree Genetic Algorithm

**Input**: A given population size $S$ and a clustering ratio $c_{ratio}$
**Output**: A local Pareto set $\mathcal{P}$.
$\mathcal{P} \leftarrow \emptyset$
$population \leftarrow InitializeRandomPopulation(S)$
**for** $i \in population$ **do**
  $updateArchive(\mathcal{P}, i)$
**end**
**while** $\neg$ *TerminationCriterion()* **do**
  $k \leftarrow selectRandomIndividual()$
  $C \leftarrow selectNearest(k, S \times c_{ratio})$
  $\mathcal{F} \leftarrow LearnLinkageTree(C)$
  **for** $individual \in C$ **do**
    $individual \leftarrow MO\text{-}Optimal\text{-}Mixing(individual, C, \mathcal{F}, \mathcal{P})$
  **end**
**end**
**return** $\mathcal{P}$

## 7.2 Experiment of MO-LTGA with sampling

In this experiment we look at the performance of MO-LTGA on a set of benchmarking functions. In this experiment a closer look is taken at the clustering ratio $c_{ratio}$ parameter. Since the size of the cluster is defined as a percentage of the population size values are tested in the range $[0.05, ..., 1]$, meaning $1/20$ the size of the population to the entire population as cluster size. All other parameters have their default value unless specified otherwise. In the experiment MO-LTGA with linkage learning is denoted as MO-LTGA-

LL, where LL stands for linkage learning. The variant that uses a random linkage tree is denoted by MO-LTGA-RL where RL stands for random linkage.

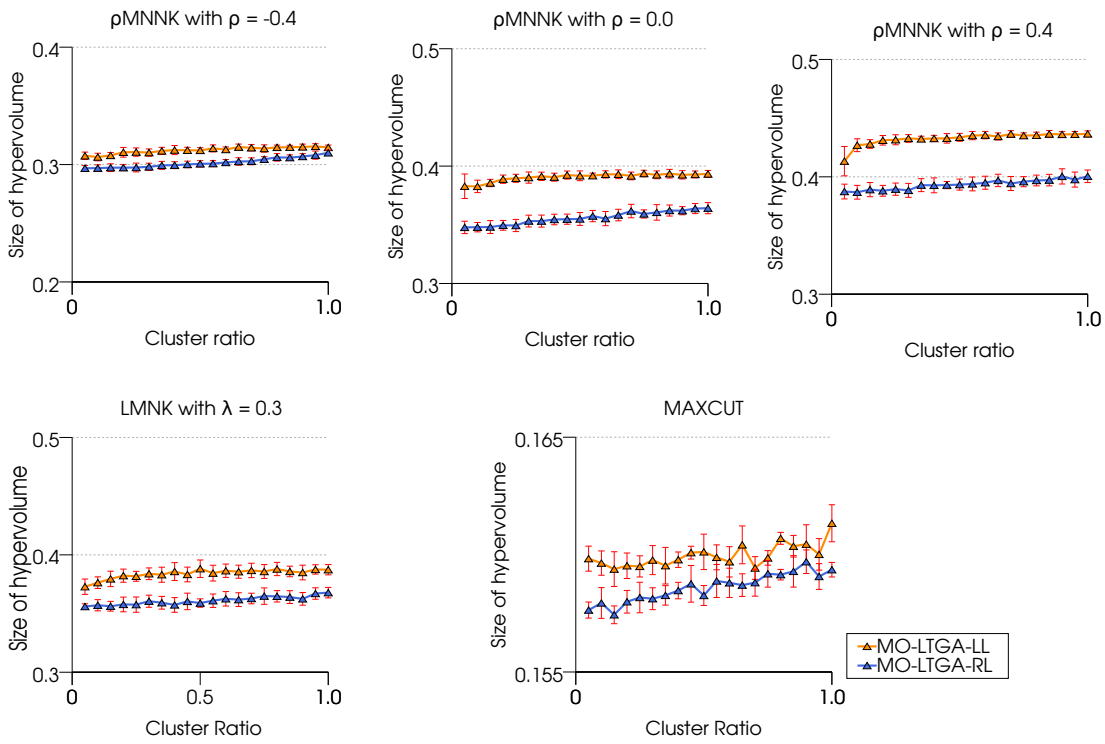


Figure 7.1: MO-LTGA on a variety of benchmarking problems with varying values for $c_{ratio}$.

As can be seen in figure 7.1 generally the cluster ratio does not have a large impact on the performance of MO-LTGA regardless of the type of problem. For all problems the performance increases very slightly for both the linkage learning and random linkage variants as the cluster ratio becomes larger. Since the random linkage variant generates a random linkage tree, the only thing that changes with a different cluster ratio is the fact that with a lower cluster ratio solutions are combined that are closer together in objective space compared to a higher cluster ratio where solutions are combined that are farther away. The only graph that stands out is Max Cut. However for Max Cut the horizontal scale is a lot more zoomed in compared to the other problems. Linkage learning and random linkage are much closer together in terms of the size of the hypervolume and the figure is much more zoomed in, which is why the variances seem much larger. Overall for Max Cut linkage learning seems less effective in comparison to $\rho$MNNK and $\lambda$MNK problems as for many data points the difference in performance is barely significant.

With linkage learning a linkage tree is learnt, but there is a tradeoff here. A lower cluster ratio means there is less information to built a tree from, but the resulting tree is a reflection of the structure in that particular area. One might expect that a cluster ratio which is too low would result in bad results because there is too little information, and that a cluster ratio which is too large would degrade performance because no local information is exploited. Apparently this is not true, since the performance curve of linkage learning is similar to that of random linkage. It is quite odd there does not seem to be any large difference in performance for varying cluster ratio's. It might be that the stochastic process of picking a random point as the center of a cluster and doing a k-nearest neighbour each iteration has some side effects which could explain this behaviour.

## 7.3 MO-LTGA with sampling: Experimental Study

In order to check whether the usage of a cluster ratio has any negative effect on the performance of MO-LTGA a comparison can be made with other similar algorithms. A different approach to extending LTGA for multi-objective problems was done by Luong et al. [2]. Their paper include benchmarks against more common benchmarking functions such as the trap - inverse trap function with a variety of algorithms. The trap - inverse trap function is described in 5.1 and it is a problem with known optima. Figure 7.2 is an excerpt from Luong et al. [2] showing the increase in the numbers of evaluations needed for solving trap - inverse trap functions with an increasing number of bits.

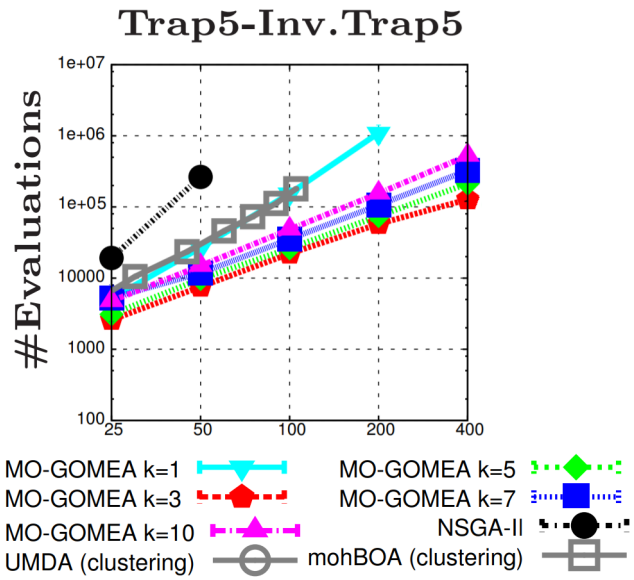


Figure 7.2: Comparison of MO-GOMEA with varying cluster sizes with other mainline multi-objective evolutionary algorithms

All variants of MO-GOMEA except for $k = 1$ are able to solve all trap functions to optimality. Other algorithms like the Nondominated Sorting Genetic Algorithm II (NSGA-II) by Deb et al. [14] and the multi-objective hierarchical Bayesian optimization algorithm (mohBOA) by Pelikan et al. [11] struggle with larger problem instances and are unable to solve instances beyond sizes 50 and 100 respectively. As both axes are logarithmic it can be seen that the number of function evaluations required by all MO-GOMEA variants with $k \geq 3$ scales polynomially with the problem size.

We can run the same experiment for MO-LTGA. The setup for this test is similar to the experiment by Luong et al. [2]. For each problem instance the algorithm is given a maximum of 10 times the number of evaluations a single objective variant of LTGA would need to solve the given instance. The population size used is decided by using bisection. The graph shows the number of evaluations used for the smallest possible population size which can still solve the trap - inverse trap problem consistently in 100 out of 100 runs.

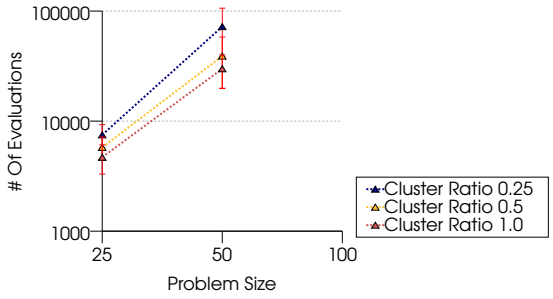


Figure 7.3: Comparison of MO-LTGA with a varying cluster ratio

The results shown in figure 7.3 are quite clear. Using cluster ratios MO-LTGA is only able to solve problem instances of sizes $n \in \{25,\ 50\}$. For $n = 100$ MO-LTGA is not able to solve the trap function anymore. For $n = 50$ MO-LTGA already needs about as many evaluations as MO-GOMEA needs for $n = 100$. This shows that using a cluster ratio is asymptotically worse than using dedicated clustering in objective space for this problem.

There are a couple of reasons why using cluster ratios leads to such poor results.

- A random point is selected and the $t - 1$ closest points are added to the cluster. Outliers are therefore less likely to be included in such a cluster as they have less points close to them. The points that are most likely to be included in the cluster are points that are closer to the middle in objective space and points that are in hotspots. This causes the search to be biased towards the area of the objective space that is the most crowded even though this is not necessarily an area where the most progress can be made.
- Each generation a different part of the population is optimized. Even though on average a solution is once every $S/t$ generations there will be a high variance between solutions because of the behaviour as mentioned previously. Some solutions are included more often than others which will effectively decrease the effectiveness of the population.

As the approach taken by MO-GOMEA to create a multi-objective variant of LTGA seems to produce better results it might be useful to take some inspirational cues from MO-GOMEA in order to improve MO-LTGA. In the next chapter we will take a look at how MO-GOMEA works and how it can be used to improve MO-LTGA.

# Chapter 8

# Multi-objective GOMEA

A different approach to extending LTGA for multi-objective problems was done by Luong et al. [2] with the Multi-objective Gene-pool Optimal Mixing Evolutionary Algorithm (MO-GOMEA). Their research was considered as a starting point, but it was decided not to take this algorithm as a starting point as the main goal of this thesis is to look at the effectiveness of linkage learning for multi-objective problems, and at first MO-GOMEA seems overly complex. MO-GOMEA performs very well against NSGA-II, but LTGA-like algorithms can also perform very well on problems without any structure. Optimal mixing can still work very well, even without a good FOS model. In that case it just performs random crossover in a local hill climbing style, which for many problems makes it still a very competitive algorithm. We can use MO-GOMEA however as a reference as the paper includes benchmarks on baseline multi-objective problems.

## 8.1 Multi-objective GOMEA

MO-GOMEA differs in a number of ways compared to traditional LTGA. Like many multi-objective algorithms it keeps an archive of all non-dominated solutions ever found, called the elitist archive. Since this elitist archive has no bound it can potentially be larger than the population size MO-GOMEA uses while searching. This means that solutions which are in the archive may be evicted from the population in favour of a new local Pareto optimal solution. Since every solution that makes it into the population is also added to the archive, the archive is always equal to or strictly better than the population. When the algorithm finishes this elitist archive is returned as solution instead of the current population.

In regular LTGA a linkage tree is learnt over a selection of the population, where the selection is determined by tournament selection. In a MOP different parts of the search space may need to be treated separately. The linkage structure in one region may be completely different from another region. MO-GOMEA does this by clustering the solutions in the population based on their position in objective space. For each cluster a separate linkage structure is learnt. In LTGA it is standard to learn the linkage tree over a subset of the population. This subset is selected through tournament selection. Since tournament selection is not trivial to do in a MOP this also needs some modification, however exactly how the tournament selection is done in MO-GOMEA is not mentioned by Luong et al. [2]. Once the model is learnt for a given cluster recombination is done. Recombination of solutions also happens within clusters, so solutions are only recombined with other solutions which are part of the same cluster. Clustering is done using the balanced k-leader-means clustering algorithm.

The balanced k-leader-means algorithm is an adaptation of the k-means clustering algorithm. In regular k-means clustering the initial $k$ clusters are selected randomly. To ensure a good initial spread of clusters balanced k-leader-means starts of by selecting an

extreme point in one of the $m$ objectives. The next leader is the point that is the furthest away from the initial leader. Then the third leader is chosen as the point that is the furthest away from the first two leaders. This process is repeated until $k$ leaders are chosen and regular k-means clustering is performed.

After the k-means clustering is done there is no guarantee how large each cluster is. To ensure that each cluster has a minimum of $t = \frac{2}{k}|P|$ solutions, where $|P|$ is the population size. Clusters which are too small are expanded to include their nearest $t$ points. Clusters that are too large are trimmed down to $t$ points by removing the points that are the furthest away. This means that there is overlap between clusters, an average solution is part of at least 2 clusters, but it is also possible solutions are not part of any cluster at all.

For extreme clusters, clusters that have the most extreme value in at least one particular objective, classic single-objective optimization is performed. Multi-objective algorithms struggle to optimize these extreme regions of the objective space and therefore single-objective optimization is preferred.

Finally the acceptance criterion is changed for the non-extreme clusters. In single-objective optimization LTGA accept solutions that have equal or better fitness. MO-GOMEA accepts a solution $o'$ over $o$ if $o'$ dominates $o$, denoted $o' \succ o$, if $o'$ is accepted into the elitist archive, or if the fitness is the same in all dimensions (but the bits are different, this is similar to accepting equal solutions in single-objective).

## 8.2 Forced Improvement

To improve certain types of problems like Max Cut forced improvement has been added to the LTGA and GOMEA family of algorithms as an optional feature. The first version of forced improvement was introduced by Bosman and Thierens [15]. When optimal mixing is unable to improve a solution, for example when there are large plateaus in the search space, forced improvement tries to improve upon the current solution by mixing the current solution with the best solution in the population. Forced improvement is only used selectively, since mixing solutions with one parent solution exclusively decreases diversity. The only situations in which forced improvement is used are:

- When after one round of optimal mixing, the solution is unchanged.
- When after $1 + \lfloor \log_{10} n \rfloor$ generations the best solution in the population has not changed. The number of generations for which the best solutions has not changed is also called the no-improvement stretch (NIS)

Whilst forced improvement is not used for every problem, Luong et al. [2] included it in MO-GOMEA. In MO-GOMEA forced improvement is done with a random solution from the elitist archive for each FOS subset, instead of the best solution for all FOS subsets. Furthermore the no-improvement stretch is redefined as the number of generations in which the elitist archive remains unchanged. When forced improvement is still unable to improve the solution, the solution is replaced by a random solution from the elitist archive.

The pseudocode for multi-objective optimal mixing can be found in algorithm 7 and the MO-GOMEA algorithm can be found in algoritm 8.

**Algorithm 7:** Multi-objective Optimal Mixing

**Input**: An individual $o$, a cluster $C$, a FOS $\mathcal{F}$ and elitist archive $\mathcal{P}$.
**Output**: An improved offspring solution $o$.

```
changed ← false
for F^i ∈ F do
    o' ← o
    p ← Random(C)                          /* Random donor from the cluster */
    o'_{F^i} ← p_{F^i}                     /* Copy over the bits from donor to child */
    addedToArchive ← updateArchive(P, o)
    if o' ≠ o and ( o' ≻ o or addedToArchive or f(o') = f(o)) then
        if addedToArchive or o' ≻ o then
            NIS ← 0                        /* Only if it is strictly better. */
        end
        changed ← true
        o ← o'
    end
end
                                           /* Do forced improvement if necesary */
if ¬changed or NIS > 1 + ⌊log10 n⌋ then
    changed ← false
    for F^i ∈ F do
        p ← Random(P)                      /* Random donor from the elitist archive */
        o'_{F^i} ← p_{F^i}                 /* Copy over the bits from donor to child */
        addedToArchive ← updateArchive(P, o)
        if o' ≠ o and ( o' ≻ o or addedToArchive or f(o') = f(o)) then
            if addedToArchive or o' ≻ o then
                NIS ← 0                    /* Only if it is strictly better. */
            end
            changed ← true
            o ← o'
        end
    end
end
if ¬changed then
    o ← Random(P)
end
return o
```

**Algorithm 8:** Multi-objective Gene-pool Optimal Mixing Evolutionary Algorithm

**Input**: A given population size $S$
**Output**: A local Pareto set $\mathcal{P}$.
$\mathcal{P} \leftarrow \emptyset$
$NIS \leftarrow 0$
$population \leftarrow InitializeRandomPopulation(S)$
**for** $i \in population$ **do**
  $updateArchive(\mathcal{P}, i)$
**end**
**while** $\neg$ *TerminationCriterion()* **do**
  $C \leftarrow performClustering(\mathcal{P}, k)$
  **for** $C_k \in C$ **do**
    $\mathcal{F}_k \leftarrow LearnLinkageTree(C_k)$
  **end**
  **for** $individual \in population$ **do**
    $l \leftarrow getCluster(C, individual)$
    **if** $isExtremeCluster(l)$ **then**
      $individual \leftarrow SO\text{-}Optimal\text{-}Mixing(individual, C_l, \mathcal{F}_l, \mathcal{P})$
    **else**
      $individual \leftarrow MO\text{-}Optimal\text{-}Mixing(individual, C_l, \mathcal{F}_l, \mathcal{P})$
    **end**
  **end**
**end**
**return** $\mathcal{P}$

In order to improve the performance of MO-LTGA it may prove fruitful to incorporate some aspects of MO-GOMEA into MO-LTGA to solve its performance problems. Chapter 9 will focus on these adaptions.

# Chapter 9

# Multi-objective Linkage Tree Genetic Algorithm with clustering

As shown in chapter 7 the approach taken with MO-LTGA is inferior to a clustering approach. To make MO-LTGA more competitive, as it will also be compared to other algorithms, and make sure that the sampling/clustering method does not prevent the algorithm from performing decently, MO-LTGA is adapted to be more like MO-GOMEA. As there is no source code available for MO-GOMEA and from the paper it is not 100% clear how everything is implemented, like the multi-objective tournament selection, the implementation is slightly different from MO-GOMEA.

Three features of MO-GOMEA are carried over to MO-LTGA:

- Use clustering with a fixed amount of clusters
- Perform single-objective optimization in extreme clusters
- Forced improvement

The clustering method MO-GOMEA uses is described in section 8.1. MO-LTGA will use a slightly different clustering method. Ideally when partitioning a population of size $S$ among $c$ clusters, each cluster has exactly $\frac{S}{c}$ clusters. MO-LTGA solves this problem by modifying the standard k-means clustering algorithm in a way that clusters have a fixed size. The reason for this is that with MO-GOMEA's clustering method a solution can be part of multiple clusters. In LTGA an offspring solution replaces its parent solution if it is equal or better. However this will lead to problems if a solution is part of multiple clusters. The same parent will be used for recombination multiple times, but what happens if another cluster already threw that parent out of the population? It is not clear how this is handled in the MO-GOMEA paper. To avoid this scenario a clustering method is used that divides the population in such a way that each solution is part of exactly one cluster.

In standard k-means clustering, each iteration a new mean is calculated and points are assigned a cluster which mean they are closest to. This process is repeated until the clusters are stable, but there is no guarantee on the final size of each cluster. In order to ensure the size of each cluster, when assigning points to a cluster, a point is only allowed to be added to that cluster if the size of that cluster does not exceed $\frac{S}{c}$. Otherwise that point will be assigned to the next closest cluster that does not exceed its size.

This method does have two drawbacks however. Firstly the algorithm is no longer guaranteed to converge to a stable arrangement. Without restrictions on cluster size with k-means clustering the mean squared error is guaranteed to decrease or remain equal. Now with this new size restriction the algorithm will stop as soon as the mean squared error stops decreasing. Secondly as the algorithm just fills a cluster until it is full, a situation might occur where a point $p_1$ could get assigned a cluster $\alpha$ and fill it up and push a next

point $p_2$ to cluster $\beta$, even though $p_1$ is closer to $\beta$ than $p_2$.

For extreme clusters single-objective optimization is performed like MO-GOMEA. Even if a cluster is the extreme cluster in multiple objectives, single-objective optimization can only be done in one objective at a time. If such a scenario occurs one objective is selected randomly to perform single-objective optimization in for that particular generation.

The implementation of forced improvement is identical to MO-GOMEA's as described in section 8.2. The pseudocode for MO-LTGA is identical to MO-GOMEA's as described in algorithm 8 as the differences between the two algorithms are contained within the various subroutines.

## 9.1 MO-LTGA with clustering: Experimental Study

In this experiment MO-LTGA with clustering is compared to MO-LTGA with sampling. The setup for this experiment is identical to the experiment in section 7.3. Figure 7.3 is included as reference.

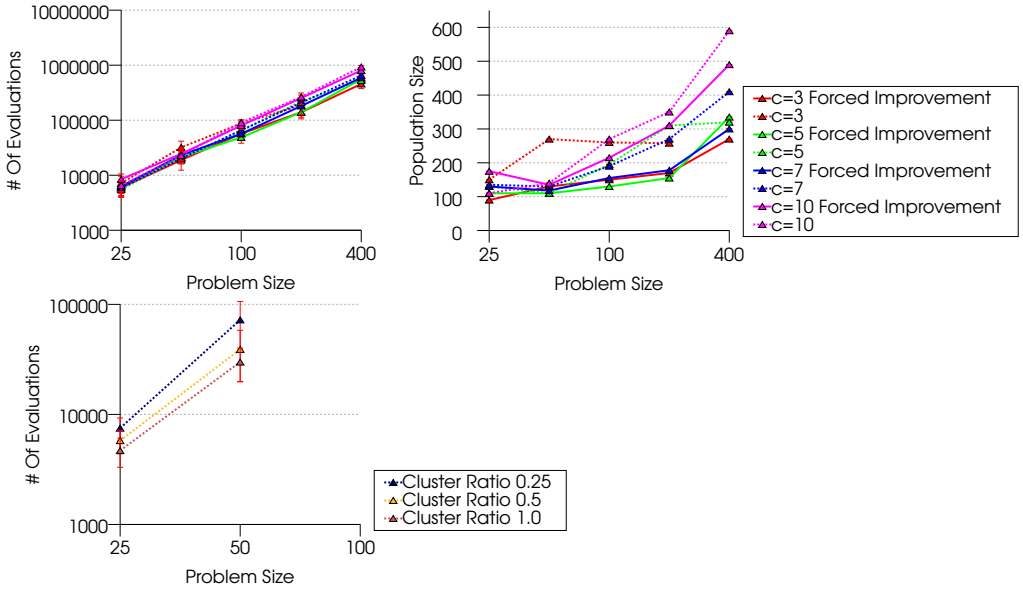


Figure 9.1: Performance of MO-LTGA with clustering with a varying amount of clusters with and without forced improvement compared to MO-LTGA with sampling

Figure 9.1 shows both the number of evaluations required as well as the resulting population size needed as found by the bisection process. The performance of MO-LTGA with clustering is much more in line with MO-GOMEA. With the exception of MO-LTGA with 3 clusters for a problem size of 400 without forced improvement, all variants are able to solve the trap - inverse trap functions up to the maximum size of 400. The general performance characteristics of all variants are the same. The number of function evaluations required scales polynomially with the problem size. When comparing the number of evaluations to MO-GOMEA as shown in figure 7.2 we can see that MO-LTGA requires more function evaluations than MO-GOMEA, but it is still in the same order of magnitude. Just like MO-GOMEA the number of clusters has no large impact, although it seems that more clusters require slightly more function evaluations.

When looking at the population sizes things get more interesting. Luong et al. [2] includes a similar graph and the population size required by variants of MO-GOMEA is between 40 for $n = 25$ to 100 for $n = 400$. If we take a look at the population sizes for MO-LTGA the picture looks very different. The required population sizes ranges from in between 90 and 200 for $n = 25$ up to 250 to 600 for $n = 400$.

Lastly if we look at how forced improvement affects performance it appears to not have a significant influence on the number of required function evaluations, but it does lower the population size needed.

#### Conclusions

One clear conclusion that can be taken from figure 9.1 is that using cluster ratios is inferior to using dedicated clustering. Cluster ratios perform asymptotically worse than clustering. The stochastic process of picking a point in the objective space and performing linkage learning and recombination in that part of the objective space iteratively without ensuring a good spread over the objective space across multiple iterations performs worse than just dedicated clustering. Using a variant of balanced k-leader-means clustering has the same performance characteristic as MO-GOMEA, with and without forced improvement. The only difference is that MO-LTGA needs a larger population as the problem size increases, whilst MO-GOMEA barely sees any increase in population size. Forced improvement does not change the performance in any significant fashion. Forced improvement was initially added to improve the performance of LTGA-like algorithms on MAXCUT by Bosman and Thierens [15]. As it has no significant impact it is better to exclude from the algorithm. It is yet another special corner case feature and it is better to keep the algorithm as simple as possible.

MO-GOMEA's clustering algorithm performs slightly differently. The clustering algorithm that MO-GOMEA uses performs a balanced k-means clustering and afterwards each cluster is expanded to a size of $t = \frac{2}{k}S$, where $S$ is the population size. MO-LTGA adapts the balanced k-means clustering itself to ensure the cluster sizes are of a certain size during the balancing, as explained in the beginning of this chapter. This might explain some of the remaining differences in performance and the lack of increase in needed population size, but further investigation would need to be done. This is out of scope for this thesis.

Although there are some differences left between MO-LTGA and MO-GOMEA they are not of major concern at this moment. The goal is not to have the best multi-objective evolutionary algorithm, but to have a good enough algorithm to analyze the effectiveness of linkage learning. The approach taken by MO-LTGA with sampling had some major flaws which prohibited it from being effective enough to be used. Now with MO-LTGA with clustering we can take a look at some other benchmark functions.

## 9.2 Random Linkage

To be able to look at how linkage learning affects the performance of MO-LTGA it needs to be compared with a variant which does no linkage learning. In order to do this a variant of MO-LTGA is constructed which uses a random linkage tree structure. Whereas MO-LTGA uses a linkage tree created with the UPGMA algorithm as described in section 3.5, this new variant will use a linkage tree that is created randomly.

LTGA creates a linkage tree by starting with a univariate FOS structure and then it merges sets of problem variables based on joint entropy. Instead of using joint entropy the merging of can be done at random and a random linkage tree is constructed.

Any difference in performance between these two variants must then be the result of the presence or absence of linkage learning.

## 9.3 Linkage Learning: Experimental Study

In the following experiments a closer look is taken on how linkage learning effects the performance of MO-LTGA. In each experiment a different parameter is varied. In the

different experiments standard MO-LTGA with clustering is denoted by MO-LTGA-LL, where LL stands for linkage learning. The variant that uses a random linkage tree is denoted by MO-LTGA-RL where RL stands for random linkage.

### 9.3.1 Clustering on λMNK and ρMNNK problems

To see how using a cluster ratio compares do dedicated clustering with and without linkage learning an experiment is done on λMNK and ρMNNK instances with varying covariances for each problem, $\lambda$ for λMNK and $\rho$ for ρMNNK. The used algorithms are MO-LTGA with sampling and MO-LTGA with clustering, each using standard linkage and random linkage. The results can be seen in figure 9.2

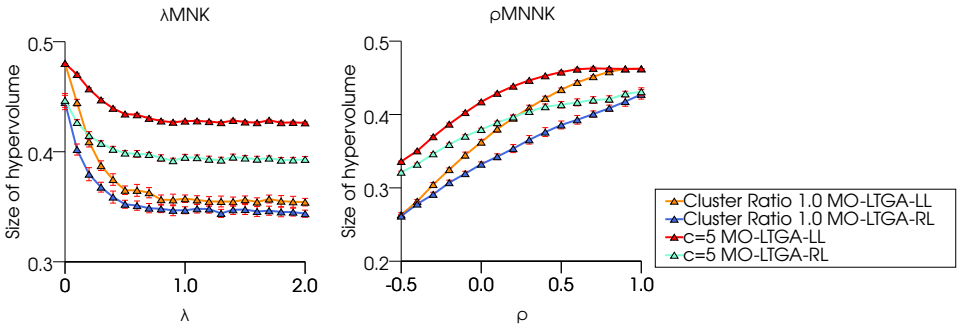


Figure 9.2: Comparison of two clustering variants of MO-LTGA both with and without linkage learning

There are some differences between the performance of all MO-LTGA variants between λMNK and ρMNNK, but they share some common trends. For λMNK with a low value of $\lambda$ there is no significant difference between clustering variants. However, when $\lambda$ increases and the correlation between the objectives decreases the performance of the cluster ratio algorithms plummets. For both clustering variants of MO-LTGA the linkage learning variants always outperform their random linkage counterparts. The type of clustering algorithm has a much larger impact though on performance. For $\lambda \geq 0.3$ the random linkage clustering variant outperforms the linkage learning cluster ratio variant. As $\lambda$ gets larger and the linkage structure is shuffled more and more differently between the objectives it is of no surprise that the size of the hypervolume of the local Pareto front drops. After $\lambda$ reaches 1 the size remains stable which suggests that for $\lambda \geq 1$ the linkage structure of λMNK is shuffled so much that it appears to be random.

For ρMNNK the results are a bit different. If we look at the extremes, $\rho = -0.5$ and $\rho = 1$, there is quite a remarkable pattern. For $\rho = -0.5$ the cluster variants perform clearly better. For $\rho = 1$ The linkage learning algorithms outperform the non-linkage learning variants, regardless of clustering used.

When $\lambda = 0$, λMNK is just a single objective problem. Therefore it is not surprising that regardless of the clustering technique it manages to outperform random linkage by quite a margin. What is more surprising is that the performance of the cluster ratio algorithms drop severely afterwards. When the correlation between the objectives decreases using dedicated clustering performs significantly better regardless whether linkage learning is used. Since it also happens for random linkage this shows that the recombination is done more efficiently using separate clusters, instead of one big cluster. The results of the ρMNNK instances also show this. With a very low correlation there are more Pareto optimal points and linkage learning seems to be off less importance, for both algorithms the linkage learning variant perform similarly to their random linkage counterpart. There is a large difference however between the two different methods of clustering, which seems to indicate that dedicated clustering performs better recombination.

For $\rho$MNNK as the correlation increases the relative performance of dedicated clustering without linkage learning drops and the performance of using a cluster ratio with linkage learning increases. For $\rho = 1$ the problem has become a single objective problem and the performance of both clustering algorithms becomes the same. For this type of instance the objective space is so focused in one direction that clustering is of less importance and the only thing that matters is whether linkage learning is used.

#### Conclusions

Linkage learning is a very effective addition for multi-objective algorithms. The more the objective functions are positively correlated, the better linkage learning performs. If there is no correlation between the objectives, or there is a highly negative correlation, and the Pareto front is larger and wider, any clustering method is superior regardless of linkage learning is used or not.

### 9.3.2 Clusters

In this experiment the performance of different amount of clusters is compared for several problems. More clusters means that each cluster is smaller, but also that within each cluster solutions are closer to each other in the objective space and less function evaluations are used per cluster. Something that also has to be taken into account is that for extreme clusters single-objective optimization is used instead of multi-objective optimization. Choosing the right amount of clusters is a non-trivial task. Too few clusters means only single-objective optimization is performed, too many might result in clusters that are too small for linkage learning to be effective.

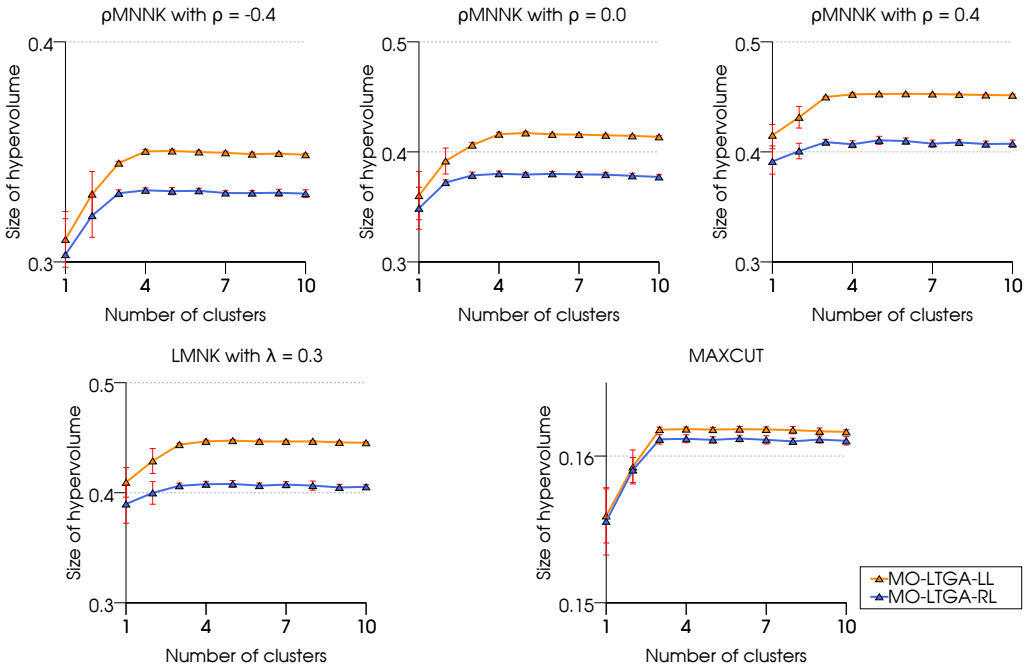


Figure 9.3: Comparison of the performance of the number of clusters $c$ for different problems

The results can be found in 9.3. The results are very similar for random linkage and linkage learning. For all problems MO-LTGA performs significantly better with 3 or more clusters. For 1 and 2 clusters not only are the results worse, there is also a lot more variance. When using 1 cluster only single-objective optimization is used, since this particular cluster is always the most extreme cluster for all 3 objectives. For $c = 2$ there is a chance that

multi-objective optimal mixing is used, if one of the two clusters is the most extreme clusters for all objectives. This is quite unlikely though, unless the objectives are very highly correlated and 1 cluster contains the good solutions, and the other cluster contains all the lesser solutions.

For 3 clusters there is already quite a high chance of one cluster not being extreme in one of the objectives, and another being extreme in 2 objectives. The results suggest that it is beneficial to have at least one cluster being optimized by MO-optimal mixing. For $c > 3$ this is always the case. In all graphs the performance for all values of $c > 3$ are very similar. Only for $\rho$MNNK with $\rho = -0.4$ and $\rho = 0$, instances with a spread Pareto front, is there a slight increase in performance between $c = 3$ and $c = 4$. With a spread Pareto front it is more likely that if $c = m$, each objective has its own cluster and only SO-optimal mixing is used. When only SO-optimal mixing is used, there is only search pressure in each individual objective, but not towards the center of the Pareto front.

When comparing MO-LTGA-LL and MO-LTGA-RL the general performance characteristics are very similar. For the $\lambda$MNK and $\rho$MNNK instances MO-LTGA-LL consistently outperforms random MO-LTGA-RL for $c \geq 2$. For $c = 1$ the variance is too large to definitively say MO-LTGA-LL is better, but as $c = 1$ performs quite poor in general this is not as much of an interest.

Max Cut shows a bit of a different story. Just like in the MO-LTGA with sampling experiment in section 7.2 the scale of the y-axis is much smaller. Furthermore the difference in performance of MO-LTGA-LL versus MO-LTGA-RL is very small. For $c = 1$ and $c = 2$ there is not even any difference between the two. For $c \geq 3$ linkage learning slightly outperforms random linkage, but the difference remains very small. Especially considering the scale of the y-axis. If we quickly compare the results for MO-LTGA with sampling from figure 7.1 we can see that MO-LTGA-LL performs much better than MO-LTGA with sampling on $\lambda$MNK and $\rho$MNNK with $\rho = -0.4$ and $\rho = 0.0$, but for $\rho$MNNK with $\rho = 0.4$ and Max Cut the difference is much smaller. For the $\rho$MNNK instance this can be explained away by the fact that with $\rho = 0.4$ the 3 objectives are highly correlated and since the linkage structure is the same this problem could be considered easy. However since each objective of Max Cut contains a single-objective Max Cut instance there is no correlation between the objectives and the linkage structure is completely different for each objective. The Max Cut problem is the hardest of the tested problems. Despite being such a hard problem and there not being any correlation between the objectives nor shared linkage structure, linkage learning still outperforms random linkage for $c \geq 3$. Even when linkage learning is not beneficial for $c \leq 2$, for $c \geq 3$ and MO-mixing starts being used linkage learning still manages to exploit some structure that is present ad-hoc.

### 9.3.3 Population Size

Determining the best population size is always a tough problem in evolutionary algorithms. Whilst not the primary goal of this thesis, a baseline population size must be set to use for all the experiments. Furthermore it is interesting to look at how the use of linkage learning affects the convergence behaviour of MO-LTGA. In this test various problems are tested with exponentially increasing population size. Tested population sizes are: $\{ 25, 50, 100, 200, 400, 800, 1600, 3200, 6400, 12800 \}$.

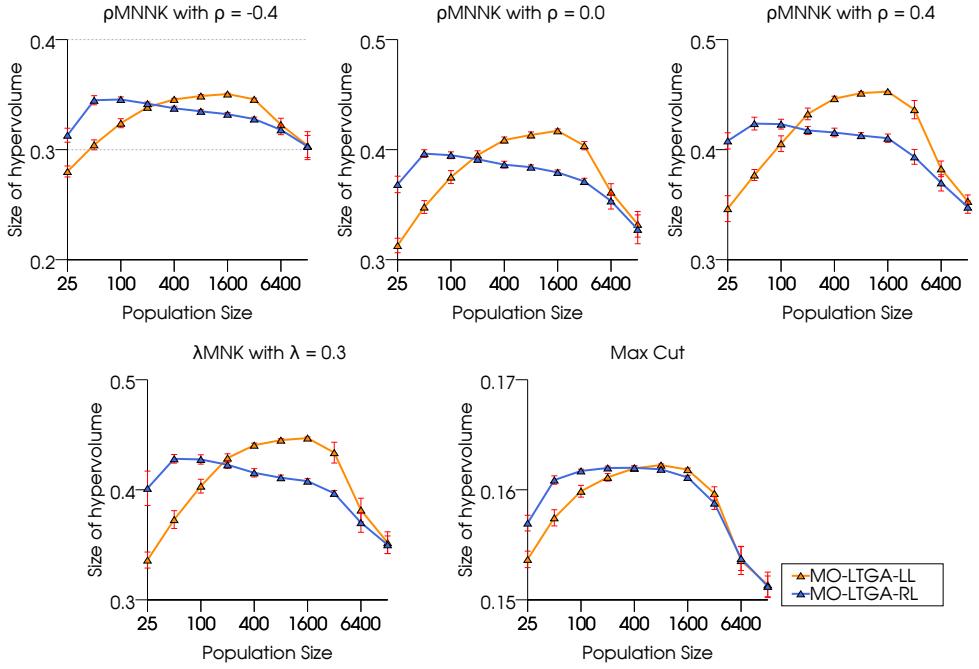


Figure 9.4: Comparison of MO-LTGA with varying population sizes

In figure 9.4 we can see that the performance of MO-LTGA with linkage learning follows a strong curve. For all the tested problems performance increases with population size until 1600 point and then drops significantly as the population sizes becomes larger. For the random linkage variant the results look very different. Apart from the tests with a population size of 25, on $\lambda$MNK and $\rho$MNNK the performance of random linkage is at a peak at 50 and only drops slowly with larger population sizes. For small population sizes it even beats the linkage learning variant. For the Max Cut the performance of random linkage is stable for population sizes between 50 and 1600. MO-LTGA-LL is only able to catch up to MO-LTGA-RL's performance for population sizes between 800 and 3200. There is not a single population size for which linkage learning has a clear advantage over random linkage.

Random linkage outperforms linkage learning for smaller population size. However when linkage learning is used with a population size around $800 - 1600$, linkage learning is equal or better than random linkage with any population size. As linkage learning exploits information present in the population it can be expected that a certain population size is required for there to be enough information present in the population so that it can be exploited. There is always the trade-off with function evaluations. In principle you would want to have the population size as large as possible, but since the amount of function evaluations is limited the population will converge too slow if the population size is too large. In all tests performance dropped with sizes equal or larger than 3200, which is because the population has not been able to converge yet.

### Conclusion

Figure 9.4 shows that random linkage performs significantly better than linkage learning for smaller population sizes. This is likely due to the different behaviour of random linkage opposed to linkage learning. In the case of smaller population sizes, populations converge too quickly. Random linkage has however a clear advantage over linkage learning when it comes to premature convergence. With random linkage a completely new random linkage tree is constructed in each generation. Linkage learning constructs a tree based on information present in the population. In the first generation both algorithms will generate a

random linkage tree. In the second generation, random linkage will generate a completely different linkage tree whilst the linkage tree that linkage learning creates will still have similarities with the linkage tree generated by the previous generation. The search space that random linkage explores is much wider than the search space linkage learning explores. However when there is too little information to go on, having a larger, albeit more random, search space works better than a smaller more focused one, that is also random in a sense, because linkage learning has too little information to go on. Linkage learning converges too quickly and random linkage is able to continue searching.

Another interesting fact is that for Max Cut random linkage does not perform any worse than linkage learning. Despite linkage learning being very useful for single-objective Max Cut, for multi-objective Max Cut even with the right population size it is barely even better than random linkage. Random linkage even has the advantage that it is much less reliant on the right population size for the right performance as it has less convergence problems.

### 9.3.4 Degree of Epistasis

In this experiment the performance of MO-LTGA with clustering is analyzed when the degree of epistasis of the problem is changed from the default value.

As the $\lambda$MNK and $\rho$MNNK problems are variants of NK-landscapes they have a problem parameter $k$ that determines with how many bits each individual bit interacts as explained in section 4.1. In the following experiment MO-LTGA with clustering is tested on a number of $\lambda$MNK and $\rho$MNNK with different values for $k$.

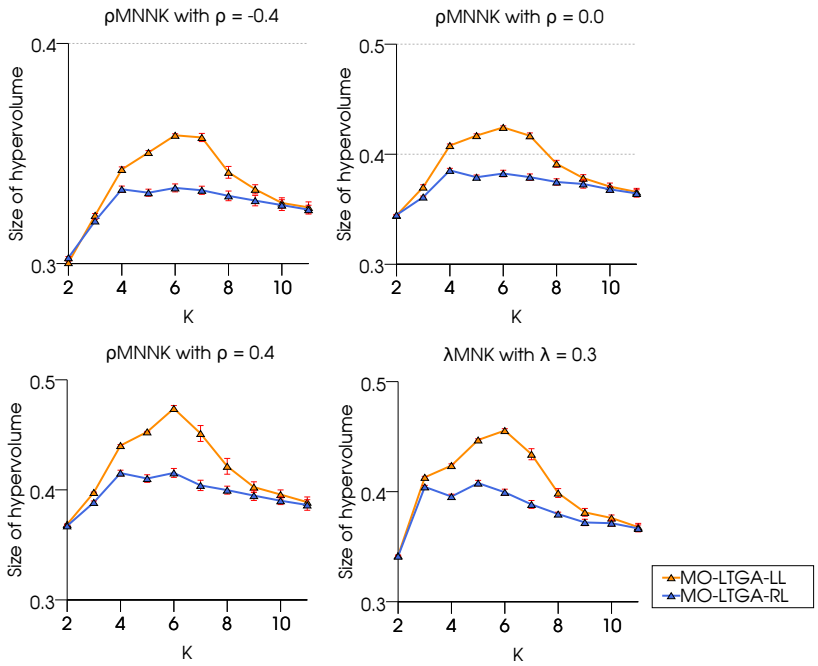


Figure 9.5: Comparison of MO-LTGA with varying values of $k$

Problems with low values of $k$ are easier since subproblems are smaller and optima for these subproblems are easier to find. When $k$ becomes larger, it gets harder to find these optima. If we look at figure 9.5 various $\lambda$MNK and $\rho$MNNK instances are tested. Linkage learning always manages to outperform random linkage for values of $k \leq 9$. For values of $4 \leq k \leq 8$ the difference is quite substantial.

For small values of $k$ the resulting problems are easy enough that random linkage also has good results. When an individual bit, as long as it is not at the edge of the

bitstring, is permutated, the change in fitness is affected by $k$ fitness functions. When $k$ is low this creates a fairly smooth fitness landscape on which even simple traditional hill climbers perform well. Furthermore the total of different number of fitness contributions each subproblem can have is $2^k$. For small values of $k$ MO-LTGA-LL is able to iterate through all those combinations for all the subproblems and as the fitness landscape is fairly smooth it acts as a standard hill climber. For values of $k \geq 4$ MO-LTGA-RL cannot do this anymore. MO-LTGA-LL detects which bits of the problem are correlated and is able to iterate through these combinations much more efficiently. When $k$ gets sufficiently large, i.e. $k \geq 9$ the fitness landscape gets so rugged and the problem instances become so hard that even MO-LTGA-LL is not able to detect the correlated bits in the $\lambda$MNK and $\rho$MNNK instances.

**Conclusion**

The results show that for moderate values of $k$ on the interval $[4, 8]$ linkage learning is effective for multi-objective variants of NK-landscapes. For values of $k \leq 3$ both MO-LTGA-LL and MO-LTGA-RL perform equally well as the fitness landscape is so smooth both algorithms are able to get very close to the Pareto front. For $k \geq 9$ the fitness landscape is so rugged that MO-LTGA-LL is not able to learn the underlying structure and there is no significant difference in performance between MO-LTGA-LL and MO-LTGA-RL.

# Chapter 10

# Scalarized LTGA

As $\rho$MNNK and $\lambda$MNK are new problems, it is not possible to compare the performance of MO-LTGA to other algorithms found in literature on these specific problems, since there are not any. To do a rough comparison of how well the Pareto based MO-LTGA performs, scalarized approach is also look at. In scalarized LTGA a number of weight vectors are generated across the objective space, and for each weight vector regular single-objective LTGA is applied, with either the linear weighting or the Tchebycheff weighting as the aggregation function. As the number of function evaluations is fixed each individual run of single-objective LTGA is given an function evaluation budged that is equal to the total number of evaluations divided by the number of weight vectors. Scalarized LTGA uses the exact same LTGA algorithm as is described in section 3.5 and does not use forced improvement. MO-LTGA puts every solution that it encounters in its elitist archive. There are multiple ways to handle this when using scalarizations. One could put the results of the individual runs in the elitist archive, but in that way one could miss Pareto optimal solutions that might not necessarily be considered interesting by the aggregation function. Since adding a solution to the elitist archive is relatively cheap computationally, every solution that is generated inside a run of single-objective LTGA is also checked against global elitist archive and added to it if it needs to be.

## 10.1 Configuration

In the following experiments MO-LTGA is compared against scalarized LTGA, iterated local search (ILS) and for $\rho$MNNK a dynamic programming approximation (DP). The used implementation of ILS is the same as in Thierens [1]. ILS flips single bits to perform hill climbing. When it is stuck in a local optimum $k$ random bits are flipped and the algorithm starts hill climbing again. A $k$ of 5 is used, the same as Thierens and Bosman [3]. This procedure is repeated until all function evaluations are used. Just as in scalarized LTGA, every solution generated by ILS is added to the elitist archive. For instances of $\rho$MNNK an approximation of the Pareto front is generated by the use of dynamic programming. When a linear weighting is applied to a $\rho$MNNK instance it is transformed back into a regular NK-landscape instance which can be solved quickly using dynamic programming as described in section 4.3. An approximation of the Pareto front can then be created by solving these single-objective NK-landscapes for a large amount of weight vectors, in this experiment 10.000. The resulting local Pareto set is not optimal as the amount of vectors is not infinite and since linear weighting is used only convex points are found, solutions which are concave will not be found. The true Pareto front is guaranteed to be equal or better than this dynamic programming approximation. By default scalarized LTGA uses 13 weight vectors and a population size of 100.

## 10.2 Scalarized LTGA: Experimental Study

Scalarized LTGA has 2 parameters of interest that differ from the Pareto based approach; The population size and the amount of weight vectors for which scalarized LTGA is performed. The first experiments will look at how these 2 parameters influence the performance of scalarized LTGA. The second experiment looks at how scalarized LTGA stacks up against MO-LTGA on $\rho$MNNK and $\lambda$MNK problems with a varying $\rho$ and $\lambda$ respectively.

### 10.2.1 Weight Vectors and Population Size

Since each run of scalarized LTGA with a certain vector can use a fixed amount of function evaluations which is inversely proportional to the total amount of weight vectors, there is a penalty to blindly increasing the amount of vectors. On the other side if too few vectors are used there will not be a good spread across the objective space.

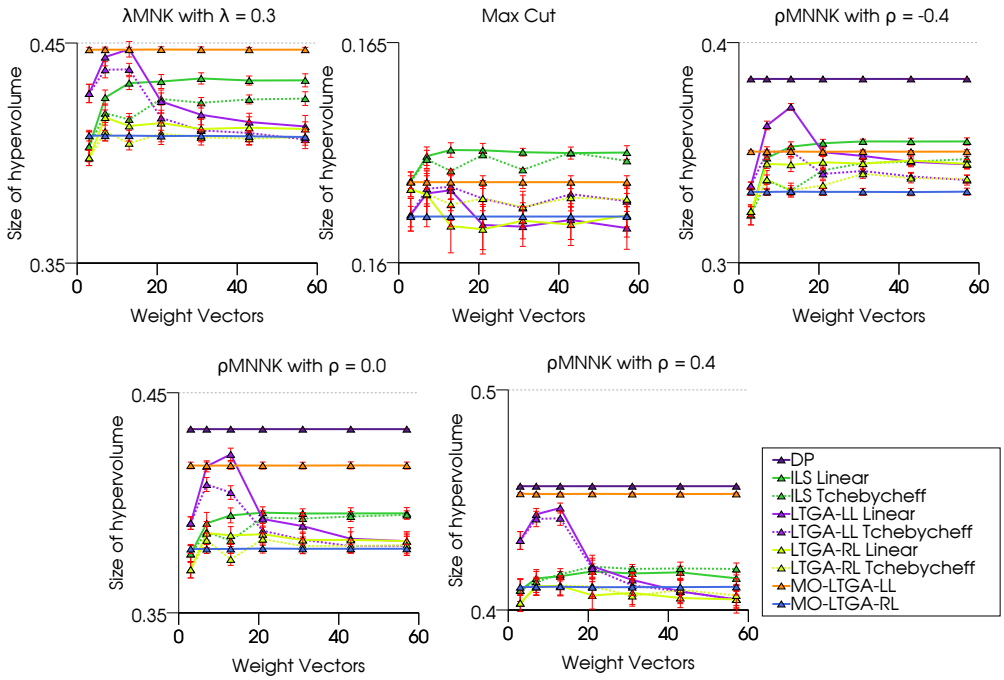


Figure 10.1: MO-LTGA compared against scalarized LTGA and ILS with a varying amount of weight vectors $\upsilon$

Figure 10.1 shows the impact the number of scalarizations has on the performance of scalarized LTGA and iterated local search. As MO-LTGA does not use weight vectors it acts as a baseline to compare against. The results show that the performance of ILS remains stable regardless of the number of weight vectors $\upsilon$ if $\upsilon \geq 13$. As $\upsilon$ increases each individual run of ILS can perform less function evaluations, but this is made up for by the fact that ILS is done more often towards different part of the objective space. On the NK-landscape-type problems a similar behaviour can be observed for scalarized LTGA-RL. For $\upsilon \geq 7$ the performance remains stable for $\lambda$MNK and $\rho$MNNK instances. This is not the case for scalarized LTGA-LL. Scalarized LTGA-LL has a clear increase in performance for $\upsilon \in \{7, 13\}$. When $\upsilon = 3$, only single-objective optimization is done into each objective, as the number of objectives is also 3. For $\upsilon \in \{7, 13\}$ there is at least some spread and scalarized LTGA-LL jumps in performance, performing on par with MO-LTGA-LL and even outperforming it for $\rho$MNNK with $\rho = -0.4$. However when the number of weight vectors increases and each individual LTGA-LL run is allowed less

function evaluations the performance drops and the effect of linkage learning dissipates entirely. If LTGA-LL is given too few function evaluations it takes too long for linkage learning to increase the performance of LTGA. The $\rho$MNNK instances show an interesting pattern. For $\rho = -0.4$ the objectives are highly negatively correlated and the Pareto front is very large. Scalarized methods perform very well compared to the Pareto based approach MO-LTGA takes. Another indicator of this is the fact that the gap between MO-LTGA and DP is much bigger than with $\rho = 0$ and $\rho = 0.4$. With $\rho = 0.4$ the objectives are positively correlated and the Pareto set is smaller and more concentrated in one area of the objective space and MO-LTGA is almost on par with DP which suggests it is very close to the Pareto set.

Another point of interest is the performance of ILS compared to scalarized LTGA-RL and MO-LTGA-RL. It is often noted that GOMEA on its own can be an effective hill climber, even if it does not have a linkage structure that can easily be exploited. The $\lambda$MNK and $\rho$MNNK instances all show that ILS always beats all LTGA-RL variants. The boost that the LTGA-LL variants get by using linkage learning is the sole reason they beat ILS. GOMEA on its own is not efficient enough to outperform ILS.

In MO-LTGA the population is used to find the entire Pareto front in one single run, which requires a larger population size compared to a single-objective problems. When translating the problem back to a single-objective problem the population can be reduced again to a size fitting for single-objective problems.

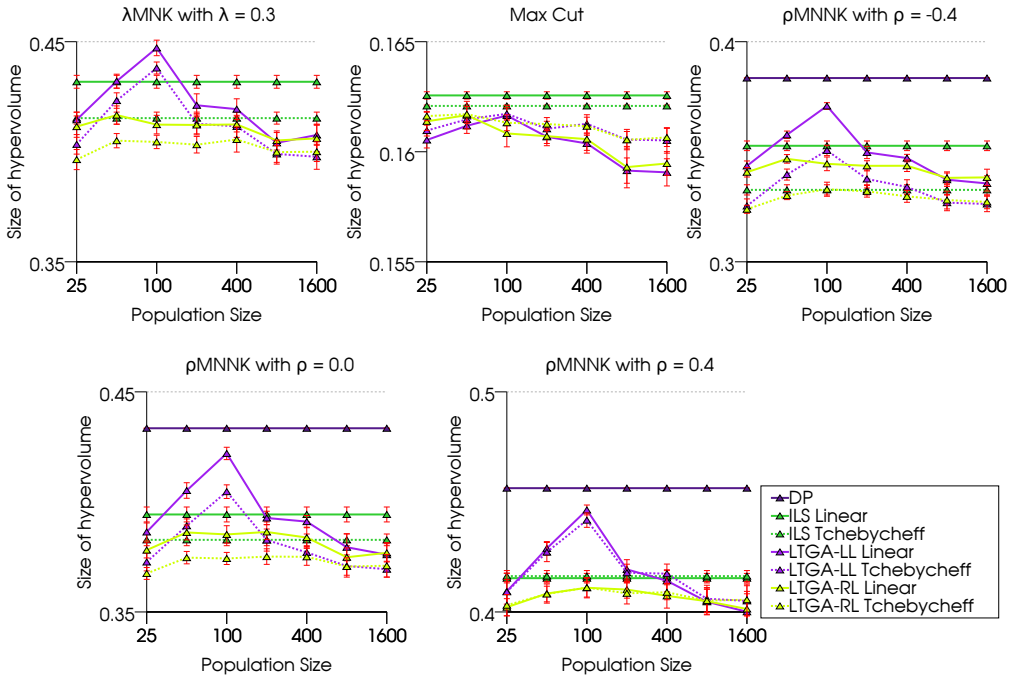


Figure 10.2: Scalarized LTGA compared against ILS with a varying population size

Figure 10.2 shows how the population size affects scalarized LTGA. Tested population sizes are: $\{25, 50, 100, 200, 400, 800, 1600\}$. Because ILS does not have a population, its performance is stable and acts as a baseline. Figure 10.2 shows a consistent curve for the population sizes. The performance increases until the population size hits 100 and then drops back down for every tested problem. For $\rho$MNNK and $\lambda$MNK problems a population size of 100 is also the only population size for which scalarized LTGA-LL outperforms ILS with linear weighting and at least matches it using Tchebycheff weighting. Scalarized LTGA-RL is less sensitive to changes in population size compared with its linkage learning counterpart. This is similar to MO-LTGA. With MO-LTGA, MO-LTGA-LL was also more sensitive to changes in population sized compared to MO-LTGA-RL.

**Conclusions**

The scalarized approaches using the linear weighting function generally outperform the Tchebycheff weighting. The only big exception to this is scalarized LTGA on Max Cut. Due to the variances of the results for Max Cut it is hard to draw any more conclusions, other than that it is also the only problem where ILS consistently outperforms all LTGA variants. Scalarized LTGA-LL manages achieve at least on par performance with MO-LTGA-LL for most problems and even exceeds performance on $\rho$MNNK with $\rho \leq 0.0$. For $\rho$MNNK the $\rho$ parameter has a direct influence on the correlation between objectives as $\rho$ is the correlation between the objectives of each individual subproblem. As $\rho$MNNK instances consist of many subproblems added together all the possible solutions together approximate a multivariate normal distribution with $\rho$ as the correlation. When the correlation is heavily negative like with the $\rho$MNNK with $\rho = -0.4$ graph in figure 10.1 the Pareto front is larger and spread across the objective space. With the scalarized method each weight vector will target a different part of the objective space and in each run new solutions are added to the local Pareto front. When the objectives are positively correlated the Pareto set is smaller and more concentrated towards one part of the objective space. In this case different weight vectors that target different parts end up finding solutions in the same part of the objective space as these solutions are good solutions for a large amount of weight vectors.

For $\lambda$MNK instances the $\lambda$ parameter has no influence on the correlation of the objectives directly, but it only affects the linkage structure. If the linkage structure is the same for all objectives there is also a perfect correlation between them. As the linkage structure gets shuffled somewhat some subproblems will have a different linkage structure in the different objectives and have different fitness contributions to the final fitness. As soon as not a single subproblem has the exact same linkage structure there is no correlation left. As a $\lambda$ of 0.3 is used there is a non-zero chance of this occurring, but experimental analysis shows that all possible solutions for a $\lambda$MNK instance with $\lambda = 0.3$ create a multivariate normal distribution without any correlation.

### 10.2.2 Linkage Learning: MO-LTGA and Scalarized LTGA

This experiment is similar to the experiment done in section 9.3.1, but now MO-LTGA with clustering is compared against scalarized LTGA. In this experiment we can take a look whether a scalarized approach is better than a Pareto based one on $\rho$MNNK and $\lambda$MNK. For $\rho$MNNK the dynamic programming approximation algorithm is also included.

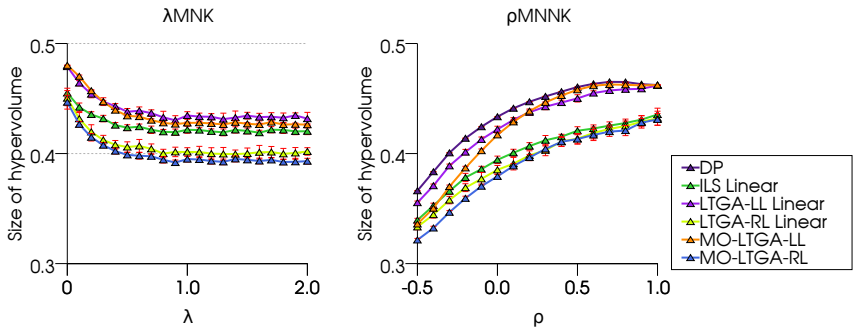


Figure 10.3: MO-LTGA compared against scalarized LTGA and ILS

The results in figure 10.3 show that there is no big difference in performance between the scalarized and Pareto based approach. On $\lambda$MNK the scalarized variants each perform slightly better overall than their Pareto based counterparts, although the difference is very minor. On $\rho$MNNK the differences are slightly bigger. When the correlation is highly

negative and the Pareto front is very spread out, the Pareto based approaches dip significantly in terms of performance. Whereas scalarized LTGA-LL manages to get close to the dynamic programming approximation regardless of the value for $\rho$, MO-LTGA-LL gets much closer to this approximation for when $\rho \geq 0.4$. Its performance drops significantly however for negative values for $\rho$. For $\rho = -0.5$ it performs on par with ILS and scalarized LTGA-RL. Also interesting to note it that wheres ILS's performance on $\rho$MNNK is slightly better than scalarized LTGA-RL, on $\lambda$MNK ILS's performance does not drop-off as significantly as all the LTGA variants. This means that for $\lambda$MNK with $\lambda \geq 1$, ILS's performance is only slightly worse than the linkage learning variants of LTGA. For high values of $\lambda$ the shuffling of the linkage structure is so different that there is no similarity left. The underlying linkage structure is completely different for these instances in each objective.

**Conclusions**

The results in section 10.2.2 somewhat overlap with those in section 10.2 in that they show that in certain situations the Pareto based MO-LTGA performs better and in others scalarized based LTGA has the upper edge. For both $\lambda$MNK and $\rho$MNNK there is always a big difference between the linkage learning variant and its random linkage counterpart and most of the time this difference is also bigger than the difference between whether a Pareto or scalarized based approach is taken. This again proves that linkage learning has a significant advantage on these type of problems. Another interesting conclusion is that ILS performs well on $\lambda$MNK instances where there is no similarity between the linkage structure between the objectives.

# Chapter 11

# Discussion

In this chapter a broader look is taken at all the newly introduced problems and algorithms. The first half analyzes the used benchmark functions. The second half discusses MO-LTGA and scalarized LTGA and its different performance characteristics.

## 11.1 Analysis of multi-objective benchmarking functions

Up until now all multi-dimensional problems in literature took one of two approaches:

- Adding extra objectives by making the underlying problem return a vector instead of a scalar value. The underlying structure of the problem is the same for each objective. Examples of this are the trap - inverse trap problem and the $\rho$MNK landscapes by Verel et al. [12].
- Generate a new instance of the problem for every objective. There is no correlation of the structure between the objectives. An example of a problem that is constructed this way is Max Cut.

In the case where the underlying structure is the same applying linkage learning is easy. A linkage tree that is learnt on solutions in one part of the objective space can be used to generate new solutions in a completely different area of the objective space as the structure is the same everywhere. The other extreme is Max Cut where for the single-objective variant the structure is not created explicitly, but appears ad-hoc by the fact that in good solutions some vertices always appear in a set together.

### 11.1.1 Max Cut

When Max Cut is extended to multiple objectives each objective has its own ad-hoc structure. Different parts of the objective space will have different structures as the structure is a mix of the different objectives. This resulting structure is very hard to exploit by LTGA algorithms as the experiments in section 9.3 shows. The difference in performance of MO-LTGA-LL and scalarized LTGA-LL compared to MO-LTGA-RL and scalarized LTGA-RL respectively is negligible and in most cases not significant. In some cases linkage learning variants outperform random linkage, but only by a very small margin. Furthermore the effect other parameters have is far bigger than any difference between linkage learning and random linkage. The experiment in section 10.2 also shows that scalarized ILS is actually a more effective approach than LTGA which signals that linkage learning is not effective for the Max Cut problem and that the structure of multi-objective Max Cut is too difficult to exploit.

### 11.1.2 $\rho$MNNK

Nearest neighbour NK-landscapes are a very popular benchmarking problem for EAs. Thus far the only multi-objective NK-landscape in literature is the $\rho$MNK landscape by Verel et al. [12]. As noted in section 4.2 regular NK-landscapes barely have any structure. In order to create a multi-objective problem based on NK-landscapes that has a clearer structure I introduced the $\rho$MNNK landscapes in section 5.4. As $\rho$MNNK are based on nearest neighbour NK-landscapes they have a clear structure. The experiments in sections 9.3 and 10.2 show that LTGA algorithms perform better with linkage learning on $\rho$MNNK instances. Research has shown that for regular single-objective nearest neighbour NK-landscapes LTGA manages to exploit the linkage structure, it is therefore not very surprising that for a simple multi-objective variant that uses the exact same linkage structure in all objectives this holds true as well. There is a $\rho$ parameter that influences the correlation between the different values of the objective vector of each subproblem. This only influences the spread of the solutions on the objective space. We can draw some conclusions based on the differences in performance between the Pareto and scalarized approaches in section 10.2, but this does not tell us as much about the effectiveness of linkage learning in the multi-objective domain. Sure we know that if the underlying linkage structure is identical in all objectives that linkage learning remains effective, but it is more interesting to know whether linkage learning remains effective even when different objectives have different linkage structures. $\rho$MNNK is not a benchmarking function with which this hypothesis can be tested well.

### 11.1.3 $\lambda$MNK

The $\lambda$MNK problem introduced in section 5.5 takes a completely different approach to creating a multi-objective problem using a nearest-neighbour NK-landscape. For the single-objective variant we saw the concept of loose linkage in section 4.4. $\lambda$MNK uses this concept to construct a multi-objective problem by making the similarity of the linkage structure between the different objectives controllable with the parameter $\lambda$. The experiment in section 10.2.2 shows how MO-LTGA and scalarized LTGA perform when modifying $\lambda$. For high values of $\lambda$ when the underlying linkage structure has no similarity anymore between the objectives LTGA manages to use the ad-hoc structure that is the result of the combination of all the separate linkage structures. This ad-hoc structure appears to be more easily exploitable than the ad-hoc structure created by a problem like Max Cut. This is shown by the fact that ILS outperforms LTGA when there is almost no ad-hoc structure, but the LTGA variants are on top for $\lambda$MNK, even for high values of $\lambda$.

$\lambda$MNK has proven itself to be a very good benchmarking function as it provides multiple parameters to control the difficulty of the instances. The difficulty of the instances can be increased by increasing $k$ and to a lesser extent $n$. Its unique property is that the similarity of the underlying structure is configurable. $\lambda$MNK is the first benchmarking function where this is possible.

## 11.2 Multi-objective LTGA

Chapter 7 introduced an initial version of LTGA that works in a multi-objective environment. Several parts of LTGA had to be adapted so that it would work for multi-objective problems. The acceptance criterion was changed to use the Pareto dominance relation and the termination criterion was altered to no longer include fitness variance as that metric no longer works for a multi-objective population. The most significant change from regular LTGA is that the first version of MO-LTGA, MO-LTGA with sampling, did not learn a linkage tree and perform recombination from the entire population. Instead it selected

a subset of the population at random and used that subset to learn a linkage tree and perform recombination for that generation. The idea behind this was that to optimize across the different parts of the objective space it would be best to choose solutions that are closer together in this regard. If you would recombine solutions that are in opposite areas of the objective space you could end up with solutions which are as good as random, depending on the problem.

This sampling approach was the simplest approach to take, but experimental analysis in sections 7.2 and 7.3 showed that this approach was inferior to the clustering approach MO-GOMEA by Luong et al. [2] took. Whilst the main focus of this thesis is to look at the effectiveness of linkage learning in a multi-objective environment, to be able to say anything meaningful about this subject any experiments should be done with an algorithm that achieves at least some acceptable level of performance. In chapter 9 MO-LTGA is adapted to incorporate certain elements of MO-GOMEA.

This improved version of MO-LTGA, MO-LTGA with clustering, has much better performance characteristics than MO-LTGA with sampling as shown in the experiment in section 9.1. The differences between the various methods can be summarized as follows:

- MO-LTGA with sampling optimizes a subset of the objective space each generation by randomly selection a solution and its closest neighbours.
- MO-GOMEA clusters the objective space in $c$ different clusters using balanced k-leader-means clustering. After the initial clustering is finished a cluster will have $\frac{|P|}{c}$ solutions on average, where $|P|$ is the population size. In practice this can vary wildly, so to ensure every cluster is equally large, every cluster is expanded or shrunk down to a size of $\frac{2}{c}|P|$
- MO-LTGA with clustering clusters the objective space into $c$ different clusters using an adapted balanced k-leader-means algorithm. During the clustering process the size of the clusters are fixed to $\frac{|P|}{c}$. This means that the final clusters also have this size and every solution belongs to exactly one cluster.

If there is one thing that can be learned from these experiments it is that the method of clustering can have a significant impact on the performance of a multi-objective algorithm. Initially a different clustering method was chosen than MO-GOMEA for simplicity and because it was not clear from their paper how optimal mixing is performed when a single solution is part of multiple clusters. There is still a difference in performance with MO-GOMEA, which might be caused by the difference in clustering method. The performance of MO-LTGA could likely be improved in the future if further research is done into this area. Another observation that can be made about the different clustering methods is the amount of function evaluations each of the methods uses per generation. As the sampling method only optimizes a subset of the population in each generation, less function evaluations are used when compared to MO-LTGA with clustering. On the other hand with MO-GOMEA each solution is part of two clusters and because the number of function evaluations scales with the size of the clusters, twice as much function evaluations will be used per generation.

One instance where linkage learning does not have a positive effect on performance is when the used population size is too low as shown by the experiment in section 9.3.3. In hindsight it might have been worth considering running the experiments for random linkage with a lower population size. MO-LTGA-LL suffers from some premature convergence problems that MO-LTGA-RL does not have as it uses a new randomly created linkage tree for each generation. MO-LTGA-LL needs quite a large population before it outperforms

MO-LTGA-RL.

## 11.3 Scalarized LTGA

In chapter 10 a different approach was taken. Instead of adapting LTGA to work in a multi-objective environment, scalarized LTGA tackles the problem by turning the multi-objective problem back into a single-objective one. The different approaches result in different performance behaviours as the experiments in section 10.2 show. The experiments on $\rho$MNNK with the various values of $\rho$ show that as the solutions in the Pareto front get less correlated in the objective space a scalarized approach works better than a Pareto based one. In these cases the amount of local Pareto optimal solutions increase and the Pareto dominance relation does not manage to guide the population towards the Pareto front efficiently.

The scalarized approach however has a number of problems that the Pareto based MO-LTGA does not have to worry about. The scalarized variant introduces yet another parameter that needs to be tuned properly, the number of weight vectors, which has a large impact on the performance. The population size is the most disliked parameter of EAs and there are ongoing efforts to eliminate it, as will be mentioned in the future work section, therefore introducing yet another parameter that will need to be tuned is only making it harder to use. Furthermore currently the weight vectors are uniformly distributed across the objective space. The performance might very well be improved if the weight vectors are targeted towards the objective space. However as EAs are designed to be black-box algorithms this is not possible to do beforehand, as there is no way of knowing what the objective space looks like upfront.

## 11.4 ILS

Chapter 10 also included a scalarized ILS implementation that acted as a reference algorithm. First it shows that LTGA performs relatively poor on Max Cut as discussed previously in section 11.1.1, but secondly it also shows that ILS always outperforms the random linkage variants. This can be seen when comparing the performance of MO-LTGA-RL and scalarized LTGA-RL in section 10.2. This means that the only reason MO-LTGA-LL and scalarized LTGA-LL beat ILS is their use of linkage learning.

## 11.5 Future Work

This section discusses a number of ways in which the algorithms presented in this thesis could be improved upon in future research.

### 11.5.1 Hill Climbing

There are single-objective evolutionary algorithms, like DSMGA-II by Hsu and Yu [16], that perform a local search hill climber as part of their initialization of the population, instead of starting with a completely random population. When learning a linkage tree this could be beneficial as the hill climber can let the population converge to already expose some of the underlying structure which a linkage learning algorithm like LTGA or DSMGA-II can then use instead of starting from a random state. Improving the starting population by a first-improvement hill climber is fairly common amongst EAs, it could be of interest to also extend this into the multi-objective domain. It is still very unlikely however that such a method would allow MO-LTGA-LL to perform better at the very small populations where MO-LTGA-RL performs best. If you take into account that the

population is also split across a number of clusters there are barely any solutions left to learn a linkage tree from if the population size is only 50 or 100.

### 11.5.2 Changing the Linkage Model

LTGA uses a linkage tree as its linkage model. Even though most benchmark functions have an underlying structure that cannot directly be represented by a tree structure, the linkage tree provides a good generic model. For some problems the smaller less dominant substructures that are not directly contained within the linkage tree are still likely to be included indirectly. A larger node in the linkage tree could contain this substructure as a subset, or a part of the substructure can be present as a smaller node.

Nevertheless it would be interesting to see whether MO-LTGA could be improved by using a different linkage model, as multi-objective problems create even more complex linkage structures compared to their single-objective counterparts. DSMGA-II has shown it is able to achieve good performance for single-objective problems compared to LTGA whilst it uses a different linkage model. Likewise the Linkage Neighbors Genetic Algorithm and the Multiscale Linkage Neighbors Genetic Algorithm by Bosman and Thierens [17] have shown that a linkage model based around the concept of linkage neighbours can perform well in certain situations.

### 11.5.3 Parameter-less Population Pyramid

As with all EAs using the correct population size is of utmost importance for the performance. To eliminate this parameter the parameter-less population pyramid (P3) was introduced by Harik and Lobo [18]. Further research was done by Besten et al. [19] which incorporated P3 into LTGA and improved on it further with the introduction of multiple insertion. Eliminating the population size as a parameter would mean we would no longer have to worry about using the correct population sizes and perhaps some of the convergence issues of MO-LTGA would be solved as well by adapting this parameter-less population pyramid.

### 11.5.4 Hybrid Search

The Pareto dominance relation used by MO-LTGA struggles to guide the search towards the Pareto front in certain situations. This is mitigated in part by performing single-objective optimization in extreme clusters. On the other hand the linear weighting of scalarized LTGA does this very well, but is hindered by the inherent disadvantages of the scalarized approach. It could be interesting to see if a hybrid approach where the Pareto dominance relation of MO-LTGA is replaced by a dynamic linear weighting vector based on the location in the objective space manages to combine the two methods.

# Chapter 12

# Conclusions

The primary goal of this thesis was to look at the effectiveness of linkage learning for multi-objective problems. This goal was formulated as three research questions. In this chapter we will look back and try to answer the research questions.

## 12.1 Summary

The Linkage Tree Genetic Algorithm is a widely used black-box evolutionary algorithm for single-objective problems with structure. To investigate whether linkage learning is also beneficial for multi-objective problems two new benchmarking functions were introduced, the multi-objective nearest neighbour NK-landscape with correlated objectives, $\rho$MNNK, and the multi-objective nearest neighbour NK-landscape with correlated linkage structure, $\lambda$MNK. As $\rho$MNNK uses the same linkage structure for each objective function it is not suited well as a benchmarking problem even though it is a multi-objective problem, it has no distinctive multi-objective structure. $\lambda$MNK is a better benchmarking problem as it allows for a configurable amount of similarity of the structure between the objective functions. It is a valuable addition to the existing set of benchmarking functions as it is the first problem in literature that allows this.

In order to test the effectiveness of linkage learning on multi-objective problems like $\lambda$MNK and $\rho$MNNK a number of new multi-objective variants of LTGA were introduced. First was the Multi-objective Linkage Tree Genetic Algorithm with sampling which takes a Pareto based approach to solving multi-objective problems. It turned out MO-LTGA with sampling had performance problems due to the used clustering method. An approach where in each generation the population is split up into a number of separate clusters, a linkage tree is learnt for each cluster, and then recombination is performed within each cluster, is superior to an approach where a randomly chosen part of the objective space is optimized each generation.

An improved version called the Multi-objective Linkage Tree Genetic Algorithm with clustering was introduced. Linkage learning improved its performance with 2 notable exceptions. For very small population sizes linkage learning performed worse than random linkage due to convergence problems for MO-LTGA with linkage learning. On the Max Cut problem linkage learning had no significant impact when taking into account the population size issue.

To compare MO-LTGA with a scalarized method a different multi-objective variant of LTGA was introduced, scalarized LTGA. Scalarized LTGA tackled the problem of multi-objective optimization by turning the multi-objective problem back into a single-objective one using a weighting function and a weighting vector. The experiments showed for $\lambda$MNK and $\rho$MNNK instances linear weighting is better and for Max Cut the Tchebycheff weighting works the best for scalarized LTGA. Even though scalarized LTGA is quite a simple and rudimentary method it stacks up quite well against the Pareto based MO-LTGA and

it performs better when the Pareto front is widely spread across the objective space. In cases where the Pareto front is smaller and more targeted towards a smaller area of the objective space the Pareto based MO-LTGA performs better.

The experiments in this thesis haven proven that in most circumstances linkage learning is beneficial for the performance of a multi-objective LTGA variant, but there is no clear winner between the scalarized and Pareto approaches.

# Appendix A

# Quad Tree

To make sure that keeping track of a local Pareto front does not have too big of an impact on the performance a quick experiment is done to see how the performance of the quadtree by Mostaghim et al. [4] performs compared to a linked list. To mimic a real use case the following is done: For a problem of relatively small problem instance, brute force through all possible solutions and add them to the archive. The chosen problem instances are $\rho$MNNK instances of sizes $n = \{10, ..., 15\}$. The blue line is the time that it takes to just evaluate all the solutions. This is used as a baseline comparison. The red and green line represent the time that it takes to evaluate the solutions and add them to the quadtree and linear archive respectively. The results are averaged out over a total of 25 runs.

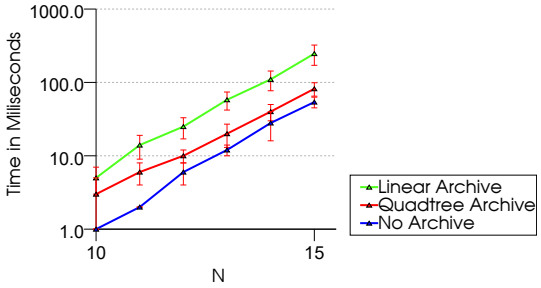


Figure A.1: Finding the optimal Pareto set for varying $n$

Figure A.1 shows that the quadtree is an order of magnitude faster than the linked list. Since the y axis is logarithmic the difference is even larger than at first glance. The additional computing time required to put the solutions in the quadtree is low, performing the function evaluation on the solution actually takes more time than inserting it in the quadtree. For the linked list inserting the solution is the bottleneck by quite a clear margin. This experiment shows that using the quadtree makes sure we do not hit a performance bottleneck when inserting new solutions in our elitist archive.

# Bibliography


[1] Dirk Thierens. "The Linkage Tree Genetic Algorithm". In: *Parallel Problem Solving from Nature - PPSN XI*. Springer, 2010, pp. 264–273.

[2] Ngoc Hoang Luong, Han La Poutr, and Peter A. N. Bosman. "Multi-objective gene-pool optimal mixing evolutionary algorithms". In: *Proceedings of the 2014 conference on Genetic and evolutionary computation*. ACM. 2014, pp. 357–364.

[3] Dirk Thierens and Peter A. N. Bosman. "Optimal mixing evolutionary algorithms". In: *Proceedings of the 13th annual conference on Genetic and evolutionary computation*. ACM. 2011, pp. 617–624.

[4] S. Mostaghim, J. Teich, and A. Tyagi. "Comparison of data structures for storing Pareto-sets in MOEAs". In: *Proceedings of the 2002 Congress on Evolutionary Computation*. Vol. 1. 2002, pp. 843–848.

[5] Carlos M. Fonseca, Lus Paquete, and Manuel Lpez-Ibnez. "An improved dimension-sweep algorithm for the hypervolume indicator". In: *Proceedings of the 2006 Congress on Evolutionary Computation*. IEEE. 2006, pp. 1157–1163.

[6] Andrzej Jaszkiewicz, Hisao Ishibuchi, and Qingfu Zhang. "Multiobjective memetic algorithms". In: *Handbook of Memetic Algorithms*. Springer, 2012, pp. 201–217.

[7] Martin Pelikan, David E. Goldberg, and Fernando G. Lobo. "A survey of optimization by building and using probabilistic models". In: *Computational optimization and applications* 21.1 (2002), pp. 5–20.

[8] Ilan Gronau and Shlomo Moran. "Optimal implementations of UPGMA and other common clustering algorithms". In: *Information Processing Letters* 104.6 (2007), pp. 205–210.

[9] Stuart A. Kauffman and Edward D. Weinberger. "The NK model of rugged fitness landscapes and its application to maturation of the immune response". In: *Journal of Theoretical Biology* 141.2 (1989), pp. 211–245.

[10] Martin Pelikan, Kumara Sastry, Martin V. Butz, and David E. Goldberg. "Performance of Evolutionary Algorithms on Random Decomposable Problems". In: *Parallel Problem Solving from Nature - PPSN IX*. Vol. 4193. Springer, 2006, pp. 788–797.

[11] Martin Pelikan, Kumara Sastry, and David E. Goldberg. "Multiobjective hBOA, Clustering, and Scalability". In: *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation*. ACM, 2005, pp. 663–670.

[12] Sbastien Verel, Arnaud Liefooghe, Laetitia Jourdan, and Clarisse Dhaenens. "On the structure of multiobjective combinatorial search space: MNK-landscapes with correlated objectives". In: *European Journal of Operational Research* 227.2 (2013), pp. 331–342.

[13] Harold Hotelling and Margaret Richards Pabst. "Rank Correlation and Tests of Significance Involving No Assumption of Normality". In: *The Annals of Mathematical Statistics* 7.1 (1936), pp. 29–43.

[14] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. "A fast and elitist multiobjective genetic algorithm: NSGA-II". In: *IEEE Transactions on Evolutionary Computation* 6.2 (2002), pp. 182–197.

[15] Peter A. N. Bosman and Dirk 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.

[16] Shih-Huan Hsu and Tian-Li Yu. "Optimization by Pairwise Linkage Detection, Incremental Linkage Set, and Restricted / Back Mixing: DSMGA-II". In: *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. ACM, 2015, pp. 519–526.

[17] Peter A.N. Bosman and Dirk 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.

[18] Georges R. Harik and Fernando G. Lobo. "A Parameter-less Genetic Algorithm". In: *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation*. Morgan Kaufmann Publishers Inc., 1999, pp. 258–265.

[19] Willem den Besten, Dirk Thierens, and Peter A. N. Bosman. "The Multiple Insertion Pyramid: A Fast Parameter-Less Population Scheme". In: *Parallel Problem Solving from Nature - PPSN XIV*. Springer, 2016, pp. 48–58.