Utrecht University

Department of Information and Computing Sciences

Master Thesis Game and Media Technology

ICA-4087658

Performance and effectiveness of Linkage Tree Gene-pool Optimal Mixing Evolutionary Algorithm on Tree Decomposition Mk Landscapes

Author:
Tobias van Driessel, BSc

Supervisor:
dr. ir. D. Thierens

July 9, 2021

Abstract

Whitley et al.[23] recently introduced Mk Landscapes and Tree Decomposition (TD) Mk Landscapes, which are generalizations of NK Landscapes and Adjacent NK Landscapes, respectively. TD Mk Landscapes are convenient for benchmarking optimization algorithms, as the global optimum can be found in polynomial time using the problem structure, but could be difficult to find for a black-box optimization algorithm. Contrary to Whitley et al., who tested gray box algorithms (algorithms with problem structure knowledge), we test a black box linkage learning (LL) algorithm, namely Linkage Tree Gene-pool Optimal Mixing Evolutionary Algorithm (LT-GOMEA), on TD Mk Landscapes. We test the performance and effectiveness of LT-GOMEA on various subclasses of TD Mk Landscapes with specific codomain and topological properties, and use the results to quantify the effects of certain subclasses and landscape features. In particular, we are interested in the effects of an increase of overlap between the cliques in the clique tree (= TD) and an increase in the branching factor of the clique tree, on TD Mk Landscapes with the deceptive trap codomain. To generate the TD Mk Landscapes for our experiments, we use the recently introduced CliqueTreeMk algorithm by Thierens et al.[20]. Interestingly, our results show that the performance and effectiveness of LT-GOMEA does not solely decrease with increasing overlap on the deceptive trap codomain. Instead, the performance and effectiveness decrease prior to increasing. Furthermore, our results indicate that both the overlap and branching increase the interference between the subfunctions, possibly leading to decreased deceptiveness and decreased importance of learning the (exact) linkage. Finally, the branching factor of the clique tree has a great effect on the number of global optima of the TD Mk Landscape and should therefore be further researched. In conclusion, our results suggest that the TD Mk Landscapes is a promising and convenient benchmark for black box genetic algorithms.

Contents

Chapter 1

Introduction

For some problems, we do not know the underlying problem structure, but still want to find a (reasonably) good solution. Algorithms that solve these kinds of problems are called black box optimizers, whereas algorithms that make use of the problem structure are called gray box optimizers. An example black box problem is a computationally expensive simulation for which we can get a fitness score for a given solution, but we do not know the underlying mechanisms of the simulation, or it is prohibitively expensive to use this problem structure. For this example problem, we have no knowledge of the underlying problem structure, so we can not use gray box optimizers. Because more such black box problems exist, it is interesting to study black box optimizers and their performance in various scenarios, to identify the strong and weak points of each.

A black-box algorithm that achieved state-of-the-art performance for discrete, Cartesian-space optimization problems[3] is the Linkage Tree Gene-pool Optimal Mixing Evolutionary Algorithm (LT-GOMEA) by Thierens et al.[18]. It is a population-based search algorithm that mixes genes of solutions in the population to generate offspring, and every generation it tries to learn the problem structure from the solutions in the population. This learned problem structure is then used during the mixing to prevent good partial solutions from being disrupted during the mixing. Since its conception in 2010, it has been improved and has been tested on various problems, such as (Adjacent) NK Landscapes, 2D spin glasses, Deceptive Trap problems and MAXCUT. Adjacent NK Landscapes was often used, as its global optimum can be calculated in polynomial time using dynamic programming, while being a non-trivial black box problem for LT-GOMEA and other black box optimizers. Ideally, the structure of a benchmark should be completely known and importantly, its global optimum. A known global optimum allows for the measurement of the overall performance and effectiveness, and is therefore an important property of benchmarks. This possibility to calculate the global optimum explains the popularity of Adjacent NK Landscapes.

Although (Adjacent) NK Landscapes are popular as a benchmark for optimization algorithms, its constraints ($M = N$, $k = K + 1$, and variable x_i must appear in subfunction f_i) are unnecessary for most benchmark purposes, as they turn out not to be important for most fundamental theoretical properties of NK Landscapes [22]. Whitley et al.[23] therefore recently introduced the term *Mk Landscapes* to refer to any k-bounded pseudo-Boolean optimization problem, thus a generalization of NK Landscapes without these constraints. Additionally, they introduced the term *Tree Decomposition Mk Landscapes* to refer to any Mk Landscape with a known and bounded tree-width of k. This is a generalization of Adjacent NK Landscapes, as Adjacent NK Landscapes control tree-width by only considering adjacent variables for the subfunctions, but this constraint can be loosened to allow for any Mk Landscape that still has a bounded tree-width. Ultimately, this bounded tree-width is the key to calculate the global optimum (or optima) in polynomial time.

Conveniently, the overall performance and effectiveness of algorithms can be evaluated due to this polynomial time global optimum calculation. And although the global optimum is known, black box algorithms do not know the problem structure and global optimum, and therefore linkage learning will be necessary for particular codomains to find the global optimum reliably and efficiently. The possibility of evaluating the performance and effectiveness of algorithms, together with the difficulty of Tree Decomposition (TD) Mk Landscapes for particular codomains (for blackbox algorithms), make TD Mk Landscapes well suited as a benchmark function for blackbox Genetic Algorithms. As the global optimum can be calculated efficiently by a dynamic programming algorithm, TD Mk Landscapes are not suitable in the context of graybox algorithms, however, as these do know the problem structure.

In their work, Whitley et al.[23] introduced a construction algorithm to construct TD Mk Landscapes, however, it only constructs TD Mk Landscapes for which the subfunctions form a chain, much like an Adjacent NK Landscape. Recently, Thierens et al.[20] introduced an algorithm, CliqueTreeMk, to construct any TD Mk Landscape and calculate its global optimum (or optima) efficiently using dynamic programming when its codomain values are known. In this work, we introduce CliqueTreeMk in more detail and use the implementation by van Driessel et al.[6](included in Appendix B) to generate TD Mk Landscapes. Note that the algorithm was implemented as a part of this master thesis research, but has already been published.

Furthermore, to the best of our knowledge, there has not been a comparative study to the performance and effectiveness of LT-GOMEA for different subclasses of TD Mk Landscapes. Therefore, in this work, we try to find the performance and effectiveness of LT-GOMEA on various subclasses of TD Mk Landscapes, to identify properties of subclasses and their landscapes that influence the performance and effectiveness of LT-GOMEA. Importantly, this will also provide us with insights into TD Mk Landscapes, their subclasses and their landscapes. Therefore, our work contributes to the understanding of TD Mk Landscapes, and lays the foundation for future research and application of TD Mk Landscapes for benchmarking black box algorithms.

In summary, our contributions include 1) a detailed introduction of the CliqueTreeMk algorithm (construction and global optimum calculation) for TD Mk Landscapes, and 2) an experimental study into the performance and effectiveness of LT-GOMEA on certain subclasses of TD Mk Landscapes, which are defined by their codomain and topological properties.

We will introduce our research questions in Chapter 2, any necessary background knowledge in Chapters 3 to 6, the experiments with their results and discussion in Chapters 7 to 11, and finally the conclusion and future work in Chapter 12.

Chapter 2

Research Questions

For our research, we are interested in the effects of certain subclasses of TD Mk Landscapes on the performance and effectiveness of LT-GOMEA, to enhance our understanding of both TD Mk Landscapes and of LT-GOMEA. As we are curious to the results for LT-GOMEA, a linkage learning algorithm, the used codomain should not be trivial to solve. Therefore, we selected the deceptive trap function as the codomain for the subfunctions, and compare with a random codomain to see the differences in effect between the codomains. Furthermore, we vary two of the TD Mk Landscape's parameters: the overlap between the subfunctions o and the branching factor of the clique tree b. As the performance of various algorithms is already known for non-overlapping deceptive trap problems, it is interesting to look at the performance and effectiveness of LT-GOMEA for increasing overlap. Note that although Adjacent NK Landscapes do overlap (although it is fixed at $o = k - 1 = K$), and there are even Adjacent NK Landscapes variants with a different overlap setting, they do not use the deceptive trap codomain. Finally, the new TD Mk Landscapes offer a branching factor setting, which certainly is interesting to look into. In summary, our research questions are:

1. What is the performance and effectiveness of LT-GOMEA on *certain subclasses* of TD Mk Landscapes?
   (a) How does an increasing *overlap* affect the performance and effectiveness of LT-GOMEA for TD Mk Landscapes problems with the *deceptive trap codomain*?
   (b) How does an increasing *branching factor* affect the performance and effectiveness of LT-GOMEA for TD Mk Landscapes problems with the *deceptive trap codomain*?
   (c) How does an increasing *overlap* affect the performance and effectiveness of LT-GOMEA for TD Mk Landscapes problems with the *random codomain*?

Chapter 3

Pseudo-Boolean optimization problem

In this work, NK Landscapes will be discussed briefly and Mk Landscapes will be discussed in more detail, both of which are (k-bounded) pseudo-Boolean optimization problems.

Pseudo-Boolean optimization problems correspond to a class of functions $f : \mathbb{B}^n \to \mathbb{R}$ that map a boolean vector to a real value. Boolean functions are a special class of pseudo-Boolean functions where the codomain is also Boolean. k-bounded pseudo-Boolean optimization problems refers to the class of pseudo-Boolean optimization problems with a bounded nonlinearity of order k. In other words, k bounds the nonlinearity of the subfunctions; every subfunction has at most k terms. k-bounded pseudo-Boolean optimization problems can be expressed by

$$f(x) = \sum_{i=1}^{M} f_i(x, mask_i)$$

[23], where $x \in X$, X represents the set of solutions over a bit string with length N, M is the number of subfunctions, and f_i is the subfunction that uses a $mask_i$ to select k bits from the bit string x; these k bits are then used to evaluate f_i.

In the following chapters, we will discuss NK Landscapes and dynamic programming algorithms for NK Landscapes, after which we will discuss Tree Decomposition Mk Landscapes in more detail.

Chapter 4

NK Landscapes

NK Landscapes were introduced by Kauffman[13][12] to model fitness landscapes of problems found in biology. An NK Landscape refers to any k-bounded pseudo-Boolean optimization problem with 3 constraints: 1) The number of subfunctions M is equal to the problem size N, 2) $k = K + 1$, where K is the number of neighbours, and 3) variable x_i must appear in subfunction f_i. This can be expressed by

$$f(x) = \sum_{i=1}^{N} f_i(S_i)$$

where S_i is a subset of variables in x, which includes x_i and K random other variables.

Although Kauffman et al. considered multiple options for the choice of neighbours, Weinberger et al.[21] first introduced a more formal definition of the Adjacent NK Landscapes. Adjacent NK Landscapes introduce an additional constraint: the K neighbours must be restricted to the following K bits; subfunction f_i takes as input variables x_i to $x_{(i+K) \bmod N}$. This can be expressed by

$$f(x) = \sum_{i=1}^{N} f_i(x_i, x_{i+1}, ..., x_{(i+K) \bmod N})$$

Note the wrap around in the definition, which makes the original definition cyclic. Whitley et al.[23] therefore use the terms Cyclic Adjacent NK Landscape and Acyclic Adjacent NK Landscapes, for the wrapping around and non-wrapping around variants, respectively. Additionally, they introduce an algorithm to convert any Cyclic Adjacent NK Landscape into an Acyclic Adjacent NK Landscape with $N_{acyclic} = N_{cyclic}$ and $K_{acyclic} = 2K_{cyclic}$[23].

4.1 Dynamic Programming for Adjacent NK Landscapes

The complexity of optimizing NK Landscapes depends on its parameters, with the value of K being the most important: for $K = 0$, the NK Landscape is unimodal and the global optimum can be found in linear time. For $K = 1$, the global optimum can be obtained in polynomial time, but for $K > 1$, the problem is NP-complete[21][24]. However, it is polynomially solvable if the variable interaction graph of the NK Landscape has a bounded tree width [5][4], e.g. when the neighbours are restricted to adjacent positions (Adjacent NK Landscapes) [4][21][24], or if generated according to a certain distribution[7]. Finally, there exists a polynomial-time approximation algorithm for $K > 1$, with approximation threshold $1 - 1/2^{K+1}$[24].

For benchmarking purposes, being able to calculate the global optimum of Adjacent NK Landscapes in polynomial time is very convenient, as one can determine whether a tested black-box optimization algorithm has found the global optimum. It is then possible to report the effectiveness and overall performance of an algorithm. Multiple dynamic programming algorithms have been proposed to find the global optimum of Adjacent NK Landscapes; in the following subsections we introduce the algorithms by Hammer[9], Weinberger[21], Wright[24], and Pelikan[17].

It is important to note that all these algorithms require knowledge of the underlying fitness structure to calculate the global optimum in polynomial time. The evolutionary algorithms we will introduce in Chapter 6 do not have such knowledge of the underlying fitness structure and thus we call these black box optimizers. There are many cases in which one does not know the underlying fitness structure of the problem at hand, but can only calculate a resulting fitness, and for these cases the black box optimizers offer a way to calculate a (reasonably) good solution.

4.1.1 Hammer

Hammer et al. [9] introduced a dynamic programming algorithm to calculate the global optimum of pseudo-Boolean functions, of which a simpler version was later introduced by Hammer et al.[11]. Crama et al.[4] then showed that the algorithm runs in polynomial time ($O(N \cdot 2^{2K})$) when the variable interaction graph of the NK Landscape has a bounded tree width. Additionally, they introduced a branch-and-bound version of the algorithm to improve the run time and considered some implementation details to decrease memory usage.
The basic idea of Hammer's algorithm is to recursively decrease the problem size to $(N-1)$ by introducing a function that has the same optimum as the previous one, but eliminates one variable.

Let f_1 be the function to be maximized and write:

$$f_1(x_1, x_2, ..., x_N) = x_1 g_1(x_2, x_3, ..., x_N) + h_1(x_2, x_3, ..., x_N),$$

where g_1 and h_1 do not depend on x_1. As f_1 needs to be maximized, x_1 should be 1 if $g_1(x_2, x_3, ..., x_N) > 0$ and 0 if $g_1(x_2, x_3, ..., x_N) \leq 0$. Define ψ_1 such that $\psi_1 = g_1(x_2, x_3, ..., x_N)$ if $g_1(x_2, x_3, ..., x_N) > 0$ and $\psi_1 = 0$ if $g_1(x_2, x_3, ..., x_N) \leq 0$. Assume that a polynomial expression of ψ_1 has been obtained. Let $f_2 = \psi_1 + h_1$, now the problem size is reduced by 1, as f_2 is not dependent on x_1. If we iteratively apply this procedure until f_N, we have reduced the problem to size 1, as it is then only dependent on x_N. We can use f_N to set x_N to its maximizing value, x_N^*. Finally, we can construct the global optimum by iterating in the reversed order and setting $x_i = 1$ if $\psi_i(x_{i+1}^*, x_{i+2}^*, ..., x_N^*) > 0$, for $(i = 1, 2, ..., N-1)$.

4.1.2 Weinberger

Weinberger [21] introduced a polynomial dynamic programming algorithm ($O(2^k N)$) to calculate the global optimum for cyclic Adjacent NK Landscapes. It calculates the best achievable score for all variables up to a certain variable, and iteratively adds all variables to end up with a best achievable score for the whole variable string. We consider $K = 1$, with $b_i \in \Sigma$ and $\Sigma \in (0,1)$ for $i = (1, 2, ..., N)$. $f_i^{b_{i-1} b_i b_{i+1}}$ is the subfunction for variable i, and $F_i^{b_N b_1 | b_{i+1} b_{i+2}}$ is the maximum value of the sum $\sum_{j=1}^{i+1} f_j^{b_{j-1} b_j b_{j+1}}$, over the variables $b_2, b_3, ..., b_i$, given the values of variables b_N, b_1, b_{i+1}, and b_{i+2}. The algorithm first calculates $F_1^{b_N b_1 | b_2 b_3} = f_1^{b_N b_1 b_2} + f_2^{b_1 b_2 b_3}$. Then every $F_i^{b_N b_1 | b_{i+1} b_{i+2}}$ is calculated for all $2 \leq i \leq N-1$, which is defined as $F_i^{b_N b_1 | b_{i+1} b_{i+2}} = \max\limits_{b_i}(F_{i-1}^{b_N b_1 | b_i b_{i+1}} + f_{i+1}^{b_i b_{i+1} b_{i+2}})$.

So, we are adding variables iteratively and finally $F_N^{b_N b_1 | b_N b_1}$ stores the maximum scores for the whole string, for the given b_N and b_1 values. If we then iterate over all possible b_N and b_1 values and choose the ones with the maximum score, we have calculated the maximum score for the problem: $F_{\text{MAX}} = \max\limits_{b_N b_1}(F_{N-1}^{b_N b_1 | b_N b_1})$.

4.1.3 Wright

Wright et al.[24] introduced a polynomial dynamic programming algorithm ($O(N \cdot 2^{3K}/K)$) to calculate the global optimum for cyclic Adjacent NK Landscapes. Given an example problem with size $N = 4$; (a_0, a_1, a_2, a_3), we first consider $K = 1$, with again $a_i \in \Sigma$ and $\Sigma \in (0,1)$ for $i = (0, 1, ..., N-1)$. The idea is to reduce the problem to size $N-1$ by eliminating one problem variable and its fitness function (a_3), which is done by storing in a matrix the best fitness achievable for the given values of the two surrounding variables (a_2, a_0). We can then use this stored value to calculate the best fitness achievable while eliminating the next variable, and so on, until we reach $N = 2$. Then we iterate over all 2^{K+1} possible instances for the last remaining variables, to calculate the global optimum.
Now we shortly consider $K > 1$: Wright et al. handle $N \bmod K = 0$ and $N \bmod K \neq 0$ separately. Here we highlight the $N \bmod K = 0$ case and reference the reader to their work for the other, as it follows the same general idea with some tweaks, and thus does not add any value. A string of length N over Σ can be considered as a string of length N/K over Σ^K. The evaluation of the subfunction f_i needs access to $K+1$ variables, which are contained in just two positions in the

string over alphabet Σ^K. If $\widetilde{f}_i = \sum_{j=K\cdot i}^{K\cdot i+K-1} f_j$, then $\widetilde{f}_i$ only depends on $\widetilde{a}_i$ and $\widetilde{a}_{i+1}$ of the string over alphabet Σ^K. Therefore, the same algorithm as for $K = 1$ can be used.

4.1.4 Pelikan

Pelikan et al. [17] introduced a dynamic programming algorithm to calculate the global optimum of an acyclic Adjacent NK Landscape in $O(N \cdot 2^K)$, and thus a cyclic Adjacent NK Landscape in $O(N \cdot 2^{2K})$. They consider an Adjacent NK Landscape variant with a variable number of overlapping bits o between subfunctions.

For $i = (0, 1, ..., M - 1)$ and $j \in \{0, 1, ..., 2^o - 1\}$, the maximum fitness of the first $i + 1$ subfunctions for integer value j of the overlapping bits shall be stored in a matrix G. For the first subfunction ($i = 0$), iterate over all 2^{K+1} instances and calculate their fitness. Record for each value j of the o overlapping bits the maximum fitness of all 2^{K-o} instances of the remaining $K - o$ bits, and store this fitness in $g_{i,j}$. Then, for the ith subfunction ($i = (1, ..., M - 1)$), go over all 2^{K+1} instances and calculate what the maximum fitness would be for the first $i + 1$ subfunctions combined: sum the stored maximum fitness for $i - 1$ for the overlapping bits ($g_{i-1,j}$) and the fitness for this subfunction. Just as for $i = 0$, we store the maximum (summed) fitness in $g_{i,j}$ for each value j of the o overlapping bits. Finally, the global optimum can be retrieved by choosing the maximum fitness in $g_{M-1,j}$ for $j \in \{0, 1, ..., 2^o - 1\}$. To get the resulting bit string, backtrack over the decisions made for each subfunction.

Chapter 5

Mk Landscapes

Although (Adjacent) NK Landscapes are popular as a benchmark for optimization algorithms, its constraints ($M = N$, $k = K + 1$, and variable x_i must appear in subfunction f_i) are unnecessary for most benchmark purposes, as they turn out not to be important for most fundamental theoretical properties of NK Landscapes, according to Whitley[22]. Whitley et al. therefore recently introduced the term *Mk Landscapes* to refer to any k-bounded pseudo-Boolean optimization problem, thus a generalization of NK Landscapes without these constraints. M is the number of subfunctions and k is a constant that provides an upper bound on the interaction order size of the subfunctions, with M being polynomial in N. Note that this is a new term for an old concept, as spin glass problems (originated in physics) are general enough to refer to any k-bounded pseudo-Boolean optimization problem as well. According to Whitley et al., *"None of the requirements of NK Landscapes turn out to be important to most fundamental theoretical properties of NK Landscapes. [...] By transferring what we know about NK Landscapes to Mk Landscapes, we remove unnecessary restrictions and create new connections to problems such as MAX-kSAT and spin glass problems."*[23].

We now introduce some background knowledge, before introducing some specific variants of Mk Landscapes that allow dynamic programming algorithms to calculate the global optimum in polynomial time. Again, note that these dynamic programming algorithms use the underlying problem structure to calculate the global optimum in polynomial time.

5.1 Background knowledge

5.1.1 Variable Interaction Graph

Definition 1 ([23])**.** *A Variable Interaction Graph is a graph $G(V, E)$ where the vertex set $V = x_1, ..., x_N$ corresponds to the set of variables in the Mk Landscape and edge $(x_i, x_j) \in E$ if and only if variables x_i and x_j have a nonlinear interaction.*

We will, just as Whitley et al. did, use the following subfunctions to illustrate the variable interaction graph concept. $N = 12$ and $k = 3$. The subfunctions are:

$$\begin{array}{ll}
f_1(x_1, x_4, x_8), & f_2(x_2, x_3, x_5), f_3(x_3, x_2, x_{10}), \\
f_4(x_4, x_2, x_1), & f_5(x_5, x_7, x_4), f_6(x_6, x_8, x_1), \\
f_7(x_7, x_3, x_5), & f_8(x_8, x_9, x_{11}), f_9(x_9, x_7, x_8), \\
f_{10}(x_{10}, x_6, x_2), & f_{11}(x_{11}, x_7, x_3), f_{12}(x_{12}, x_1, x_6).
\end{array}$$

This set of subfunctions results in the Variable Interaction Graph as shown in Figure 5.1. It is assumed that the subfunctions induce nonlinearity on all variables in the subfunction.

5.1.2 Tree Decomposition / Clique Tree

Before introducing the Tree Decomposition / Clique Tree definition, we need to introduce some additional definitions:

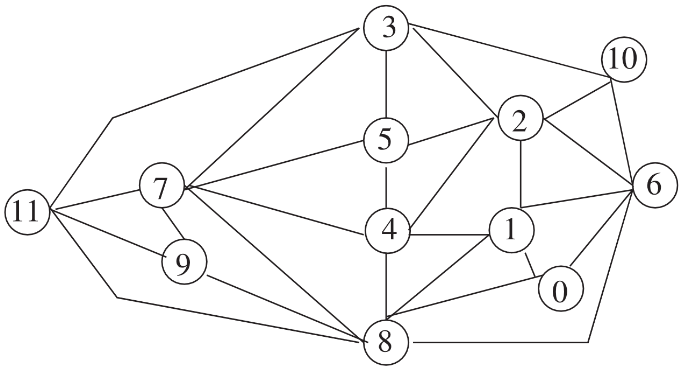


Figure 5.1: "The Variable Interaction Graph (VIG) tracks sources of nonlinearity."[23]

Definition 2. *A clique is a set of vertices such that every pair of vertices in that set is adjacent; the subgraph induced by the clique is complete.*

Definition 3. *A maximal clique is a clique that can not be extended by adding one more adjacent vertex.*

Definition 4 ([14])**.** *Let G be a connected undirected graph, and let $C_1, ..., C_M$ be the set of maximal cliques in G. Let T be any tree-structured graph whose nodes correspond to the maximal cliques $C_1, ..., C_M$. Let C_i, C_j be two cliques in the tree that are directly connected by an edge; we define $S_{i,j} = C_i \cap C_j$ to be a separator between C_i and C_j. Let $W_{<(i,j)}$ ($W_{<(j,i)}$) be all of the variables that appear in any clique on the C_i (C_j) side of the edge. We say that a tree T is a clique tree for G if:*

1. *each node corresponds to a clique in G, and each maximal clique in G is a node in T*
2. *each separator $S_{i,j}$ separates $W_{<(i,j)}$ and $W_{<(j,i)}$ in G.*

Note that a clique tree is just a different term for the same concept as a tree decomposition or a junction tree. We will use 'tree decomposition' and 'clique tree' interchangeably, depending on what suits the context.

As mentioned in the NK Landscape chapter, Hammer's algorithm calculates the global optimum in $O(N \cdot 2^{2K})$ for Adjacent NK Landscapes. This can be explained by Hammer's reported run time of $O(N \cdot 2^t)$, where t is the tree-width of the Tree Decomposition of the Variable Interaction Graph. For an acyclic Adjacent NK Landscape, the Tree Decomposition is a chain with node i consisting of the variables in subfunction i (f_i). Therefore, acyclic Adjacent NK Landscapes have a tree-width of K, as the tree-width is the maximum node size -1 and every subfunction has $K+1$ variables, resulting in $O(N \cdot 2^K)$ for Hammer's algorithm. However, as mentioned in the introduction for NK Landscapes, Whitley et al. have shown that every Cyclic Adjacent NK Landscape can be converted into an Acyclic Adjacent NK Landscape with $N' = N$ and $K' = 2K$. Therefore, any Adjacent NK Landscape can be optimized in $O(N \cdot 2^{2K})$ using Hammer's algorithm.

We illustrate a Tree Decomposition for an Acyclic Adjacent NK Landscape with $N = 7$ and $K = 2$ in Figure 5.2 below:

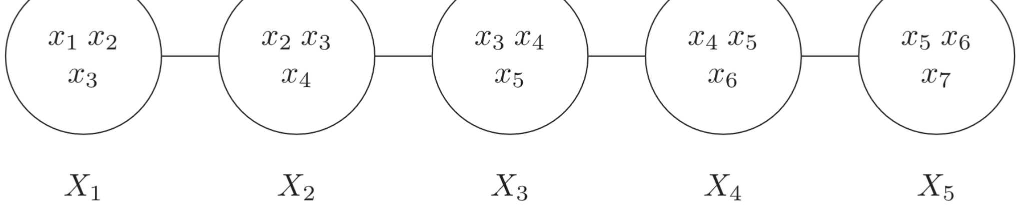


Figure 5.2: "Example of Tree Decomposition for an Acyclic Adjacent NK Landscape. The tree-width is 2."[23]

5.2 Localized and TD Mk Landscape

The fact that Adjacent NK Landscapes allow for their global optimum to be calculated in polynomial time raises the question whether general Mk Landscapes do as well. Computing the tree-width is NP-hard[4], but an upper bound on the tree-width would be enough to calculate the global optimum in polynomial time (using Hammer's algorithm). This upper bound for the tree-width can be found using greedy algorithms [23]. In short, polynomial optimization is dependent on whether a polynomial upper bound can be found for the tree-width.

Adjacent NK Landscapes control tree-width by only considering adjacent variables for the subfunctions, Whitley et al. used this idea to create Localized Mk Landscapes (analogous to Adjacent NK Landscapes) which can be generalized to create Tree Decomposition Mk Landscapes. Both control the tree-width, thereby allowing for optimization in polynomial time, if $k \in O(\log N)$.

Definition 5 ([22]). *A Localized Mk Landscape is an Mk Landscape where each subfunction f_i is defined over a window of variables from x_i to $x_{((i+K) \bmod N)}$, however, the number of variables used in each subfunction can be less than $K + 1$.*

Whitley et al. illustrate Localized Mk Landscapes using the following example:

$$f(x) = f_1(x_1, x_3, x_4) + f_2(x_4, x_5) + f_3(x_5, x_6).$$

The function is not an Adjacent NK Landscape, as the subfunctions' variables are not adjacent. However, dummy variables can be introduced to make these subfunctions adjacent:

$$f(x) = f_1(x_1, x_2, x_3, x_4) + f_2(x_2, x_3, x_4, x_5) + f_3(x_3, x_4, x_5, x_6).$$

Localized Mk Landscapes can be optimized in polynomial time [22], while allowing for a greater variety in functions than Adjacent NK Landscapes. They propose a more general result in their later work, as not this adjacency, but the bounded tree-width is of importance:

Definition 6 ([23]). *A Tree Decomposition Mk Landscape (TD Mk Landscape) is any Mk Landscape which has a known and bounded tree-width of k.*

It is easy to see that Tree Decomposition Mk Landscapes can be optimized in polynomial time if $k \in O(\log N)$, given that a known and bounded tree-width is required to optimize in polynomial time ($O(N \cdot 2^{2k})$). TD Mk Landscapes allow for more problems to be used as part of a benchmark and real-world problems can be transformed into a TD Mk Landscape if a tree decomposition of some fixed width is already known.

Next, we will introduce two construction algorithms and one dynamic programming algorithm for TD Mk Landscapes. First, the construction algorithm by Whitley et al.[23] is introduced, then the novel construction algorithm and global optimum dynamic programming algorithm by Thierens et al.[20] are introduced.

5.3 TD Mk Landscape Algorithms

5.3.1 Construction Whitley

Whitley et al.[23] introduced a construction algorithm for TD Mk Landscapes, however, it is important to note that it limits the construction to TD Mk Landscapes with a tree decomposition of a chain, just like Adjacent NK Landscapes. It is therefore still limited and can not construct all TD Mk Landscapes.

An $M \times k$ matrix is constructed, where the rows correspond with the subfunctions and their variables. The variables must appear in contiguous rows and all N variables must appear in at least one row. If constructed in this way, a tree decomposition can be made with tree-width $k-1$, where every row of the matrix is represented by a node in the tree. They illustrated the construction with a figure as depicted in Figure 5.3, where $N = 23$, $M = 10$, and $k = 6$.

5.3.2 CliqueTreeMk Introduction

We introduce Thierens's CliqueTreeMk construction and global optimum calculation algorithms[20], as the CliqueTreeMk algorithm consists of these two phases. The construction algorithm allows for

	V1	V2	V3	V4	V5	V6
f_1	23	22	20	8	9	21
f_2	7	10	20	8	9	21
f_3	7	10	6	8	9	1
f_4	7	10	6	5	2	1
f_5	3	4	6	5	2	1
f_6	3	4	6	5	2	15
f_7	3	4	6	5	11	15
f_8	3	14	12	5	11	15
f_9	13	14	12	16	11	15
f_{10}	13		19	16	17	18

Figure 5.3: "Example of $M \times k$ lookup table of variables of a TD Mk Landscape. In the example $N = 23$, $M = 10$, and $k = 6$. Each row of the table can become a subfunction in an Mk Landscape, with variables V1 to V6. The table also corresponds to a Tree Decomposition of that same set of functions."[23]

the construction of any TD Mk Landscape, as, contrary to Whitley's algorithm, a branching factor b can be passed, which determines the branching factor in the constructed Tree Decomposition of the Variable Interaction Graph. Essentially, Thierens's algorithm can construct TD Mk Landscapes with an actual *tree* decomposition, whereas Whitley's algorithm is limited to landscapes with a *chain* decomposition, due to the implicit use of a branching factor of 1.

In the context of CliqueTreeMk, the term clique tree is more suitable than tree decomposition, as we use the terms clique and separator intensively. Whitley's algorithm output could then be regarded as a clique *chain* instead of a clique *tree*. We use the term *clique* to represent the variables in a subfunction, and the term *separator* to represent the overlapping variables between two cliques/subfunctions, as these terms reflect their properties in a tree decomposition/clique tree in a succinct manner.

The idea behind CliqueTreeMk's construction algorithm is to construct the TD Mk Landscape by directly generating a clique tree with the exact properties as required by the parameters, in order to ensure that a clique tree with the required properties can be constructed. Its input topology parameters are the number of subfunctions/cliques M, number of variables per subfunction/clique k, number of overlapping bits between subfunctions/cliques o, and branching factor b. The branching factor represents the number of branches in the clique tree. The problem length N can be represented by $N = (M - 1) \cdot (k - o) + k$, as the first clique/subfunction takes k variables, and every other clique/subfunction overlaps o variables with another clique/subfunction and adds $k - o$ unused variables to get to length k.

The general idea of CliqueTreeMk's *construction* algorithm is to first construct clique C_0 as the root of the clique tree by assigning the first k variables from the shuffled variable list, and then generate b children cliques ($C_{j \in \text{children}_i}$) for every clique C_i until we have constructed M cliques. Each child C_j overlaps with its parent C_i for o variables, described by the separator S_j between C_i and C_j, and the remaining $k - o$ variables are taken from the shuffled variable list to complete C_j.

The *global optimum dynamic programming* algorithm then uses this clique tree structure with its cliques and separators to calculate the global optimum. It is comparable to Pelikan's[17] dynamic programming approach in the way it stores the $k - o$ remaining variables's maximizing values for the values of the o overlapping variables (separator variables). Starting at the leaves of the tree, for each separator S_j we store for each of the instances of the separator variables the maximizing variable values for its child clique C_j and the resulting score. Then, we can iterate in the reverse direction and assign values to the clique variables in C_j based on the maximizing values for its variables stored in its parent separator S_j.

We illustrate the CliqueTreeMk algorithm during these phases using an example instance with number of subfunctions/cliques $M = 7$, subfunction/clique size $k = 3$, and overlap $o = 2$. Together, these define length $N = 9$. Furthermore, we choose a branching factor $b = 2$. The construction algorithm uses fixed values for k, o, and b, but the algorithm can be extended to allow for non-fixed values during construction. Likewise for the dynamic programming algorithm. The variables are randomly ordered: $(x_4, x_2, x_7, x_5, x_1, x_9, x_3, x_8, x_6)$.

5.3.3 CliqueTreeMk Construction

The algorithm is described in a textual version below and a pseudocode version in Algorithm 1.

1. Initially, take the first k variables as the root clique C_0. Otherwise, take the next clique C_i to expand.
2. Choose o random variables from parent clique C_i, assign to separator S_j
3. Take next $(k-o)$ not chosen variables and add the variables from S_j to construct child clique C_j
4. Go to step 2 until b branches have been built
5. Go to 1 to expand the next clique

Algorithm 1: CliqueTreeMk Construction

```
Input: $M$, $k$, $N$, $b$, $o$, shuffled list of variables
Result: Clique tree
$C_0 \leftarrow$ first $k$ variables;
$count \leftarrow 1$;
for $i \leftarrow 0$ to $M-2$ do
    for $j \leftarrow 0$ to $b-1$ do
        $S_{count} \leftarrow o$ random variables from clique $C_i$;
        $x \leftarrow$ next $(k-o)$ unused variables;
        $C_{count} \leftarrow S_{count} \cup x$;
        $count \leftarrow count + 1$;
        if $count == M$ then
            return clique tree;
    end
end
```

Following the algorithm with the given example instance could result in the following list of cliques: (x_4, x_2, x_7), (x_4, x_7, x_5), (x_4, x_2, x_1), (x_7, x_5, x_9), (x_4, x_7, x_3), (x_4, x_1, x_8), (x_2, x_1, x_6)
In Figure 5.4 we illustrate the constructed clique tree with its separators.

Essentially, the algorithm creates a clique tree / tree decomposition that adheres to the given constraints, defined by the input topology parameters. Importantly, it adheres to the running intersection property, as problem variables are either part of a single clique C_i or part of multiple cliques that are directly connected by separators. This follows from steps 2 and 3 of the textual version: during construction of a clique C_j, each variable is either taken from the unused problem variables list or copied from the parent clique C_i (and added to the separator S_j), with C_j being a child of C_i. The dynamic programming algorithm that calculates the global optimum requires this running intersection property to select the best value for variables in isolation: for $k-o$ variables at every clique and for o variables at every separator. It is able to calculate the global optimum in polynomial time due to the bounded (and known) tree-width.

5.3.4 CliqueTreeMk Global Optimum Dynamic Programming Algorithm

To explain the dynamic programming algorithm, we first introduce it in a textual form, and then we introduce it in more detail using some formulas.

The CliqueTreeMk global optimum solver follows very similar steps to the dynamic programming algorithm by Pelikan et al.[17]. The CliqueTreeMk global optimum solver traverses the clique tree from the leaves to the root, storing for each instance of separator S_i (o overlapping bits) the maximizing values for the $k-o$ variables in $C_i \setminus S_i$ with its score. The maximizing values for $C_i \setminus S_i$ are stored in K_i and the accompanying score is stored in h_i. Then, for each possible instance of the clique root C_0, the best achievable score g_0 is calculated using its children separators S_j and the

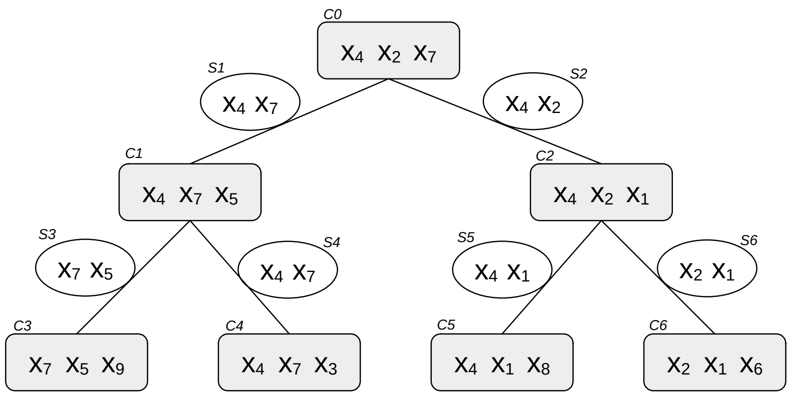


Figure 5.4: Example clique tree with cliques C0 to C6 and separators S1 to S6.

stored best achievable score in h_j for that instance of the separator variables. The highest score of these possible instances is the global optimum (or global optima). To assemble the global optimum solution, C_0's maximizing instance is written to the solution and the clique tree is traversed from the root to the leaves, storing the maximizing values for the $k-o$ variables from each K_i into the solution.

If there are multiple global optima, then there are multiple maximizing instances for one or more separators S_i. Each of these maximizing instances for S_i is stored in K_i. When one of these cases of multiple maximizing instances is encountered during the assembly of the global optima, the current global optimum is copied a number of times, according to the number of maximizing instances in K_i (minus one). Finally, each of these copies is assigned one of the maximizing instances and the traversal of the clique tree is continued. Each of these global optima solutions is now considered at every remaining separator in the clique tree. More specifically, we can define $\forall$ separators S_i:
$h_i(a_1, ..., a_o) = g_i(a_1, ..., a_o, a^*_{o+1}, ..., a^*_k)$ with
$a_1, ..., a_o \in S_i$, $a_{o+1}, ..., a_k \in C_i \setminus S_i$ and $a^*_{o+1}, ..., a^*_k$ maximizing g_i for values $a_1, ..., a_o$.
$K_i(a_1, ..., a_o) = \{a^*_{o+1}, ..., a^*_k\}$
And $\forall$ cliques C_i:
$g_i(a_1, ..., a_k) = f_i(a_1, ..., a_k) + \sum_{j \in \text{children}_i} h_j(b_1, ..., b_o)$

To illustrate these, we can define the previous specifically for our example instance. We define $\forall$ separators S_i:
$h_i(x_a, x_b) = g_i(x_a, x_b, x^*_c)$ with $x_a, x_b \in S_i$ and $x_c \in C_i \setminus S_i$ and x^*_c maximizing g_i for x_a and x_b values.
$K_i(x_a, x_b) = \{x^*_c\}$
And $\forall$ cliques C_i: $g_i(x_p, x_q, x_r) = f_i(x_p, x_q, x_r) + h_{child1}(x_p, x_q) + h_{child2}(x_p, x_r)$

Using the above formulas, we can write a shorter version of the algorithm: For every possible instance of the problem variables in C_0, calculate g_0. Calculating g_0 will recursively calculate all the g_i, h_i, and K_i values for $i > 0$. The maximum of these g_0 values is the global optimum of the TD Mk Landscape and can be used to retrieve the bit string that achieves this fitness. This is done by acquiring the stored maximizing values for each separator S_i from K_i and assigning their values to the global optimum solution. Or in a more pseudo code way:

1. For each possible instance of problem variables in C_0, calculate g_0
2. Maximum g_0 is global optimum
3. Take next separator, starting with S_1
4. Take maximizing values from K_i, for problem variable values already in global optimum solution, and put them in global optimum solution
5. Go to step 3 to assign all problem variable values

We illustrate the algorithm using the example used in the previous subsection. We use the following deceptive trap function for each subfunction:

$$\begin{aligned} f_i(x_a, x_b, x_c) : 111 &=> 4 \\ 000 &=> 2 \\ \text{otherwise} &=> 2 - c(x_a, x_b, x_c) \end{aligned}$$

where c returns the number of ones in the passed variable values.

We show the calculated h_i and K_i values for S_6 and S_1, as $i \in \{3, 4, 5, 6\}$ have the same h_i and K_i values and likewise for $i \in \{1, 2\}$. Then we show the construction of the global optimum using the calculation of g_0 for C_0.

$$C_6 = \{x_2, x_1, x_6\},\ S_6 = \{x_2, x_1\}$$
$$g_6(x_2, x_1, x_6) = f_6(x_2, x_1, x_6)$$

$S_6 = x_2x_1$	00	01	10	11
$h_6(x_2, x_1)$	2	1	1	4
$K_6 = x_6^*$	0	0	0	1

In the above table, we list the possible instances of the separator variables x_2 and x_1, the maximizing values of the remaining variable x_6 in C_6 for these instances (K_6), and the resulting scores for these maximizing values (h_6). Because C_6 is one of the leaves, g_6 is equal to f_6. We can see the deceptive attractor at work here, attracting any instance of the separator variables that does not contain a part of the local optimum.

$$C_1 = \{x_4, x_7, x_5\},\ S_1 = \{x_4, x_7\}$$
$$g_1(x_4, x_7, x_5) = f_1(x_4, x_7, x_5) + h_4(x_4, x_7) + h_3(x_7, x_5)$$

$S_1 = x_4x_7$	00	01	10	11
$h_1(x_4, x_7)$	$2 + 2 + 2$ $= 6$	$0 + 4 + 1$ $= 5$	$0 + 1 + 4$ $= 5$	$4 + 4 + 4$ $= 12$
$K_1 = x_5^*$	0	1	1	1

Because C_1 does have children cliques, the calculation of h_1 and thus of g_1 does involve the h_i values of its children, h_4 and h_3.

$$C_0 = \{x_4, x_2, x_7\},\ S_0 = \emptyset$$
$$g_0(x_4, x_2, x_7) = f_0(x_4, x_2, x_7) + h_2(x_4, x_2) + h_1(x_4, x_7)$$

$x_4x_2x_7$	000		111
$g_0(x_4, x_2, x_7)$	$2 + 6 + 6$ $= 14$	...	$4 + 12 + 12$ $= 28$

Finally, we calculate the g_0 values for all possible instances of the problem variables in C_0. Here we have illustrated just two cases, instances 000 and 111 for $x_4x_2x_7$. Note that this table differs from the two before in the things we calculate; here we don't calculate h_i values, as there is no separator. Instead, we calculate all g_0 values and record the maximum value as the global optimum (or global optima).

For this example, the global optimum value is 28. The maximizing instance for C_0, while considering the rest of the clique tree using dynamic programming, is $x_4^*x_2^*x_7^* = 111$. We can now traverse the clique tree to assign the other bits of the global optimum solution. First, $S_1 = \{x_4, x_7\}$, as is shown in the table for C_6 / S_6, so we insert the values of x_4 and x_7 from our global optimum solution, which are 1 and 1. For instance $x_4x_7 = 11$, $K_1 = x_5^* = 1$, so we assign value 1 to x_5 in our global optimum solution. After doing this for all separators, our global optimum solution is 111111111.

5.3.5 CliqueTreeMk Discussion

The construction and global optimum algorithms of CliqueTreeMk are shown for TD Mk Landscapes with fixed subfunction size k, overlap o, and branching factor b, but CliqueTreeMk allows for these parameters to be non-fixed, although the algorithms need to be adjusted for this to be possible.

Chapter 6

LT-GOMEA

In this chapter we introduce Evolutionary Algorithms and specifically Linkage Tree Gene-pool Optimal Mixing Evolutionary Algorithm (LT-GOMEA), as in our experiment we will test LT-GOMEA's performance and effectiveness on the previously introduced TD Mk Landscapes. These Evolutionary Algorithms are black box optimizers, and hence have no knowledge of the underlying problem structure, contrary to the dynamic programming algorithms seen for the Adjacent NK Landscapes and TD Mk Landscapes.

6.1 EA

Evolutionary Algorithms are inspired by evolutionary biology and apply some of its principles, such as 'survival of the fittest' / selection, recombination, and mutation. It applies these on a population of solutions to optimize problems without knowing the underlying problem structure. Important for the quality of the results are the choices for these aspects; when in accordance with the underlying problem structure, the performance will be better. Often Problem-Specific Knowledge (PSK) is used to improve the quality of the solutions or the run time required to find the global optimum. For an EA, convergence speed and diversity are two key aspects, which are both essential to find a solution within reasonable time and of high enough quality. If, for example, selection pressure is too high, convergence will be too great and we speak of premature convergence; the algorithm has not had enough time to diverge sufficiently in order to find the global optimum. Another important aspect of EAs are building blocks, low-order high-performance subsets of the variables when considered and recombined together. Good recombination operators will recombine these building blocks without disturbing/destroying them.

Every EA has the following aspects: initialization, selection, recombination, and mutation. An EA often initializes the population by drawing random samples from the solution space, however, heuristics and problem-specific knowledge can be used to improve initialization of the population. Then, a selection of the population is made to apply recombination and generate offspring. During recombination, mutation can be used to increase diversity. Finally, a selection from the parent and offspring pools is made to arrive at a new generation of the population, which will be used in the next iteration.

In the following sections, we will discuss evolutionary algorithms that try to explicitly learn the structure of the problem and use this knowledge to generate new solutions. We will consider LT-GOMEA in more detail at the end of this chapter.

6.2 FOS

There are various choices for the considered structure of a problem, which are all instances of the Family of Subsets (FOS) model[18]. The FOS model describes, as the name suggests, the structure of a problem as a family of subsets. Let L be the set of all variable indices, $L = \{0, 1, ..., N-1\}$, with N the problem size. A FOS $\mathcal{F}$ is a set of subsets of L, or in other words, $\mathcal{F}$ is a subset of the powerset of L, $\mathcal{F} \subseteq \mathcal{P}(L)$ and can be written as $\mathcal{F} = \{\mathbf{F}^0, \mathbf{F}^1, ..., \mathbf{F}^{|\mathcal{F}|-1}\}$, where $\mathbf{F}^i \subseteq \{0, 1, ..., N-1\}$. Subset $\mathbf{F}^i$ is also called a linkage set, a term that better reflects the dependent nature of the variables in $\mathbf{F}^i$.

Various FOS model instances exist, of which we'll shortly consider univariate, marginal and linkage tree. The univariate FOS model is the set of subsets where every subset exists of just 1 variable index: $\mathcal{F} = \{\mathbf{F}^0, \mathbf{F}^1, ..., \mathbf{F}^{N-1}\}$, with $\mathbf{F}^i = \{i\}$, $\forall i \in \{0, 1, ..., N-1\}$. This is a very limited model that doesn't consider any dependencies between variables. The marginal model does consider dependencies between variables, but every variable must be contained in just 1 subset: $\mathcal{F} = \{\mathbf{F}^0, \mathbf{F}^1, ..., \mathbf{F}^{|\mathcal{F}|-1}\}$, with $\mathbf{F}^i \cap \mathbf{F}^j = \emptyset$, for $\forall i, j \in \{0, 1, ..., |\mathcal{F}|-1\}$ and $i \neq j$. Whereas the marginal model considers variables thus as dependent or independent, the linkage tree model considers the variables to be both dependent and independent, but at different levels in a hierarchy: $\mathcal{F} = \{\mathbf{F}^0, \mathbf{F}^1, ..., \mathbf{F}^{|\mathcal{F}|-1}\}$, with $\mathbf{F}^i \subseteq \{0, 1, ..., N-1\}$, $\forall i \in \{0, 1, ..., |\mathcal{F}|-1\}$. This allows for the linkage tree to capture both low-order and higher-order dependencies[2]. An example linkage tree instance of $L = \{0, 1, 2, 3, 4\}$ is $\mathcal{F} = \{\{0\}, \{1\}, \{2\}, \{3\}, \{4\}, \{5\}, \{0, 1\}, \{2, 3\}, \{0, 1, 4\}, \{0, 1, 2, 3, 4\}\}$.

Importantly, these FOS models can be used by linkage-learning EAs to learn the structure of the problem and then to generate new solutions using this knowledge of the structure. This way, highly correlated problem variables will be considered together (as a linkage set), thereby preventing disruption during recombination and mixing building blocks effectively.

6.3 EDAs

One such linkage-learning EA is the Estimation-of-Distribution Algorithm (EDA) by Pelikan et al.[16], which learns the problem structure by building a probabilistic model of the current population. It uses an estimation of the distribution of the variable values in the current population to do so, and draws samples from the constructed distribution to create the offspring for the next generation. It uses an appropriate FOS model while estimating the distribution to learn the problem structure.

6.4 GOMEA

Different from EDAs, the Gene-pool Optimal Mixing Evolutionary Algorithm (GOMEA) uses the constructed FOS model to recombine parts of the parents to generate offspring. It applies the Gene-pool Optimal Mixing (GOM) operator to every parent $\mathbf{p}$, first creating the first offspring solution by copying the parent ($\mathbf{o} \leftarrow \mathbf{p}$), then iterating over all linkage sets $\mathbf{F}^i$ in the FOS model in random order and for each linkage set $\mathbf{F}^i$ doing the following: select a random donor solution $\mathbf{d}$ and select the linkage set ($\mathbf{F}^i$) bits from that donor. Write a copy of the current offspring $\mathbf{o}$ to $\mathbf{o}'$ and replace the linkage set bits in $\mathbf{o}'$ by the donor $\mathbf{d}$'s linkage set bits. If $\mathbf{o}'$'s fitness is equal to or higher than $\mathbf{o}$'s fitness, $\mathbf{o} \leftarrow \mathbf{o}'$, otherwise simply discard $\mathbf{o}'$. Continue to the next linkage set with this new $\mathbf{o}$.

Although applying the GOM operator over all linkage sets requires more fitness evaluations per generation compared to GA and EDA, it explicitly exchanges building blocks (the linkage sets $\mathbf{F}^i$ in the FOS model) to increase the fitness. For problems efficiently solvable with correctly detected linkage, this leads to requiring much smaller population sizing and far fewer generations, resulting in a more efficient overall performance[19][15].

A very successful GOMEA variant with the Linkage Tree FOS model, Linkage Tree Gene-pool Optimal Mixing (LT-GOMEA), is discussed in the following section.

6.5 LT-GOMEA

The Linkage Tree Genetic Algorithm was introduced by Thierens[18] and has since been successful, showing state-of-the-art performance for discrete, Cartesian-space optimization problems[3]. The Linkage Tree FOS model offers a powerful linkage model at a low computation cost[2], and mixes well with the GOM operator: the operator can exchange both low-order and higher-order building blocks due to the hierarchy in the LT model. There have been some revisions of the Linkage Tree Gene-pool Optimal Mixing Evolutionary Algorithm (LT-GOMEA), in this paper we will use the latest[2], but with some rollbacks so that it can be used on Cartesian-space problems instead of the permutation problems the referred version is made for. The latest version of LT-GOMEA uses Forced Improvements (FI)[1] and the Linkage Tree (LT) construction uses Unweighted Pair Group Method with Arithmetic Mean (UPGMA) and the reciprocal nearest-neighbour chain technique[8].

Starting with the latter, we shortly illustrate the construction of the Linkage Tree, and then continue with an introduction of the Forced Improvements phase. In short, the LT is constructed bottom-up by merging the two most dependent linkage sets until a merged linkage set has been added with all variable indices ($= L$). First, all singleton linkage sets $\mathbf{F}^i = \{i\}$ are added to $\mathcal{F}$. Then we iteratively merge the two most dependent linkage sets, $\mathbf{F}^i = \mathbf{F}^j \cup \mathbf{F}^k$, and add the newly constructed linkage set $\mathbf{F}^i$ to $\mathcal{F}$, excluding the merged linkage sets ($\mathbf{F}^j$ and $\mathbf{F}^k$) from further merging.

To determine which linkage sets are the most dependent, mutual information (I) is used to calculate the degree of dependency. The Unweighted Pair Group Method with Arithmetic Mean (UPGMA) is used to extend the notion of pairwise dependency to linkage sets of length > 1 [3]:

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

with

$$I(X, Y) = H(X) + H(Y) - H(X, Y)$$

where $H(X)$ and $H(X, Y)$ are the marginal and joint entropies, respectively. Using the reciprocal nearest-neighbour chain technique by Gronau and Moran[8], the computational complexity of building the linkage tree is $O(N^2 p)$, where $p = |P|$ is the population size, with P being the population.

As explained in the GOMEA subsection, the GOM operator iterates over all linkage sets in random order, chooses a random donor for each linkage set, and continues the iteration with the modifications when the generated offspring has an equal or better fitness. This can be termed phase 1 of the GOM. Phase 2 is introduced to move off fitness plateaus and make sure the fitness is increased: when a parent wasn't changed in phase 1, or the best solution hasn't improved for a number of generations (the no-improvement-stretch), Forced Improvement[1] (phase 2) is entered. The no-improvement-stretch (NIS) is required to move off fitness plateaus, which can occur because phase 1 allows equal fitness changes. If the NIS is bigger than $1 + \lfloor \log_{10}(N) \rfloor$[3], phase 2 is entered.

Forced Improvement (phase 2) again iterates over all linkage sets, but now chooses the best solution in the population as the donor and only allows strict improvements in the fitness. As soon as an improvement is found, Forced Improvement is exited. If no offspring with higher fitness was generated, the parent's offspring is set to a copy of the best solution.

Importantly, the LT linkage set consisting of all variable indices (L) is ignored during GOM, as it would simply replace the whole parent with the donor, introducing unnecessary additional selection pressure[3].

6.6 Population Sizing-Free Scheme

The population size is crucial for the performance of an EA: if too small, premature convergence will occur, if too big, the algorithm will take overly long. Furthermore, although in theory it's interesting to determine the optimal population size, in practice it's convenient to run a black-box optimizer on a problem without needing to determine the appropriate population size. For these reasons, Bosman et al.[2] introduced the population sizing-free scheme by Harik and Lobo[10] in their latest revision of LT-GOMEA. In essence, it interleaves generations of the EA for different population sizes, allowing smaller instances more generations than bigger instances. Specifically, for every d generations of instance P_i, instance P_{i+1} is run for 1 generation, with $p_i = p_{base} \cdot 2^i$ for P_i, $i \geq 0$, $d \geq 1$, and where p_{base} is the population size of the starting population. When the instance of a bigger population size has a higher average fitness than a smaller population size, the smaller population instance and all instances with an even smaller population size are halted, as the bigger population size instance is more efficiently spending the evaluation budget. In their work, Bosman et al. use $d = 4$ and $p_{base} = 1$, so for every 4 generations of P_i, 1 generation of P_{i+1} is run. This value of d ensures that the smaller populations converge faster, as they are allowed twice as many evaluations.

For LT-GOMEA, Bosman et al. use the best solution accros all different population size instances as the donor in phase 2 (FI). The no-improvement-stretch to determine whether to enter phase 2, however, is still calculated for the best solution in the current instance. As in previous versions of LT-GOMEA, an instance is also stopped when there is no more diversity in the population (all equal solutions). They note an increase in the number of evaluations of a factor

1 to 4 because of the use of the Harik-Lobo scheme, for GOMEA to solve various benchmark problems.

Chapter 7

Experiment: Increasing Overlap

To get a first intuition for the performance and effectiveness of LT-GOMEA on deceptive trap problems, we conducted a simple experiment: we generated deceptive trap problems with increasing problem size N and overlap o, and ran the Linkage Tree Gene-pool Optimal Mixing Evolutionary Algorithm (LT-GOMEA) on these generated problems to quantify the effect of this increase in o for the difficulty of the problem. LT-GOMEA has shown state-of-the-art performance for discrete, Cartesian-space optimization problems[3], and should therefore show just how difficult and non-trivial TD Mk Landscapes can be. Because we know the global optimum (or optima) of the generated problems, we can evaluate the overall performance and effectiveness of LT-GOMEA.

In short, we want to answer the following question in this experiment:

How does an increasing overlap affect the performance and effectiveness of LT-GOMEA for TD Mk Landscapes problems with the deceptive trap codomain?

We expect to see an increase in the required number of evaluations (first hitting time) and a decrease in effectiveness, as the problem size N increases. With a bigger problem size, LT-GOMEA needs a bigger population size to have sufficient diversity in the initial population to have all the required bits of the global optimum appear at least once in the population and to be able to learn the problem structure.

Additionally, we expect to see an increase in difficulty when the number of overlapping bits o is increased. When we speak of a strict increase of difficulty, the number of required evaluations increases and the effectiveness decreases. Increasing overlap should increase the difficulty, as the Linkage Tree can only represent the problem structure exactly when there is no overlap and every increase in overlap decreases the accuracy of the Linkage Tree. If the Linkage Tree can not represent the problem structure well, the mixing will be less effective. These effects should be especially visible with the deceptive trap codomain, as it requires knowledge of the problem structure to overcome the deceptiveness of the problem and thus not being able to represent the structure well will result in being less able to overcome the deceptiveness of the problems.

7.1 Experimental setup

7.1.1 Benchmark problems

Configuration input: $M \in \{m \,|\, N \leq 150\}$, $k = 5$, $o \in \{0, 1, ..., 4\}$, $b = 2$. Where problem size $N = (m-1) \cdot (k-o) + k$.

The codomain used for the experiment is the *deceptive trap function*, where we generate for each subfunction a random bit string of length k to be the local optimum and its inverse to be the deceptive attractor. The local optimum has a score of 1.0, the deceptive attractor has a score of 0.9 and any other bit string has score $0.9 - d \cdot \frac{0.9}{k}$, where d is the hamming distance to the local deceptive attractor.

To generate the TD Mk Landscape problems, we use the implementation of the CliqueTreeMk algorithms by van Driessel et al.[6].

7.1.2 Evaluation

Per configuration instance we generated 25 problems, and for each of these problems, we ran LT-GOMEA 3 times. For the runs where LT-GOMEA manages to find the global optimum, we record the first hitting time. The first hitting time is the number of function evaluations until the global optimum or one of the global optima was found by the algorithm. To record the first hitting times, we need to ignore any unsuccessful LT-GOMEA runs, as these did not find the global optimum. So, for the 3 runs of LT-GOMEA, we filter out any runs that did not find the global optimum and take the median first hitting time for the remaining successful runs. Then we take the median value from the median first hitting times for the 25 generated problems, where again any runs that did not find the global optimum were filtered out. This median value is recorded together with the problem size of the configuration. Besides this first hitting time, we record the effectiveness of LT-GOMEA for every configuration. We measure the effectiveness by counting the number of problems out of 25 (for the current configuration) for which at least 1 LT-GOMEA run found the global optimum, or one of the global optima in case the fitness function has multiple global optima (note that LT-GOMEA is not designed to be a multi-modal EA, so one should not expect it to return all global optima simultaneously). Also note that when we filter out unsuccessful runs in the first hitting time calculation, we still record these unsuccessful runs in the effectiveness for that configuration instance.

To check our results for statistical significance, we test the difference in evaluations between different configurations by comparing the difference in mean values for shared problem sizes. For example, if we compare overlap configurations $o \in \{3, 4\}$, the different configurations have different problem sizes when we increase the number of subfunctions M (recall $N = (M - 1) \cdot (k - o) + k$). To do a fair comparison, we need to compare at equal problem sizes. For the equal problem sizes, we compare the mean first hitting times and report the number of times the difference was significant. When the difference is statistically significant for >= 70% of the compared problem sizes, we regard the results for the two configurations as statistically significant. An example result is that the difference in first hitting time for 4/12 (4 out of 12) problem sizes was statistically significant, another is that **11/13** were significantly different. Importantly, we do not compare problem sizes for which one of the configurations has a number of subfunctions $M \leq 5$, as the differences between the configurations only really show for bigger subfunction numbers. This is due to the first subfunction being completely identical in everything, apart from the difference in codomain value order. Furthermore, we do not compare problem sizes for which one of the configurations has an effectiveness of $< 50\%$, as we do not consider these samples as being reliable. Finally, for the actual statistical significance test, we assume normal distribution for the first hitting time results and therefore run independent student's t-tests. We use $\alpha = 0.05$ and do not assume (in)equal variance, but calculate whether the variance is equal using the Levene test for equal variances.

7.1.3 LT-GOMEA configuration

We use LT-GOMEA with the population sizing-free scheme as introduced by Bosman et al.[2], but we use its discrete Cartesian version. LT-GOMEA instance i with population size P_i is run 4 times for every run of instance $i + 1$, with $P_0 = 1$ and $P_{i+1} = 2 \cdot P_i$. The maximum number of running LT-GOMEA instances is 25. We set the No Improvement Stretch (NIS) to $1 + log_{10}(P_i)$, where P_i is the population size of LT-GOMEA instance i. Forced Improvement (FI)[1] is run if the best fitness in a population did not improve for more generations than this NIS. We use premature stopping to stop any LT-GOMEA instance when a LT-GOMEA instance with a bigger population size has a higher average fitness. A LT-GOMEA instance is also stopped when the fitness variance in the population of a LT-GOMEA instance is equal to or smaller than 0.00001. When the global optimum score is found, we stop execution (of all LT-GOMEA instances) and record the current number of evaluations as the first hitting time. Finally, we use a computation budget of 300,000 evaluations, with every partial evaluation counting as an evaluation as well. When the computation budget is spent, we stop execution.

o	1	2	3	4
0	**5/5**	**5/5**	**11/11**	5/24
1	X	0/8	0/14	0/29
2	X	X	1/16	0/32
3	X	X	X	0/66

(a) $o \in \{0, 1, 2, 3\}$ vs $o \in \{1, 2, 3, 4\}$, H_1: increasing overlap o **increases** first hitting time.

o	3	4
2	7/16	**29/32**
3	X	38/66

(b) $o \in \{2, 3\}$ vs $o \in \{3, 4\}$, H_1: increasing overlap o **decreases** first hitting time.

Table 7.1: Statistical results for the differences in mean values between first hitting times for different overlap values, for LT-GOMEA and $b = 2$. First hitting time is the number of required evaluations to find the global optimum. Printed is the number of times the difference was significant out of the total number of times a difference was compared. For example, in Figure 7.1a, 5/5 comparisons were statistically significant for $o = 0$ vs $o = 1$, with the alternative hypothesis that $o = 1$ has a higher first hitting time than $o = 0$.

7.2 Results

The results are shown in Figure 7.1 and Tables 7.1a & 7.1b. The tables show the statistical significance for different alternative hypotheses, with per comparison the number of significant comparisons and the number of comparisons. The figure shows in the upper graph the first hitting time for increasing values of the problem size N, with a line per overlap o setting, and the lower graph shows the effectiveness. When the effectiveness of an overlap configuration decreases below 50%, it is not considered reliable anymore and therefore its performance is not plotted anymore in the upper graph. To emphasize this decision, we have highlighted the 50% effectiveness mark in the lower graph with a horizontal line.

We hypothesized that the problems would become more difficult to solve for LT-GOMEA with increasing overlap o, however, the results paint a different picture. As expected, problems with overlap 1 and 2 do require more evaluations and have a lower effectiveness than problems without any overlap. However, we already see a change here with respect to the increase of overlap from 0 to 1: although the effectiveness does decrease when increasing the overlap from 1 to 2, the required number of evaluations does not increase significantly. If we increase the overlap further, we see this change continuing: problems with overlap setting 3 have a (insignificantly) lower number of required evaluations and a higher effectiveness than overlap settings 1 and 2, so one could regard problems with overlap 3 as easier. Likewise, problems with an overlap of 4 variables require (insignificantly) fewer evaluations and have a higher effectiveness than problems with overlap setting 3. However, the difference between overlap values 2 and 4 in required number of evaluations is significant, with a clear difference in the effectiveness. Finally, and perhaps most surprising of all, the results show that problems with overlap 4 are solved using a similar number of evaluations compared to problems with overlap 0. Importantly, however, problems with overlap 0 are always solved, whereas problems with overlap 4 are not.

Although we have now seen what the effect of the overlap is on the performance and effectiveness of LT-GOMEA, we need to dive deeper into the mechanisms behind the behavior we observe in the results. In the next sections we try to answer the question: *Why does the difficulty first increase with increasing overlap, before decreasing for $o > 2$?*.

Due to an issue with the number of global optima while calculating the global optima for some problems, we suspect a role for the number of global optima on the difficulty of a problem. For this reason, the next chapter investigates this further.

7.3 Conclusions

- The difficulty of TD Mk Landscape problems with the deceptive trap codomain does not strictly increase with increasing overlap. Instead, the difficulty increases prior to decreasing.

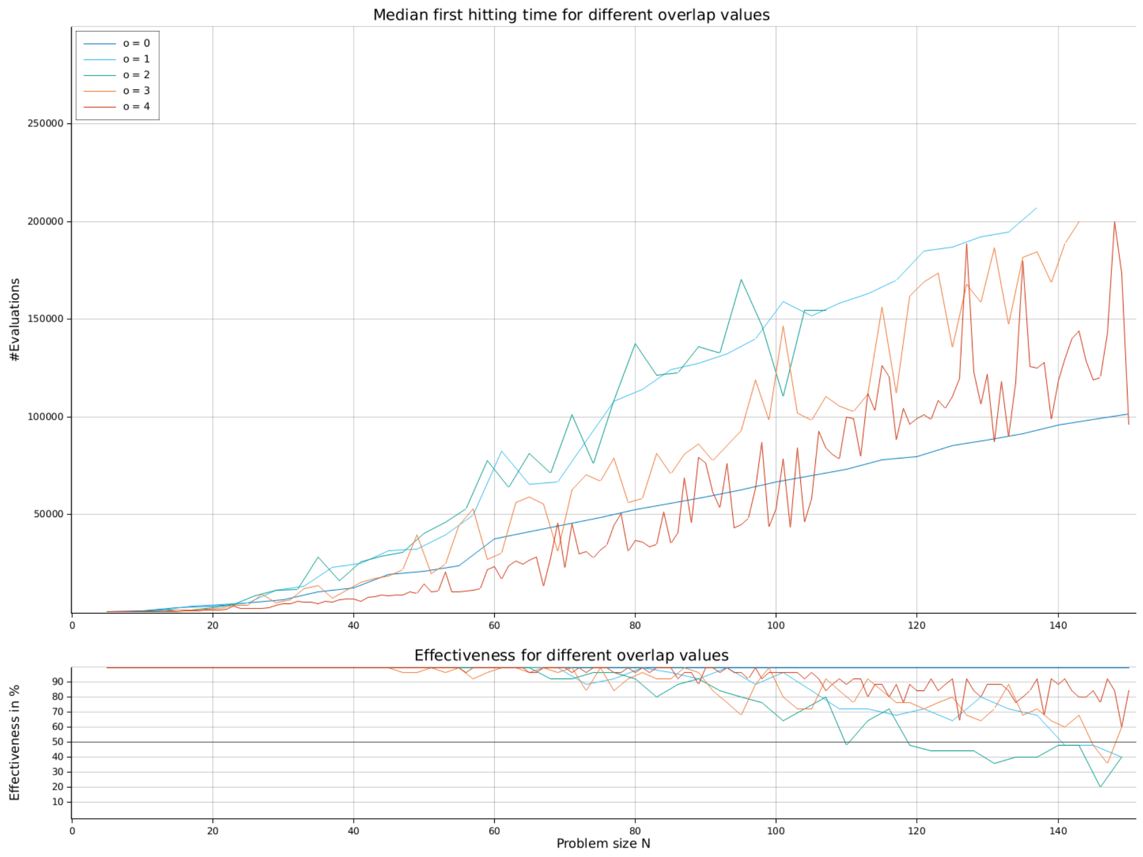


Figure 7.1: Performance and effectiveness of LT-GOMEA for different overlap values

Chapter 8

Experiment: Branching / Global Optima

During preliminary experiments for the previous experiment, we discovered that the number of global optima exploded after $N = 120$ for $o = 4$, $b = 1$, and the deceptive trap codomain. Do note that the same issue may arise for $o = 3$ with $b = 1$ and much higher problem sizes, or a different codomain. It is not immediately clear why there are so many global optima for $b = 1$, nor is it immediately clear why the number of global optima changes significantly between branching factors.

For the mentioned setting, if we enlarge the problem size beyond 120, we encounter problems with (for example) an incredible number of ≈ 600.000 global optima (when $N \approx 130$). To see if this number of global optima is the reason deceptive problems get easier when the overlap is increased beyond 2 (as we saw in the previous experiment), we benchmark LT-GOMEA with problems that differ only in their branching factor and thus in their number of global optima. We expect to see that the difficulty increases significantly between $b = 1$ and $b = 2$, due to the significantly smaller number of global optima. This could then potentially explain why a larger overlap makes problems easier from $o = 3$ onwards, as we see a higher number of global optima with increasing overlap.

In short, in this experiment we try to answer the question:

How does the number of global optima affect the performance and effectiveness of LT-GOMEA for TD Mk Landscapes problems with the deceptive trap codomain?

8.1 Experimental setup

We use many of the same configuration settings as in the previous experiment, but generate problems with a different configuration and expand the graph to show the median number of global optima for each configuration instance. For the problems, we use configuration input: $M \in \{m \,|\, N \leq 120\}$, $k = 5$, $o = 4$, $b \in \{1, 2\}$. Where problem size $N = (m - 1) \cdot (k - o) + k$. Again we use the deceptive trap function as codomain.

8.2 Results

The results are plotted in Figure 8.1 and the statistical (in)significance is shown in Tables 8.1a and 8.1b. We can see that the number of global optima does differ greatly; at some points an up to 10 times difference. The effectiveness of $b = 1$ is higher, as expected, but the difference is less pronounced than what we expected. Surprisingly, the number of required evaluations is higher for $b = 1$ than for $b = 2$, but not significantly so.

This can be explained by a different factor affecting the results here, unbeknownst to us at the moment. So the effect of more global optima might still be significant, but it could be overshadowed by another factor with an even greater effect. To be able to say more about the different factors at play and the magnitude of their effect, we need to isolate the factors at play here. We hypothesize that the two factors are: 1) The number of global optima, and 2) The branching factor. In the next chapters, we will look at each in more detail. It should be noted that these two factors

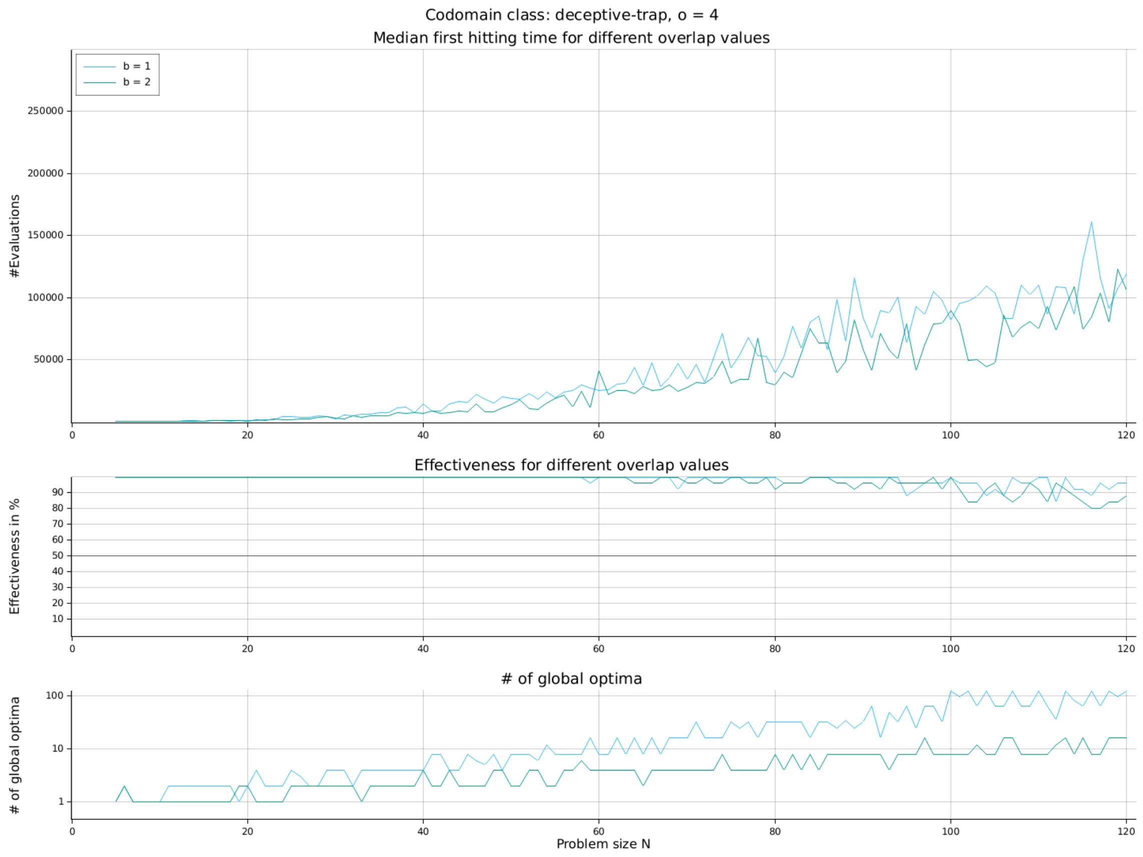


Figure 8.1: Performance and effectiveness of LT-GOMEA for different branching factor values: $b \in \{1, 2\}$

b	2
1	0/110

(a) H_1: increasing branching **increases** first hitting time.

b	2
1	28/110

(b) H_1: increasing branching **decreases** first hitting time.

Table 8.1: Statistical results for the differences in mean values between different branching values ($b = 1$ vs $b = 2$), for LT-GOMEA and $o = 4$. See section 7.1.2 and the common caption of Tables 7.1a and 7.1b for more details.

aren't the only two factors, but we will look into this further in a later chapter. It is still unclear what underlying mechanisms of an increasing branching factor could affect the performance of LT-GOMEA, but it will be related to the increased overlap for some cliques/variables. Importantly, we have not answered the question we set out to answer with this experiment and will try to do so with our next experiment.

8.3 Conclusions

- The branching factor does not just change the number of global optima, but introduces a different effect as well, which is probably related to the increased overlap for some cliques/variables. However, what this effect is exactly, is still unclear.

Chapter 9

Experiment: Global Optima

In the previous experiment, we encountered mixed results. To identify the effect of the number of global optima on the performance and effectiveness of LT-GOMEA, we isolate the number of global optima as the only difference between two problem sets: a set of problems with a high number of global optima and a set of problems with a low number of global optima. We expect to see that the difficulty is decreased significantly when there is a greater number of global optima, potentially explaining half of the effect we saw in the previous experiment. The lower difficulty could be explained by the presence of multiple global optima, while LT-GOMEA only needs to find one global optimum.

In this experiment we again try to answer the question:

How does the number of global optima affect the performance and effectiveness of LT-GOMEA for TD Mk Landscapes problems with the deceptive trap codomain?

9.1 Experimental setup

9.1.1 Benchmark problems

To stay as close to our implementation of the problem generation as possible, we generate $8 \cdot 25$ problems and choose either the $\frac{1}{8}$ problems with highest or lowest number of global optima, depending on what problem set we are generating. We do this for both problem sets, so in total we generate $2 \cdot 8 \cdot 25$ problems, and use $2 \cdot 25$ problems for the two sets of problems.

Again, we use the deceptive trap function as described in the previous experiments and run LT-GOMEA on the generated problems. We set $N \leq 120$, $k = 5$, $o = 4$, $b = 1$.

9.2 Results

9.2.1 LT-GOMEA

The results are shown in Table 9.1 and Figure 9.1. As can be observed from the bottom chart in the figure, the dark blue line represents the set of problems with a high number of global optima and the light blue line the set of problems with a low number of global optima. The median number of global optima differs between the two sets up to a factor of 100, and there is a visible difference in the required number of evaluations and effectiveness. The table tells us the difference in the required number of evaluations is significant.

The results show that a greater number of global optima does have a positive impact on the performance and effectiveness of LT-GOMEA. To verify this result, we plotted the results for random linkage LT-GOMEA as well, in Appendix Figure A.1. The observed positive effect could be explained simply by the greater set of correct solutions leading to an easier landscape to traverse for LT-GOMEA, however, it might not be as simple as that. The results do not show the effect one would expect from a hundredfold increase in number of global optima, so the high number of global optima does not decrease the difficulty just because of their sheer number. There seems to be a limiting factor here, we suspect it is the fact that solutions can be quite far apart and therefore have a negative effect on the mixing of LT-GOMEA; mixing two solutions that are close to a global optimum could only improve the solutions for the identical parts of the global optima.

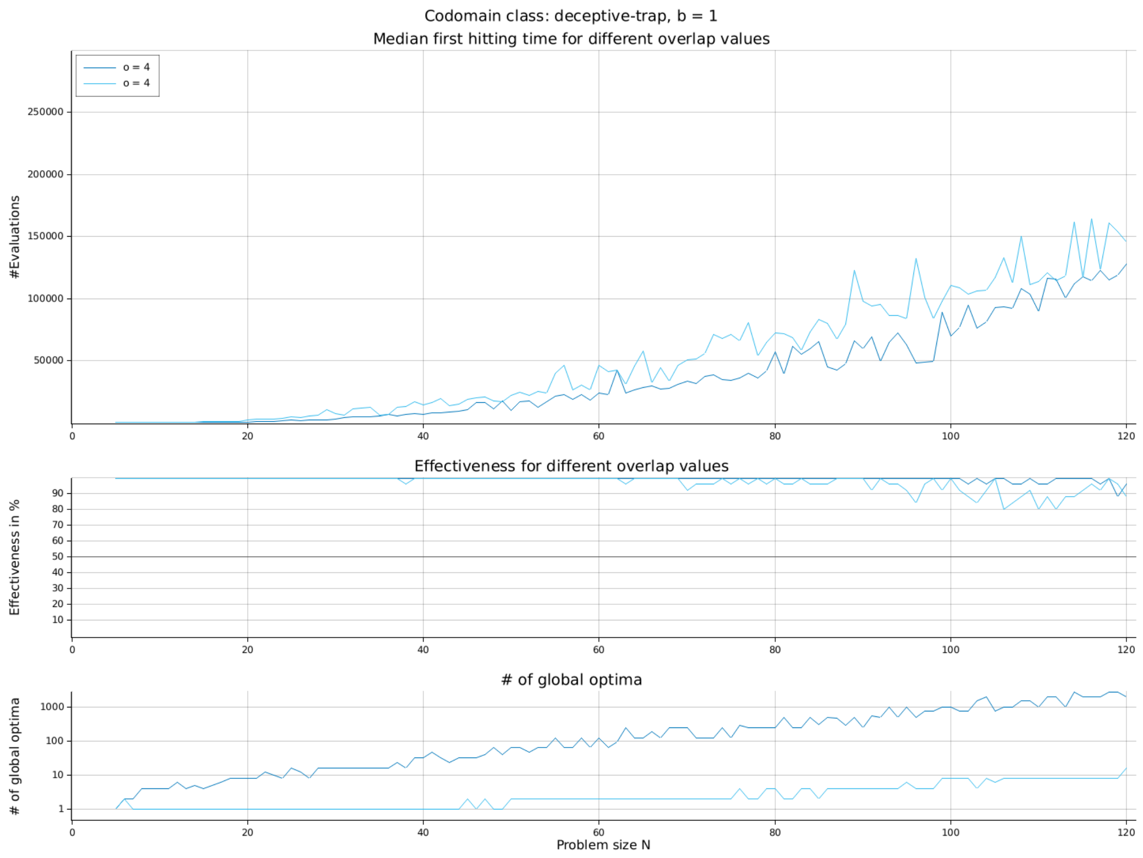


Figure 9.1: Performance and effectiveness of LT-GOMEA for high/low number of global optima

#opt.	high
low	**78/110**

Table 9.1: Statistical results for the differences in mean values between problems with either a low or high number of global optima, for LT-GOMEA, $o = 4$, $b = 1$. H_1: Problems with a *high* number of global optima have a **lower** first hitting time than problems with a *low* number of global optima, for an otherwise equal configuration.

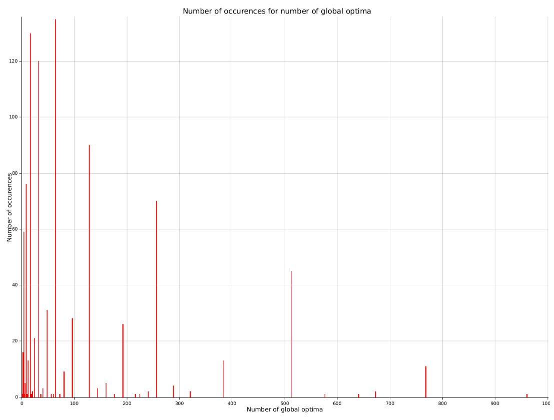
To test this hypothesis, we will now look into the global optima: first we will take a look at the distribution of the number of global optima, and then we will take a look at the hamming distance between global optima.

9.2.2 # of Global Optima Distribution

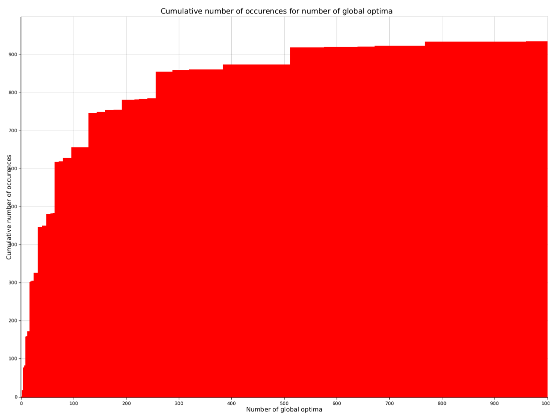
To gain an insight into the occurences of the different number of global optima and especially for problems with a lot of global optima, we draw the distribution of the number of global optima for 1000 generated problems with $N = 100$, $o = 4$ and $b = 1$. The results are plotted in Figure 9.2: as a distribution in Figure 9.2a and a cumulative distribution in Figure 9.2b. Note that there are problem instances with a number of global optima higher than 1000, but we limited the range of the figures for the sake of clarity.

Interestingly, the number of global optima is not distributed normally. Instead, we see spikes at $x = 2^a$ and $x = 2^a + 2^b$, where $a, b \in \{0, 1, ...\}$ and $a \neq b$. For example, we can clearly see that the number of global optima with the most occurences are $x \in \{16, 32, 64\}$, but we also see spikes at $x \in \{192, 386, 768\}$.

This could be caused by just a variables/bits being different for the global optima. $x = 2^a$ could then be explained by every one of the a variables/bits leading to an equal score for both its possible values, for the global optima solutions constructed so far in the global optimum calculation



(a) Distribution



(b) Cumulative distribution

Figure 9.2: Distribution of number of global optima for 1000 generated problems with $N = 100$, $o = 4$, $b = 1$. On the x-axis is the number of global optima, on the y-axis is the number of problems that have this many global optima.

process. $x = 2^a + 2^b$ could be similarly explained. One example leading to this number of global optima would be when again all a variables/bits lead to equal scores for both its possible values, for the global optima solutions constructed so far. However, the second bit now only leads to the same score for both its values for half of the global optima solutions constructed so far. To illustrate this, consider the following example:

Problem with $M = 2$, $k = 2$, $o = 1$, and $b = 1$. Our cliques are: $C_0 = \{x_1, x_2\}$ and $C_1 = \{x_2, x_3\}$. Then we have one separator $S_1 = \{x_2\}$. For clique C_0, the codomain of the subfunction is such that there are two local optima, for $x_1 = 1$ and $x_2 \in \{0, 1\}$. For clique C_1 we consider two cases, case A in which all possible instances of C_1 have the same (highest) score, and case B in which $x_2 = 1, x_3 \in \{0, 1\}$ and $x_2 = 0, x_3 \in \{1\}$ have the highest score. In case A, there are $1 \cdot 2 + 1 \cdot 2 = 4$ global optima, as there are 2 possible instances of x_2 that have the highest score in C_0 and 2 possible instances of x_3 that have the highest score in C_1 for each of these 2 x_2 instances. In case B, there are $1 \cdot 1 + 1 \cdot 2 = 3$ global optima, as there are 2 possible instances of x_2 that have the highest score in C_0, and there are 2 possible instances of x_3 that have the highest score in C_1 for $x_2 = 1$ and 1 instance of x_3 that has the highest score in C_1 for $x_2 = 0$.

To test our hypothesis of spikes being caused by $x = 2^a$, we will look into the hamming distance between the global optima next.

9.2.3 Hamming Distance Between Global Optima

As is described in the previous experiment, we expect a small average hamming distance between the global optima, contrary to what we hypothesized using the results for LT-GOMEA on the high and low number of global optima problem sets. At the time, we expected the hamming distance to be big so that the small difference in performance and effectiveness between $o = 4, b = 1$ with low and high number of global optima could be explained. Here, we take a look at all the problems from our previous experiment with a number of global optima equal to 512 and record the hamming distances between the global optima. We then calculate the average hamming distance D_{h_avg} and the min and max hamming distances. The results are listed in Table 9.2, with the number of occurences of a specific average, minimum and maximum hamming distance aggregated.

Our expectation of a small average hamming distance D_{h_avg} turns out to be correct; the average hamming distance is indeed small for most of the problems that have 512 global optima. In fact, the explanation we suggested in the previous experiment seems to be confirmed by the big number of problems with a maximum hamming distance of 9. If the maximum hamming distance is 9, then all global optima are equal except for these 9 variables/bits that take all possible $2^9 = 512$ values. Therefore, we can conclude that for these cases there are a few variables that have the same score for the possible values for all separator instances in the current partial global optima solutions.

This result changes the way we view the results of the low vs high number of global optima problems for LT-GOMEA. We hypothesized that the small effect of the number of global optima

num	avg	min	max
33	4.51	1	9
3	5.01	1	10
1	5.26	1	11
2	5.51	1	11
2	6.01	1	12
1	6.51	1	13
1	8.77	1	18
1	11.40	1	24
1	13.78	1	29

Table 9.2: The hamming distances between the global optima in the problems with 512 global optima. Problems with the same average, min, and max aggregated using a counter in the first column. In total there are 45 problems considered here.

was due to the fact that the hamming distance between the global optima was high, therefore making the mixing less effective. The result showing a low hamming distance means that the gain from having many global optima is very limited; the global optima are very close in terms of hamming distance, so LT-GOMEA will not find a global optimum much quicker, as it must already be close to one of the global optima to find the others.

Finally, this results also shows that TD Mk Landscapes with the deceptive trap codomain, $o = 4$, $b = 1$, and a lot of global optima are not interesting for benchmarking niching algorithms, as the global optima are too close together to really test their capabilities.

9.2.4 Short Discussion

We hypothesize it is the interference of the deceptive subfunctions (due to the increased overlap o) that decreases the deceptiveness of the problem. This interference and decreased deceptiveness is then also the cause of the high number of global optima. In other words, the high number of global optima is not the cause of the decreased difficulty, but is a consequence of the cause behind this decreased difficulty. Here, the cause of the decreased difficulty is the increasing overlap, which causes interference and a decreased deceptiveness.

Furthermore, we hypothesize that the increased interference between the subfunctions decreases the importance of learning the (exact) linkage, as the variables become increasingly interconnected.

9.3 Conclusions

- In this chapter, we have seen that the number of global optima does have an effect on the first hitting time, but that it is smaller than we hypothesized. We think this is due to the small hamming distance between the global optima; it must already be close to one of the global optima to find the others, therefore there is only a small gain in the number of evaluations and effectiveness.

- Furthermore, we think the high number of global optima is caused by interference of deceptive subfunctions, which also lead to decreased deceptiveness. The interference of deceptive subfunctions is caused by an increase in overlap between the subfunctions. This could be further increased with a larger branching factor.

- We think the increased interference between the subfunctions also decreases the importance of learning the (exact) linkage, as the variables become increasingly interconnected.

- The number of global optima for problems with $N = 100$ are often a power of 2 (2^a) or a sum of two powers of 2 ($2^a + 2^b$).

Chapter 10

Experiment: Increasing Overlap and Branching

In the previous chapter we have taken a closer look at the effect of the number of global optima on the performance and effectiveness of LT-GOMEA. Here we take a closer look at the effect of the branching factor b. Together, these experiments should enable us to understand the mechanisms at play when we increase the branching factor and thus understand exactly what happened in the experiment where we only looked at branching factor values $b \in \{1, 2\}$. In this chapter we try to answer the question:

How does an increasing branching factor affect the performance and effectiveness of LT-GOMEA for TD Mk Landscapes problems with the deceptive trap codomain?

To do so, we increase the branching factor up to a value of 6 and evaluate the performance and effectiveness of LT-GOMEA. As mentioned before, we think we saw two effects colliding between $b = 1$ and $b = 2$: the number of global optima and the increased overlap for some cliques/variables. As the number of global optima already decreased significantly when the branching factor increased from 1 to 2, and we saw little effects for a 100x increase of number of global optima, we expect the number of global optima to have an insignificant effect on the performance and effectiveness of LT-GOMEA with increasing branching value. However, the increased overlap for some cliques/variables will increase further with increasing b, so we expect that the first hitting time and effectiveness will improve.

10.1 Experimental Setup

We use many of the same configuration settings as in the experiment in Chapter 8, where we looked at branching factors $b \in \{1, 2\}$, but increase the range of the branching factor to 6. So, for the problems, we use configuration input: $M \in \{m \,|\, N \leq 120\}$, $k = 5$, $o \in \{0, 1, 2, 3, 4\}$, $b \in \{1, 2, 3, 4, 5, 6\}$. Where problem size $N = (m - 1) \cdot (k - o) + k$. Again we use the deceptive trap function as codomain.

10.2 Results

In Figures 10.1a and 10.1b we show the results for $b = 1$ and $b = 6$, respectively. As the biggest changes occur for $o = 4$, we highlight the results for $o = 4$ and $b \in \{2, 3\}$ in Figure 10.2a and for $o = 4$ and $b \in \{4, 5, 6\}$ in Figure 10.2b. We do not print the results for $b = 1$ in Figure 10.2a, as they make the graph harder to read due to the effectiveness being higher than for $b = 2$, which decreases the clear overview. See Figure 8.1 for the clutter we get when plotting $b \in \{1, 2\}$.

In Figures 10.1a and 10.1b we can see that the results for $o = 0$ don't change, as expected. If the lines for $o \in \{1, 2\}$ change at all, it is barely visible. However, the lines for $o \in \{3, 4\}$ do change visibly, with $o = 4$ most noticeably. They indeed seem to get easier with increasing branching factor b. For $o = 3$, the change between sequential b values is barely noticeable, but for $o = 4$, the effects are clearly visible. For this reason, we have plotted the results for $o = 4$ separately in Figures 10.2a and 10.2b. In these figures we can clearly see that the first hitting time

o	1	2	3	4
	3/24	6/30	19/53	**106/110**

Table 10.1: Statistical results for the differences in mean values between $b = 1$ vs $b = 6$, for LT-GOMEA, $o \in \{1, 2, 3, 4\}$. H_1: First hitting time **decreases** with increasing branching factor b.

o	3	4
2	8/17	**34/34**
3	X	**52/53**

(a) $o \in \{2, 3\}$ vs $o \in \{3, 4\}$, H_1: Increasing overlap **decreases** first hitting time.

o	3	4
0	2/9	**18/18**

(b) $o = 0$ vs $o \in \{3, 4\}$, H_1: **Different** first hitting time.

Table 10.2: Statistical results for the differences in mean values between different overlap values, for LT-GOMEA, $b = 6$. Statistics for Figure 10.1b.

(required number of evaluations) decreases with increasing b. If we look carefully, one can see that the effectiveness increases with increasing b. The decreased number of evaluations is statistically significant for $o = 4$, as can be seen in Table 10.1. This is supported by Table 10.3.

Interestingly, as is visible in 10.1b, the first hitting time of $o = 4$ is well below the first hitting time of $o = 0$, which is supported by the statistical significance in Table 10.2b. However, the effectiveness is still lower than for $o = 0$, so it's not strictly easier. Clearly, the effectiveness of $o = 0$ is 100% everywhere (this is better visible in 10.1a), whereas the effectiveness of $o = 4$ is not. In Figures 10.2a and 10.2b we can see the effectiveness of $o = 4$ for $N \leq 120$ creeps closer and closer to the $y = 100\%$ line.

The statistically significant difference between $o = 0$ and $o = 3$ for $b = 1$ is not present for $b = 6$, which is also visible in Figure 10.1b. The first hitting times for these configurations are closer for $b = 6$.

Furthermore, we can see in Figures 10.1a and 10.1b that the difference between $o = 3$ and $o = 4$ is increased by increasing the branching factor from $b = 1$ to $b = 6$, which is supported by Tables 7.1b and 10.2a: in Table 7.1b, the difference between $o = 3$ and $o = 4$ is not significant for $b = 1$, but in Table 10.2a, for $b = 6$, the difference *is* significant.

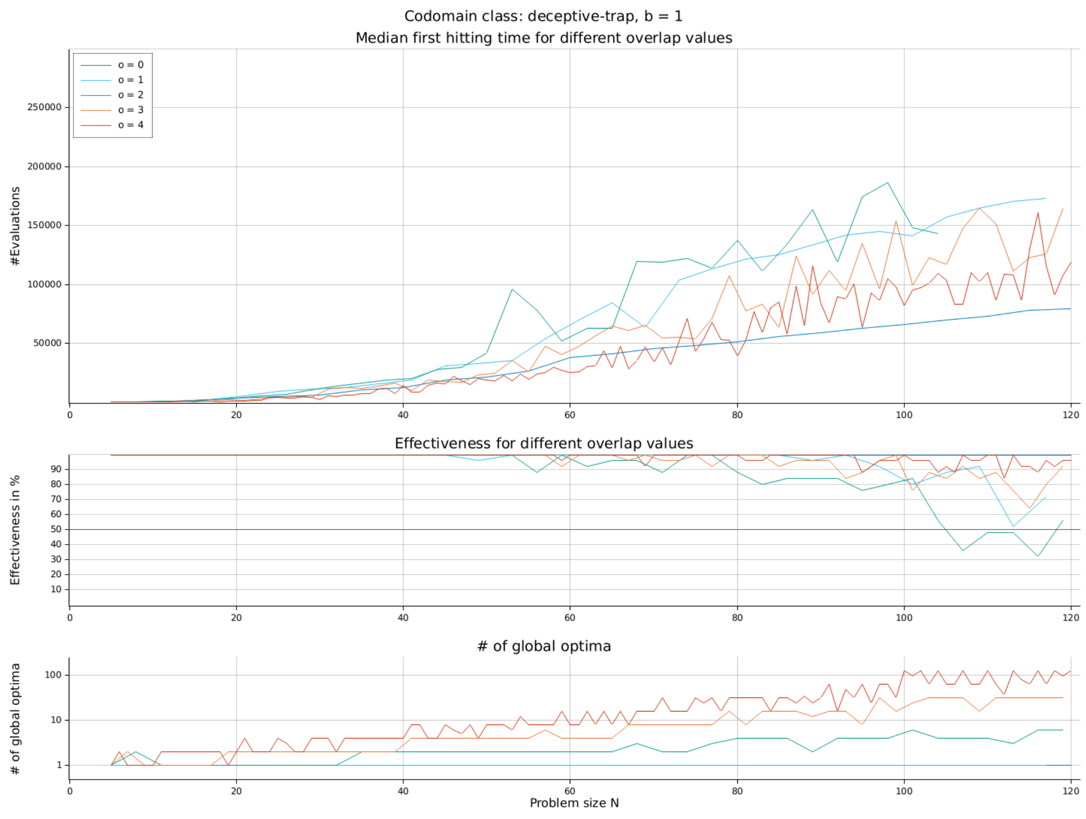
The median number of evaluations required for $o = 3$ with a branching factor $b = 6$ is similar to that of $o = 4$ with a branching factor of $b = 1$. However, the effectiveness of overlap setting $o = 3$ is still lower, so $o = 4$ with $b = 1$ can still be regarded as being less difficult for LT-GOMEA than $o = 3$ with $b = 6$.

Now that we have observed the decrease of difficulty with increasing b (for $b > 1$, as the number of global optima improves the performance and effectiveness of $b = 1$), we have identified the factors at play in the difference between $b = 1$ and $b = 2$: the number of global optima and the other effects of the branching factor. We will come back to this at the end of this chapter, after we have taken a closer look at the mechanisms behind the branching factor's effect.

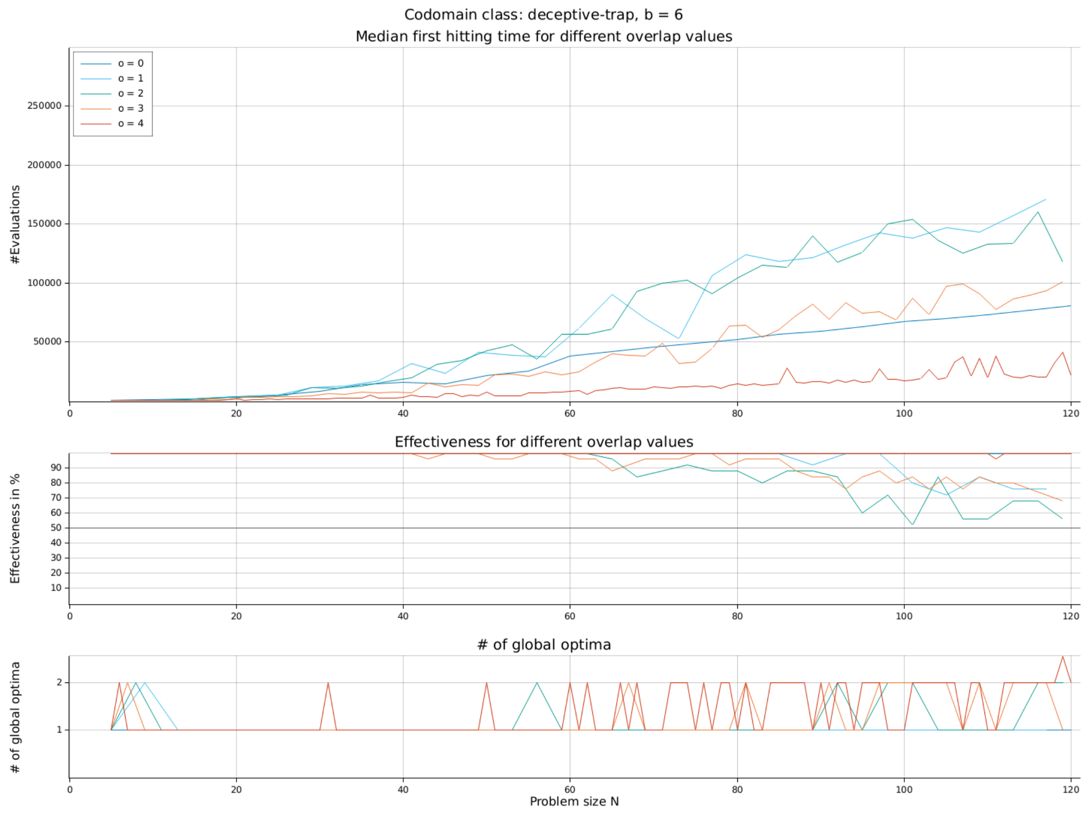
How do we explain these observations? The branching increases the overlap between the cliques further; some cliques/variables will overlap with even more cliques/variables for $b = 6$ than for $b = 1$. We hypothesize, just as in the discussion of the previous chapter, that this increases the interference of the deceptive cliques/subfunctions, leading to decreased deceptiveness and decreased importance of linkage learning. To test this hypothesis, we will now look at the performance and effectiveness of GOMEA with a Univariate Family of Subsets, U-GOMEA, and of LT-GOMEA with random linkage.

10.2.1 Random Linkage LT-GOMEA & U-GOMEA

To test the hypothesis that increasing the branching factor increases the interference of the deceptive subfunctions, leading to decreased deceptiveness and decreased importance of linkage learning, we test LT-GOMEA with random linkage (linkage is not learned, but randomly initialized) and U-GOMEA (GOMEA with the univariate FOS). If these manage to perform well, then this suggests that the problems have lost a significant part of their deceptiveness and the requirement to perform linkage learning, as these will not perform well at all when the problems are fully deceptive due to them being unable to learn/represent the structure and therefore not effectively mixing solutions.

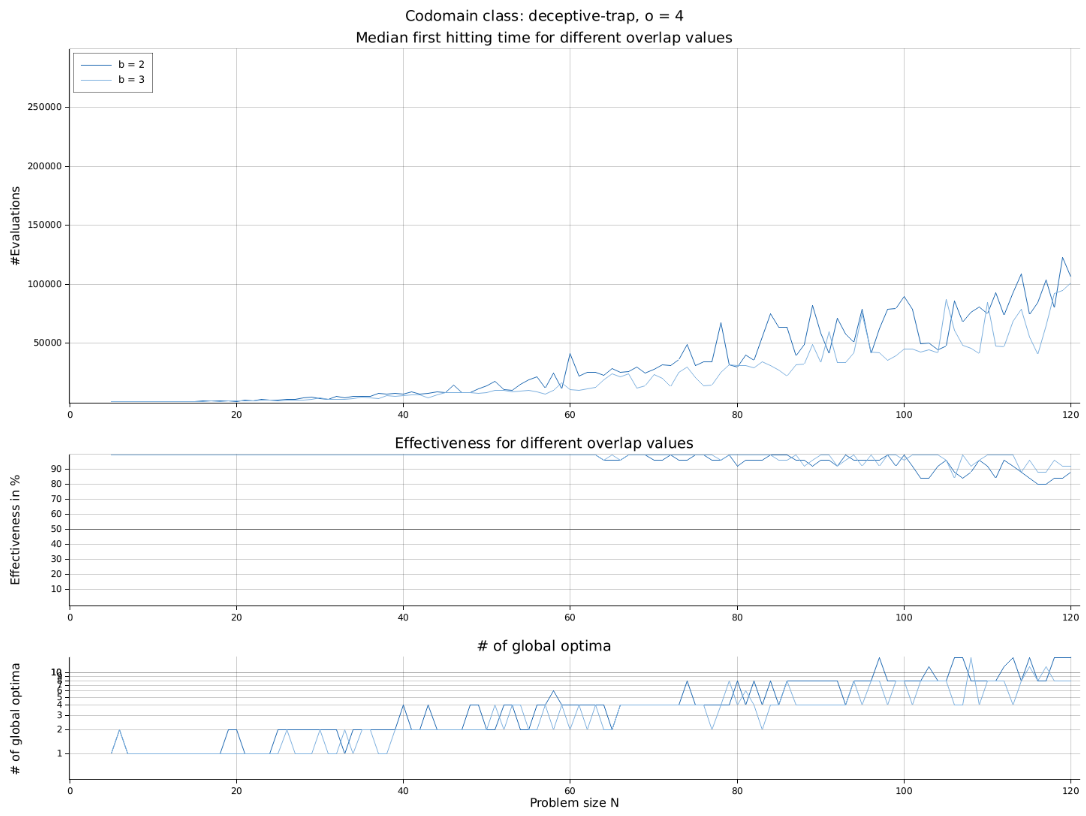


(a) $b = 1$

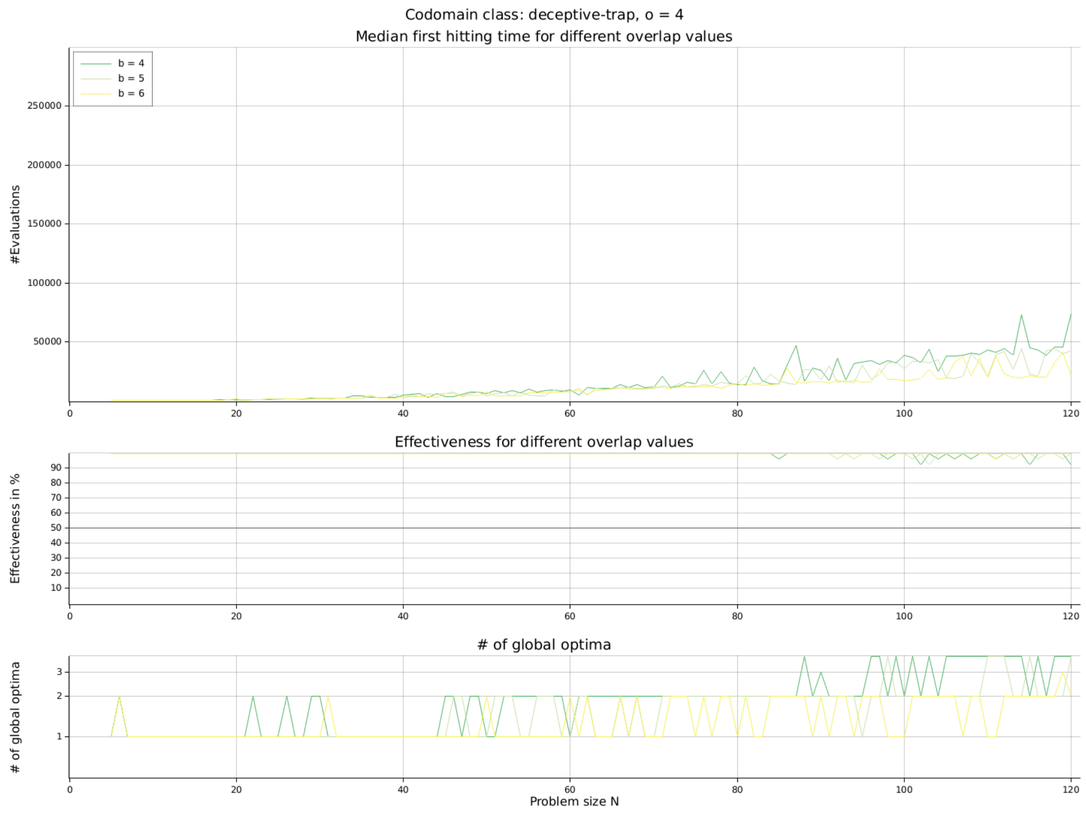


(b) $b = 6$

Figure 10.1: Performance and effectiveness of LT-GOMEA for different overlap values with different branching values: a) $b = 1$ and b) $b = 6$.



(a) $b \in \{2, 3\}$



(b) $b \in \{4, 5, 6\}$

Figure 10.2: Performance and effectiveness of LT-GOMEA for overlap value $o = 4$ with different branching values: a) $b \in \{2, 3\}$ and b) $b \in \{4, 5, 6\}$.

b	2	3	4	5	6
1	28/110	72/110	**99/110**	**103/110**	**106/110**
2	X	32/110	73/110	**91/110**	**99/110**
3	X	X	28/110	51/110	71/110
4	X	X	X	18/110	26/110
5	X	X	X	X	19/110

Table 10.3: Statistical results for the differences in mean values between different branching values, for $b \in \{1, 2, 3, 4, 5\}$ vs $b \in \{2, 3, 4, 5, 6\}$, $o = 4$, LT-GOMEA. H_1: Increasing branching b **decreases** first hitting time. Statistics for Figure 10.2.

o	1	2	3	4
	1/1	0/6	8/19	**51/58**

Table 10.4: Statistical results for the differences in mean values between $b = 1$ vs $b = 6$, for random linkage LT-GOMEA, $o \in \{1, 2, 3, 4\}$. H_1: First hitting time **decreases** with increasing branching factor b. Statistics for Figure 10.3.

In other words, these would be lucky to find the global optimum when there is no overlap.

More specifically, U-GOMEA can only find the global optimum for separated / non-overlapping / fully-deceptive problems when its subfunctions are hamming distance 0 or 1 away from the local optimum and it manages to mix the remaining bits (that are not equal to the global optimum yet) with the correct value. If any of the subfunctions flip / mix one of the bits that are already correctly set, it will not reach the local optimum anymore and neither will it reach the global optimum anymore, due to the fully deceptiveness. For LT-GOMEA with random linkage, the chance of finding the global optimum is higher, as it is not just reliant on single bit-flips, but can mix multiple bits and therefore reach the local optimum from hamming distances bigger than 1 as well. However, due to the random linkage, the mixing will still not be very effective, but more effective than doing single bit-flips.

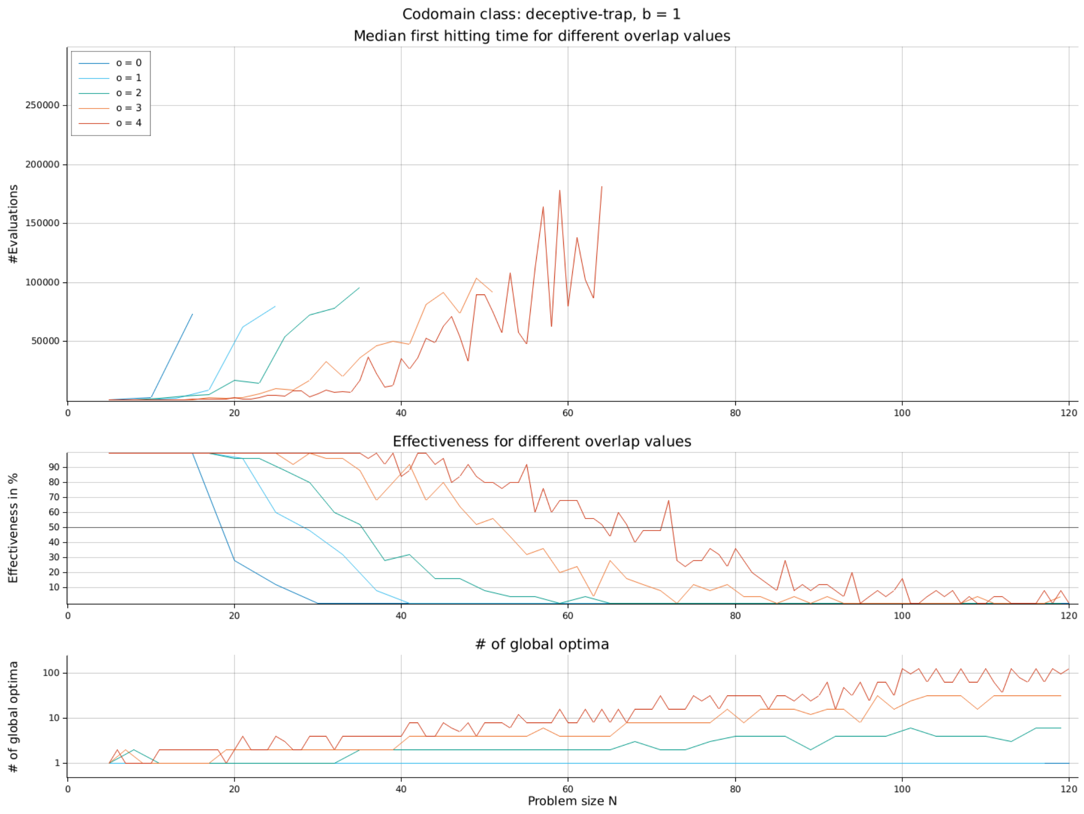
Now, in this experiment there *is* overlap ($o > 0$), and we again expect the random linkage LT-GOMEA to perform better than U-GOMEA, because the subfunctions still contain 5 variables and not all of these variables overlap. Therefore, we expect there to be an advantage in the exchange of building blocks, even when the deceptiveness decreases due to the interference between subfunctions. However, the difference between the two algorithms might get smaller and smaller with increasing overlap if there really is increased interference between the deceptive subfunctions, therefore decreasing the advantage of mixing with building blocks. Furthermore, we expect them both to perform better with increasing overlap, due to the decrease in deceptiveness and decrease in importance of linkage learning as a result of the interference. And finally, we expect the difference in performance and effectiveness between LT-GOMEA on the one hand and random linkage LT-GOMEA and U-GOMEA on the other hand to become smaller with increasing overlap.

In this experiment we also increase the branching factor up to a value of 6, just as in the previous experiment. As mentioned before, we think that this will further increase the interference between the deceptive subfunctions, so we expect the effects mentioned in the last paragraph to be increased. In other words, we expect that with increasing branching factor 1) the difference in performance and effectiveness between LT-GOMEA and U-GOMEA to get even smaller, 2) their performance and effectiveness to improve even further, and 3) the difference in performance and effectiveness between LT-GOMEA on the one hand and random linkage LT-GOMEA and U-GOMEA on the other hand to become even smaller.

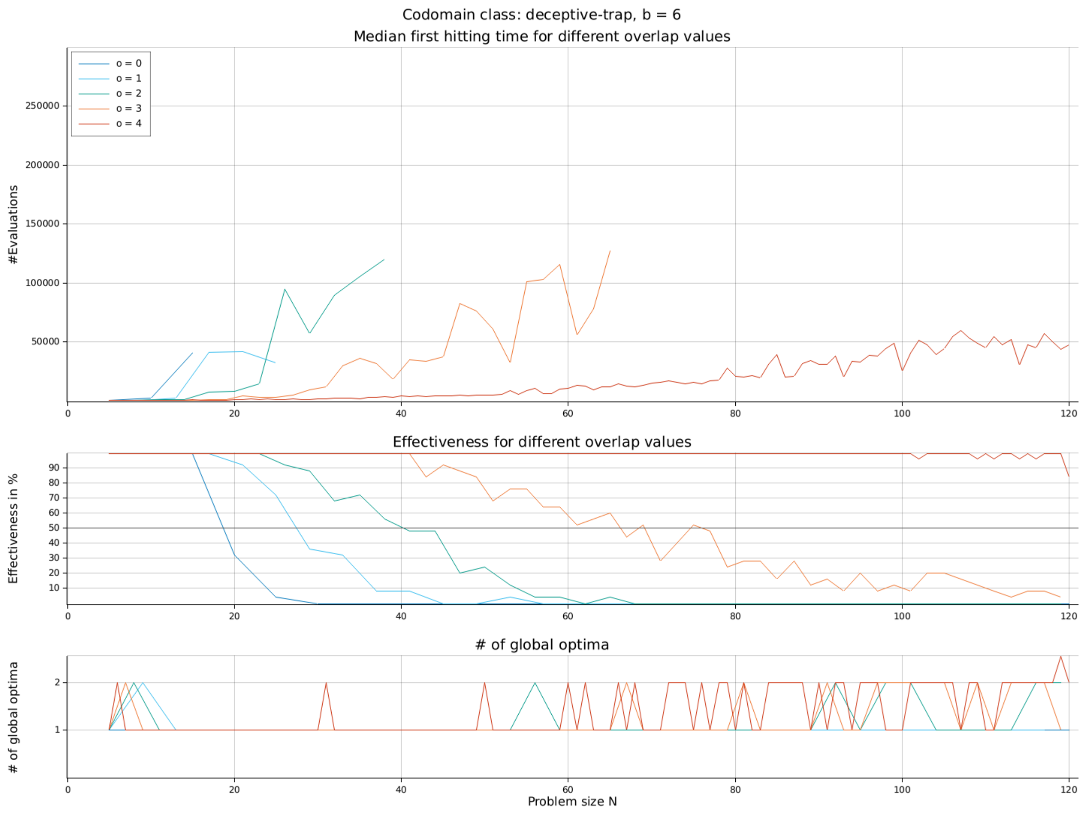
We use the same problems as were generated in the previous experiment, to be able to make a fair comparison.

The results are plotted in Figures 10.3 and 10.4, for random linkage LT-GOMEA and U-GOMEA respectively.

In the Figures, we can see that for all algorithms (LT-GOMEA, random linkage LT-GOMEA, and U-GOMEA) the performance and effectiveness are equal or increase with increasing b, except for the case of LT-GOMEA between $b = 1$ and $b = 2$, as mentioned before when we discussed the possible explanations. However, only for $o = 4$ is the change significant for the first hitting



(a) $b = 1$



(b) $b = 6$

Figure 10.3: Performance and effectiveness of Random Linkage LT-GOMEA for different overlap values with different branching values: a) $b = 1$ and b) $b = 6$.

o	1	2	3	4
0	0/0	0/0	0/0	0/0
1	X	0/0	0/0	**1/1**
2	X	X	**3/3**	**6/6**
3	X	X	X	10/19

(a) $b = 1$, for Figure 10.3a.

o	1	2	3	4
0	0/0	0/0	0/0	0/0
1	X	0/0	0/0	**1/1**
2	X	X	**3/3**	**7/7**
3	X	X	X	**27/28**

(b) $b = 6$, for Figure 10.3b.

Table 10.5: Statistical results for the differences in mean values between different overlap values, for $o \in \{0, 1, 2, 3\}$ vs $o \in \{1, 2, 3, 4\}$, random linkage LT-GOMEA, $b \in \{1, 6\}$. H_1: Increasing overlap **decreases** first hitting time. Statistics for Figure 10.3.

o	1	2	3	4
	0/1	1/5	6/15	**41/48**

Table 10.6: Statistical results for the differences in mean values between $b = 1$ vs $b = 6$, for U-GOMEA, $o \in \{1, 2, 3, 4\}$. H_1: First hitting time **decreases** with increasing branching factor b. Statistics for Figure 10.4.

o	1	2	3	4
0	0/0	0/0	0/0	0/0
1	X	0/0	0/0	**1/1**
2	X	X	**3/3**	**5/5**
3	X	X	X	5/15

(a) $b = 1$, for Figure 10.4a.

o	1	2	3	4
0	0/0	0/0	0/0	0/0
1	X	0/0	0/0	**1/1**
2	X	X	1/4	**6/6**
3	X	X	X	**16/17**

(b) $b = 6$, for Figure 10.4b.

Table 10.7: Statistical results for the differences in mean values between different overlap values, for $o \in \{0, 1, 2, 3\}$ vs $o \in \{1, 2, 3, 4\}$, U-GOMEA, $b \in \{1, 6\}$. H_1: Increasing overlap **decreases** first hitting time. Statistics for Figure 10.4.

o	0	1	2	3	4
	0/0	**1/1**	**6/6**	**16/19**	**41/58**

(a) $b = 1$

o	0	1	2	3	4
	0/0	**1/1**	**7/7**	19/28	45/110

(b) $b = 6$

Table 10.8: LT-GOMEA vs random linkage LT-GOMEA, H_1: LT-GOMEA has a **lower** first hitting time than U-GOMEA.

o	0	1	2	3	4
	0/0	**1/1**	**5/5**	**11/15**	**34/48**

(a) $b = 1$

o	0	1	2	3	4
	0/0	**1/1**	**6/6**	**14/17**	71/110

(b) $b = 6$

Table 10.9: LT-GOMEA vs U-GOMEA, H_1: LT-GOMEA has a **lower** first hitting time than U-GOMEA.

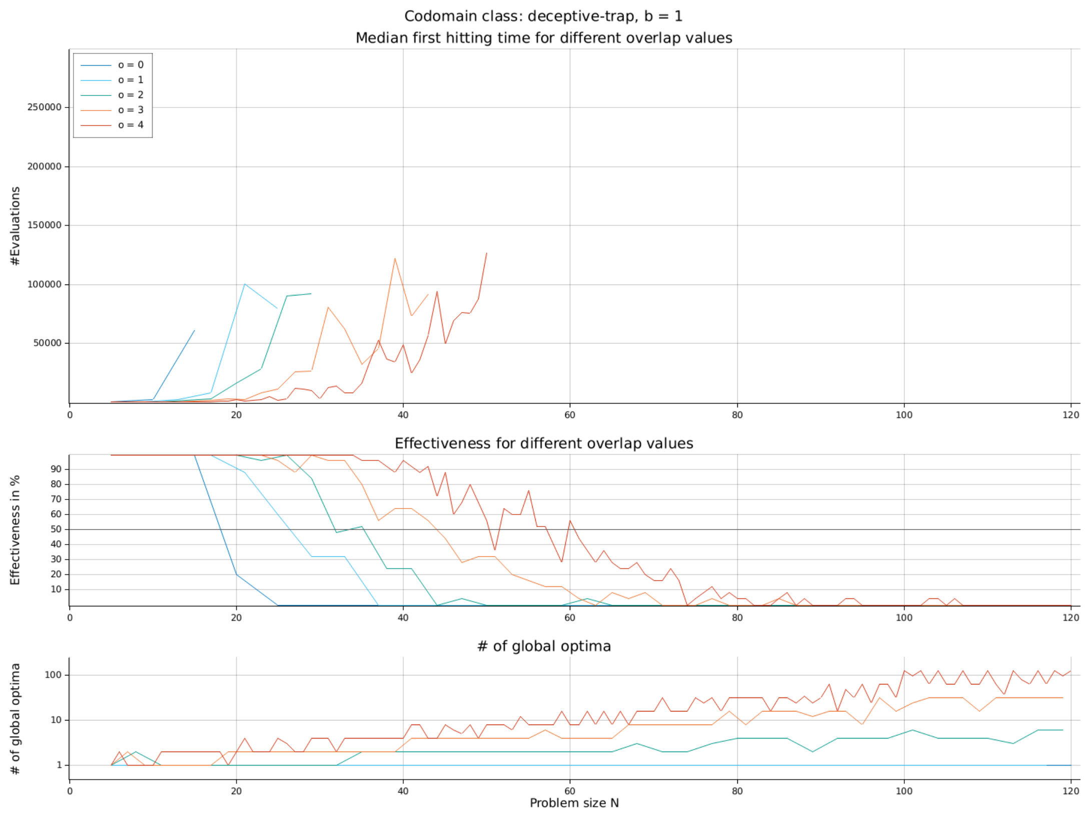
o	0	1	2	3	4
	0/0	0/1	0/5	0/15	3/48

(a) $b = 1$

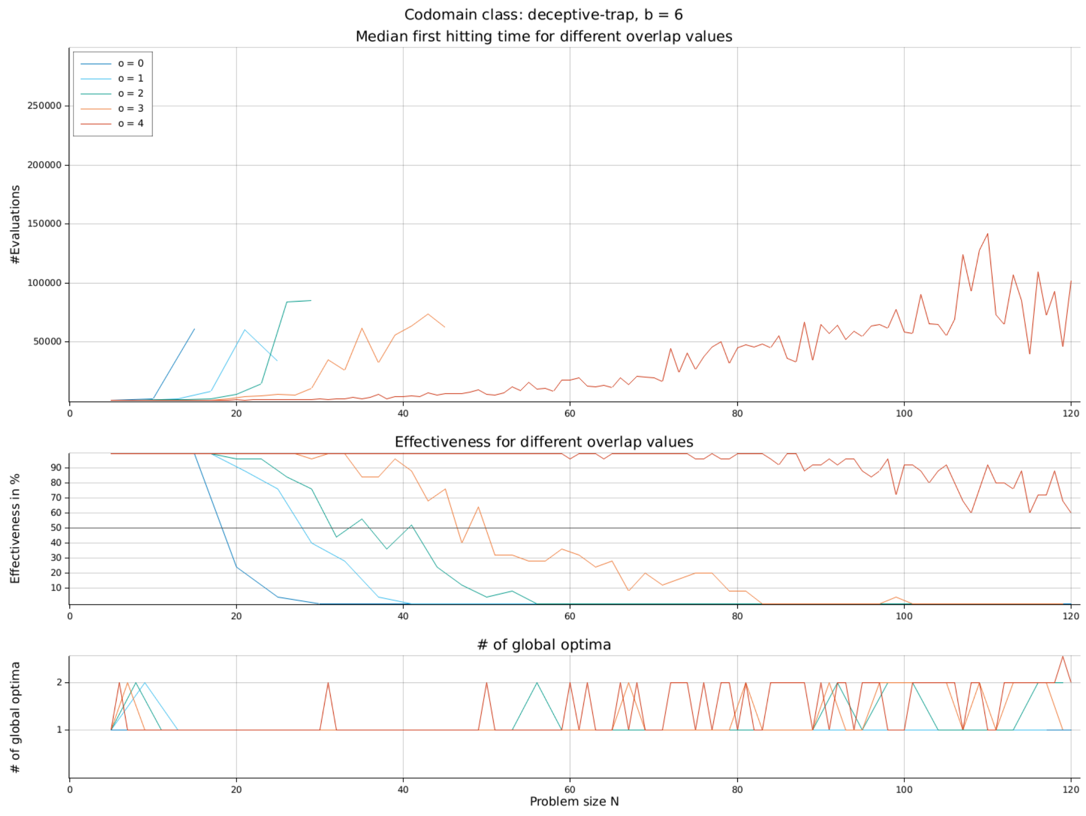
o	0	1	2	3	4
	0/0	0/1	0/5	1/17	32/110

(b) $b = 6$

Table 10.10: random linkage LT-GOMEA vs U-GOMEA, H_1: **Different** first hitting time.



(a) $b = 1$



(b) $b = 6$

Figure 10.4: Performance and effectiveness of U-GOMEA for different overlap values with different branching values: a) $b = 1$ and b) $b = 6$.

time, as can be read from Tables 10.1, 10.4, and 10.6. This equal or increase in performance and effectiveness hints at the possibility of a higher branching factor leading to a lower difficulty in the general case. Furthermore, we can see that with increasing overlap, the performance and effectiveness of random linkage LT-GOMEA and U-GOMEA increase. This is supported by Tables 10.5a, 10.5b, 10.7a, and 10.7b. This might support our hypothesis that increasing overlap increases interference between subfunctions, leading to decreased deceptiveness and decreased importance of linkage learning, thus to easier problems. We can also see that random linkage LT-GOMEA and U-GOMEA are closer to the performance and effectiveness of LT-GOMEA with increasing b, especially for $o = 4$. See Tables 10.8a, 10.8b, 10.9a, and 10.9b. This again hints at the possibility of the increasing overlap and branching factor, especially together, increasing the interference between deceptive subfunctions. This would then lead to a decreased deceptiveness and decreased importance of linkage learning of the problem. If the importance of linkage learning would indeed decrease, the advantage from having the linkage learning to learn the deceptive trap structure would be decreased and the disadvantage of not knowing the deceptive trap structure would be decreased as well. This would then definitely explain the relatively smaller difference in performance and effectiveness between LT-GOMEA and random linkage LT-GOMEA and U-GOMEA.

We see that the difference in first hitting times between random linkage LT-GOMEA and U-GOMEA for equal overlap values is insignificant, however, random linkage LT-GOMEA does have a higher effectiveness than U-GOMEA, most noticeably so for the random codomain. This is as we hypothesized, so the mixing with blocks of size > 1 does give an advantage. Interestingly, the difference between the three gets very small with $o = 4$ and $b = 6$. This supports our claim that the increasing overlap and branching decreases deceptiveness and the importance of linkage learning, as U-GOMEA really does not perform well when there is deceptiveness, so a lot of the deceptiveness must have been removed. This applies in part to random linkage LT-GOMEA as well. Another way of looking at this is that learning the structure is not essential to find a global optimum for $o = 4$ and $b = 6$, but there still is a big difference in performance.

The relatively close scores for $o = 4$ and $b = 6$ for the three algorithms begs the question what kind of structure the problems have, if the deceptive trap structure really is decreased. Given the fact that random linkage LT-GOMEA performs better than U-GOMEA, it is clear that the problem is not represented best by a univariate FOS, so there must still be some higher-level structure. This is supported by the fact that LT-GOMEA still performs better than random linkage LT-GOMEA, which tells us there is still benefit in linkage learning.

Interesting is also the difference of increasing the overlap for on the one hand LT-GOMEA and on the other hand random linkage LT-GOMEA and U-GOMEA. For LT-GOMEA, the difficulty first increases, before decreasing from $o = 3$ onwards. But for random linkage LT-GOMEA and U-GOMEA, any increase in overlap means a decrease in difficulty. As mentioned before, we think this is caused by the effects of the increase in interference between the subfunctions: 1) The decreased deceptiveness decreases the difficulty for random linkage LT-GOMEA, but for LT-GOMEA it was not a problem in the first place due to the linkage learning (and block mixing) and therefore its decrease does not really increase the performance for LT-GOMEA. 2) The decreased importance of linkage learning decreases the difficulty for random linkage LT-GOMEA, but for LT-GOMEA it does not initially decrease the difficulty, as it is able to learn the linkage.

In fact, when increasing the overlap initially, the Linkage Tree can not represent the overlapping structure fully, so the mixing will be less effective. However, with increasing overlap, the deceptiveness and the importance of learning the (exact) linkage decreases, overshadowing the inability to represent and learn the problem structure fully, and decreasing the difficulty.

Interestingly, as can be seen in Appendix Figures A.2 and A.3, we do not see the conflicted results between $b = 1$ and $b = 2$ for random linkage LT-GOMEA and U-GOMEA, which we do see for LT-GOMEA. As we have seen that the number of global optima does affect the performance of random linkage LT-GOMEA (and therefore possibly also of U-GOMEA), we assume that this is due to the bigger difference in performance and effectiveness we see with increasing branching factor b, which must then be greater than the effect of a higher number of global optima.

10.2.2 # Cliques per Variable

To give us a better sense of how the higher branching factor b could influence the topography and the overlap of the problem, we will take a look at the number of cliques each variable is in. We expect the number of cliques per variable to strongly increase due to the higher branching and think this could be a reason for the observation that problems with high overlap and branching values are easier.

To test this, we generate 25 problems for $N = 100$, $k = 5$, $o = 4$, $b \in \{1, 2, 3, 4, 5, 6\}$ and note the number of cliques each variable is in. For these values, we calculate the statistics and draw the distribution. For 3 of these problems we show the statistics in Table 10.11 and for the first two of these problems in the table (with $b \in \{1, 2, 6\}$) we show the distributions in Figure 10.5.

The results show that with the increase of the branching factor, the distribution is pushed to the extremes. The change between $b = 1$ and $b = 2$ is the most dramatic, with a big increase in the variance, visible in the figure with an almost double the amount of occurrences for 1 clique and a much higher maximum value of ≈ 70 cliques per variable compared to $\approx 20 - 35$. For increasing b beyond 2, a higher b increases the variance further, with more occurrences for 1 clique and more occurrences having a number of cliques per variable of > 50. These changes are caused by the increasing shallowness of the clique tree; more and more cliques end up as leafs, so their non-overlapping variable will occur in just 1 clique, and more and more cliques overlap with the same clique, leading to higher maxima and/or more variables that are in a lot of cliques.

So how does this explain the lower difficulty of problems with a higher branching factor? We hypothesized that it is due to the increased overlap between cliques, but we now additionally hypothesize that it is partly due to the number of occurrences of 1 clique per variable. In other words, we hypothesize/think that a higher branching factor changes the overlap to be less evenly divided over the cliques, which decreases the difficulty. The overlap is shifted to some variables that occur in a high number of cliques and a lot of variables that occur in just 1 clique. We consider two possible explanations of the results:

In explanation 1, the value of the variables that are in so many cliques does not matter, as they are in so many cliques and in each there is a random local optimum. Therefore, there is a 50/50 chance per clique what the best value for this variable is; 0 or 1. Because of this, the deceptive trap structure is still quite intact as the value of these high-profile variables does not have a big impact on the problem as a whole, but it does on the current clique. The lower difficulty of the higher branching value problems should then be totally explained by the high number of variables that are in just 1 clique. The only impact we can think of that this could have is that maybe the problems become more separated, as the high-profile variables can take either value and are therefore independent? Then it's just solving the cliques separately, as the variables in it that occur only once are also independent. However, the high-profile variables can of course only be set to one value in the end, so we expect that it's actually the case that the high-profile variables are supposed to be set to one variable in order to find the global optimum:

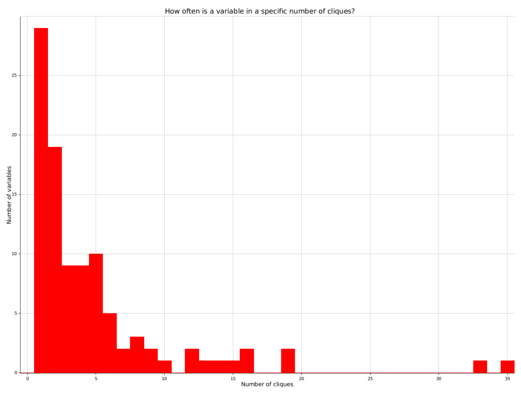
In explanation 2, the value of the high-profile variables does matter in order to find the global optimum or optima. Then the resulting difference in fitness when flipping the high-profile variables' bits must be big. Additionally, the cliques that contain these high-profile variables have lost some of their deceptive trap structure, as some of their variables already have a set value. Solving the problems is then a matter of first finding the best value for these variables and then finding the best value for the remaining variables in the clique.

We have answered our question of what the effects are of an increasing branching factor, and have looked shortly at what might be the reason for that. Now, in the following chapter we will further investigate the hypothesis that we put forth on the effects of the increase overlap, to answer the question of *How does an increasing overlap affect the performance and effectiveness of LT-GOMEA?*. We do this by taking a closer look at another codomain to test our hypothesis.

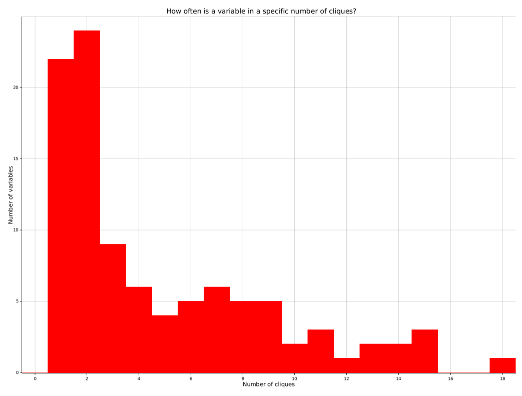
10.3 Conclusions

10.3.1 General

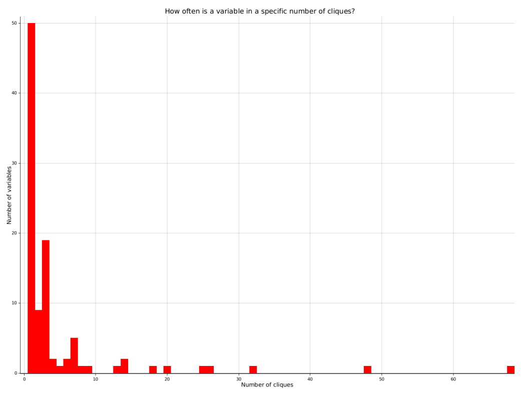
- The difficulty of the problems (in terms of first hitting time and effectiveness) decreases with increasing branching factor ($b = 1 \to 6$), for all algorithms considered (LT-GOMEA, random linkage LT-GOMEA, and U-GOMEA) and $o = 4$.
- Factors at play between $b \in \{1, 2\}$ for $o = 4$ are 1) the number of global optima and 2) a further decrease of deceptiveness and importance of linkage learning. The second factor is caused by the distribution of cliques per variable shifting to the extremes: there are some variables that appear in a lot of cliques and a lot of variables that appear in just 1 clique. This increases the interference between the subfunctions.



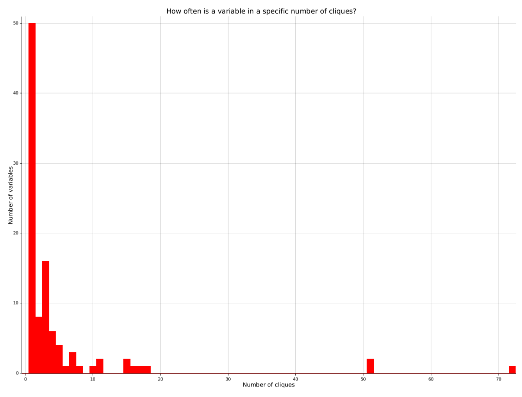
(a) $b = 1$



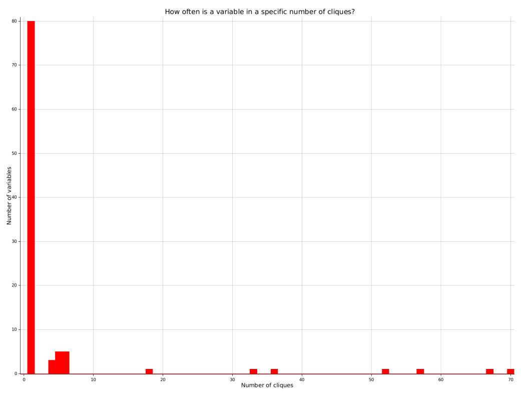
(b) $b = 1$



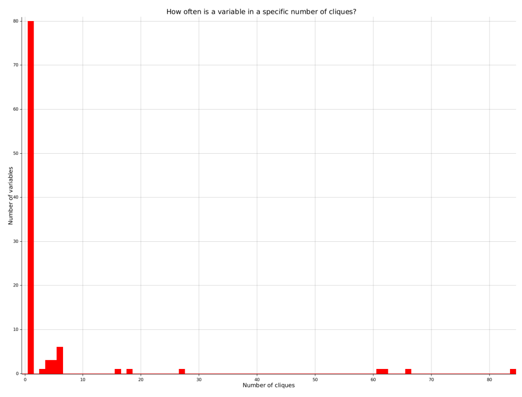
(c) $b = 2$



(d) $b = 2$



(e) $b = 6$



(f) $b = 6$

Figure 10.5: Distribution of number of cliques per variable. On the x-axis is the number of cliques a variable is in, on the y-axis is the number of occurrences for that number of cliques per variable. For deceptive trap codomain, with $N = 100$, $k = 5$, $o = 4$, so $M = 96$ and thus there are 96 cliques in total. Per b setting, we plotted the distribution for 2 problem instances.

b	min	max	std_dev	var	median	p25	p75
	1	35.00	5.89	34.69	3	1	5.00
1	1	18.00	4.18	17.43	3	2	7.00
	1	22.00	4.05	16.40	3	2	7.00
	1	68.00	9.54	91.01	1.5	1	3.00
2	1	72.00	10.26	105.17	1.5	1	4.00
	1	57.00	8.77	76.99	1.5	1	4.00
	1	71.00	11.21	125.74	1	1	3.00
3	1	64.00	10.91	118.99	1	1	4.00
	1	75.00	11.65	135.64	1	1	3.00
	1	76.00	12.67	160.65	1	1	3.00
4	1	73.00	12.16	147.82	1	1	3.58
	1	76.00	12.36	152.85	1	1	3.00
	1	73.00	12.01	144.30	1	1	1.00
5	1	75.00	12.95	167.66	1	1	1.00
	1	67.00	12.31	151.45	1	1	1.00
	1	70.00	12.78	163.43	1	1	1.00
6	1	84.00	13.64	185.96	1	1	1.00
	1	69.00	12.59	158.40	1	1	1.00

Table 10.11: The statistics for the number of cliques per problem variable. For each b value, we show the statistics for 3 problems. The average is always 4.8. For deceptive trap codomain, with $N = 100$, $k = 5$, $o = 4$, so $M = 96$ and thus there are 96 cliques in total.

- Thus, we hypothesize that increasing overlap and branching both increase the interference between the subfunctions, which leads to a decrease in the deceptiveness and importance of (exact) linkage learning. This makes the problems easier in all cases for random linkage LT-GOMEA and U-GOMEA, and easier in some cases for LT-GOMEA.

10.3.2 LT-GOMEA

- For LT-GOMEA, the first hitting time of $o = 4$ is significantly lower than the first hitting time of $o = 0$ for $b = 6$ and $N \leq 120$, however the effectiveness is still lower. The first hitting time of $o = 3$ is no longer significantly higher than the first hitting time of $o = 0$ for $b = 6$, whereas it was for $b = 1$. Unsurprisingly then, $o = 3$ has a significantly higher first hitting time than $o = 4$ for $b = 6$.

- For LT-GOMEA, the performance is first decreased with increasing overlap, due to its inability to represent the structure fully and the resulting less efficient mixing. The decreasing deceptiveness a) does not initially increase the performance of LT-GOMEA, as this was not a problem, or b) it does increase the performance, but the effect of the inability to represent the problem structure is stronger. At $o = 3$, the inability to represent the problem structure fully is overshadowed by the interference between subfunctions, leading to decreased deceptiveness and importance of (exact) linkage learning.

10.3.3 Random linkage LT-GOMEA & U-GOMEA

- Although there is no significant difference in first hitting time, random linkage LT-GOMEA does have a higher effectiveness than U-GOMEA, so mixing with blocks of size ≥ 1 is advantageous.

- Learning the linkage is not essential to find the global optimum for problems with $o = 4$ and $b = 6$, but there still is a big difference in performance.

10.3.4 Branching

- The distribution of the number of cliques a variable is in, is pushed to the extremes with increasing b; there are some variables that appear in a lot of cliques and a lot of variables that appear in just 1 clique. The biggest change is for $b = 1 \rightarrow 2$, but the distribution changes further with increasing b. This change in the distribution is caused by the increasing shallowness of the clique tree; an increasing number of cliques are leafs in the clique tree.

- This means that some variables occur in a big number of cliques (≈ 60) and we hypothesize that it is very important to get the value of these variables right to find a global optimum. After setting these variables' values, most of the remaining variables' values can be set by just checking which value is higher (0 or 1). This would also partly explain the further decrease in deceptiveness.

Chapter 11

Experiment: Random vs. Deceptive Trap Codomain

In the previous experiments, we used the deceptive trap codomain as described in section 7.1.1, for which we saw the behavior as listed in the previous chapters. In this chapter, we perform some experiments to see how much of this behavior is due to the used codomain, by comparing the results of the deceptive trap codomain with the random codomain. We test using both the regular LT-GOMEA, that we have used in most experiments, and the random linkage LT-GOMEA, to identify any differences in the results due to the differences in the algorithms.

In this experiment, we try to answer the question:

How does an increasing overlap affect the performance and effectiveness of LT-GOMEA for TD Mk Landscapes problems with the random codomain?

In the very first experiment, we saw the pattern that the difficulty first increased with increasing overlap, before decreasing when the overlap is increased further. The difficulty first increasing before decreasing was, as explained in the results section in the previous chapter, due to the combination of 1) LT-GOMEA getting worse at representing the structure and 2) the deceptiveness and importance of (exact) linkage learning decreasing. For the random codomain, there is no deceptiveness to decrease, but the importance of linkage learning will decrease with increasing overlap o. The latter should counter the increasing inability of the Linkage Tree to represent the problem structure, but it is yet unclear how these will interact exactly. We hypothesize that we see a similar pattern here as for the deceptive trap codomain. For random linkage LT-GOMEA, which does not learn the linkage, the only active factor is the decreasing importance of the linkage learning. Therefore, we expect the performance and effectiveness to improve with increasing overlap o.

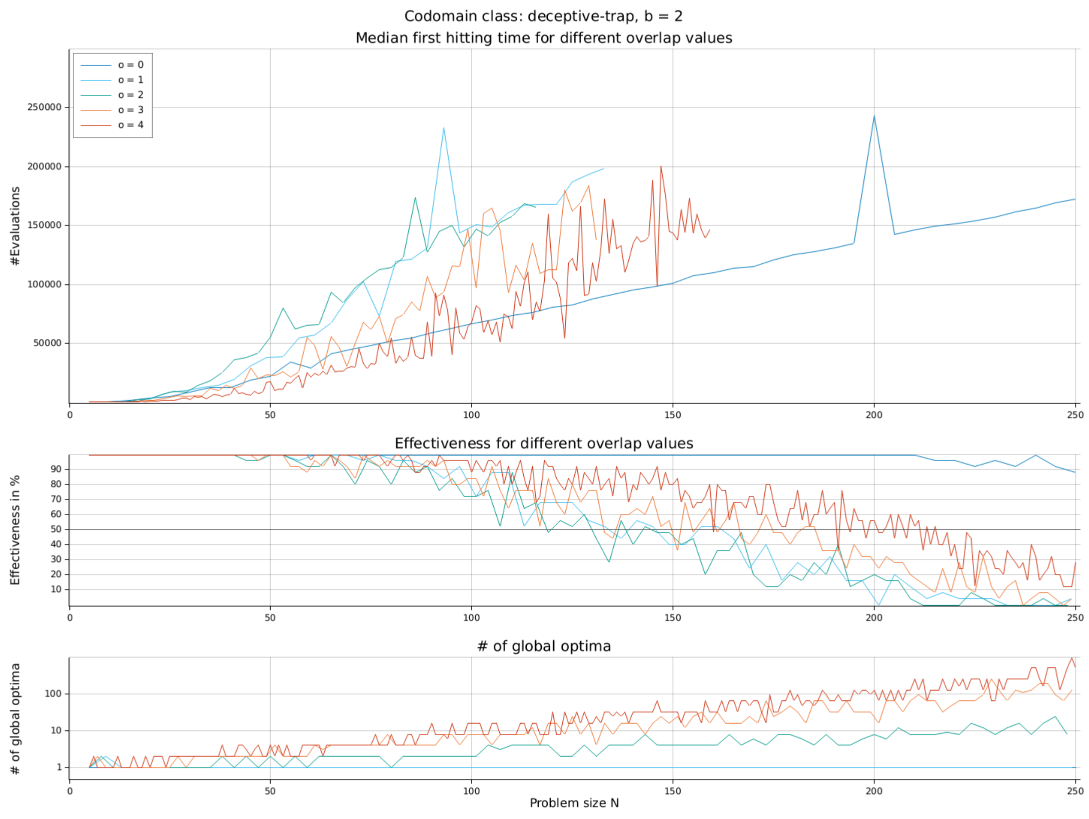
11.1 Experimental Setup

We use many of the same configuration settings as in the experiment in the previous chapter, where we looked at branching factors $b \in \{1, 2, 3, 4, 5, 6\}$, but limit the branching factor here to a value of $b = 2$ to eliminate the high number of global optima problem. As the number of global optima is not an issue then anymore, we use a maximum problem size of 250. So, for the problems, we use configuration input: $M \in \{m \,|\, N \leq 250\}$, $k = 5$, $o \in \{0, 1, 2, 3, 4\}$, $b = 2$. Where problem size $N = (m - 1) \cdot (k - o) + k$. We test on the random and deceptive trap codomains, and use LT-GOMEA and random linkage LT-GOMEA.

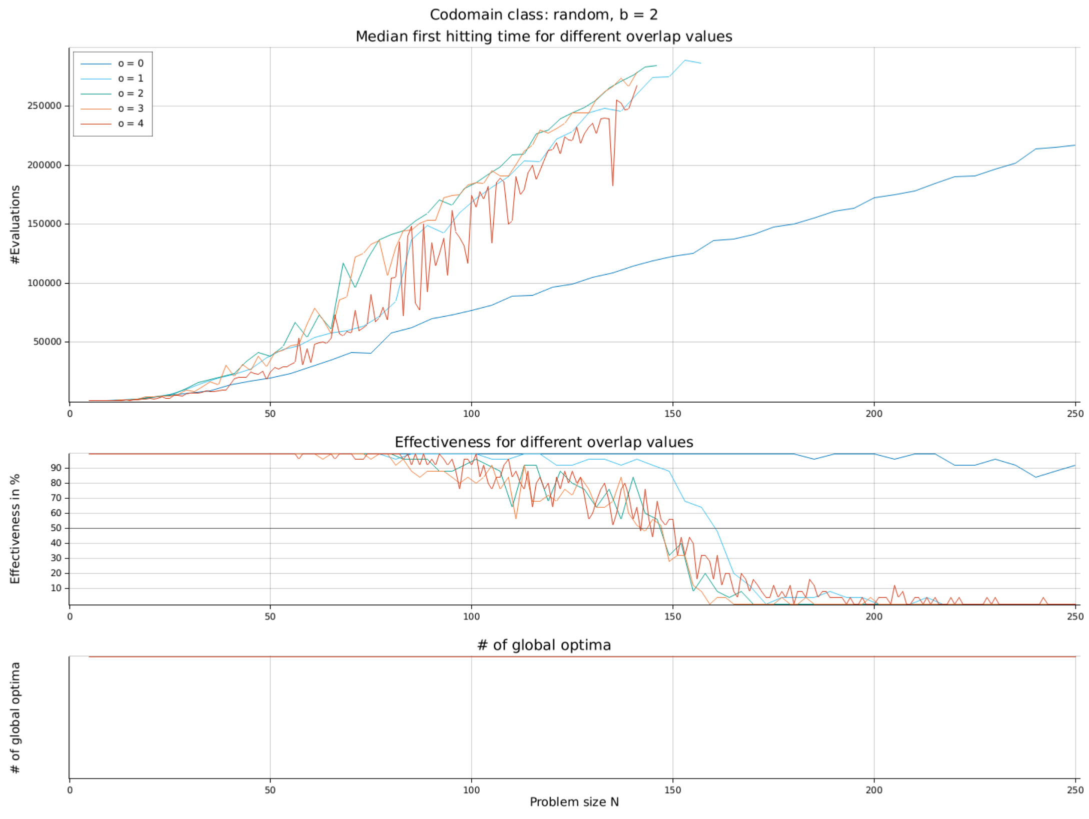
11.2 Results

11.2.1 LT-GOMEA

The results for LT-GOMEA are plotted in Figures 11.1a and 11.1b, for the deceptive trap and random codomains respectively.



(a) deceptive trap



(b) random

Figure 11.1: Performance and effectiveness of LT-GOMEA for different overlap values on a) deceptive trap and b) random codomain.

o	1	2	3	4
0	**6/6**	**8/8**	**12/12**	**21/24**
1	X	4/10	6/15	5/31
2	X	X	2/20	**32/43**
3	X	X	X	30/65

Table 11.1: Statistical results for the differences in mean values between different overlap values, for $o \in \{0, 1, 2, 3\}$ vs $o \in \{1, 2, 3, 4\}$, LT-GOMEA, random codomain, $b = 2$. H_1: Increasing overlap results in a **different** first hitting time.

o	0	1	2	3	4
	20/44	11/31	13/38	34/64	88/139

Table 11.2: Statistical results for the differences in mean values between the deceptive trap and random codomain, for $o \in \{0, 1, 2, 3, 4\}$, LT-GOMEA, $b = 2$. H_1: Deceptive trap has a **different** first hitting time than random codomain.

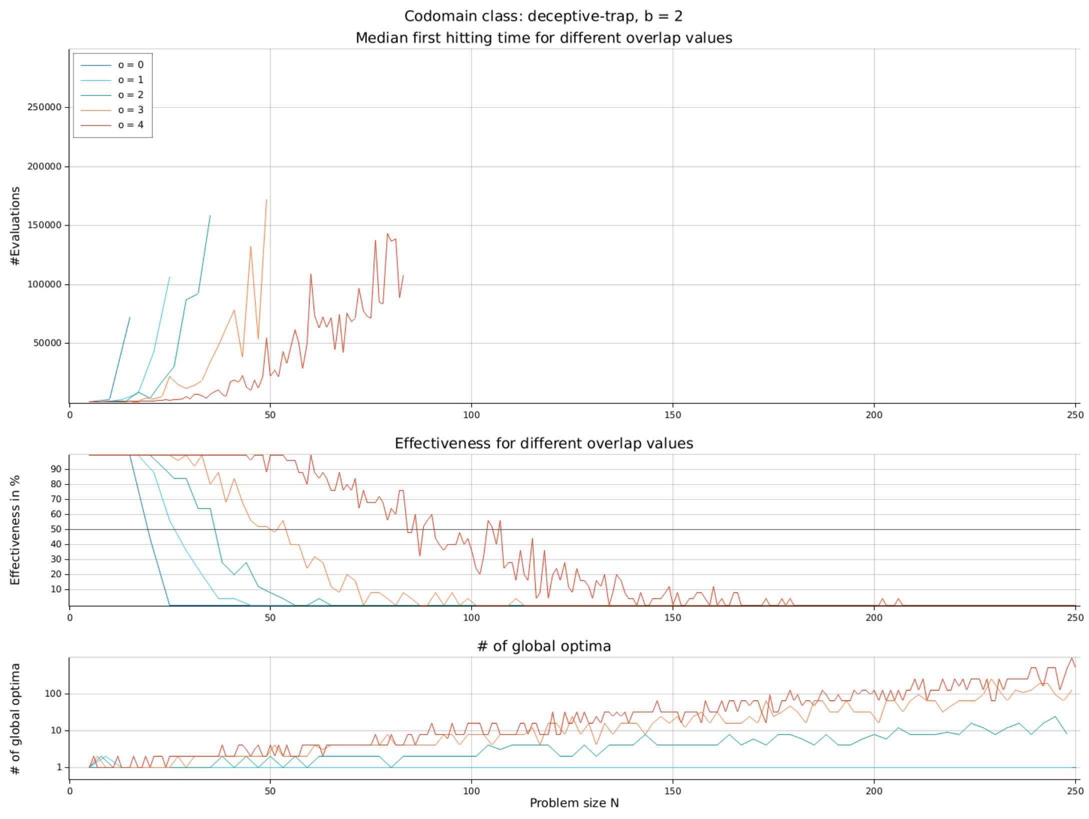
First, we discuss the results for the random codomain. Something that stands out is the sudden drop in the effectiveness around a problem size of 140, this is due to our configured maximum number of evaluations for LT-GOMEA of 300,000. After this maximum number of evaluations is spent, LT-GOMEA is stopped. If the median first hitting time approaches 300,000, then many runs exceed this number and are therefore stopped. This decreases the effectiveness quickly.

Furthermore, we indeed see a pattern similar to the one we saw for the deceptive trap codomain. For increasing overlap o, the difficulty first increases in both effectiveness and first hitting time for $o = 0 \to 1$ and in the effectiveness for $o = 1 \to 2$. Then, the difficulty decreases between $o = 2 \to 4$ in the first hitting time. See Table 11.1 for the statistical data. Our hypothesis correlates with the results, so it could be right. The difficulty could indeed first increase due to the increasing inability to represent the problem structure, before decreasing due to the linkage learning importance decreasing and having a greater effect than the inability to represent the structure.

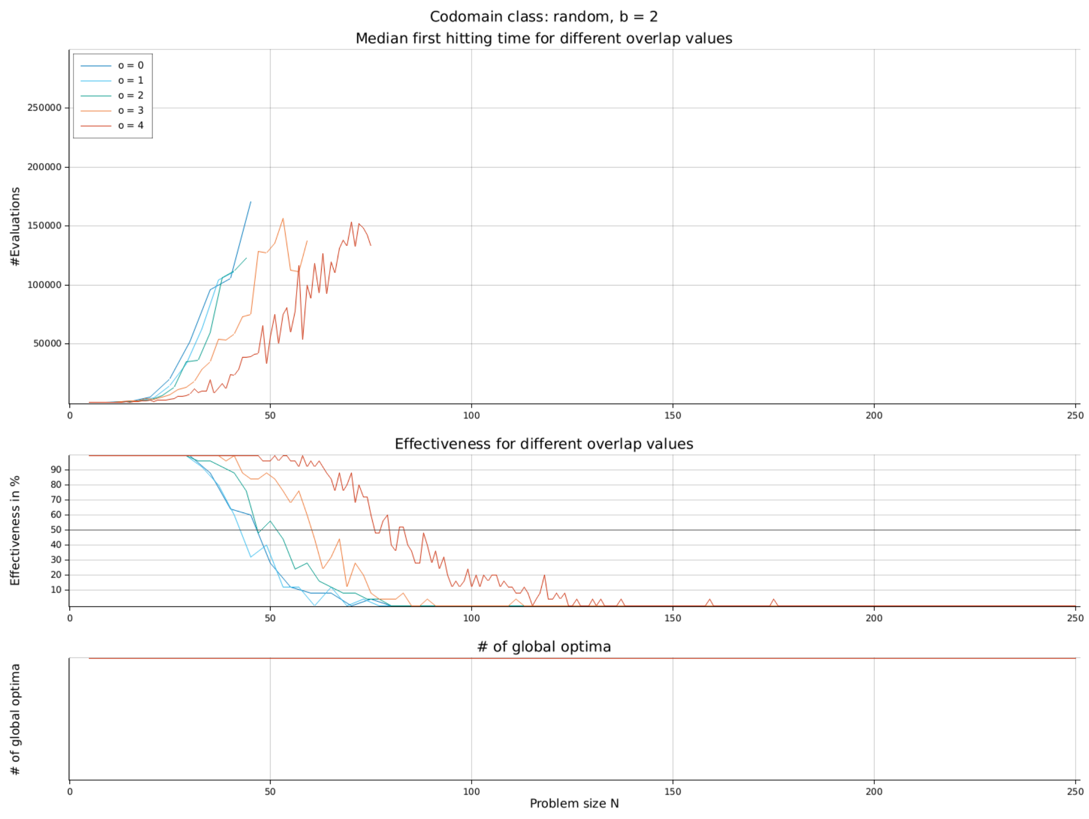
So now that we understand why the results of LT-GOMEA on TD Mk Landscapes with random codomain are the way they are, we can compare the results for the deceptive trap and random codomain. Table 11.2 shows that any difference in the first hitting time is insignificant. This is an unexpected result, as the Linkage Tree is harder to learn for LT-GOMEA when a random codomain is used, so we expected a significant difference for at least $o = 0$. Although the differences in first hitting times are insignificant, in the figure it is clearly visible that $o = 1$ has a higher effectiveness for the random codomain, until the number of evaluations reach the maximum we set for the experiment. The reason for this is unclear, but one explanation could be that deceptive trap codomain is more difficult due to the deceptiveness and the inability to fully represent the structure, whereas for the random codomain not completely being able to represent the structure has less of an effect.

o	1	2	3	4
0	0/0	**1/1**	**2/2**	**4/4**
1	X	0/2	**2/2**	**5/5**
2	X	X	2/4	**9/10**
3	X	X	X	**19/23**

Table 11.3: Statistical results for the differences in mean values between different overlap values, for $o \in \{0, 1, 2, 3\}$ vs $o \in \{1, 2, 3, 4\}$, random linkage LT-GOMEA, random codomain, $b = 2$. H_1: Increasing overlap **decreases** first hitting time.



(a) deceptive trap



(b) random

Figure 11.2: Performance and effectiveness of random linkage LT-GOMEA for different overlap values on a) deceptive trap and b) random codomain.

o	0	1	2	3	4
	0/0	**1/1**	3/6	4/19	16/70

Table 11.4: Statistical results for the differences in mean values between the deceptive trap and random codomain, for $o \in \{0, 1, 2, 3, 4\}$, random linkage LT-GOMEA, $b = 2$. H_1: Deceptive trap has a **different** first hitting time than random codomain.

o	0	1	2	3	4
	4/4	**5/5**	**9/10**	**19/23**	44/70

Table 11.5: LT-GOMEA vs random linkage LT-GOMEA. H_1: LT-GOMEA has a **lower** first hitting time than random linkage LT-GOMEA.

11.2.2 Random Linkage LT-GOMEA

The results for random linkage LT-GOMEA are plotted in Figures 11.2a and 11.2b, for the deceptive trap and random codomains respectively.

As per our hypothesis, the difficulty decreases with increasing overlap. However, as can be seen in Table 11.3, any differences in first hitting time between $o \in \{0, 1, 2\}$ are not significant. For the increase in overlap from 0 to 1, there is no significant difference, as there are not enough problem sizes for $M > 5$ that can be compared. From 1 to 2, there is a difference, but it is not significant. However, from $o = 2$ onwards, the increase in overlap decreases the difficulty. From $o = 2$ to $o = 3$, this is only the case in terms of effectiveness, but for $o = 3$ to $o = 4$ both the first hitting time and the effectiveness improve. We suggested that the decrease in importance of the linkage learning could be an explanation, this can indeed be the case. Noteworthy is the equal difficulty for $o \in \{0, 1, 2\}$, therefore possibly suggesting that the importance of linkage learning only begins to decrease when o is increased beyond 2.

From Table 11.4, we can draw the conclusion that there is no significant difference in the first hitting time between the deceptive trap and random codomain, for equal overlap settings and the random linkage LT-GOMEA algorithm. Note that, although there is a significant difference in the first hitting time for overlap setting 1, it is recorded for just one problem size, so it is not enough to reliably conclude that there is a significant difference. However, the effectiveness does differ quite a bit between the two codomains for $o \in \{0, 1, 2\}$. For $o = 0$, this could be explained by the fact that the random linkage LT-GOMEA does not learn any linkage, so it will not be affected by the higher difficulty of learning of the random codomain's structure and is thus only affected by the lower difficulty codomain (for algorithms that do not learn linkage) as it is not necessarily deceptive. This would extend to overlap settings $o \in \{1, 2\}$ as well. Furthermore, for the deceptive trap codomain, the deceptiveness decreases with increasing overlap, so this could explain the bigger differences in difficulty between overlap settings for the deceptive trap codomain in comparison with the random codomain.

From Table 11.5 we can conclude that the difference in first hitting time between LT-GOMEA and random linkage LT-GOMEA for the random codomain is significant for $o \in \{0, 1, 2, 3\}$. For these overlap values, LT-GOMEA performs better. Interestingly, for the deceptive trap codomain, the difference was significant for $o \in \{2, 3, 4\}$, as we saw in Table 10.8a. Importantly, LT-GOMEA has a higher effectiveness than random linkage LT-GOMEA in all tested configurations with $b = 2$ (both deceptive trap en random codomains).

11.3 Conclusions

- As we hypothesized, the first hitting time of LT-GOMEA follows a similar pattern for the random codomain as for the deceptive trap codomain. We think the difficulty first increases due to the increasing inability to represent the problem structure, before decreasing due to the linkage learning importance decreasing and having a greater effect than the inability to represent the structure.
- Furthermore, as we hypothesized, the first hitting time of random linkage LT-GOMEA for the random codomain does decrease with increasing overlap. However, the first hitting time

is similar for overlap values $o \in \{0, 1, 2\}$. We suggested that the first hitting time would decrease due to the decreasing importance of linkage learning, and still agree, but the results suggest that the importance of the linkage learning only starts to decrease from $o = 2 \rightarrow 3$ onwards.

- Interestingly, for random linkage LT-GOMEA, the first hitting time and effectiveness between overlap values $o \in \{0, 1, 2\}$ is very similar for the random codomain, as mentioned above, but different for the deceptive trap codomain. Above we tried to explain this result for the random codomain, here we just add a comment about the difference we see here between the overlap values for deceptive trap: contrarily to the random codomain, deceptiveness does decrease for the deceptive trap codomain with increasing overlap and this explains the relatively bigger differences in difficulty between the overlap values. In other words, the effect of the overlap o on the difficulty of the problems is greater for the deceptive trap codomain than for the random codomain, due to not just the importance of the linkage learning decreasing, but the deceptiveness decreasing as well. Finally, the effectiveness for $o = 0$ differs between the two codomains, this could be explained by the easier subfunctions of the random codomain (as it is not necessarily deceptive).
- For both LT-GOMEA and random linkage LT-GOMEA, there is no significant difference in the first hitting time between the deceptive trap and random codomains, for $o \in \{0, 1, 2, 3, 4\}$.
- LT-GOMEA has a similar or better first hitting time and a better effectiveness than random linkage LT-GOMEA for all tested configurations. Thus, this suggests that linkage learning is recommended for the tested configurations.

Chapter 12

Conclusions

In this work, we have answered the research question *'What is the performance and effectiveness of LT-GOMEA on certain subclasses of TD Mk Landscapes?'* for the subclasses considered in this work: codomains = {deceptive trap, random}, and topological properties = {overlap, branching}. For these subclasses, we have looked at the interaction between the above subclasses of TD Mk Landscapes and the landscape features = {number of global optima, deceptiveness}. Furthermore, we have reported the effect of changes in the subclasses and the landscape features on the performance and effectiveness of LT-GOMEA. This hopefully contributes to a better understanding of the TD Mk Landscape benchmark and its parameters.

In this chapter, we summarize the context and results of this work, offer an overview of the takeaway points, and list opportunities for future work.

12.1 Summary

Whitley et al.[23] recently introduced the TD Mk Landscape to generalize the Adjacent NK Landscape, by removing its unnecessary constraints (for benchmarking) and by focusing on the key property to allow for a polynomial global optimum calculation: the tree-width of the Mk Landscape (or k-bounded pseudo-Boolean optimization problem) should be known and bounded by k. We think the TD Mk Landscape is very suitable to benchmark black box optimization algorithms, due to 1) the fact that linkage learning will be required for black box algorithms to reliably and efficiently find the global optimum for some codomains, and 2) this polynomial global optimum calculation, allowing to report on the effectiveness and overall performance of an algorithm.

Thierens et al.[20] recently introduced the CliqueTreeMk algorithm to generate TD Mk Landscapes and find their global optimum (or optima). We introduced it in more detail in this work, and use the implementation by van Driessel et al.[6] to generate the TD Mk Landscapes for our experiments.

For our experiments, we use a linkage learning genetic algorithm called Linkage Tree Gene-pool Optimal Mixing Evolutionary Algorithm (LT-GOMEA) to show the difficulty of the problems. The LT-GOMEA algorithm has shown state-of-the-art performance for some discrete, Cartesian-space optimization problems[3].

In our experiments, we have tried to answer the question: *What is the performance and effectiveness of LT-GOMEA on certain subclasses of TD Mk Landscapes?* In the remainder of this summary, we summarize our experiments by listing the specific research questions and the results.

12.1.1 Increasing Overlap

We started off with our first question to get a first intuition: *How does an increasing overlap affect the performance and effectiveness of LT-GOMEA for TD Mk Landscapes problems with the deceptive trap codomain?*

Although this initial experiment does not allow for firm conclusions, we observed a pattern of the difficulty first increasing prior to decreasing with increasing overlap, which served as a starting point for our research. To begin answering our initial question, we therefore asked ourselves the question:

Why does the difficulty first increase with increasing overlap, before decreasing for $o > 2$?

12.1.2 Branching / Global Optima

Our first idea was to look into the effect of the number of global optima, therefore asking the following question:

How does the number of global optima affect the performance and effectiveness of LT-GOMEA for TD Mk Landscapes problems with the deceptive trap codomain?

Then, we discovered that the number of global optima is not the only change when we increase the branching factor. Therefore, we zoomed in on the global optima *and* on the other effects of increasing the branching factor.

12.1.3 Global Optima

First, as our previous question was already considering the number of global optima, we again tried to answer it, now by really only having the number of global optima differ between two problem sets:

How does the number of global optima affect the performance and effectiveness of LT-GOMEA for TD Mk Landscapes problems with the deceptive trap codomain?

We concluded that the number of global optima does have an effect on the first hitting time and effectiveness, but it is smaller than we expected. We think the difference is so small, because the hamming distance between the global optima is small; LT-GOMEA must already be close to one of the global optima to find the others.

Furthermore, we put forth our hypothesis that the interference of the deceptive subfunctions, caused by the increasing overlap, causes 1) the high number of global optima, 2) a decrease in deceptiveness of the problem, and 3) a decrease in importance of learning the (exact) linkage. It could be the case this interference is further increased by increasing the branching factor.

12.1.4 Increasing Overlap and Branching

Now we turned to the other effects of increasing the branching factor; we tried to answer the question: *How does an increasing branching factor affect the performance and effectiveness of LT-GOMEA for TD Mk Landscapes problems with the deceptive trap codomain?*

As per our hypothesis, the first hitting time and effectiveness was equal or improved with increasing branching ($b = 1 \to 6$) for all algorithms (LT-GOMEA, random linkage LT-GOMEA, and U-GOMEA), but just for $o = 4$. The only exception was for LT-GOMEA and $b = 1 \to 2$.

When the branching is increased for $o = 4$, 1) the number of global optima decreases and 2) the deceptiveness and importance of linkage learning decrease. The second factor is caused by the further increased interference between subfunctions, which is due to the distribution of cliques per variables shifting to some variables appearing in a lot of cliques and many variables appearing in just 1 clique.

We hypothesize that the variables that appear in a big number of cliques (≈ 60), due to this shifted distribution, are very important to finding a global optimum. If these variables are set to the right value, most of the remaining variables can be set by checking what value is higher. If this hypothesis is right, this would partly explain the further decrease in deceptiveness.

So, increasing overlap and branching both increase the interference between the subfunctions, therefore decreasing the deceptiveness and importance of learning the (exact) linkage. This decreases the difficulty in all cases for random linkage LT-GOMEA and U-GOMEA, and in some cases for LT-GOMEA.

Our contemporary understanding then led us to hypothesize that the following happens when the overlap is increased for LT-GOMEA: Due to its inability to represent the problem structure fully, the mixing is less efficient and the performance and effectiveness is decreased. The decreasing deceptiveness either does not initially increase the performance and effectiveness of LT-GOMEA, as this was not a problem, or it does increase the performance and effectiveness, but the effect of the inability to represent the problem structure is stronger. At $o = 3$, the inability to represent the problem structure is overshadowed by the interference between subfunctions, leading to decreased deceptiveness and importance of (exact) linkage learning, and thus to increased performance and effectiveness.

12.1.5 Random vs. Deceptive Trap Codomain

Now that we answered the question on the effects of increased branching and looked shortly into the mechanism behind it, we turn our attention to testing our hypothesis on the effects of increasing overlap. We do so by answering the question:

How does an increasing overlap affect the performance and effectiveness of LT-GOMEA for TD Mk Landscapes problems with the random codomain?

Just as we expected, the first hitting time of LT-GOMEA for the random codomain follows a pattern similar to the one visible for the deceptive trap codomain, with increasing overlap. We think this is due to just the decreasing deceptiveness missing: the difficulty first increases due to the increasing inability to represent the problem structure, before decreasing due to the linkage learning importance decreasing and having a greater effect than the inability to represent the problem structure.

Furthermore, the first hitting time of random linkage LT-GOMEA for the random codomain decreases with increasing overlap, also as we expected based on our hypothesis. Again, this is similar to the pattern we saw for deceptive trap, but now with a smaller effect due to the decreasing deceptiveness missing. The results suggest that the importance of the linkage learning only starts to decrease from $o = 2 \to 3$ onwards.

To rephrase part of what was noted above: the effect of the overlap o on the difficulty of the problems is greater for the deceptive trap codomain than for the random codomain, due to the deceptiveness decreasing as well.

This now has answered our research questions, but of course leaves room for future research to confirm or deny our hypotheses and results.

Finally, it should be noted that LT-GOMEA has a similar or better first hitting time and a better effectiveness than random linkage LT-GOMEA for all tested configurations (different codomains, o, b). Thus, this suggests that linkage learning is beneficial, even when the Linkage Tree FOS can not represent the problem structure fully due to overlap between the subfunctions.

12.2 Takeaways

- With increasing **overlap**
  - the Linkage Tree's ability to (learn/) represent the structure decreases
  - being able to learn or represent the structure becomes less important (perhaps only from $o = 2 \to 3$ onwards for $k = 5$) due to interference between subfunctions
  - (deceptive trap codomain) the deceptiveness decreases due to interference between deceptive trap subfunctions
  - ($b = 1$) the number of global optima increases
- With **high overlap** *and* increasing **branching**
  - the distribution of the cliques per variable in the clique tree shifts to one that has a few variables that occur in a lot of cliques and many that occur in just 1 clique. These variables that appear in a lot of cliques might be important to find a global optimum.
  - being able to learn or represent the structure becomes even less important due to a further increased interference between deceptive trap subfunctions (due to the shifted distribution)
  - (deceptive trap codomain) deceptiveness is further reduced due to an increased interference between deceptive trap subfunctions(due to the shifted distribution)
  - The number of global optima is reduced, with the biggest reduction from branching value $1 \to 2$. This increases the difficulty of the problems, but only by a little, due to the small hamming distance between the global optima.
- Linkage learning with the LT FOS is beneficial in our experiments.
- For the algorithms:

  - For **LT-GOMEA**, the difficulty of the TD Mk Landscape problems do not strictly increase with increasing overlap. Instead, we see it increase prior to decreasing. Using the above observations, we can explain these results:

    The Linkage Tree's ability to represent the structure decreases with increasing overlap, so this makes the mixing less effective, this is why the difficulty is increased initially. With increasing overlap, the ability to learn or represent the structure becomes less important, decreasing the negative effect of not being able to represent or learn the structure. For deceptive trap codomain problems, the deceptiveness also decreases due to the interference of the deceptive trap subfunctions.

    When the branching factor is increased in combination with a high overlap, the difficulty is further decreased for the deceptive trap codomain (random codomain was not tested). However, the exact value of b at which the difficulty decreases is dependent on the number of global optima for the different branching factors. For $N \leq 120$ and $o = 4$, the difficulty starts to decrease when the branching factor is increased to a value > 2, but this is specific to the experiment configuration. This is due to a big increase in the number of global optima decreasing the difficulty. As for the other factor at play when increasing the branching factor: the increased branching factor causes the distribution of cliques per variable to be concentrated at a handful of variables that occur in a high number of cliques, leaving a lot of variables in just 1 clique. This increases the interference between the subfunctions further and therefore decreases deceptiveness (in the case of deceptive trap codomain) and the importance of linkage learning.

  - For **random linkage LT-GOMEA**, we see a different picture in the results due to this random linkage. Any increase in overlap for the tested codomains (deceptive trap and random) results in a decrease in difficulty (in terms of first hitting time and effectiveness). Any increase in branching does so as well for the deceptive trap codomain, for high overlap settings. Due to its random linkage, it does not learn the structure of the problems and therefore mixes less effectively then normal LT-GOMEA. This absence of learning does mean, however, that the effect of not being able to learn or represent the structure is not a factor affecting the results with increasing overlap. Therefore, the fact that being able to learn or represent the structure becomes less important and the deceptiveness decreasing make the problems easier with increasing overlap.

    Random linkage LT-GOMEA performs worse than LT-GOMEA in all tests, unsurprisingly, due to the linkage learning leading to more effective mixing for LT-GOMEA.

  - For univariate FOS GOMEA, **U-GOMEA**, the differences between overlap values and branching factor values are very similar to that of random linkage LT-GOMEA. U-GOMEA performs equal to or worse than Random linkage LT-GOMEA in our tests, due to the mixing with blocks of size > 1 leading to more effective mixing.

- The effect of the overlap o on the difficulty of the problems is greater for the deceptive trap codomain than for the random codomain, due to the deceptiveness decreasing as well.

12.3 Future work

We think our results show that the TD Mk Landscape certainly is an interesting benchmark, due to its many unexplored properties, while our background knowledge chapters show its convenience. We hope our work has provided ample opportunities to continue the research, and we provide a non-exhaustive list of possible future work areas below:

12.3.1 Global optima & cliques per variable

- We plotted the number of cliques per variable for our experiment on the increasing branching factor, and with increasing branching factor (for $o = 4$), the number of variables that appeared in a lot of variables increased. It is possible that these 'high-profile' variables are essential for finding a global optimum, and could then be regarded as a *backbone*. Therefore, it would be interesting to first check if these backbone variables are indeed essential for the global optima and thus have a specific value they have for all global optima. If this is the case, then we could test this hypothesis by passing LT-GOMEA these global optima values for the backbone variables from the start and seeing if this improves the performance of LT-GOMEA by much. If the backbone is indeed essential to find a global optimum, and if the problem is easily solved when the backbone is correctly set, then the landscape might have lost a lot of its deceptiveness. Additionally, it would be interesting to report for the problems with a lot of global optima, in how many cliques the variables appear that differ between the global optima. Especially for problems that have a number of global optima equal to a power of 2 (most of them), and a max hamming distance between the global optima equal to $\log_2$ of the number of global optima (also most of them), as the variables that differ between the global optima then all have two maximizing values for all global optima constructed so far. This could provide a start for the analysis of why it is possible that there are so many global optima.

- As mentioned in the previous item, it would be interesting to analyze what exactly causes the difference in the number of global optima for the deceptive trap codomain and how one can make a high number of global optima appear. This is relevant for codomains such as deceptive trap that even for $b = 2$ start to have a high number of global optima for $o = 4$ and $N > 200$ (see Figure 11.1). It would furthermore be of interest to analyze the problems with a lot of global optima further to measure exactly how often the number of problems is a power of 2.

- To provide an insight into (part of) the mechanisms under the hood when the overlap is increased, the distribution of cliques per variable for overlap value $o \neq 4$ could be plotted, as an extension to our current $o = 4$. This could provide an insight especially in combination with the previously mentioned option to investigate the presence of a *backbone*. Additionally, the relation between the distribution and the number of global optima could be studied.

- Very closely related to the previous suggestions, one could look into the overlap settings 0 and 4 to see which is more difficult in terms of performance and effectiveness with $b = 6$ and $N > 250$. The previous suggestions could help explain what causes the difference in performance and effectiveness.

12.3.2 Other

- Currently, we have hypothesized that the deceptiveness of the deceptive trap problems decreases due to interference between the subfunctions when the overlap and branching is increased, but an analysis of the deceptiveness is still missing. It would certainly put our hypotheses to the test when the actual deceptiveness of the problems is analyzed with increasing overlap and branching.

- We did just two experiments on the random codomain, and it would certainly be interesting to perform many of the same experiments as we did for the deceptive trap codomain. Therefore, one could test U-GOMEA as well to test our hypotheses on random linkage LT-GOMEA's results. Furthermore, one could increase the branching factor b to compare our hypotheses on the effect of increasing o and b with the actual results.

- Our experiments used the population sizing-free scheme for LT-GOMEA to get rid of the population size parameter, however, it remains of interest to analyze the scaling of the minimum required population size with increasing N. Therefore, one future work opportunity would be to use a fixed population size for LT-GOMEA during each run.

- The current CliqueTreeMk algorithms are designed to use fixed k, o, b values, however, the definition of TD Mk Landscape does not require fixed values, so one could extend the algorithm to allow for more possible problems. Then, the algorithms would allow for truly any Tree Decomposition Mk Landscape to be generated.

Bibliography

[1] Peter A N Bosman and Dirk Thierens. “Linkage Neighbors , Optimal Mixing and Forced Improvements in Genetic Algorithms Categories and Subject Descriptors”. In: *Gecco 2012* (x 2012), pp. 585–592.

[2] Peter A.N. Bosman, Ngoc Hoang Luong, and Dirk Thierens. “Expanding from discrete cartesian to permutation Gene-Pool Optimal Mixing Evolutionary Algorithms”. In: *GECCO 2016 - Proceedings of the 2016 Genetic and Evolutionary Computation Conference* (2016), pp. 637–644. DOI: `10.1145/2908812.2908917`.

[3] Peter A.N. Bosman and Dirk Thierens. “More concise and robust linkage learning by filtering and combining linkage hierarchies”. In: *GECCO 2013 - Proceedings of the 2013 Genetic and Evolutionary Computation Conference* (2013), pp. 359–366. DOI: `10.1145/2463372.2463420`.

[4] Yves Crama, Pierre Hansen, and Brigitte Jaumard. “The basic algorithm for pseudo-Boolean programming revisited”. In: *Discrete Applied Mathematics* 29 (2-3 1990), pp. 171–185. ISSN: 0166218X. DOI: `10.1016/0166-218X(90)90142-Y`.

[5] Andreas WM Dress. “On the computational complexity of composite systems”. In: *Fluctuations and Stochastic Phenomena in Condensed Matter*. Springer, 1987, pp. 377–388.

[6] Tobias van Driessel and Dirk Thierens. “Benchmark Generator for TD Mk Landscapes”. In: *2021 Genetic and Evolutionary Computation Conference Companion*. GECCO ’21. Association for Computing Machinery, 2021, pp. 1227–1233.

[7] Yong Gao and Joseph Culberson. “An analysis of phase transition in NK landscapes”. In: *Journal of Artificial Intelligence Research* 17 (2002), pp. 309–332.

[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] PL Hammer, I Rosenberg, and S Rudeanu. “Application of discrete linear programming to the minimization of Boolean functions”. In: *Rev. Mat. Pures Appl* 8 (1963), pp. 459–475.

[10] Georges R Harik and Fernando G Lobo. “A parameter-less genetic algorithm.” In: *GECCO*. Vol. 99. 1999, pp. 258–267.

[11] Petru L Ivanescu, Sergiu Rudeanu, and Peter L Hammer. *Boolean methods in operations research and related areas.* 1968.

[12] Stuart A Kauffman et al. *The origins of order: Self-organization and selection in evolution.* Oxford University Press, USA, 1993.

[13] 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.

[14] Daphne Koller and Nir Friedman. “Structured Probabilistic Models: Principles and Techniques”. In: *MIT Press. To appear* 48 (2009), pp. 54–61.

[15] Ngoc Hoang Luong, Han La Poutré, and Peter AN Bosman. “Exploiting linkage information and problem-specific knowledge in evolutionary distribution network expansion planning”. In: *Evolutionary computation* 26.3 (2018), pp. 471–505.

[16] 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.

[17] Martin Pelikan et al. "Performance of evolutionary algorithms on NK landscapes with nearest neighbor interactions and tunable overlap". In: *Proceedings of the 11th Annual Genetic and Evolutionary Computation Conference, GECCO-2009* (May 2014 2009), pp. 851–858. DOI: 10.1145/1569901.1570018.

[18] Dirk Thierens. "The linkage tree genetic algorithm". In: *Lecture Notes in Computer Science* 6238 LNCS (PART 1 2010), pp. 264–273. ISSN: 03029743. DOI: 10.1007/978-3-642-15844-5_27.

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

[20] Dirk Thierens and Tobias van Driessel. "A Benchmark Generator of Tree Decomposition Mk Landscapes". In: *Proceedings of the Genetic and Evolutionary Computation Conference 2021.* GECCO '21. Association for Computing Machinery, 2021, pp. 229–230.

[21] Edward D. Weinberger. "NP Completeness of Kauffman's N-k Model, A Tuneable Rugged Fitness Landscape". In: *Santa Fe Institute Working Papers* 96-02-003 (Feb. 1996).

[22] Darrell Whitley. "Mk landscapes, NK landscapes, MAX-kSAT: A proof that the only challenging problems are deceptive". In: *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation.* 2015, pp. 927–934.

[23] L Darrell Whitley, Francisco Chicano, and Brian W Goldman. "Gray box optimization for Mk landscapes (NK landscapes and MAX-kSAT)". In: *Evolutionary computation* 24.3 (2016), pp. 491–519.

[24] Alden H Wright, Richard K Thompson, and Jian Zhang. "The computational complexity of NK fitness functions". In: *IEEE Transactions on Evolutionary Computation* 4.4 (2000), pp. 373–379.

Appendix A

Additional Tables & Figures

A.1 Global Optima

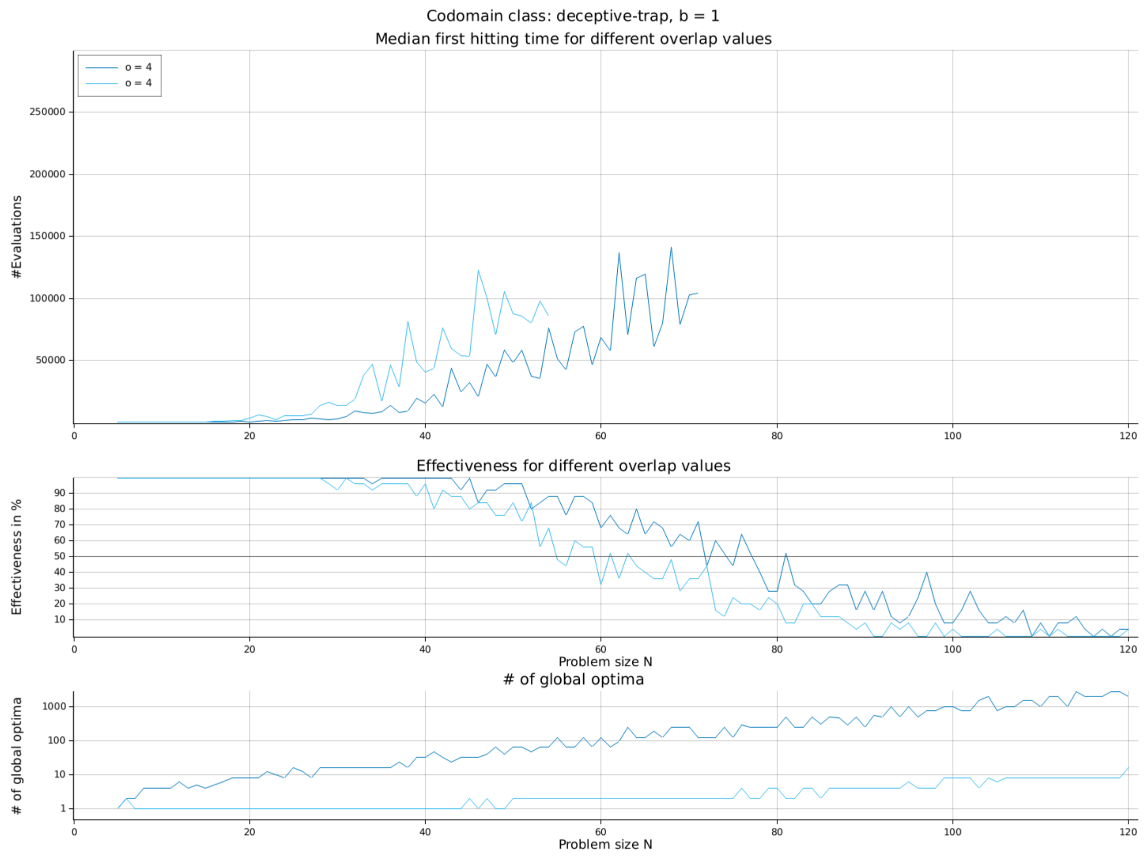


Figure A.1: Performance and effectiveness of random linkage LT-GOMEA for high/low number of global optima

#opt.	high
low	**38/50**

Table A.1: Statistical results for the differences in mean values between problems with either a low or high number of global optima, for random linkage LT-GOMEA, $o = 4$, $b = 1$. H_1: Problems with a *high* number of global optima have a **lower** first hitting time than problems with a *low* number of global optima, for an otherwise equal configuration.

A.2 Increasing Overlap and Branching

A.2.1 Random Linkage LT-GOMEA

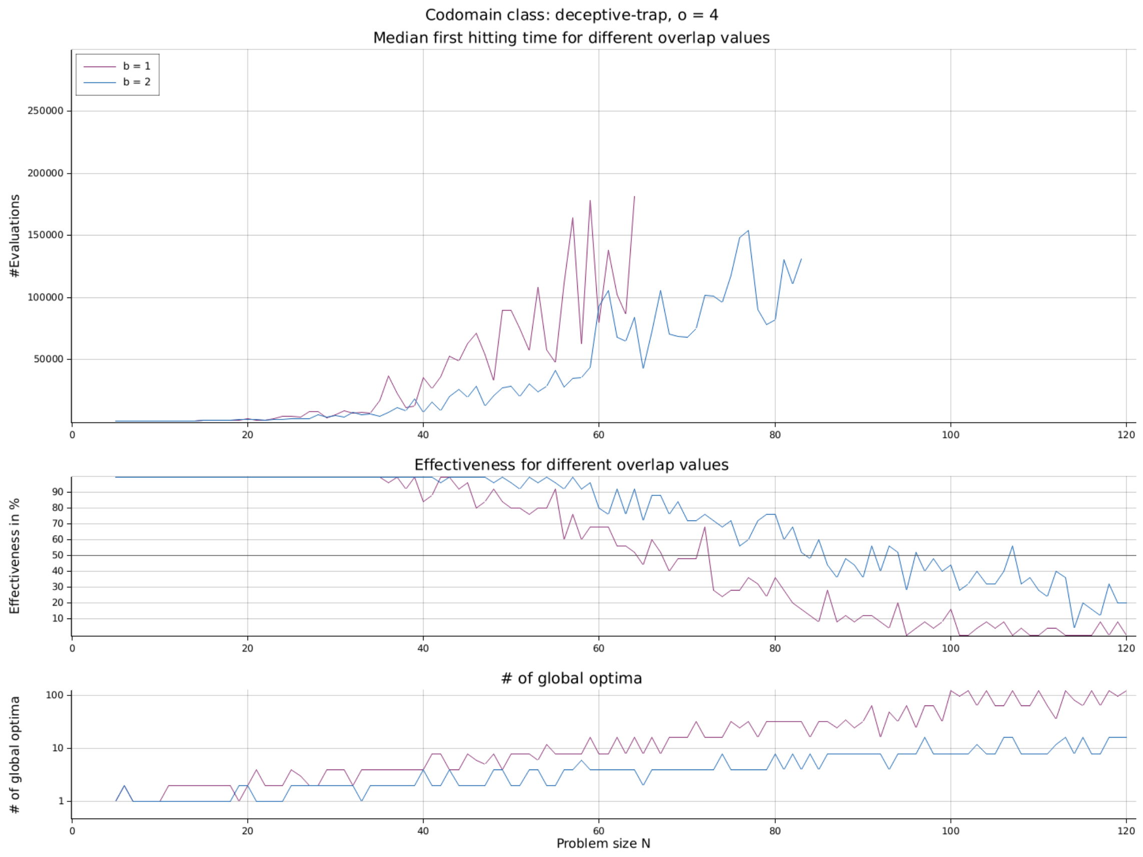


Figure A.2: Performance and effectiveness of random linkage LT-GOMEA for overlap value $o = 4$ with different branching values: $b \in \{1, 2\}$.

A.2.2 U-GOMEA

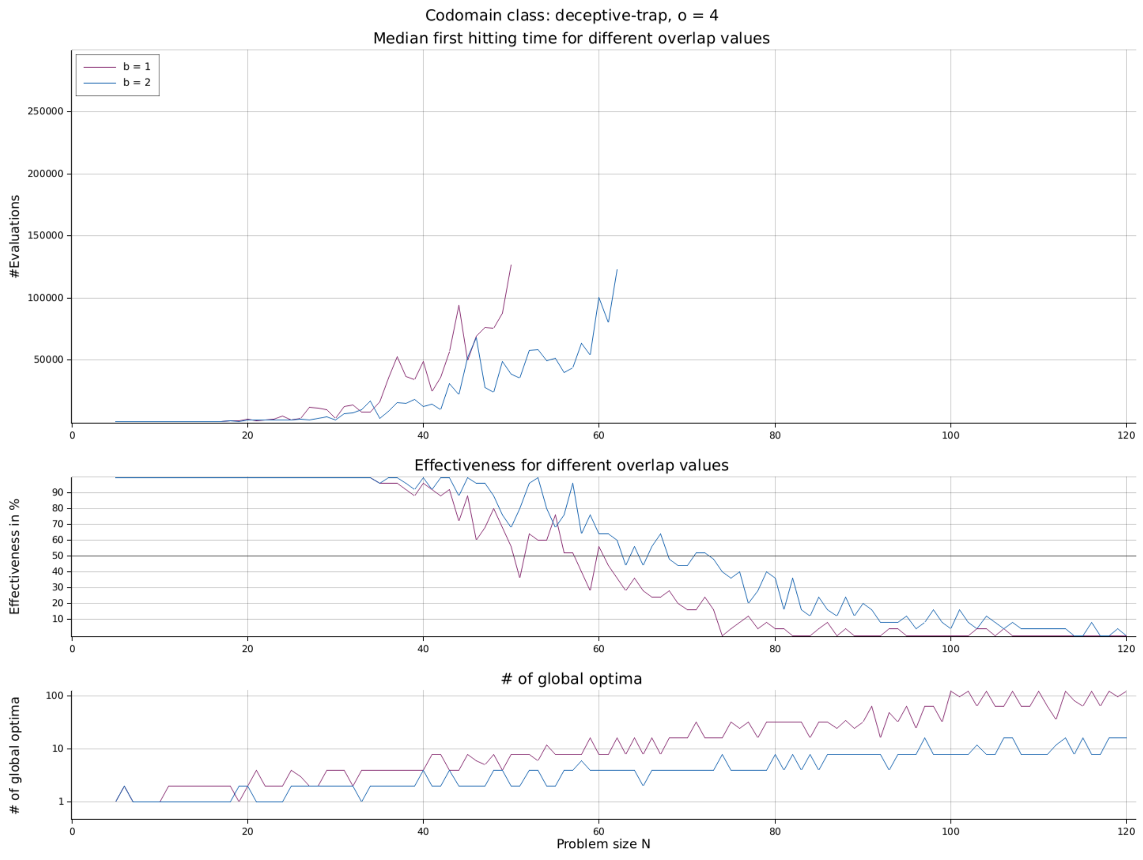


Figure A.3: Performance and effectiveness of U-GOMEA for overlap value $o = 4$ with different branching values: $b \in \{1, 2\}$.

Appendix B

GECCO '21 Workshop Paper

Workshop paper 'Benchmark Generator for TD Mk Landscapes' @ Analysing Algorithmic Behaviour of Optimisation Heuristics [6].

Benchmark Generator for TD Mk Landscapes

Tobias van Driessel
Utrecht University
Utrecht, Netherlands
tobiasvandriessel@startmail.com

Dirk Thierens
Utrecht University
Utrecht, Netherlands
d.thierens@uu.nl

ABSTRACT

We introduce a publicly available benchmark generator for Tree Decomposition (TD) Mk Landscapes. TD Mk Landscapes were introduced by Whitley et al. to get rid of unnecessary restrictions of Adjacent NK Landscapes while still allowing for the calculation of the global optimum in polynomial time. This makes TD Mk Landscapes more lenient while still being as convenient as Adjacent NK Landscapes. Together, these properties make it very suitable for benchmarking blackbox algorithms. Whitley et al., however, introduced a construction algorithm that only constructs Adjacent NK Landscapes. Recently, Thierens et al. introduced an algorithm, CliqueTreeMk, to construct any TD Mk Landscape and find its optimum. In this work, we introduce CliqueTreeMk in more detail, implement it for public use, and show some results for LT-GOMEA on an example TD Mk Landscape problem. The results show that deceptive trap problems with higher overlap do not necessarily decrease performance and effectiveness for LT-GOMEA.

CCS CONCEPTS

• **Computing methodologies → Heuristic function construction**.

KEYWORDS

Benchmarking, Decomposable Landscapes, Dynamic Programming,

ACM Reference Format:
Tobias van Driessel and Dirk Thierens. 2021. Benchmark Generator for TD Mk Landscapes. In *2021 Genetic and Evolutionary Computation Conference-Companion (GECCO '21 Companion), July 10–14, 2021, Lille, France.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3449726.3463177

1 INTRODUCTION

Suitable benchmark functions are vital to test the effectiveness and performance of evolutionary algorithms. Ideally, these benchmark functions should be completely understood in the sense that we know their structure and, importantly, their global optimum (or optima) so that we can check if a given EA has actually found the best possible solution. A problem with designing benchmark functions is that for many interesting problem classes it is not possible to compute the global optimum efficiently. Not knowing whether an EA has found the best solution limits the practical use of the benchmark and only allows relative comparisons between different algorithms - or different parameter settings of a given algorithm - but it does not allow to evaluate the overall performance and effectiveness. For example, [6] propose an interesting class of benchmark functions, but unfortunately there is no way to efficiently compute the global optimum. Similarly, the well known NK Landscapes does not allow to compute the global optimum. For this reason, EA researchers often use the Adjacent NK Landscapes where the interaction between the variables is limited to adjacent problem variables, allowing the use of dynamic programming to compute the global optimum.

NK Landscapes form a subset of k-bounded pseudo-Boolean optimization problems due to its additional constraints: the number of subfunctions is equal to the number of variables N (= problem size), and every subfunction f_i contains variable x_i and K neighbours, thus setting the subfunction size k to $k = K + 1$. For NK Landscapes, these neighbours are K random variables, and for Adjacent NK Landscapes, these neighbours are the subsequent K variables.

Although (Adjacent) NK Landscapes are popular as a benchmark for optimization algorithms, its constraints are unnecessary for most benchmark purposes, as they turn out not to be important for most fundamental theoretical properties of NK Landscapes [9]. Whitley et al.[10] therefore recently introduced the term *Mk Landscapes* to refer to any k-bounded pseudo-Boolean optimization problem, thus a generalization of NK Landscapes without these constraints. Additionally, they introduced the term *Tree Decomposition Mk Landscapes* to refer to any Mk Landscape with a known and bounded tree-width of k. This is a generalization of Adjacent NK Landscapes, as Adjacent NK Landscapes control tree-width by only considering adjacent variables for the subfunctions, but this constraint can be loosened to allow for any Mk Landscape that still has a bounded tree-width. Ultimately, this bounded tree-width is the key to calculate the global optimum (or optima) in polynomial time.

Conveniently, the overall performance and effectiveness of algorithms can be evaluated due to this polynomial time global optimum calculation. And although the global optimum is known, black box algorithms do not know the problem structure and global optimum, and therefore linkage learning will be necessary for particular codomains to find the global optimum reliably and efficiently. The possibility of evaluating the performance and effectiveness of algorithms, together with the difficulty of Tree Decomposition (TD) Mk Landscapes for particular codomains (for blackbox algorithms), make TD Mk Landscapes well suited as a benchmark function for blackbox Genetic Algorithms. As the global optimum can be calculated efficiently by a dynamic programming algorithm, TD Mk Landscapes are not suitable in the context of graybox algorithms, however, as these do know the problem structure.

GECCO '21 Companion, July 10–14,2021, Lille, France
© 2021 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-8351-6/21/07…$15.00
https://doi.org/10.1145/3449726.3463177

In their work, Whitley et al.[10] introduced a construction algorithm to construct TD Mk Landscapes, however, it only constructs TD Mk Landscapes for which the subfunctions form a chain, much like an Adjacent NK Landscape. Recently, Thierens et al.[8] introduced an algorithm, CliqueTreeMk, to construct any TD Mk Landscape and calculate its global optimum (or optima) using dynamic programming when its codomain values are known. In this work, we introduce CliqueTreeMk in more detail, introduce a benchmark generator that implements the algorithm and is available on GitHub, and show indicative results for the Linkage Tree Gene-pool Optimal Mixing Evolutionary Algorithm (LT-GOMEA)[3][1], a linkage learning blackbox evolutionary algorithm. These contributions aim to provide a better understanding of the CliqueTreeMk algorithm, let researchers use our implementation to generate TD Mk Landscapes and benchmark their algorithms, and show that TD Mk Landscapes could be of interest to benchmark blackbox algorithms.

2 TREE DECOMPOSITION MK LANDSCAPES

Whitley et al.[9] recently introduced the term *Mk Landscapes* to refer to any k-bounded pseudo-Boolean optimization problem, a generalization of NK Landscapes without the unnecessary constraints ($M = N, k = K+1$, and variable x_i must appear in subfunction f_i). M is the number of subfunctions and k is a constant that provides an upper bound on the interaction order size of the subfunctions, with M polynomial in N. In a later work, Whitley et al.[10] introduced Tree Decomposition Mk Landscapes, a generalization of Adjacent NK Landscapes: *Tree Decomposition Mk Landscapes* refer to any Mk Landscape with a known and bounded tree-width of k. Tree Decomposition (TD) Mk Landscapes focus on the key property to allow for the global optimum be calculated in polynomial time ($O(N \cdot 2^{2K})$ with Hammer's algorithm[5][4]); a tree decomposition with bounded and known tree-width k must be constructable from (the Variable Interaction Graph of) the Mk Landscape, with $k \in O(\log N)$. TD Mk Landscapes can be expressed by

$$f(x) = \sum_{i=1}^{M} f_i(x, C_i)$$

where $x \in X$, X represents the set of solutions over a bit string with length N, M is the number of subfunctions, f_i is the ith subfunction, and C_i is the ith subset of problem variables that form the input of f_i.

Whitley et al.[10] introduced a construction algorithm to construct TD Mk Landscapes, however, it limits the output to TD Mk Landscapes with a chain-like tree decomposition, similar to the structure of Adjacent NK Landscapes. It is therefore still limited and can not construct all TD Mk Landscapes.

It constructs a $M \times k$ matrix, where the rows correspond with the subfunctions and their variables. The variables must appear in contiguous rows and all N variables must appear in at least one row. If constructed in this way, a tree decomposition can be made with tree-width $k-1$, where every row of the matrix is represented by a node in the tree.

3 CLIQUE TREE MK

To construct any TD Mk Landscape and calculate its global optimum, Thierens et al.[8] introduced the CliqueTreeMk algorithm. First it constructs a TD Mk Landscape and then uses the structure of the generated landscape to calculate its global optimum (or optima) efficiently.

In the context of this algorithm, we use the term clique tree rather than tree decomposition, as it makes heavily use of the concepts of cliques and separators. The output of Whitley's construction algorithm could then be regarded as a clique *chain* rather than a clique *tree*. We use the term *clique* to represent the set of problem variables in a subfunction, and the term *separator* to represent the set of overlapping problem variables between two cliques/subfunctions, as these terms reflect their properties in a clique tree/tree decomposition in a succinct manner.

The idea behind CliqueTreeMk's construction algorithm is to construct the TD Mk Landscape by directly generating a clique tree with the exact properties as required by the input topology parameters, in order to ensure that a clique tree with the required properties can be constructed, which is required by the definition of TD Mk Landscapes. Its input topology parameters are the number of subfunctions/cliques M, number of variables per subfunction/clique k, number of overlapping bits between subfunctions/cliques o, and branching factor b. The branching factor represents the number of branches in the clique tree. The problem length N can be represented by $N = (M-1) \cdot (k-o) + k$, as the first clique/subfunction takes k variables, and every other clique/subfunction overlaps o variables with another clique/subfunction and adds $k-o$ unused variables to get to length k.

The general idea of CliqueTreeMk's *construction* algorithm is to first construct clique C_0 as the root of the clique tree by assigning the first k variables from the shuffled variable list, and then generate b children cliques ($C_{j \in \text{children}_i}$) for every clique C_i until we have constructed M cliques. Each child C_j overlaps with its parent C_i for o variables, described by the separator S_j between C_i and C_j, and the remaining $k-o$ variables are taken from the shuffled variable list to complete C_j.

The *global optimum dynamic programming* algorithm then uses this clique tree structure with its cliques and separators to calculate the global optimum. It is comparable to Pelikan's[7] dynamic programming approach in the way it stores the $k-o$ remaining variables's maximizing values for the values of the o overlapping variables (separator variables). Starting at the leaves of the tree, for each separator S_j we store for each of the instances of the separator variables the maximizing variable values for its child clique C_j and the resulting score. Then, we can iterate in the reverse direction and assign values to the clique variables in C_j based on the maximizing values for its variables stored in its parent separator S_j.

We illustrate the CliqueTreeMk algorithm during these phases using an example instance with number of subfunctions/cliques $M = 7$, subfunction/clique size $k = 3$, and overlap $o = 2$. Together, these define length $N = 9$. Furthermore, we choose a branching factor $b = 2$. The construction algorithm uses fixed values for k, o, and b, but the algorithm can be extended to allow for non-fixed values during construction. Likewise for the dynamic programming algorithm. The variables are randomly ordered: $(x_4, x_2, x_7, x_5, x_1, x_9, x_3, x_8, x_6)$.

3.1 Construction

The algorithm is described in a textual version below and a pseudocode version in Algorithm 1.

(1) Initially, take the first k variables as the root clique C_0. Otherwise, take the next clique C_i to expand.
(2) Choose o random variables from parent clique C_i, assign to separator S_j
(3) Take next $(k-o)$ not chosen variables and add the variables from S_j to construct child clique C_j
(4) Go to step 2 until b branches have been built
(5) Go to 1 to expand the next clique

Algorithm 1: CliqueTreeMk Construction

```
Input: $M$, $k$, $N$, $b$, $o$, shuffled list of variables
Result: Clique tree
$C_0 \longleftarrow$ first $k$ variables;
$count \longleftarrow 1$;
for $i \longleftarrow 0$ to $M-2$ do
    for $j \longleftarrow 0$ to $b-1$ do
        $S_{count} \longleftarrow o$ random variables from clique $C_i$;
        $x \longleftarrow$ next $(k-o)$ unused variables;
        $C_{count} \longleftarrow S_{count} \cup x$;
        $count \longleftarrow count + 1$;
        if $count == M$ then
            return clique tree;
    end
end
```

Following the algorithm with the given example instance could result in the following list of cliques: (x_4, x_2, x_7), (x_4, x_7, x_5), (x_4, x_2, x_1), (x_7, x_5, x_9), (x_4, x_7, x_3), (x_4, x_1, x_8), (x_2, x_1, x_6)
In Figure 1 we illustrate the constructed clique tree with its separators.

Essentially, the algorithm creates a clique tree / tree decomposition that adheres to the given constraints, defined by the input topology parameters. Importantly, it adheres to the running intersection property, as problem variables are either part of a single clique C_i or part of multiple cliques that are directly connected by separators. This follows from steps 2 and 3 of the textual version: During construction of a clique C_j, each variable is either taken from the unused problem variables list or copied from the parent clique C_i (and added to the separator S_j), with C_j being a child of C_i. The dynamic programming algorithm that calculates the global optimum requires this running intersection property to select the best value for variables in isolation: for $k-o$ variables at every clique and for o variables at every separator. It is able to calculate the global optimum in polynomial time due to the bounded (and known) tree-width.

3.2 Global Optimum Dynamic Programming Algorithm

To explain the dynamic programming algorithm, we first introduce it in a textual form and then we introduce it in more detail using some formulas.

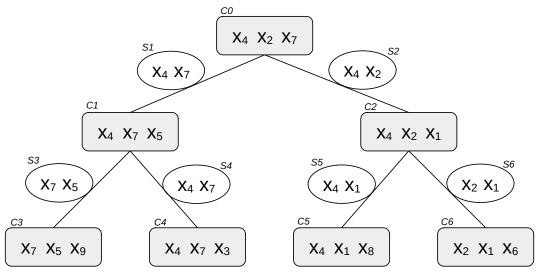


Figure 1: Example clique tree with cliques C0 to C6 and separators S1 to S6.

The CliqueTreeMk global optimum solver follows very similar steps to the dynamic programming algorithm by Pelikan et al.[7]. The CliqueTreeMk global optimum solver traverses the clique tree from the leaves to the root, storing for each instance of separator S_i (o overlapping bits) the maximizing values for the $k-o$ variables in $C_i \setminus S_i$ with its score. The maximizing values for $C_i \setminus S_i$ are stored in K_i and the accompanying score is stored in h_i. Then, for each possible instance of the clique root C_0, the best achievable score g_0 is calculated using its children separators S_j and the stored best achievable score in h_j for that instance of the separator variables. The highest score of these possible instances is the global optimum (or global optima). To assemble the global optimum solution, C_0's maximizing instance is written to the solution and the clique tree is traversed from the root to the leaves, storing the maximizing values for the $k-o$ variables from each K_i into the solution.

If there are multiple global optima, then there are multiple maximizing instances for one or more separators S_i. Each of these maximizing instances for S_i is stored in K_i. When one of these cases of multiple maximizing instances is encountered during the assembly of the global optima, the current global optimum is copied a number of times, according to the number of maximizing instances in K_i (minus one). Finally, each of these copies is assigned one of the maximizing instances and the traversal of the clique tree is continued. Each of these global optima solutions is now considered at every remaining separator in the clique tree.

More specifically, we can define $\forall$ separators S_i:
$h_i(a_1, ..., a_o) = g_i(a_1, ..., a_o, a^*_{o+1}, ..., a^*_k)$ with
$a_1, ..., a_o \in S_i$, $a_{o+1}, ..., a_k \in C_i \setminus S_i$ and $a^*_{o+1}, ..., a^*_k$ maximizing g_i for values $a_1, ..., a_o$.
$K_i(a_1, ..., a_o) = \{a^*_{o+1}, ..., a^*_k\}$
And $\forall$ cliques C_i:
$g_i(a_1, ..., a_k) = f_i(a_1, ..., a_k) + \sum_{j \in \text{children}_i} h_j(b_1, ..., b_o)$

To illustrate these, we can define the previous specifically for our example instance. We define $\forall$ separators S_i:
$h_i(x_a, x_b) = g_i(x_a, x_b, x^*_c)$ with $x_a, x_b \in S_i$ and $x_c \in C_i \setminus S_i$ and x^*_c maximizing g_i for x_a and x_b values.
$K_i(x_a, x_b) = \{x^*_c\}$
And $\forall$ cliques C_i: $g_i(x_p, x_q, x_r) = f_i(x_p, x_q, x_r) + h_{child1}(x_p, x_q) + h_{child2}(x_p, x_r)$

Using the above formulas, we can write a shorter version of the algorithm: For every possible instance of the problem variables in C_0, calculate g_0. Calculating g_0 will recursively calculate all the g_i, h_i, and K_i values for $i > 0$. The maximum of these g_0 values is the global optimum of the TD Mk Landscape and can be used to retrieve the bit string that achieves this fitness. This is done by acquiring the stored maximizing values for each separator S_i from K_i and assigning their values to the global optimum solution. Or in a more pseudo code way:

(1) For each possible instance of problem variables in C_0, calculate g_0
(2) Maximum g_0 is global optimum
(3) Take next separator, starting with S_1
(4) Take maximizing values from K_i, for problem variable values already in global optimum solution, and put them in global optimum solution
(5) Go to step 3 to assign all problem variable values

We illustrate the algorithm using the example used in the previous subsection. We use the following deceptive trap function for each subfunction:

$$\begin{aligned} f_i(x_a, x_b, x_c) : 111 &=> 4 \\ 000 &=> 2 \\ \text{otherwise} &=> 2 - c(x_a, x_b, x_c) \end{aligned}$$

where c returns the number of ones in the passed variable values.

We show the calculated h_i and K_i values for S_6 and S_1, as $i \in \{3, 4, 5, 6\}$ have the same h_i and K_i values and likewise for $i \in \{1, 2\}$. Then we show the construction of the global optimum using the calculation of g_0 for C_0.

$$C_6 = \{x_2, x_1, x_6\},\ S_6 = \{x_2, x_1\}$$
$$g_6(x_2, x_1, x_6) = f_6(x_2, x_1, x_6)$$

$S_6 = x_2x_1$	00	01	10	11
$h_6(x_2, x_1)$	2	1	1	4
$K_6 = x_6^*$	0	0	0	1

In the above table, we list the possible instances of the separator variables x_2 and x_1, the maximizing values of the remaining variable x_6 in C_6 for these instances (K_6), and the resulting scores for these maximizing values (h_6). Because C_6 is one of the leaves, g_6 is equal to f_6. We can see the deceptive attractor at work here, attracting any instance of the separator variables that does not contain a part of the local optimum.

$$C_1 = \{x_4, x_7, x_5\},\ S_1 = \{x_4, x_7\}$$
$$g_1(x_4, x_7, x_5) = f_1(x_4, x_7, x_5) + h_4(x_4, x_7) + h_3(x_7, x_5)$$

$S_1 = x_4x_7$	00	01	10	11
$h_1(x_4, x_7)$	$2+2+2$ $= 6$	$0+4+1$ $= 5$	$0+1+4$ $= 5$	$4+4+4$ $= 12$
$K_1 = x_5^*$	0	1	1	1

Because C_1 does have children cliques, the calculation of h_1 and thus of g_1 does involve the h_i values of its children, h_4 and h_3.

$$C_0 = \{x_4, x_2, x_7\},\ S_0 = \emptyset$$
$$g_0(x_4, x_2, x_7) = f_0(x_4, x_2, x_7) + h_2(x_4, x_2) + h_1(x_4, x_7)$$

$x_4x_2x_7$	000		111
$g_0(x_4, x_2, x_7)$	$2+6+6$ $= 14$	...	$4+12+12$ $= 28$

Finally, we calculate the g_0 values for all possible instances of the problem variables in C_0. Here we have illustrated just two cases, instances 000 and 111 for $x_4x_2x_7$. Note that this table differs from the two before in the things we calculate; here we don't calculate h_i values, as there is no separator. Instead, we calculate all g_0 values and record the maximum value as the global optimum (or global optima).

For this example, the global optimum value is 28. The maximizing instance for C_0, while considering the rest of the clique tree using dynamic programming, is $x_4^*x_2^*x_7^* = 111$. We can now traverse the clique tree to assign the other bits of the global optimum solution. First, $S_1 = \{x_4, x_7\}$, as is shown in the table for C_6 / S_6, so we insert the values of x_4 and x_7 from our global optimum solution, which are 1 and 1. For instance $x_4x_7 = 11$, $K_1 = x_5^* = 1$, so we assign value 1 to x_5 in our global optimum solution. After doing this for all separators, our global optimum solution is 111111111.

4 EXAMPLE

Our implementation of the CliqueTreeMk algorithm can be found on GitHub[1], here we show some results with our benchmark generator implementation to illustrate its ease of use. Its main functionality is the generation of problems and the calculation of these problems's global optimum, however, it can also generate some input codomain files for the problem generation. The codomain files generation should make it easy to generate a TD Mk Landscape problem from scratch and benchmark an algorithm with it.

4.1 Problem Generation

The problem generator can take as input a configuration folder, a codomain folder, a configuration file, or a codomain file. Here, we highlight how to use the generator with a configuration file and codomain file, and refer the reader to the documentation for the instructions on how to run the generator with multiple configuration files in a folder or multiple codomain files in a folder.

4.1.1 Configuration Input. We create a configuration file to generate deceptive trap problems with topology parameters in a range, in this case we use $M \in \{1, ..., 49\}$, $k = 5$, $o = 1$, $b = 1$:

```
M 1 50
k 5 6
o 1 2
b 1 2
deceptive-trap
```

As options for the codomain we currently offer: *Random*, *Deceptive Trap*, *NKq*, *NKp*, and *Random Deceptive Trap* (a combination of the two). Here we have chosen the deceptive trap function.

Then we use the executable *problem_generator* to generate the codomain files and the problems (25 for each configuration), and find the global optimum for each problem:

```
problem_generator configuration_file -n 25 FILE
    CODOMAIN_OUT PROBLEM_OUT
```

where `CODOMAIN_OUT` and `PROBLEM_OUT` are the (existing) output codomain folder and output problem folder.

[1]https://github.com/tobiasvandriessel/problem-generator

4.1.2 *Codomain Input.*

Instead of generating the codomain and then generate a problem with this generated codomain, one can use an existing codomain file to create a TD Mk Landscape problem. The executable offers the following subcommand for this purpose:

```
problem_generator codomain_file CODOMAIN_FILE
    PROBLEM_FILE_OUT
```

4.1.3 *Codomain File Structure.*

The input codomain files should have the following structure:

```
M K O B
CODOMAIN_VALUE_1
...
CODOMAIN_VALUE_LAST
```

where M, K, O, and B represent the to be inserted values of M, k, o and b, and CODOMAIN_VALUE_1 ... CODOMAIN_VALUE_LAST represent the $M \cdot 2^k$ decimal codomain values, each on a new line.

4.1.4 *Problem File Structure.*

The output problem files have the following structure:

```
M K O B
GLOB_OPT_VAL
NUM_GLOB_OPT
GLOB_OPT_1
...
GLOB_OPT_LAST
CLIQUE_INDICES_1
...
CLIQUE_INDICES_LAST
```

where GLOB_OPT_VAL represents the global optimum (optima) value, NUM_GLOB_OPT represents the number of global optima, GLOB_OPT_1 ... GLOB_OPT_LAST represent the global optima solutions, and CLIQUE_INDICES_1 ... CLIQUE_INDICES_LAST represent the problem variables in each clique.

An example problem generated:

```
2 5 1 1
1.9
2
101000111
010111000
5 3 2 1 7
1 0 6 4 8
```

4.2 Experiment

To show the potential of the TD Mk Landscape benchmark, we conducted a simple experiment: We generated deceptive trap problems with increasing problem size N and overlap o, and ran the Linkage Tree Gene-pool Optimal Mixing Evolutionary Algorithm (LT-GOMEA) on these generated problems to quantify the effect of this increase in o for the difficulty of the problem. LT-GOMEA is a blackbox algorithm, and thus does not have any problem structure information, that tries to learn the linkages between the problem variables to learn the problem structure. LT-GOMEA has shown state-of-the-art performance for discrete, Carthesian-space optimization problems[2], and should therefore show just how difficult and non-trivial TD Mk Landscapes can be. Because we know the global optimum (or optima) of the generated problems, we can evaluate the overall performance and effectiveness of LT-GOMEA.

Configuration input: $M \in \{m \mid N \leq 150\}$, $k = 5$, $o \in \{0, 1, ..., 4\}$, $b = 2$. Where problem size $N = (m - 1) \cdot (k - o) + k$. Note that preliminary experiments indicate that the branching factor b seems to have a big impact on the number of global optima.

The codomain used for the experiment is the deceptive trap function, where we generate for each subfunction a random bit string of length k to be the local optimum and its inverse to be the deceptive attractor. The local optimum has a score of 1.0, the deceptive attractor has a score of 0.9 and any other bit string has score $0.9 - d \cdot \frac{0.9}{k}$, where d is the hamming distance to the local deceptive attractor.

Per configuration instance we generated 25 problems, and for each of these problems, we ran LT-GOMEA 3 times. For the runs where LT-GOMEA manages to find the global optimum, we record the first hitting time. The first hitting time is the number of function evaluations until the global optimum or one of the global optima was found by the algorithm. To record the first hitting times, we need to ignore any unsuccessful LT-GOMEA runs, as these did not find the global optimum. So, for the 3 runs of LT-GOMEA, we filter out any runs that did not find the global optimum and take the median first hitting time for the remaining successful runs. Then we take the median value from the median first hitting times for the 25 generated problems, where again any runs that did not find the global optimum were filtered out. This median value is recorded together with the problem size of the configuration. Besides this first hitting time, we record the effectiveness of LT-GOMEA for every configuration. We measure the effectiveness by counting the number of problems out of 25 (for the current configuration) for which at least 1 LT-GOMEA run found the global optimum, or one of the global optima in case the fitness function has multiple global optima (note that LT-GOMEA is not designed to be a multi-modal EA, so one should not expect it to return all global optima simultaneously). Also note that when we filter out unsuccessful runs in the first hitting time calculation, we still record these unsuccessful runs in the effectiveness for that configuration instance.

We use LT-GOMEA with the population sizing-free scheme as introduced in [1], but we use its discrete cartesian version. LT-GOMEA instance i with population size P_i is run 4 times for every run of instance $i + 1$, with $P_0 = 1$ and $P_{i+1} = 2 \cdot P_i$. The maximum number of running LT-GOMEA instances is 25. We set the No Improvement Stretch (NIS) to $1 + log_{10}(P_i)$, where P_i is the population size of LT-GOMEA instance i. Forced Improvement (FI)[3] is run if the best fitness in a population did not improve for more generations than this NIS. We use premature stopping to stop any LT-GOMEA instance when a LT-GOMEA instance with a bigger population size has a higher average fitness. A LT-GOMEA instance is also stopped when the fitness variance in the population of a LT-GOMEA instance is equal to or smaller than 0.00001. When the global optimum score is found, we stop execution (of all LT-GOMEA instances) and record the current number of evaluations as the first hitting time. Finally, we also stop execution as soon as we hit 300,000 evaluations, with every partial evaluation counting as an evaluation as well.

The results are shown in Figure 2; the upper graph shows the first hitting time for increasing values of the problem size N, with a line per overlap o setting, and the lower graph shows the effectiveness. When the effectiveness of an overlap configuration decreases below 50%, it is not considered reliable anymore and therefore its performance is not plotted anymore in the upper graph. To emphasize this decision, we have highlighted the 50% effectiveness mark in the lower graph with a horizontal line.

We hypothesised that the problems would become more difficult to solve for LT-GOMEA with increasing overlap o, however, the results paint a different picture. Problems with overlap 1 and 2 do require more evaluations and have a lower effectiveness than problems without any overlap, and are very close in both the required number of evaluations as well as the effectiveness. Interestingly, problems with overlap setting 3 have a lower number of required evaluations and a higher effectiveness than overlap settings 1 and 2, so one could regard problems with overlap 3 as easier. Likewise, problems with an overlap of 4 variables require less evaluations and have a higher effectiveness than problems with overlap setting 3. Finally, and perhaps most surprising of all, the results show that problems with overlap 4 are solved using fewer evaluations than problems with overlap 0, for problem sizes $N \leq 60$. Importantly, however, problems with overlap 0 are always solved, whereas problems with overlap 4 are not.

5 CONCLUSIONS

This paper aimed 1) to provide a better understanding of the CliqueTreeMk algorithm, 2) let researchers use our implementation to generate TD Mk Landscapes and benchmark their algorithms, and 3) show that TD Mk Landscapes could be of interest to benchmark blackbox algorithms.

First, we introduced the CliqueTreeMk benchmark generator for TD Mk Landscapes by Thierens et al.[8] in more detail. We have shown the main difference with the construction algorithm by Whitley et al.[10]: it is able to construct TD Mk Landscapes with a clique *tree* rather than a clique *chain*, due to the branching factor configuration parameter b. With this branching factor, it is able to construct any TD Mk Landscape with fixed k, o, and b. In the example section, we have illustrated the usage of our implementation of CliqueTreeMk, which is publicly available on GitHub and designed to be easy to use for researchers. Finally, we have reported on a simple experiment to show the variation in difficulty one can already achieve with the change of one parameter of the TD Mk Landscape; the number of overlapping bits between subfunctions o. The results show that the performance and effectiveness of LT-GOMEA do not necessarily decrease with increasing overlap o for deceptive trap problems.

By varying the codomain of the landscape, multiple types of problems can be created. TD Mk Landscapes are well suited to serve as benchmark functions for blackbox Genetic Algorithms that are not given the structural problem information as specified by the clique tree. Specifically, for particular codomains - including deceptive functions - linkage learning techniques will be necessary to be able to find the global optima reliably and efficiently. Experimental studies of genetic algorithms greatly benefit from the availability of suitable and well understood benchmark functions.

In the future, one might consider extending the CliqueTreeMk algorithm for variable (but bounded) k, o, and b. If implemented, this would then allow for truly any TD Mk Landscape to be generated and its optimum calculated.

REFERENCES

[1] Peter A.N. Bosman, Ngoc Hoang Luong, and Dirk Thierens. 2016. Expanding from discrete cartesian to permutation Gene-Pool Optimal Mixing Evolutionary Algorithms. *GECCO 2016 - Proceedings of the 2016 Genetic and Evolutionary Computation Conference* (2016), 637–644. https://doi.org/10.1145/2908812.2908917

[2] Peter A.N. Bosman and Dirk Thierens. 2013. More concise and robust linkage learning by filtering and combining linkage hierarchies. *GECCO 2013 - Proceedings of the 2013 Genetic and Evolutionary Computation Conference* (2013), 359–366. https://doi.org/10.1145/2463372.2463420

[3] Peter A N Bosman and Dirk Thierens. 2012. Linkage Neighbors , Optimal Mixing and Forced Improvements in Genetic Algorithms Categories and Subject Descriptors. *Gecco 2012* x (2012), 585–592.

[4] Yves Crama, Pierre Hansen, and Brigitte Jaumard. 1990. The basic algorithm for pseudo-Boolean programming revisited. *Discrete Applied Mathematics* 29, 2-3 (1990), 171–185. https://doi.org/10.1016/0166-218X(90)90142-Y

[5] PL Hammer, I Rosenberg, and S Rudeanu. 1963. Application of discrete linear programming to the minimization of Boolean functions. *Rev. Mat. Pures Appl* 8 (1963), 459–475.

[6] Kei Ohnishi, Shota Ikeda, and Tian-Li Yu. 2020. A Test Problem with Difficulty in Decomposing into Sub-Problems for Model-Based Genetic Algorithms. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion* (Cancún, Mexico) *(GECCO '20)*. Association for Computing Machinery, New York, NY, USA, 221–222. https://doi.org/10.1145/3377929.3389993

[7] Martin Pelikan, Kumara Sastry, David E. Goldberg, Martin V. Butz, and Mark Hauschild. 2009. Performance of evolutionary algorithms on NK landscapes with nearest neighbor interactions and tunable overlap. *Proceedings of the 11th Annual Genetic and Evolutionary Computation Conference, GECCO-2009* (2009), 851–858. https://doi.org/10.1145/1569901.1570018

[8] Dirk Thierens and Tobias van Driessel. 2021. A Benchmark Generator of Tree Decomposition Mk Landscapes. In *Proceedings of the Genetic and Evolutionary Computation Conference 2021 (GECCO '21)*. Association for Computing Machinery, 229–230.

[9] Darrell Whitley. 2015. Mk landscapes, NK landscapes, MAX-kSAT: A proof that the only challenging problems are deceptive. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. 927–934.

[10] L Darrell Whitley, Francisco Chicano, and Brian W Goldman. 2016. Gray box optimization for Mk landscapes (NK landscapes and MAX-kSAT). *Evolutionary computation* 24, 3 (2016), 491–519.

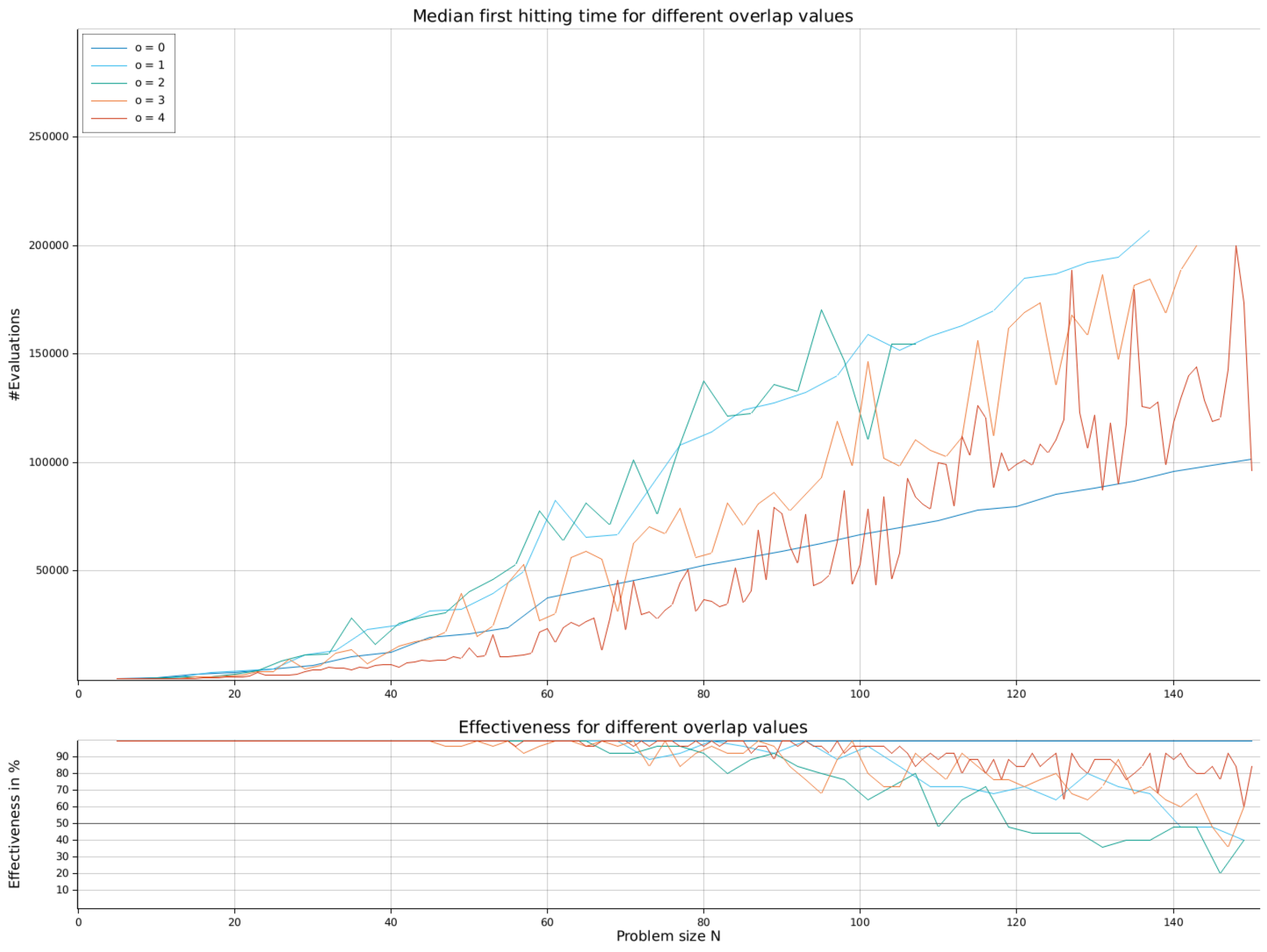


Figure 2: Performance and effectiveness of LT-GOMEA for different overlap values