Utrecht University

Department of Information and Computing Sciences

Master Thesis Computing Science

Analyzing the Performance of the Linkage Tree Gene-pool Optimal Mixing Evolutionary Algorithm on Tree Decomposition Mk Landscapes using Local Optima Networks

Author
Erik van den Hoogen

Supervisor
dr. ir. D. Thierens

January 11, 2023

Abstract

This thesis analyses the performance of Linkage Tree Gene-pool Optimal Mixing Evolutionary Algorithm (LT-GOMEA) on Tree Decomposition Mk Landscapes for problems generated by the benchmark function CliqueTreeMk introduced by Thierens et al. [16]. To understand the challenges faced by LT-GOMEA on these landscapes, the search process is visualized using Local Optima Networks (LONs) introduced by Ochoa et al. [13]. Problems are generated in both a *random* and *deceptive-trap* codomain, and the input parameters for CliqueTreeMk are varied to create problems with different characteristics. The main parameter being varied is the overlap between subfunctions, and the number of subfunctions increases as well to maintain a constant problem length. Two implementations of LT-GOMEA are tested: the standard population-based version and a non-population-based version in which the building blocks are learned from a hill-climbed population. For population-based LT-GOMEA, the performance for both codomains increased prior to decreasing with an increase in overlap, but the deceptive-trap codomain performed better at higher overlap. This is thought to be due to a lower number of local optima (than for the random codomain) and decreased deceptiveness with an increase in overlap. The non-population based LT-GOMEA had similar results for the deceptive-trap codomain, but performed poorly on the random codomain with overlap higher than 0, suggesting that an evolving population is necessary to utilize the linkage learning of LT-GOMEA. We also discovered that the intended deceptiveness of the deceptive-trap codomain decreases with overlap, and this could also contribute to the increase in global optima for the deceptive-trap neutral codomain. The findings about the challenges faced by LT-GOMEA are supported by analyzing the search process through the use of LONs. The LONs reveal that problems generated with the same input parameters can vary significantly in difficulty, and gaining a deeper understanding of the reasons behind this variance could improve the effectiveness of CliqueTreeMk as a benchmark function, by being able to more precisely estimate the difficulty of the generated problems.

Contents

Chapter 1

Introduction

This thesis evaluates the performance of LT-GOMEA (Linkage Tree Gene-pool Optimal Mixing Evolutionary Algorithm), a state-of-the-art black-box optimizer developed by Bosman et al. [2], on problems created by the CliqueTreeMk benchmark function introduced by Thierens et al. [16]. The difficulty of these problems will be analyzed in relation to different input parameters. Understanding the difficulty changes caused by varying the input parameters will be beneficial for utilizing CliqueTreeMk as a benchmark function. CliqueTreeMk creates Tree-Decomposition (TD) problems in the category of Mk-Landscapes introduced by Whitley et al. [23].
Gray-box optimizers can be utilized to solve search problems for which the underlying structure is known, but black-box optimizers can also learn the structure of the problem during the search process. When the underlying structure is not provided, gray-box optimizers are not effective. In these situations, black-box optimizers are needed to solve the problem. LT-GOMEA is a population-based black-box optimizer, which learns the structure of a problem instance by recognizing structures within an evolving population. By benchmarking LT-GOMEA on TD Mk Landscape problems, the effectiveness of the algorithm as a black-box optimizer can be evaluated and its suitability for different problem scenarios can be determined. The CliqueTreeMk benchmark function allows the creation of different problem instances with different input parameters. The goal of this thesis is to understand how the difficulty of these problems varies with different input parameters. To understand how LT-GOMEA performs on the presented problems, the search of LT-GOMEA is visualized using LONs (Local optima networks), proposed by Ochoa et al. [13].

To better understand Mk Landscapes, it is helpful to first explain the concept of NK Landscapes. NK Landscapes are popular for benchmarking search algorithms, where a problem instance can be represented by a bitstring of length N. Every variable in the bitstring contributes to one or more subfunctions. The subfunctions contribute to the solution's total fitness, and the fitness per subfunction depends on its variable configuration. The number of subfunctions $M = N$ and the subfunction size $k = K + 1$, where $0 \leq K \leq N - 1$. For adjacent NK landscapes, variable x_i must occur in $subf_i$ [23] and only adjacent variables can be represented in a subfunction. A benefit of adjacent NK landscapes is that the global optimum can be calculated in polynomial time. This is beneficial for benchmarking because solutions of black-box optimizers can be verified with the global optimum. The constraints of NK landscapes are overcome by Mk landscapes proposed by Whitley et al. [23]. The CliqueTreeMk benchmark function is able to generate these Mk landscapes. This allows for more flexibility and a wider range of problem

instances to be created. Additionally, the tree-width is bounded to k, this makes the problem polynomial time solvable with dynamic programming when the TD is known. TD MK landscapes have several parameters that can be used to generate different instances, including tree-width, branching factor, problem size, and overlap between subfunctions. The fitness score for each subfunction configuration is determined by a codomain. In some cases, such as when using deceptive-trap codomains, it can be difficult to solve the problem without using linkage learning to identify important building blocks. When evaluating a search algorithm, simply knowing whether it can find the optimal solution and how many function calls it takes, may not provide sufficient insight into why it succeeds or fails. These metrics do not give a complete picture of the relative difficulty of different problems, because they do not provide information about the internal structure of the problem and how the search algorithm handles these structures. To truly understand the strengths and weaknesses of an algorithm, it is necessary to consider a broader range of information. Ochoa and Verel did a lot of research to reveal these structures, and this thesis is inspired by their concepts, for revealing the difficulties when solving TD Mk Landscapes.

Ochoa et al. [11] came up with techniques to reveal problem structures and search difficulties within NK Landscapes using a best-improvement local search (BILS) algorithm. They make a so-called "Local optima Network" (LON), which is a graph where the nodes represent local optima found by BILS and the edges represent local optima (LOs) with common neighbours (non-optimized solutions). In a later paper, Verel et al. [19] use Iterated Local Seach (ILS) where an edge arises between 2 LOs when a mutation successfully moved to a better LO. A mutation is defined by D bitflips in a bitstring. An Edge connects LO_i with LO_j, when a mutation of LO_i and the optimization by BILS leads to LO_j, where $fitness(LO_i) <= fitness(LO_i)$. Different mutation sizes D can result in different edge configurations for the LON. A LON is a product of the used search algorithm together with the problem. The LON nodes also keep track of how many times they are visited by the search algorithm, this is called the node strength. These strengths can give a lot of insight into the more visited and attractive regions within a LON.

In Ochoa et al. [13] the same concept is used for making LONs for the *Travelings Salesman Problem* (TSP). ILS is used for exploring a LON with multiple runs. Where a run tries to optimize a solution till it did not improve for n times. They use methods for recognizing regions of attractions within a LON, in later chapters these methods will be clarified. Almost inescapable regions form the endpoint of the search and are called a *funnel*. All the visited LOs of a run belong to the same funnel as well as the visited LOs of runs with the same endpoint. A LO can belong to multiple funnels because mutation is random which can lead to different endpoints. When analysing these funnels, observations can be made about the performance of a search algorithm and the difficulty of the problem. For example: when the global optimum is located within a relatively small badly connected funnel the algorithm shall have a difficult time finding the global optimum, but when there is only one funnel that contains the global optimum, finding the global optimum can be simpler.

This thesis aims to test LT-GOMEA on TD MK Landscapes and to reveal and understand the changes in difficulty emerging by changing the CliqueTreeMk input-parameters. A LON will be created for analysing the performance of LT-GOMEA on different TD MK Landscape instances. The analysis will be conducted using a LON (Local Optima Network) that is constructed differently from Ochoa's [13] approach, as it uses a population-based search algorithm. Unlike in their paper, here the state of a generation in the search

algorithm does not consist of a single solution, and the next solution is not determined solely by alterations to a single solution. Therefore, decisions must be made about what the nodes represent in each state of the search algorithm. LONs for Population-based algorithms have been proposed [10][3][18], but these were made for either gray-box optimizers, or the state represented within the nodes is so complex, that the LONs are mainly useful for capturing general trends in difficulty and for comparing different population-based algorithms. This thesis is concerned with one population-based algorithm and tries to understand the difficulty on the level of the used codomain. We will investigate if a LON can tell something about the difficulties for LT-GOMEA, where the nodes represent each best solution of a population within a generation. Besides the regular implementation of LT-GOMEA, it would be interesting to see how LT-GOMEA performs when the building blocks are learned from solely a population of hill-climbed (by use of Best-improvement local search) solutions. The building blocks have to be learned once. These building blocks are then used to improve a single solution, where this solution tries to improve by inheriting the bits belonging to a building block of a random hill-climbed solution out of the population. This search algorithm has a lot in common with ILS, only the alterations are not caused by mutation, but by targeting the learned building blocks. The advantage of this approach is that a LON represents all solutions that made an improvement during the search, which results in a more complete LON than for the first approach of LT-GOMEA. Hopefully, this will contribute to a better understanding of how LT-GOMEA performs on these types of problems.

Chapter 2

Research Questions

In order to fully understand the performance of LT-GOMEA, it is not sufficient to simply know whether it is successful or not. It is also important to investigate how the algorithm searches through the search space, as this can provide insight into why the algorithm may struggle with some problems more than others. The LONs introduced by Ochoa et al. [13], are capable of visualizing the search and hopefully can give a more comprehensive understanding of LT-GOMEA's performance. The LONs are capable of identifying regions of attraction. These regions are important features for understanding search difficulties. When the global optimum is well connected to the largest regions of attraction, this is generally a positive sign for the algorithm's performance. In contrast, if the global optimum is located outside of these regions, it can be more difficult for the algorithm to reach the optimal solution. Additionally, the nodes in a LON can capture how often each solution has been found during multiple searches, providing insight into the reachability of different solutions and the relative difference in the frequency with which they are visited. Although useful, the approach for creating LONs for population-based algorithms is different. This may result in the loss of some information, as decisions must be made about what to include in the LON. As a result, it is interesting to investigate whether LONs can still provide sufficient information to give insight into how population-based algorithms search through the search space.
Not only is this thesis concerned with the performance of LT-GOMEA, but also with how different configurations of CliqueTreeMk influence the search difficulty. In particular, it is interesting how different levels of overlap affect search difficulty. For problems of a constant length, the overlap can be increased either by multiplying the number of subfunctions or by increasing the length of the subfunctions. The latter approach is similar to increasing the ruggedness of NK landscapes, which is known to substantially increase search difficulty for NK landscapes. It will be interesting to see how the increase in overlap affects the TD Mk landscape, and to compare the results with different codomains, including the random and deceptive-trap codomains.

The following research questions will be investigated:

- How does the amount of overlap in TD MK Landscapes affect the performance of LT-GOMEA within the random and deceptive-trap codomain?
- How does LT-GOMEA perform when the building blocks are learned from a population of hill-climbed solutions?
- Can LONs effectively help to explain the difficulties encountered by LT-GOMEA?

- Can LONs, as used in this study, provide useful insights into the search processes of population-based algorithms?

Chapter 3

NK and Mk Landscapes

3.1 NK Landscapes

Kauffman et al. [8][7] introduced NK Landscapes and used it to predict evolutionary processes. In this thesis, a solution is represented by a bit string of length N. S is the set of possible solutions (i.e. search space). The search space exists out of 2^N solutions. The fitness of a solution is dependent on interconnected bits. Every bit is dependent on at most K other bits, where $0 \leq K \leq N-1$. A subset existing of dependent bits from a bit string is called a *subfunction*. Every subfunction contains at most k variables, therefore $k = K + 1$. For NK landscapes, the number of subfunctions $M = N$ and also variable x_i must occur in subfunction $subf_i$ [23]. The fitness of solution s is the sum of all the subfunction evaluations. A landscape can be visualized in 3D where the horizontal dimensions can be determined by policy choices, where neighbouring solutions are 1 bit apart. The vertical axis represents the solution fitness. When $K = 0$, every bit is independent and $subf_i$ only contains x_i. An effective strategy of finding the global optimum can be hill-climbing, because dependencies do not have to be learned [4]. When K increases so do the ruggedness of the landscape [11], this increases the peaks and hills and when the problem is big enough it is unlikely for the hill-climber to find the global optimum, and linkage learning may be necessary.
NK Landscapes are proved to be NP-complete [22], but there is a subclass of NK Landscapes called Adjacent NK Landscapes, which is very popular for benchmarking purposes because it is polynomial time solvable. There are multiple dynamic programming algorithms introduced for this, one of them by Weinberger et al. [22]. Adjacent NK Landscapes are only polynomial solvable when treated as gray-box, so the problem structure is known, this means that the fitness scores for each subfunction configuration are given. This makes Adjacent NK landscapes in particular suitable for benchmarking black-box optimizers since the optimum of the black-box optimizer can be compared to the global optimum which is calculated in polynomial time.

$$F(s) = \sum_{i=1}^{M} subf_i(s, mask_i) \tag{3.1}$$

In the equation above, $mask_i$ selects all the bits from s that belong to $subf_i$. An optimization problem where $F(s)$ has the form of Equation 3.1 and s is represented by a bit-string is called a pseudo-Boolean optimization problem. Here follows an example, where bit x_i is dependent on the next bit and because every x_i must represent a subfunction and the subfunction size is 2, the last bit is dependent on the first bit. Here are the subfunction

fitnesses for every combination:
$f(0,0) = 2$
$f(1,0) = 0$
$f(0,1) = 0$
$f(1,1) = 4$
Next, the fitness will be calculated for an example bit-string (1, 1, 1, 0, 0, 1), where N=6 and K=1:
$F(1,1,1,0,0,1) = f(1,1)+f(1,1)+f(1,0)+f(0,0)+f(0,1)+f(1,1) = 4+4+0+2+0+4 = 14$
Note that it is not necessary for Nk Landscape problems that each subfunction has the same fitness evaluation for all combinations of subfunction variables. The case above only serves as an example.

3.2 Mk Landscapes

NK Landscapes are very useful for benchmarking, but they also impose redundant constraints such as $M = N$ and the required occurrence of variable x_i in $subf_i$. Therefore Whitley et al. [23] introduced a generalization of NK landscapes known as Mk landscapes, where the obligatory variable subfunction occurrences are not necessary and M does not have to be equal to N. M again represents the number of subfunctions and k indicates the upper bound for the subfunction size. Whitley et al. [24] also introduced a tree decomposition (TD) Mk Landscape, where the global optimum can be calculated in polynomial time when the fitness of the subfunction configurations is given. So it is polynomial-time solvable by a gray-box optimizer. Thierens et al.[16] introduced CliqueTreeMk which can generate any TD Mk Landscape, and these are also polynomial-time solvable.

3.2.1 Clique Tree Mk

The tree decomposition is being built from top to bottom. And the input parameters are subfunction size k, overlap o and branching factor b. The algorithm initially takes k variables from the input string, these form a clique in the root node. In example 3.1 these are $x4, x2, x7$. A clique represents the variables within a subfunction. The cliques in the branches are formed in BFS order. Each branch takes o random variables from the parent and fills this child by adding $k - o$ of the non-selected variables. The overlapping variables are denoted by S_i and the subfunction variables, i.e. the overlapping variables plus the additional variables are denoted by C_i, where the root node is C_0. The algorithm stops when all the problem variables are present in the tree. In the following example, variables $x_1..x_9$ are randomly ordered (x4, x2, x7, x5, x1, x9, x3, x8, x6), where $b = 2$, $o = 2$ and $k = 3$.

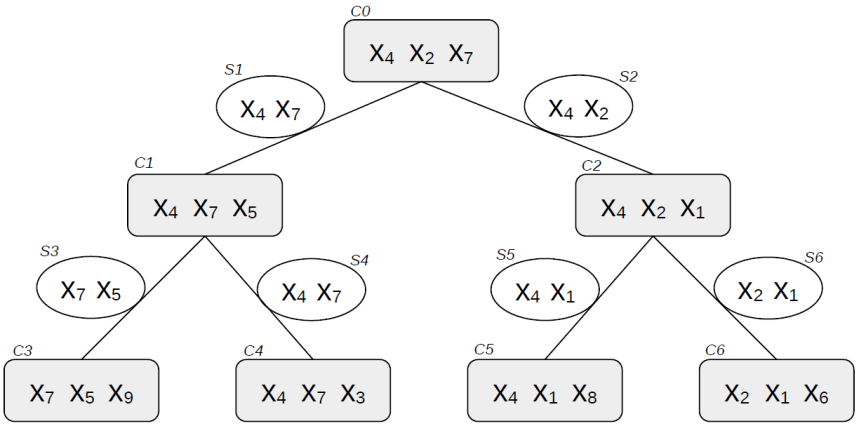


Figure 3.1: Example branching tree. Figure is retrieved from Thierens et al. [16]

In the resulting TD Mk landscape, $M = 7$. The cliques in the tree represent the subfunctions similar to the subfunctions for NK landscapes. The global optimum can be calculated in polynomial time using dynamic programming when the tree decomposition is known. The following steps will go over the algorithm as listed in their paper [16]:

1. Initially, take the next k variables as clique C_0. Otherwise, take the already constructed clique C_i.
2. Choose o random variables from parent clique C_i, assign to separator S_j .
3. Take next $(k - o)$ unchosen 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 build the next clique.

3.2.2 Global Optimum by Dynamic Programming

Every clique C_i except the root node C_0 introduces variable(s) C_i/S_i which are not present in the rest of the tree. This property can be used to solve the global optimum by dynamic programming in polynomial time. The global optimum is calculated by going through the branching tree from bottom to top in the following way: go over all the possibilities of the combination of all the variables S_i and choose the variable(s) C_i/S_i such that it maximizes the combination used in S_i. When C_i is a root node the maximizing value for C_i for each combination of S_i can simply be calculated by choosing the maximizing values for C_i/S_i. When C_i is not a leaf node, the maximizing value for the combination S_i is dependent on the C_i values of the children.

3.2.3 Codomains

The chosen codomain will to a great extent determine the structure of the landscape. This paper will investigate the structural differences when varying the input variables of the CliqueTreeMk algorithm within a certain codomain. The problem size N will be chosen such that it is computationally manageable. Besides a random codomain, also

a deceptive-trap codomain will be used. These codomains are ready to use in the TD Mk Landscape generator by Van Driessel [17]. Where the deceptive-trap codomain, gives fitnesses to the subfunctions accordingly: Each subfunction gets a random substring as an optimum for that subfunction with length k, this optimum has a fitness of 1. The inverse acts as an attractor and has a fitness of 0.9. The rest of the substrings have a fitness of $0.9 - d\frac{0.9}{k}$, where d is the hamming distance with respect to the optimal substring. The hamming distance is the number of bits that differ, so the hamming distance between the attractor and the optimum is k. This codomain makes it so that a hill-climber will tend to move to the attractor. The random codomain assigns random fitness scores between 0 and 1 for each substring of the subfunction.

Chapter 4

Constructing LONs

For understanding the difficulties within a landscape, a lot of concepts are used of how Ochoa et al. [13] constructed LONs for TSP instances and NK Landscapes [11] with escape edges introduced by Verel et al. [19], this is also done by Herman et al. [6] for NK Landscapes. This chapter will start by explaining their methods and later on in chapter 6 some of their ideas will be used for making LONs with LT-GOMEA as search algorithm. Ochoa et al. [13] used ILS as a search algorithm and they used Best-Improvement-Local-Search (BILS) for exploring the landscape. BILS starts by picking an initial solution $s \in S$ and calculating its fitness $f(s)$. All its neighbours $V(s)$ will be explored and if the best neighbour s' is better than s, s becomes s' and the hill-climbing starts again. If there is no better solution in the neighbour space, s is marked as a LO [14]. Hence, s is a LO in a maximizing problem when: $\forall s' \in V(s), f(s) \geq f(s')$.

Algorithm 1 Best Improvement Local Search

$s \leftarrow$ *Initial solution* $s \in S$
$s' \leftarrow x_0 \in V(s)$, *such that* $\forall x \in V(s), f(x_0) \geq f(x)$
while $f(s) < f(s')$ **do**
    $s \leftarrow s'$
    $s' \leftarrow x_0$
end while
$LO \leftarrow s$

The BILS operator is denoted by $h(s)$. The vertices of a LON are represented by LOs, where $LO_i \in S$. The basin of attraction of LO_i is the set of all the solutions that arrive at LO_i after BILS, $bi = \{\forall s \in S | h(s) = LO_i\}$. Verel et al. [19] proposed basin-transition edges and escape edges for the edges between LOs. This paper only uses escape edges because calculating all the edges (basin-transition-edges) uses a brute-force method, which involves searching through all possible solutions, which is too computationally expensive. Instead, escape edges can serve as an informative model and are less computationally expensive [19].
Escape edges are explored using a mutation operator. An edge e_{ij} between LO_i and LO_j exists when one can use at most D (where, $D > 0$) mutations to go from LO_i to a solution belonging to b_j. The mutation operator flips 2 random bits. So when $D = 2$, four random bits will be flipped. The edge weight w_{ij} is being sampled by the number of times this edge is traversed, note that a LON is directed and if the edges e_{ij} and e_{ji} exist, w_{ij} and w_{ji} do not have to be equal. The following algorithm shows how a LON is created using ILS, where the mutation operator mutates solutions when stuck in a LO. The algorithm

below optimizes an initial solution by BILS (alg. 1) and mutates the solution at most n times, if it does not result in a better or equal solution after BILS optimized the mutated solution. When the mutated solution arrives at a better or equal LO after BILS, an escape edge between these LOs is added. When this happens, the search will continue and starts mutating this new LO, otherwise, it continues mutating the previous LO. When the global optimum is found, or n mutations did not lead to a different LO, this ILS run stops. The example below performs 1000 ILS runs in total.

Algorithm 2 LON creation

```
LONs ← {}
Edges ← {}
Attractors ← {}
n ← 1000                                   ▷ n is empirically determined
for 1 to 1000 do
    s ← initial solution
    s ← h(s)
    for i from 1 to n do
        s' ← mutation(s)
        s' ← h(s')
        if f(s') ≥ f(s) then
            LONs.add(s')
            Edges.add((s, s'))
            s ← s'
            i ← 0
        end if
    end for
end for
```

4.1 Mining Funnels

The notion of *funnels* is used for explaining search difficulties within the landscape. Ochoa et al. [13] used this concept for TSP landscapes. It is called a funnel because for minimization problems including the TSP problems the valleys contain LOS which are hard to escape from. When a local search algorithm finds a LO located in these valleys, it becomes very hard to move between valleys and the best solution derivable is probably the optimum solution within this funnel. Finding the global optimum for TD Mk landscapes is a maximization problem, but still the term *funnel* will be used for describing the reversed valleys. For grasping the structure of the landscape it is interesting to find the optimum solutions located within these funnels because this is where effective evolutionary algorithms are likely to end up. Consequently, this thesis will go over a method to find these so-called *funnel floors* (best LOs within funnel), which are similarly used in the TSP landscape analyses of Ochoa. For finding the funnels, it is necessary to find the funnel floors. For recognizing these funnel floors, *attractor* nodes are used, which are funnel floor candidates. These *attractor* nodes are those nodes that did not improve or did not find an equal solution by applying the mutation function n times in algorithm 2. Therefore attractor nodes are LOs that do not improve easily after mutation. It can be that for independent runs, some attractor nodes did find an improvement or

equal solution after mutation and are therefore connected to nodes with higher fitness or equal fitness in the LON. If an attractor node is connected to an attractor with higher fitness, it can not be a funnel floor, because nodes belonging to a funnel floor never improved. For generalizing the structure of the landscape a network is created consisting of only attractor nodes. These attractor nodes are used for finding the sinks meaning the funnel floors of the network. The words sink and funnel-floor are used interchangeably. The attractor network per definition contains the sinks because these funnel floors are nearly impossible to escape and did not improve after n mutations. For neutral networks (networks that contain neighbouring nodes of similar fitness [20]), connected attractor nodes with equal evaluation are compressed to one plateau node within the attractor network. An attractor node lies within a sink if there are no outgoing edges to attractor nodes of better evaluation. By going through the nodes of the attractor network, the sink nodes can be recognized, because these are the nodes (plateau sinks for neutral network) without outgoing edges in the attractor network.

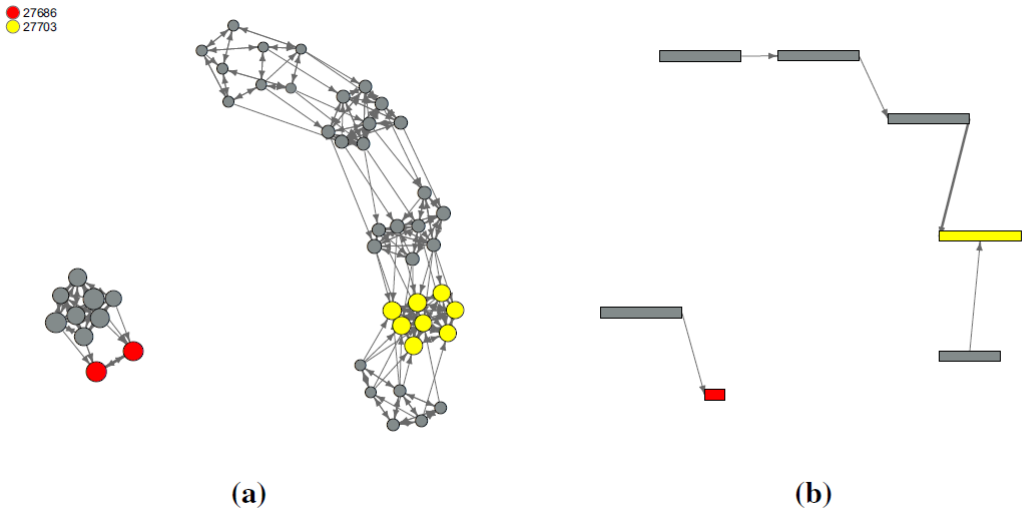


Figure 4.1: Example of neutral TSP Attractor network retrieved from Ochoa et al. [13]. The colors represent the fitness of the funnel sinks. Two funnels sinks can be recognized, where red is the global optimum, and yellow is a suboptimal sink. Graph (**a**) shows the attractor network, in (**b**) the connected attractor nodes of (**a**), which have equal evaluations, are compressed to a plateau. The plateau length indicates the number of attractor nodes. Grey nodes belong to a funnel but are not sinks themselves.

In algorithm 3, a sink node is denoted by $s \in S$, where S is the set of sinks extracted from the attractor network. When the network is neutral, s can also be a sink plateau node. The most important thing about the sink nodes is that it is used to divide the LON into different funnels. The sink nodes are used for detecting to which basin the LOs belong. For detecting the funnel basins, a BFS starting from the sink is done for each sink. The edges are reversed so that BFS will reach all the nodes that can reach the sink in the original LON. In other words, the nodes that can reach a particular sink lie in the funnel basin of this sink. The outgoing edges of the sinks within the LON are removed because the connected nodes are not attractor nodes themselves and belong to a different funnel basin [13]. The operator that removes edges from sinks to non-attractor nodes, is denoted by $DelOutgoingSinkEdges(LON, S)$. When the BFS algorithm is finished, all the LOs within the LON will belong to one or more funnels, these properties will be

visualized in a 2D representation. When the basins are established, the LOs which belong to multiple basins are represented by the set *Overlap*.

Algorithm 3 Calculate Basins

```
Basins ← {}
S ← funnel sinks
DisconnectedLON ← DelOutgoingSinkEdges(LON, S)

for s ∈ S do
    Basins.add(BFS(DisconnectedLON, s))
end for

Overlap ← {}                                  ▷ LOs that belong to multiple Basins
for i, j in range(S.length) do
    Overlap.add(Basins[i] ∪ Basins[j])
end for
RemoveDoubleNodes(Overlap)
```

4.2 Visualization

By visualizing the LON, the funnel basins can be recognized which will give insight into the structure of the landscape. The changes in funnel structures caused by varying input parameters for the CliqueTreeMk algorithm will be investigated. Also, the mutation size will influence the edge distribution. It is important to note that LT-GOMEA does not use a standard mutation size. Instead, it learns building blocks that are swapped. The changes that have been made to adapt it for the use of LT-GOMEA are mentioned in section 6.0.1. The x position of an LO within this LON can be determined by a graph visualization module. The fitness of an LO can be shown by its y position. LOs can be colored according to which funnel they belong. The size of the node can be visualized according to the node's strength. When the network is relatively big and the important structures are only visible in certain parts of the network, it can be pruned by removing nodes that have a fitness lower than a certain threshold [12]. This threshold has to be of a magnitude such that all sinks can be displayed. This pruning process can alter the structure, which has to be taken into account when analyzing.

Chapter 5

LT-GOMEA

An Evolutionary Algorithm (EA), is a population-based search algorithm that uses biological processes to optimize a population of solutions by reproduction, mutation, selection, and recombination [21]. The performance of a solution can be determined by a fitness function. There are various ways to optimize a population by using different crossover policies. A lot of those policies ignore structural information that can be observed from the population, and the crossover process stays the same disregarding the population. For harder problems, it can be useful to learn structures and use this to make crossover policies. This is what the Gene-Pool Mixing Evolutionary Algorithm (GOMEA) does, it supposes that an improving population yields beneficial structural information and dependencies between variables that can be used to establish building blocks (linkage-learning). These building blocks are then used in crossover to preserve important structures in the offspring.

5.1 FOS and Linkage Tree

Let S be the set of all variable indices: $\{0, 1, ..., l-1\}$, where l is the problem size. The dependencies between variables can be captured in subsets of S. The linkage tree contains the sets that are considered as blocks for crossover in LT-GOMEA. A set of subsets that capture the dependencies between variables is called a Family of Subsets (FOS) [15]. Where FOS F is a set of subsets from S. F is a subset of the superset S, $F \subseteq P(S)$. Three examples of FOS models are: univariate, marginal, and linkage tree FOS structure. Let the length of S be, $l = 6$. For the univariate FOS structure all the subsets consist of 1 variable, e.g. $F = \{\{0\}, \{1\}, \{2\}, \{3\}, \{4\}, \{5\}\}$. This structure is limited because it cannot represent dependencies between variables. The marginal FOS structure can contain sets that consists of more than 1 variable, but every variable can only occur once in F, e.g. $F = \{\{0, 3\}, \{1, 2, 4\}, \{5\}\}$. This structure can represent dependencies between variables, but the limitation of a variable only occurring once in the FOS excludes higher-order dependencies [1]. The Linkage Tree is a hierarchical representation of dependencies [1], where the leaf nodes are the single problem variables of the univariate FOS structure and the root node contains all the variables of S. The variables in a subset are dependent on each other, and the children are a marginal structure of the parent, and therefore the variables become independent within the children if they belong to another child. This hierarchical structure makes it very effective to capture the higher and low-order dependencies [1].

5.1.1 Building the Linkage Tree

The Linkage tree is built from bottom to top with the reciprocal nearest chain algorithm [5], starting with the univariate subsets. This algorithm runs in $O(l^2)$. The goal is to combine the most dependent subsets, therefore the Mutual Information (I) between these subsets is computed. Equation 5.1, measures the mutual dependence $I(X, Y)$ between variable X and Y, and determines the similarity between the joint distribution $p(X, Y)$ and the product of marginal distribution, $p(X)$ and $p(Y)$. In our problem, the variables $X, Y \in 0, 1$. Computing $I(X, Y)$ between all pairs of variables is an l^2 operation. When the linkage sets have a length > 1, the Unweighted Pair Group Method with Arithmetic Mean (UPGMA)[5][2] is used. In Equation 5.2, the compared subsets are denoted by F_i and F_j. The process of combining subsets is repeated until every subset is combined once. Hence, the root node contains all the variables. We do not combine the 2 last subsets for LT-GOMEA, because using the root node and therefore all the variables for a swap, essentially means that a solution inherits all variables from another solution, which is not desired for recombination. Now the LT will contain $2l - 1$ subsets.

$$I(X, Y) = \sum_{x \in X} \sum_{y \in Y} p(x, y) log \frac{p(x, y)}{p(x)p(y)} \tag{5.1}$$

$$I^{UPGMA}(F_i, F_j) = \frac{1}{\mid F_i \mid\mid F_j \mid} \sum_{X \in F_i} \sum_{Y \in F_j} I(X, Y) \tag{5.2}$$

5.1.2 GOMEA

After the LT is learned from the population, the Gene-Pool Optimal Mixing (GOM) operator, uses the linkage sets to generate new offspring. The linkage sets represent the variables that the parent node inherits from a donor to create offspring. GOM goes over all the solutions in the population in an attempt to make improvements. The process for every parent solution to create offspring goes as follows: The linkage sets are traversed in order from last merged to first merged, and as mentioned before, the linkage set belonging to the root node is not used because the offspring would then be exactly the same as the donor and no recombination would take place. A random donor is taken out of the population for every traversed linkage set. And the parent inherits the variables of the current linkage set from the donor. If this makes an improvement or generates a solution of equal fitness, continue the search with this new solution, otherwise, keep the current one. The offspring is the solution that remains after the linkage sets are traversed.

5.1.3 LT-GOMEA

At first, the population needs to be initialized, this can be a population that consists of n random solutions. The process of building LT from the population and then generating new offspring by the GOM operator is repeated till a stopping criterion is met.

5.1.4 LT-GOMEA with Local Search

One reason for the success of LT-GOMEA is its ability to identify and learn important building blocks, rather than relying on random mutations for improvement like some other search algorithms such as ILS. It learns from the population as a whole instead of only the fittest solutions in the population. ILS often fails to solve harder problems, because it has to be extremely lucky to find the global optimum for deceptive codomains and large problems. But what if the building blocks are learned from a population of BILS-optimized solutions? It would be interesting to know if optimized solutions yield enough information to capture the right building blocks. This raises the question, of whether the LT could be learned out of a population of LOs alone. The advantage of this method is that the LT only has to be learned once out of the LOs and then the building blocks can be used to improve a solution. These building blocks are then used to improve one particular solution instead of a whole population. Now, the search becomes very similar to ILS, but the mutations are now learned blocks from the LOs.

Chapter 6

LON of LT-GOMEA on TD MK Landscapes

6.0.1 LON for LT-GOMEA

Making a LON for population-based search algorithms is not as straightforward as for ILS as explained in section 4. For ILS, there is one improving solution, where every passed LO is a node and every successful perturbation makes an edge between LOs. For LT-GOMEA there is an improving population instead of just one improving solution. To overcome this change, an LT-GOMEA run will be handled similarly to an ILS run, but the best solutions in the population are used for representing the nodes instead of one improving solution. If a new generation yields solutions that are equal to or better than the current best solutions, an edge will go from the nodes of the previous best solutions to the nodes of the current best solutions. For non-neutral landscapes, there is only one best solution. To complete the LON, multiple LT-GOMEA runs can be executed. The process for defining attractor nodes will differ from that used for ILS, where a node becomes an attractor if the solution does not improve after n hill-climbed perturbations. In ILS, the value of n used to determine when a node becomes an attractor is often quite large (for example, $n = 10000$ in Ochoa et al. [13]). However, in LT-GOMEA, the population tends to converge quickly when the best solution has not improved for a few generations, so continuing the search for a large value of n would be inefficient. Therefore, a lower value of n must be maintained. A stopping criterion must be defined for LT-GOMEA such that when the algorithm takes too long, the best solutions become attractors and the LT-GOMEA run is terminated. These LT-GOMEA runs are done multiple times for the creation of a LON.

6.0.2 LON for LT-GOMEA with building blocks learned from LOs

As described in section 5.1.4, LT-GOMEA will learn its building blocks from a population of hill-climbed LOs generated by BILS. The building blocks are learned from this population, and constructing a LON for this population is similar to the method used for ILS, since there is no improving population and the edges in the LON are the direct result of LOs that have been improved using the learned blocks and hill-climbed LOs. The solutions that made improvements are all present in the LON which is not the case for the population-based LT-GOMEA. The experiment setup will specify the stopping

criteria for the runs. If these criteria are too strict, many nodes may become attractors and the search may terminate before reaching the optimal solution. On the other hand, if the criteria are too soft, the search may take too long to find the global optimum and may become stuck in a local optimum. These criteria are tested by checking the lower bound for the attractor threshold, to which the performance does not drop.

Chapter 7

Experiment: Increasing Overlap for LT-GOMEA

This experiment is performed to investigate how an increase in overlap affects the performance of LT-GOMEA on problems within the random and deceptive-trap codomain. The deceptive-trap codomain, as described in section 3.2.3, produces a neutral landscape with many plateaus, where different solutions with equal fitness exist. As a result, there can be multiple nodes belonging to the best solutions in a generation. This can lead to a LON with many edges and nodes. Although these neutral landscapes may not closely resemble real-world problems where each solution typically has a unique fitness, the deceptive component allows for challenging problems that may require linkage-learning to solve. Therefore, this experiment also looks into a slightly altered version of the deceptive-trap codomain, where a random value between 0 and 0.01 is added to each fitness belonging to the configurations of the subfunctions. These increments are so little in comparison with the differences in fitness for different subfunction configurations, that they do not disrupt the deceptive behaviour. The performances of the deceptive-trap codomain with and without random increments are compared to see if they act similarly. Furthermore, the performance of LT-GOMEA on the random codomain will be investigated.

7.1 Experimental Setup for Regular LT-GOMEA

The input parameters for the CliqueTreeMk problem generator, are subfunction size $k = 8$ and branching factor $b = 1$. The problem length is kept to a length of $N = 80$ for all instances. To maintain this length while increasing the overlap, the number of subfunctions M is increased. To determine the corresponding number of subfunctions for a problem with length N and the desired overlap, the following formula is used: $N = Mk - o(M - 1)$, where Mk is the sum of the lengths of the subfunctions. The overlapping bits with a count of $o(M-1)$ are subtracted because this results in the problem length. To ensure that the problem length is N and that the number of subfunctions M is an integer, only overlap values that meet these criteria are used. The formula can be rewritten to M: $M = \frac{L-o}{k-o}$. In the experiments $N = 80$, because it allows for a lot of overlap values that produce integer values for M and the problem length is sufficient for generating complicated problems relative to the population size. To generate the CliqueTreeMk problems, 6 different parameter configurations are used, which consist of the following values for (o,M): (0,10),(2,13),(4,19),(5,25) and (6,37). The population size is chosen such that LT-GOMEA does not always converge to a global optimum, as it

is expected that especially there where LT-GOMEA struggles, interesting features can be observed in the LON which can provide insights in identifying the challenges that LT-GOMEA faces on the selected problems. A population size $P = 600$ is chosen. For each CliqueTreeMk parameter configuration, 10 problem instances are generated. Each instance is solved using 20 runs of LT-GOMEA, and the corresponding LON for this instance is updated with each run. During the search, edges form between the nodes of the best solution(s) of the previous generation to the best solution(s) of the current generation (there can be multiple solutions for the neutral network). The edges and nodes found in the 20 runs per instance of CliqueTreeMk are combined into one LON for each instance. As a result, if an edge is formed in one run, it will be present in the LON. The strength of a node is the number of generations in which it was a part of the best solution(s). In the LON, the node strengths for each run are combined. If a node becomes an attractor in one run, it will remain an attractor in the combined LON.
The stopping criteria for each LT-GOMEA run are as follows: If during the search, LT-GOMEA did not introduce a new best solution(s), compared to the best solution(s) of the previous generation, for three consecutive generations, the run stops and the best solution(s) of the final generation, becomes an attractor(s). This stopping criterium is based on how fast the population converges when not improving.

7.2 Results

7.2.1 Results for Deceptive-Trap Codomain

Figure 7.1 shows the number of times that LT-GOMEA successfully found the optimum for each experiment. The CliqueTreeMk parameters are displayed on the left side of the diagrams as M_k_o (number of subfunctions M, subfunction size k, overlap o). All problems are generated with a branching factor of 1. The experiments labelled *deceptive-trap neutral* (in red) refer to the deceptive-trap codomain as described in Section 3.2.3, while those labelled *deceptive-trap non-neutral* (in blue) refer to the randomly incremented codomain. The x-axis represents the experiment number, where all experiments of the diagram are generated with the same input parameters. Note that the experiments that are generated by the same input parameters, do not create the same problem. The *deceptive-trap non-neutral* and *deceptive-trap neutral* codomains both use the same generated instance by CliqueTreeMk, with the only difference being that the subfunction fitnesses are slightly increased for the *deceptive-trap non-neutral*.

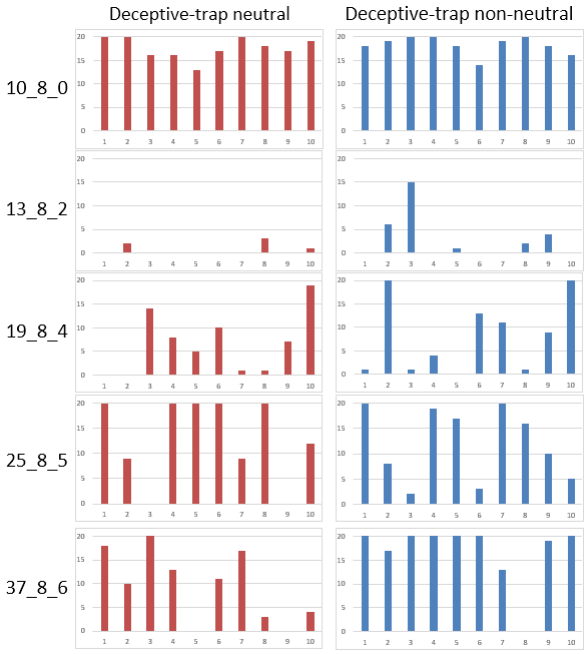


Figure 7.1: In red the experiments for Deceptive-trap neutral and in blue for Deceptive-trap non-neutral. The x-axis shows the experiment number and the y-axis shows the number of successful runs for 20 LT-GOMEA runs per experiment. Branching factor = 1, Population size = 600. Each diagram shows the input parameters displayed as M_k_o.

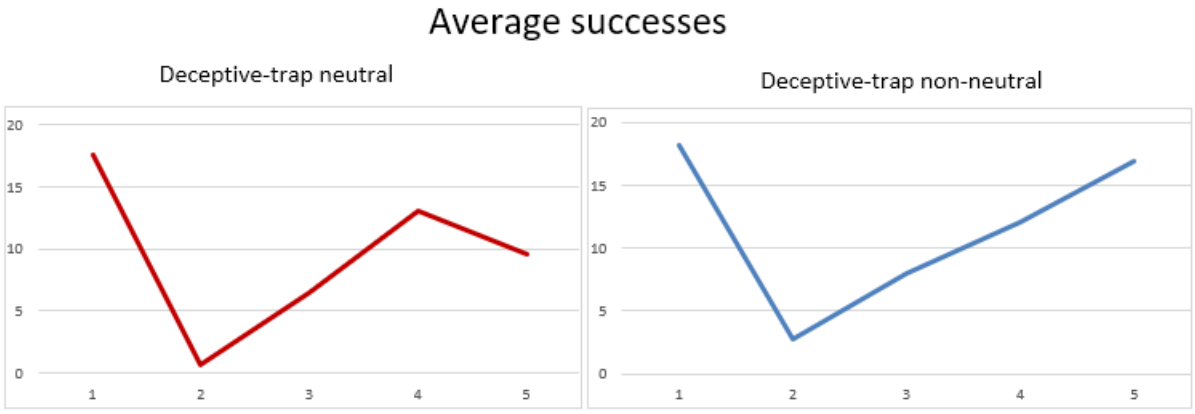


Figure 7.2: These graphs display a line representing the average of successful runs across all experiments for each CliqueTreeMk input configuration. The x-axis indicates the input parameter configurations in the order shown in Figure 7.1.

10_8_0	13_8_2	19_8_4	25_8_5	37_8_6
1	1	4	160	256
1	1	8	64	64
1	1	2	32	32
1	1	4	32	64
1	4	48	64	64
1	1	20	160	32
1	3	32	16	256
1	2	16	32	64
1	1	256	16	128
1	2	32	24	16

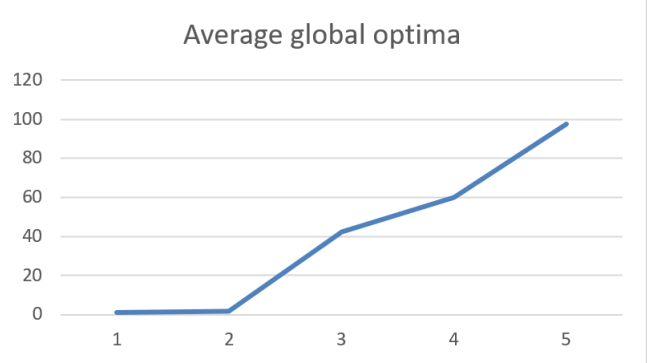


Figure 7.3: The table displays the number of global optima for each experiment of each CliqueTreeMk input configuration within the deceptive-trap neutral codomain. The line graph illustrates a line representing the average values per configuration.

The experiments of input parameter configuration 10_8_0 show similar results for both codomains. Where in all experiments, the majority of LT-GOMEA runs found the global optimum. However, configuration 13_8_2 shows different results. For both codomains, at least half of the experiments did not find the global optimum at all. Only experiment 3 for the deceptive-trap non-neutral codomain had a majority of successful runs. This configuration had the worst performance in terms of finding the global optimum for both codomains. As seen in Figure 7.2, the two codomains showed differences for configuration 37_8_6, with the deceptive-trap non-neutral codomain showing an upward trend, unlike the neutral codomain. In figure 7.11 can be seen that for the deceptive-trap neutral codomain, configuration 25_8_5 and 37_8_6 each have 2 experiments (exp 2 and exp 8 for 25_8_5 and exp 4 and exp 8 for 37_8_6) where the number of vertices is significantly larger compared to the other experiments of the input configuration. These experiments also have no successful runs. Figure 7.3 shows that the average number of global optima of each input configuration increases as the overlap increases for the deceptive-trap neutral codomain.

LON details

In the following part, examples of LONs for the deceptive-trap non-neutral codomain for each input configuration will be displayed, starting with an explanation of how they are visualized. The colour of a node indicates to which funnel it belongs. Grey nodes belong to multiple funnels. The sink nodes can be identified by their fitness values being displayed. The x-position of each node is determined by a layout algorithm using the Pyvis module in Python. Some x-positions have been manually adjusted for better readability. The y-position is determined by the fitness value of the node, with nodes at the top having the lowest fitness and nodes at the bottom having the highest fitness with a linear relationship. The strength of each node is represented by the diameter. The node strength in this experiment is the number of times a solution belonged to the best solutions in the population. Therefore, a node must have a strength of at least 4 to become an attractor, according to the stopping criteria described in section 7.1. Namely, 3 consecutive non-improving generations. The relationship between node strength and diameter is not linear, as it would result in a too-large size difference between nodes. The edges in the LONs are directed from top to bottom or between nodes of equal fitness. However, the

latter does not occur in the following figures due to their non-neutral property. Self-loops are edges that connect a node to itself and occur when a node belongs to the best solution(s) for consecutive generations. Self-loops can be seen in the bottom left corner of Figure 7.6b and in the global optimum. Figure 7.6b will be discussed and used to explain some additional concepts. Most of the nodes have a strength of one, which means that these solutions were only part of a single LT-GOMEA run within a single generation. The node belonging to the global optimum has a strength of 76, this can be explained by the fact that LT-GOMEA was successful for 19 out of 20 runs. The strength increases 19 times when the global optimum is found and also increases three consecutive times for each run that LT-GOMEA is unable to find an improvement because it has already found the global optimum, therefore: $Strength = 19 + 3 * 19 = 76$. Note that the strengths of the sinks combined can be less than $20+3*20 = 80$, because there can be attractor nodes that eventually improved in a different generation and therefore are not marked as sinks. The top of the LON displays 20 starting nodes that are relatively random, as they are the best nodes for a random population that have been optimized by BILS. Given the size of the search space, it is not surprising that these starting nodes do not have a strength higher than one, otherwise, this would indicate that separate runs started with the same best solution as the starting node. The biggest improvement in fitness for each run can be observed between the first and second generations of LT-GOMEA. In this case, there is only one funnel in which the global optimum is the sink node.

LON Results

This section discusses the LONs of the deceptive-trap non-neutral codomain. All the LONs where the input parameter configuration is 10_8_0 are similar to Figure 7.6b, with the global optimum found in all of them and most LONs having only one funnel. There are some LONs with additional funnels, but their basins are smaller compared to the basin of the funnel containing the global optimum. The strength of the global optimum is the highest among all nodes in all instances, and the majority of nodes have a strength of one.

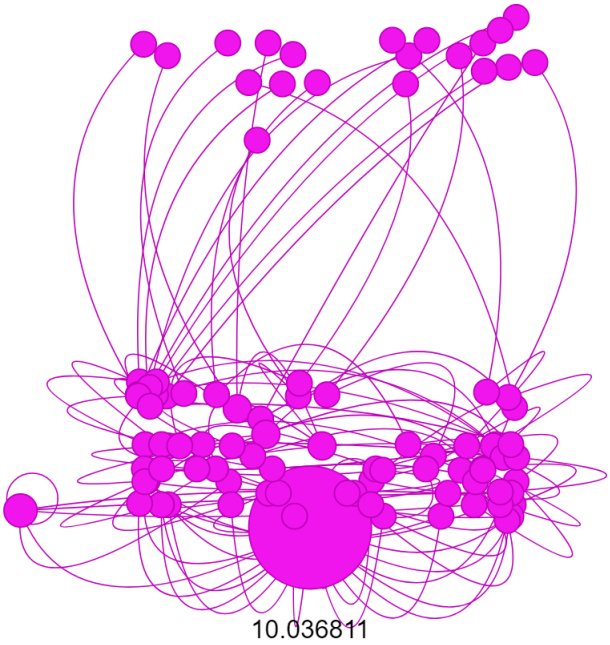
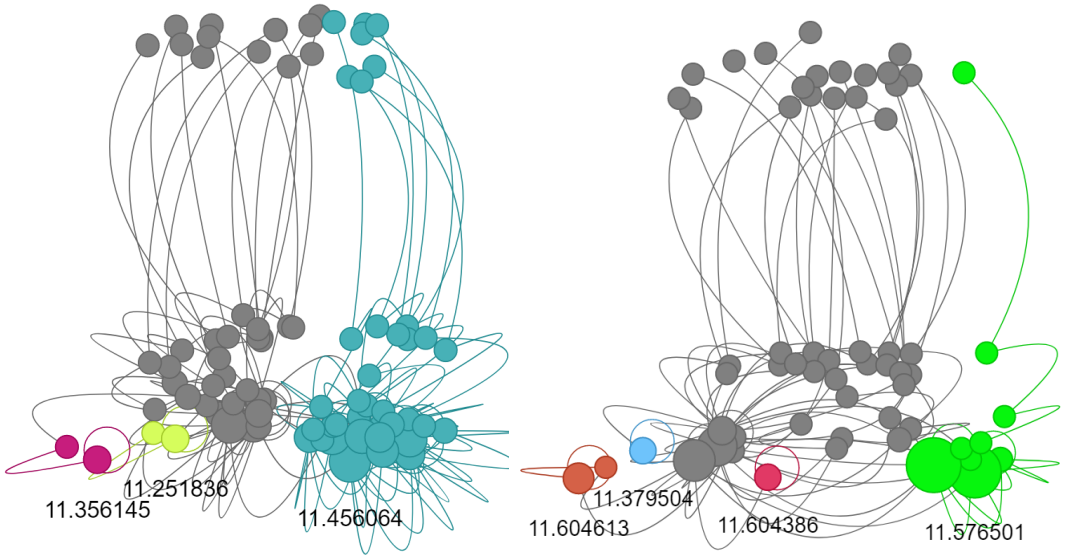


Figure 7.4: This is the 2nd experiment from the "deceptive-trap non-neutral" codomain where the input parameters are (M=10, k=8, o=0). There is one funnel containing the global optimum.

In half of the instances for input configuration 13_8_2, the global optimum was never found. In the cases where it was found, it was not found frequently, except for experiment 3. The number of funnels varies between 1 and 4 for the instances where the global optimum was found and between 1 and 5 for the instances where it was not found. In 8 out of 10 instances, the best-found solution is not the one with the highest strength within the LON. The two cases where the best-found solution has the highest strength are also the cases where the global optimum was found. One of these is shown in Figure 7.5a, where all three funnels share a large overlapping portion of their basin, with the blue funnel containing the global optimum having the largest non-shared basin. Most nodes with a strength greater than 1 are blue nodes close to the global optimum.

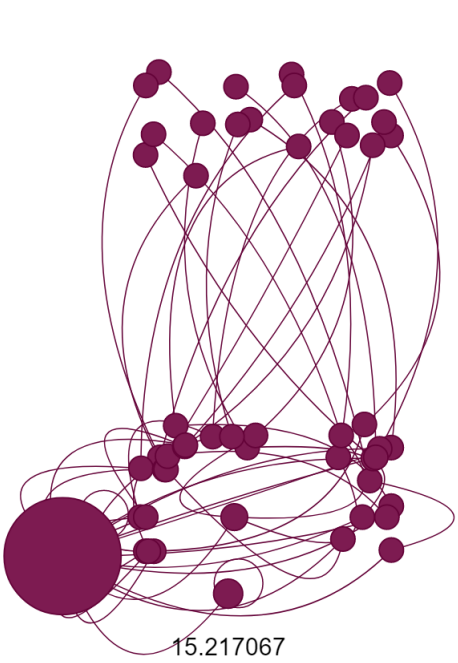


(a) This is the ninth experiment. It has three funnels, with the blue funnel containing the global optimum.

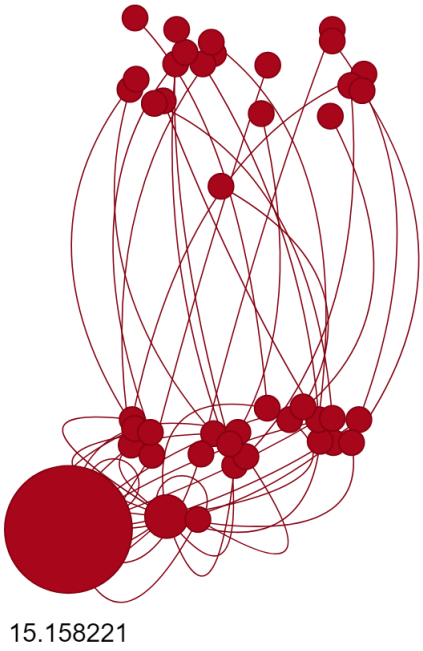
(b) This is the eighth experiment. It has four funnels. The brown sink is the global optimum.

Figure 7.5: The input parameters are (M=13, k=8, o=2) and the deceptive-trap non-neutral codomain is used.

Figure 7.5b also shows 1 bigger funnel as well as smaller funnels, and all funnels share an overlapping basin. In both figures, the grey nodes with high strength, are relatively high in fitness and are well connected to colored nodes. In figure 7.5b these are the grey nodes between the blue and red sink. A difference with figure 7.5a is that the largest funnel does not contain the global optimum.

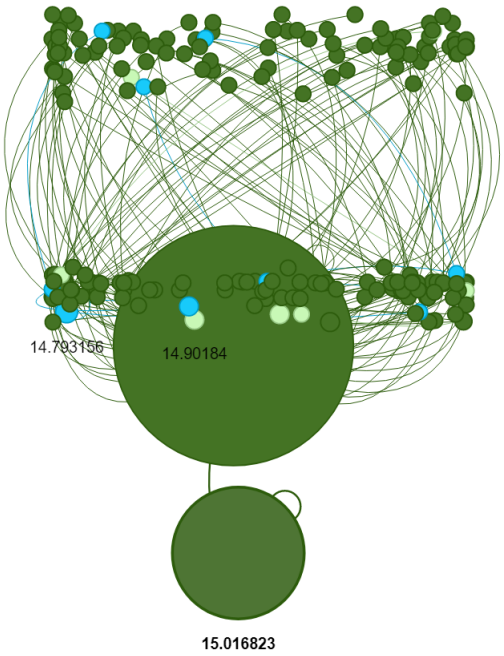


(a) This is the first experiment. It has one funnel, containing the global optimum.

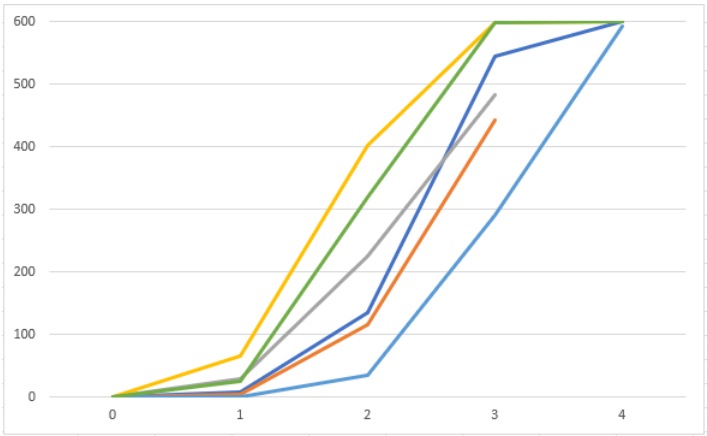


(b) This is the second experiment. It has one funnel, where the global optimum is the largest node.

Figure 7.6: The input parameters are (M=19, k=8, o=4), and the deceptive-trap non-neutral codomain is used.



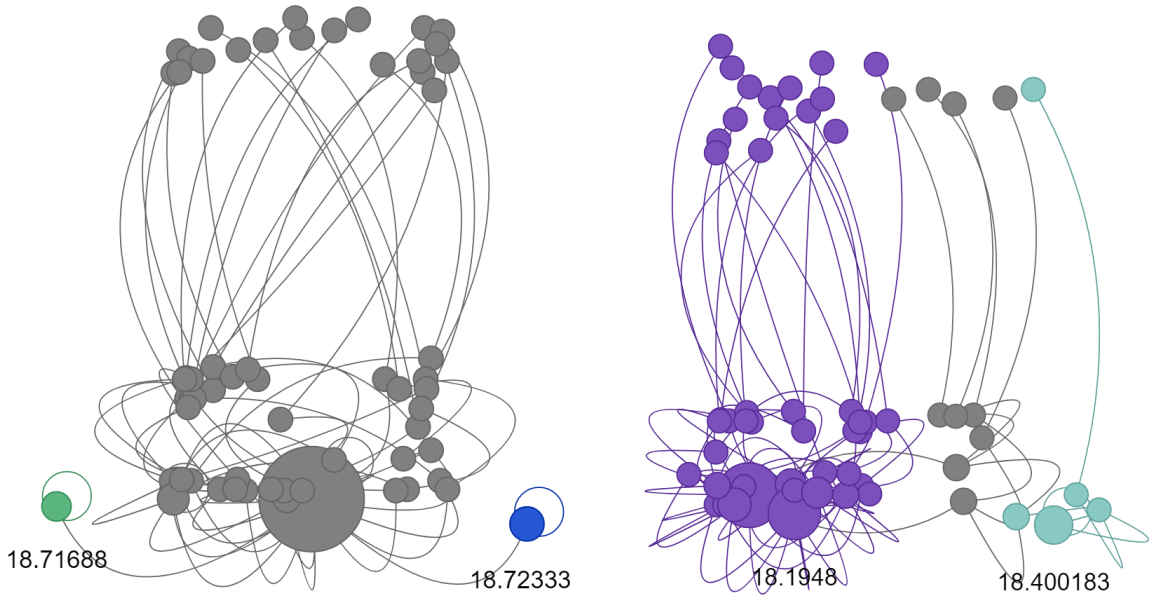
(a) Input parameters are (M=19, k=8, 0=4), and the deceptive-trap non-neutral codomain is used. This LON shows 100 runs.



(b) The y-axis shows the occurrences of the large attractor (above the global optimum) in the population (600) for 6 separate runs. The x-axis represents the generation, with generation 1 being the first time the attractor is present and the line ending when LT-GOMEA stopped.

Figure 7.7

At first glance, Figure 7.6a and Figure 7.6b appear similar. Both problems were generated using input parameter configuration 19_8_4, but LT-GOMEA found the global optimum once for the first experiment shown in Figure 7.6a and 20 times for the second shown in Figure 7.6b. Both figures show one funnel and one large node (high strength) in relation to the others. The large nodes are reached for each LT-GOMEA run, but for experiment 1 this node is the global optimum, while for experiment 2 it is not. In experiment 1, the global optimum is found once after finding the solution belonging to the large node.
For input parameter configuration 19_8_4, 4 out of the 10 experiments have a main funnel in which the global optimum is the node with the highest strength. In 2 of the experiments, there is an attractor node with a high strength relative to the other nodes, but it is not the global optimum. Figure 7.7b and 7.7a show an instance of configuration 19_8_4 containing 100 runs. The biggest attractor has been found in almost every LT-GOMEA run. In all of the cases, the optimum is found after this attractor has been found. And for almost half of the cases, this attractor was a parent or modified solution to generate the global optimum. Figure 7.7b shows 6 runs of LT-GOMEA demonstrating the speed at which the population can converge to an attractor. Many runs in this particular problem exhibit this behaviour. The x-axis represents the generation, with generation 1 showing the first generation in which the attractor appears in the population, and the following generations show the increase until LT-GOMEA meets its stopping criteria. For these 6 runs, LT-GOMEA converged in 2 to 3 generations after the attractor became present. This shows how fast a population can converge existing of almost one single solution.



(a) Experiment 3. 2 funnels, where the blue sink is the global optimum.

(b) Experiment 6. 2 funnels, where the light blue sink is the global optimum.

Figure 7.8: Configuration 25_8_5, from the Deceptive-trap non-neutral codomain

In 6 out of 10 experiments, for input parameter configuration 25_8_5, the biggest funnel contains the global optimum which also resembles the node with the highest strength. The global optimum has been found in all problems within this configuration. Figures 7.8a and 7.8b show instances where LT-GOMEA performed the worst in terms of the number of times the global optimum was found. Figure 7.8a displays one large attractor

and two better solutions, with the blue one being the global optimum. The edge from the large attractor to the global optimum was traversed twice, indicating that the optimum was found in two separate runs after the large attractor was found. The edge from the large attractor to the other sink was traversed once. Figure 7.8b displays two funnels, with the smallest one containing the global optimum. The two funnels overlap, and the two grey nodes with the highest fitness were explored by two different runs. These runs caused different edges to lead to better nodes in different funnels.
For configuration 37_8_6, 9 out of 10 instances, show one main funnel where the biggest node by far is the global optimum. These 9 LONs look very similar too figure 7.6b and in general look very similar to the LONs produced for the instances of configuration 10_8_0.

Repeatability of LT-GOMEA's Results

In order to assess the repeatability of LT-GOMEA's results, 10 LONs were constructed for the same hard instance of input parameter configuration 19_8_4. The global optimum was found in 9 out of 10 LONs, ranging from 1 to 7 times. The identified attractors were similar across all cases, with all LONs showing a prominent and common attractor. In 8 out of the 9 cases where the global optimum was found, this attractor had an edge connecting it to the global optimum and both belonged to the largest funnel. The number of sinks and funnels varied, with some instances having attractors that became sinks while others did not, resulting in mostly small funnels aside from the largest one.

Increasing Ruggedness

In the previous experiments, the number of subfunctions M compensated to maintain a constant problem length N, when the overlap was increased. Alternatively, N can be kept equal by changing the subfunction size k for an increase in overlap. This is done when changing the ruggedness for NK Landscapes, which is known to increase the difficulty [9]. Increasing the size of the subfunctions leads to an increase in overlap due to the constant number of subfunctions (M), which also results in an increase in ruggedness. To make experiments for Mk landscapes similar to the increase of ruggedness, the subfunction size has to compensate to keep the problem length equal. This has been done for the experiments of the following configurations:

N	M	k	o
48	6	8	0
49	6	9	1
50	6	10	2
51	6	11	3
52	6	12	4
47	6	12	5

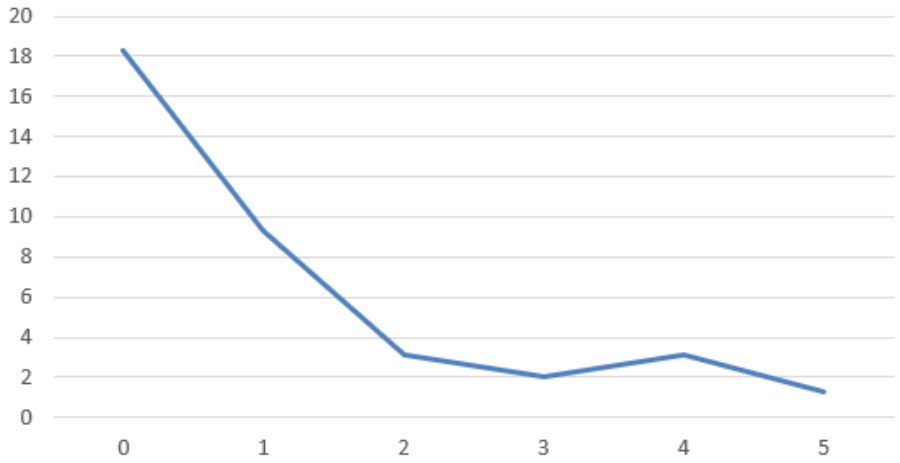
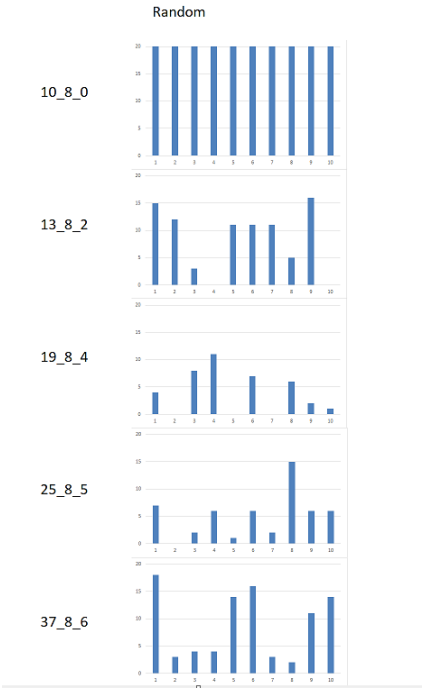


Figure 7.9: This graph shows the average number of successes across 20 runs, plotted in the order of the input parameter configurations listed in the table above (ordered by increasing overlap).

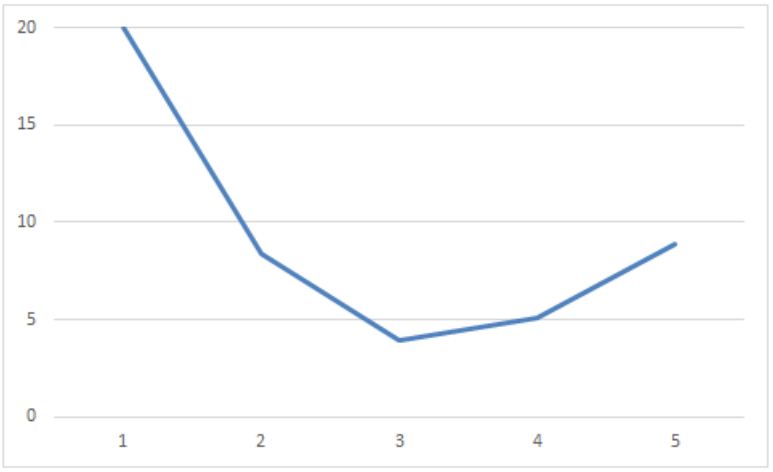
For each overlap, the subfunction size was calculated to maintain a constant problem length(N), using the following formula: $k = \frac{o(M-1)+N}{M}$. The value of k was then rounded to the nearest integer, which then can result in small variations of N. Again 10 instances are generated for each input parameter configuration, where LT-GOMEA performs 20 runs for each instance. For overlap values of 0 and 1, Lt-GOMEA performed well, however, when the overlap is greater than 2, LT-GOMEA struggles to solve problems when a population of 600 is used. Note that LT-GOMEA performs well when the population is higher. When instances of (N=47, M=6, k=12, o=5) were tested with higher populations, LT-GOMEA was able to frequently solve them when the population was above 3000.

7.2.2 Results for Random Codomain

The input parameter configurations and population size (600) used in this experiment were identical to those used in the previous experiments. For input parameter configuration 10_8_0, the global optimum was found in every run and instance, resulting in one funnel for each instance. Most nodes have a strength of 1. In figure 7.10 can be seen that LT-GOMEA performs the worst for configuration 19_8_4. So unlike the deceptive-trap codomain where LT-GOMEA fails the most on an overlap of 2, for the random codomain, this is for an overlap of 4. Also, the number of vertices (nodes and vertices mean the same thing) in the random codomain exceeds the number of vertices for the deceptive-trap non-neutral codomain except for configuration 10_8_0 (figure 7.11).



(a) Successes per generated instance of the given input parameter configuration within the random codomain.



(b) The y-axis represents the average successes over 20 runs per configuration. The x-axis represents the configurations in the order given in the bar diagram.

Figure 7.10

		Random				Deceptive-trap neutral				Deceptive-trap non-neutral		
		Sinks	Sucesses	Vertices		Sinks	Sucesses	Vertices		Sinks	Sucesses	Vertices
10_8_0	exp0	1	20	85	exp0	1	20	171	exp0	3	18	103
	exp1	1	20	82	exp1	1	20	173	exp1	1	19	99
	exp2	1	20	83	exp2	1	16	196	exp2	1	20	99
	exp3	1	20	85	exp3	1	16	181	exp3	1	20	97
	exp4	1	20	92	exp4	1	13	185	exp4	1	18	96
	exp5	1	20	85	exp5	1	17	209	exp5	2	14	95
	exp6	1	20	82	exp6	1	20	164	exp6	1	19	98
	exp7	1	20	82	exp7	1	18	169	exp7	1	20	100
	exp8	1	20	81	exp8	2	17	195	exp8	2	18	99
	exp9	1	20	82	exp9	1	19	192	exp9	3	16	100
		Sinks	Sucesses	Vertices		Sinks	Sucesses	Vertices		Sinks	Sucesses	Vertices
13_8_2	exp0	1	15	162	exp0	2	0	220	exp0	1	0	62
	exp1	1	12	153	exp1	4	2	74	exp1	1	6	53
	exp2	4	3	150	exp2	4	0	66	exp2	1	15	59
	exp3	8	0	202	exp3	1	0	77	exp3	2	0	47
	exp4	4	11	164	exp4	1	0	92	exp4	4	1	62
	exp5	1	11	211	exp5	2	0	67	exp5	1	0	51
	exp6	1	11	202	exp6	2	0	134	exp6	4	0	53
	exp7	1	5	142	exp7	1	3	144	exp7	4	2	63
	exp8	1	16	148	exp8	1	0	168	exp8	3	4	88
	exp9	1	0	148	exp9	2	1	70	exp9	5	0	54
		Sinks	Sucesses	Vertices		Sinks	Sucesses	Vertices		Sinks	Sucesses	Vertices
19_8_4	exp0	10	4	192	exp0	3	0	168	exp0	1	1	54
	exp1	5	0	169	exp1	2	0	275	exp1	1	20	44
	exp2	8	8	195	exp2	1	14	190	exp2	6	1	80
	exp3	4	11	181	exp3	3	8	272	exp3	2	4	68
	exp4	13	0	203	exp4	3	5	199	exp4	5	0	50
	exp5	8	7	185	exp5	5	10	389	exp5	1	13	68
	exp6	14	0	193	exp6	1	1	52	exp6	1	11	62
	exp7	12	6	205	exp7	3	1	347	exp7	3	1	70
	exp8	12	2	206	exp8	1	7	217	exp8	1	9	44
	exp9	8	1	183	exp9	1	19	229	exp9	1	20	41
		Sinks	Sucesses	Vertices		Sinks	Sucesses	Vertices		Sinks	Sucesses	Vertices
25_8_5	exp0	8	7	173	exp0	1	20	254	exp0	1	20	53
	exp1	6	0	170	exp1	1	9	972	exp1	1	8	67
	exp2	14	2	193	exp2	1	0	4152	exp2	2	2	64
	exp3	5	6	184	exp3	1	20	86	exp3	1	19	54
	exp4	9	1	187	exp4	1	20	119	exp4	1	17	60
	exp5	7	6	194	exp5	1	20	271	exp5	2	3	62
	exp6	14	2	180	exp6	1	9	254	exp6	1	20	44
	exp7	3	15	144	exp7	1	20	299	exp7	2	16	63
	exp8	9	6	189	exp8	4	0	2576	exp8	4	10	70
	exp9	10	6	168	exp9	1	12	300	exp9	3	5	76
		Sinks	Sucesses	Vertices		Sinks	Sucesses	Vertices		Sinks	Sucesses	Vertices
37_8_6	exp0	3	18	145	exp0	1	18	260	exp0	1	20	61
	exp1	7	3	156	exp1	1	10	180	exp1	2	17	55
	exp2	5	4	154	exp2	1	20	114	exp2	1	20	54
	exp3	3	4	167	exp3	1	13	217	exp3	1	20	43
	exp4	3	14	145	exp4	1	0	712	exp4	1	20	47
	exp5	4	16	163	exp5	3	11	447	exp5	1	20	58
	exp6	15	3	162	exp6	1	17	511	exp6	1	13	65
	exp7	7	2	177	exp7	2	3	356	exp7	3	0	51
	exp8	6	11	159	exp8	3	0	940	exp8	1	19	58
	exp9	6	14	168	exp9	1	4	148	exp9	1	20	54

Figure 7.11: This figure shows the number of sinks, successes and vertices for each codomain, input parameter configuration and experiment.

7.3 Discussion

7.3.1 Increasing Overlap for Deceptive-trap Codomain

Increase in Global Optima for the Neutral Deceptive-trap Codomain

The neutrality of the neutral deceptive-trap codomain results in a significant amount of variation in the number of nodes and edges as the overlap increases. This is due to the presence of multiple best solutions in a single LT-GOMEA generation. The number of

edges can increase rapidly, especially when the best solutions of the current generation did not improve, which leads to edges between all non-improving solutions. This can be observed by big plateaus within LON and in figure 7.3, where the number of global optima increases with overlap. These are plateaus that contain either the global optima or sub-optimal solutions. The big question is, why do the number of global optima and the number of equal solutions per generation increase when the overlap increases? There is no explanation found of why the neutrality of the landscape would increase when the overlap does, but there is an idea for the increase in solutions with equal fitness. As the overlap increases, the number of subfunctions also increases. The next experiment in section 8.3.1(figure 8.9), shows that when there are more subfunctions due to the increase in overlap, there is a higher probability that there will be subfunctions in the global optimum that do not have the same substring as the optimal substring for that subfunction. The optimal subfunction's substring refers to a substring of a subfunction that provides the most fitness, while the deceptive subfunction substring refers to the substring of the deceptive attractor. It is important to note that the deceptive attractor substring refers to the substring of a single subfunction, while a solution that is deceptive refers to the overall problem. There is only one deceptive and one optimal substring per subfunction which both have a unique fitness, but there are many different subfunction substrings with equal fitness in between (for $k > 1$). Because the substring fitness is established by the hamming distance relative to the global optimum, and the deceptive substring has the maximum hamming distance, but for each hamming distance in between, there are multiple substrings that produce this hamming distance. It is believed that when the global optimum includes more subfunctions that do not contain the optimal or deceptive substring, this can lead to an increase in global optima with the same fitness. The same goes for when LOs are considered that contain sub-optimal substrings. Because, when there are more LOs considered by LT-GOMEA that contain sub-optimal substrings, meaning other substring values than the deceptive or optimal substring, the probability increases that these overlapping subfunctions produce solutions with equal fitness because there are more subfunction substring candidates that can be part of those LOs. This finding explains the observed increase in the number of global optima and equal solutions found per generation with an increase in overlap. However, this does not necessarily mean that the search space is more neutral, but rather that there are just more equal solutions that turn out to be LOs or global optima.
An issue being observed within the LONs for neutral codomains is that the neutrality of the landscape sometimes leads to a bouncing effect between nodes with equal fitness in a plateau. This is because different nodes are interchangeably present in the set of best solutions for many generations and do not stop because the stopping criteria are not met, causing the node strengths to increase frequently without an improvement in fitness. The LON can then sometimes show unreasonably large nodes.

Analyzing Search Difficulty in the Deceptive-trap Codomain with Increasing Overlap

The least number of successful LT-GOMEA runs is found with an overlap of 2, for both deceptive-trap codomains. The instances with an overlap of 0, were clearly the least difficult to solve. The trend for average successes is similar for the two codomains, but they deviate slightly for an overlap of 6. As explained more thoroughly in section 8.3.1, it is believed that with an increase in overlap, there are fewer subfunctions within the global

optimum that have the optimal substring (fewer subfunctions with the optimal substring in relation to the total number of subfunctions). When the overlap increases, not all overlapping subfunctions can have the optimal substring, as the optimal substrings may conflict for the overlapping bits between subfunctions. Also, the increase in subfunctions with an increase in overlap causes more LOs to exist (more peaks and hills), which can enhance the process of finding deceptive solutions because there are smaller steps needed to move between LOs. Still, the problems with 0 overlap are easy to solve and the search difficulty is clearly the highest for an overlap of 2. It is believed that the overlap of 0 is easy to solve, because of linkage learning. When there is 0 overlap, there are only 2 subfunction substrings that are attractive, namely, the optimal and deceptive substring. Lt-GOMEA is highly adequate to recognize these building blocks for 0 overlap, because there is not much diversity within these subfunction substrings. With a large enough population, the optimal substrings will be present in the population. The blocks are easily recognizable, so it is not necessary for a large number of these optimal substrings to be present in the population for them to be utilized. It is believed that the counteracting changes in search difficulty, one being the increase in diversity of building blocks with increasing overlap, making it more difficult for linkage learning to identify these blocks, and the second one being the decrease in deceptiveness at higher overlap (less optimal substrings in global optima and closer LOs), contribute to the dip in performance for an overlap of 2.
In 7.11 can be seen that for overlaps 5 and 6 there are LONs with an exceptionally high number of nodes for the neutral codomain, which all do result in no successes. These influence the average successes substantially and could cause instances of the generated problems with the same input parameters to vary a lot in difficulty and explain the differences in difficulty between the deceptive codomains. The number of global optima increases substantially when the overlap surpasses an overlap of 3. It is believed that a large number of global optima do not necessarily make the search process easier, because the problems generated with the same input parameters for the non-neutral codomain are more successful for a higher overlap. Instances with an exceptionally high number of nodes appear more difficult compared to other instances generated with the same input parameters. The global optimum may be difficult to locate because LT-GOMEA becomes stuck on large plateaus, which are highly connective and contain many different solutions with the same fitness. The solutions in such a plateau may be similar in structure and deceptiveness, reducing the diversity in the population and making it harder to escape from to find better optima. This could explain why LT-GOMEA performs better with high overlap for the non-neutral codomain.

7.3.2 Random vs Deceptive neutral vs Deceptive non-neutral

For all the codomains, the configuration with an overlap of 0 is the least difficult to solve. For the other overlap values, the instances for each input parameter configuration seem to differ a lot in difficulty. Many of the difficult instances contain attractors that dominate the population within a few generations, as can be seen in 7.7b. In the problem of figure 7.7a, the large attractor was beneficial in generating the global optimum. However, when this is not the case, finding the global optimum becomes very difficult. This could explain the differences in difficulty within each input parameter configuration, where per instance there are differences in deceptiveness. This also can be observed within the random codomain. While the random codomain is not specifically designed to produce deceptiveness, the randomness can always produce deceptive subfunctions. And

as can be learned from how fast an attractor can dominate a population, there only has to be a small deceptive portion of the solution, to make a problem difficult to solve for LT-GOMEA. Note that this applies to problems with an overlap greater than 0. For an overlap of 0, linkage learning becomes more effective because there is less diversity in the population, making it easier to solve deceptive problems (when the population is big enough). For both the deceptive and random codomains the search difficulty increases with overlap and then decreases. Van Driessel [17] questioned whether the decrease in search difficulty for deceptive problems was due to the increase in global optima for the neutral deceptive-trap codomain at higher overlap. However, this decrease in search difficulty is also observed in non-neutral deceptive-trap codomain. Additionally, instances with a large number of global optima, when generated with the same input parameters, are not necessarily the easiest to solve. It is also the case for the random codomain, although the decrease in difficulty is more noticeable for the deceptive codomain. For the random codomain, the decrease in deceptiveness for increasing overlap is believed to be primarily caused by smaller steps that have to be taken to traverse local optima, while for the deceptive codomain it is shown that in addition to this, the deceptiveness decreases significantly because the optimal subfunction substrings become less present. This difference is thought to play a role in why the problems with high overlap for the deceptive comdomains are significantly easier than for the random codomain. Also, the increase in LOs is much higher for the random codomain than for the deceptive-trap non-neutral codomain, this can also be noticed by the difference in number of vertices for the problems of higher overlap. There seems to be a balance in, to what extent more LOs are beneficial, but the next experiment will go deeper into this. The random codomain can have multiple local optima for subfunction substrings when only looking at the substring of one subfunction. For the deceptive codomain, when the optimal subfunction substring is neglected there is only one local optimum and this is the deceptive attractor substring. Otherwise, there are two, and this is probably still less than for the random codomain. This does not mean that only the deceptive attractor substring (or optimal subfunction substring) is present in the global optima, but just that there are probably fewer local optima in the whole search space than for the random codomain. Note that a local optimum for a subfunction's substring means something different than a local optimum for the entire solution.
For the deceptive codomains, the smaller diversity caused by the abundance of deceptive subfunction substrings within the population can actually aid linkage learning in identifying the correct building blocks, which can then be used to select less common subfunction substrings that are part of the global optimum.

7.3.3 Usefulness of LON

This section discusses the practical uses and benefits gained by using the method described in this research to generate LONs for LT-GOMEA. It is important to note that the insights and explanations described in the previous section were greatly aided by the use of LONs. The LONs provide an overview of the search process, and this is beneficial to recognize important features. Additionally, this method allows for keeping track of useful metrics that can be used to look into these features. For example, the LONs show which nodes are highly attractive, these nodes can then be looked into about what their evolution is within the population for each generation. This gives insights into why LT-GOMEA prematurely converges, on the other hand this attractor could be beneficial for finding

the global optimum when it shares a lot of blocks with similar bits.
The vizualized LONs are especially used for comparing the instances generated with the same input parameters. These vizualized LONs showed that there can be big differences between these instances, for example in terms of deceptive attractor occurences. This raised the question of why these instances are so different for problems generated with the same input parameters. Van Driessel et al [17], mainly focused on the problems where LT-GOMEA was successful, but this ignored the fact that the problem generated by the same input parameters show differences in difficulty. The multiple LT-GOMEA runs done for generating one LON showed that these difficulties are not because of an accidental failure for not learning the right building blocks, but that some problems (with the same input parameters) are significantly more difficult than others. This insight was crucial for looking into the number of optimal subfunction orderings within the global optimum for the deceptive-trap codomain, as well as looking into the difference in plateau sizes for the neutral deceptive-trap codomain.
For understanding the differences between instances for different input parameters, the metrics such as number of vertices and of course the number of successes are useful, but this is where it becomes more difficult for the population-based method of LON creation. The LONs are incomplete in the sense that they only target the best solutions within a generation, which can give limited information and this is where the next experiments helped by understanding more about the difficulties posed by the problems generated by CliqueTreeMk. The attractors which are the best solutions in the LON, play a major role, because they can either aid the search for the global optimum, or disrupt the search by taking over the population. But they are not always the reason why the global optimum is found. Also, there can be attractors which are abundant in the population, but they are not the best solution and are therefore not present in the LON. This gives uncertainties for what the edges and therefore funnels exactly tell. For example, in the LON of figure 7.7a, there is clearly a relationship, the global optimum has only been found after the large attractor, but this is not always the case. The next experiments will use ILS and a different implementation of LT-GOMEA to generate LONs where the edges are a direct result from changes within previous solutions, therefore they may provide more interesting insights.

Chapter 8

Experiment: Increasing Overlap for LT-GOMEA on a Hill-climbed population

This experiment is executed to investigate whether LT-GOMEA can learn the right building blocks when only one initial population of hill-climbed (best-improvement) solutions is used.

8.1 Setup

The experiments are done for the same CliqueTreeMk configurations as used for the former LT-GOMEA experiment. The deceptive-trap non-neutral and random codomain are used. Note that the instances are newly generated, so they are not exactly the same as in the previous experiment. The building blocks are learned from a population, where the solutions are randomly generated and improved by hill-climbing (BILS). The building blocks are then used to optimise a solution. The population size is $P = 5000$, which is substantially larger than the population size used in the previous experiment, but this is necessary because unlike in regular LT-GOMEA, where the building blocks are computed for each generation, here it is learned once. The right building blocks cannot be learned from a much smaller population when only hill-climbing is used for optimizing the solutions. Hill-climbing these 5000 solutions is by far the most time-consuming process, in comparison to the time it takes to learn the building blocks and to optimise a solution with these learned building blocks. When these building blocks are learned, many LT-GOMEA runs can be performed similarly as to ILS. In the case of ILS random mutations are used to improve solutions (together with BILS), and in this experiment, building blocks are used to improve a solution. 600 runs are performed per instance of a configuration. Nodes are trying to improve by inheriting bits from a parent, these bits are indicated by the building blocks. A random parent is chosen out of the population of hill-climbed solutions. When all the building blocks are tried for n times without an improvement, the non-improved node becomes an attractor and the run stops. In this experiment $n = 300$. A higher n did not give more improvement, only a longer search process. The node strength is only increased when the search arrived at this solution previously to another solution. When a solution did not improve, the strength does not increase. Algorithm 4 shows the pseudo code for the search process and is also used to recognize the nodes and edges. The function $inherits()$ takes 3 arguments, the current solution, the building block and

the parent. It returns the solution that inherited bits from the parent indicated by the building block.

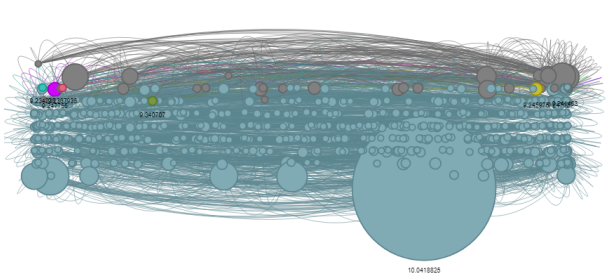
Algorithm 4 LON creation

```
LONs ← {}
Edges ← {}
Attractors ← {}
Population ← {}
for 1 to 5000 do
    s ← random solution
    s' ← BILS(s)
    Population.add(s')
end for
building_blocks ← learn_building_blocks(Population)
n ← 300
for 1 to 600 do
    s ← random solution
    for i from 1 to n do
        for BB in building_blocks do
            parent ← Population[random.index]
            s' ← inherits(s, BB, parent)
            if f(s') ≥ f(s) and s ≠ s' then
                LONs.add(s')
                Edges.add((s, s'))
                s ← s'
                i ← 1
            end if
        end for
    end for
end for
```

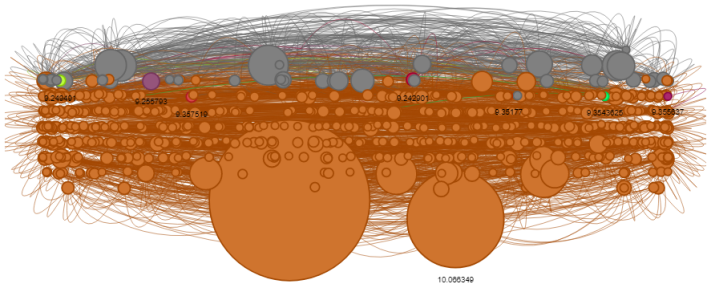
8.2 Results

In the next section, some LONs for the deceptive-trap non-neutral codomain will be shown, starting with configuration 10_8_0. What can be noticed for all LONs is that the nodes are concentrated on certain y-values, which can be observed by somewhat horizontal lines formed by the nodes. This is caused by the deceptive-trap codomain, where in the neutral codomain plateaus can be formed. Many solutions show almost equal fitness because without the random increment it is a neutral landscape. For most instances of configuration 10_8_0, LT-GOMEA found the global optimum for more than 200 runs. The node strength in these experiments indicates in how many runs this node (solution) has been reached. Most LONs show a resemblance with some other LONs within this input parameter configuration. Three different-looking LONs are featured in figure 8.3 and each LON of this configuration shows a resemblance with one of the featured LONs of this figure. As can be seen in figure 8.2, experiment 0, 2, 5 and 6 show a similar number of successes (around 420 runs out of 600). All the LONs of these experiments look similar to figure 8.1a. There is one big funnel and one big attractor with a substantially higher

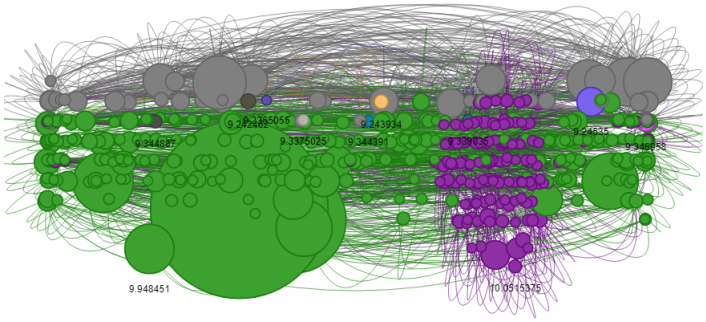
strength compared to the other nodes. This big attractor is also the global optimum in these experiments. These LONs show no signs of difficulty for LT-GOMEA. There are some sinks that have very low fitness and strength, but the basins of these sinks do overlap with other basins (except for the sink nodes). In other words, only grey nodes and the sink itself belong to these basins. The grey nodes on top of the LONs in this input parameter configuration occur because these small funnels overlap with the main funnel and other smaller funnels. Then there are experiments that look similar to figure 8.1b, these are experiments 1, 3, 7, 8 and 9. They are very similar to the LONs of experiment 5, the only clear difference is that there is another big attractor, which has a higher strength than the global optimum, but they both belong to the same funnel. This bigger attractor is not necessarily the second-best optimum. The number of different bits between the big attractor and global optimum is in all five experiments 8 bits, which is equal to the subfunction size. The node strengths of the global optimum and the big attracter are similar in all five experiments, where the global optimum has a strength of about 230 and the big attractor of about 350. Figure 8.1c shows a different looking LON which only has been observed in experiment 4. There are 2 big funnels, where the biggest funnel does not contain the global optimum and the smaller one does. There is still one big attractor but it is not in the same funnel as the global optimum. This big attractor has a strength of 192 and the global optimum a strength of 7. These two solutions have a difference of 16 bits, which is 2 times the subfunction size. There are 4 nodes that differ 8 bits with the global optimum, where one node has a strength of 94, but it is in the green funnel. The other three have a strength of less than 23 and are all three located in the purple funnel (the funnel of the global optimum).



(a) Experiment 5.



(b) Experiment 9.



(c) Experiment 4.

Figure 8.1: Configuration 10_8_0, from the Deceptive-trap non-neutral codomain

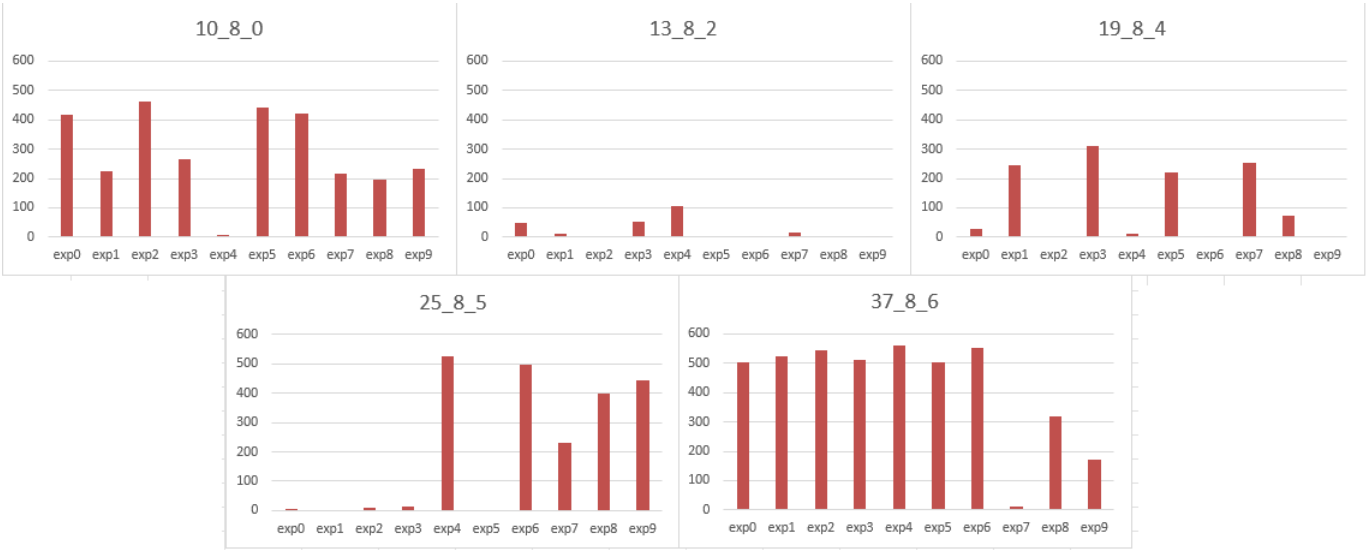
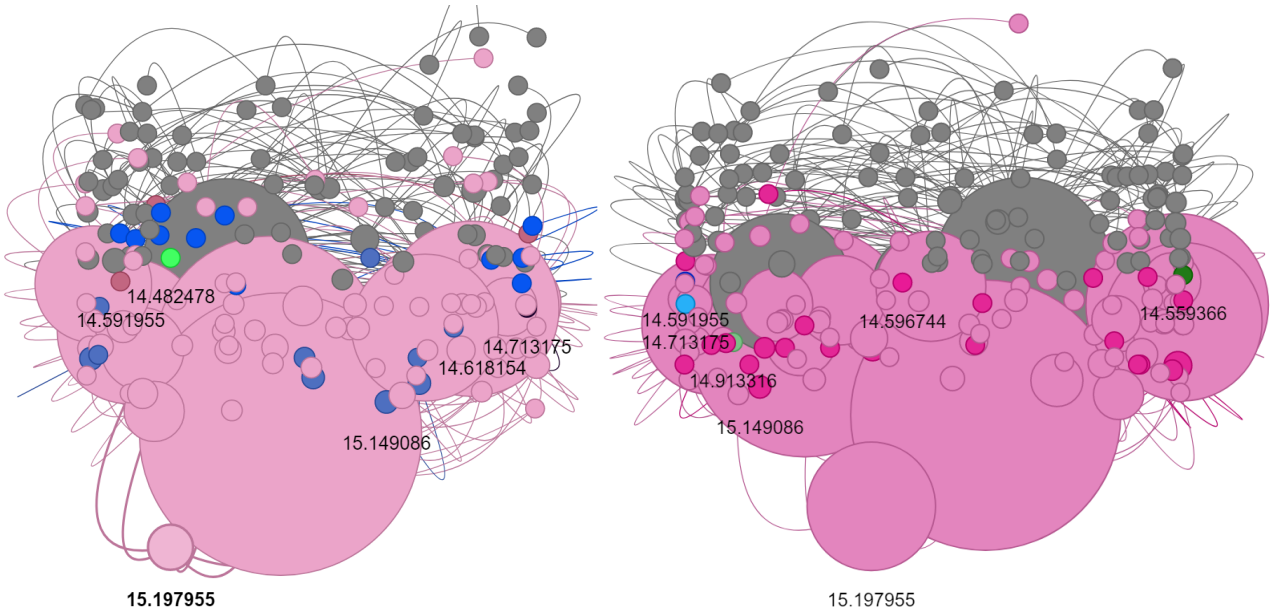


Figure 8.2: Number of successes for each experiment of each configuration.

In a two of the LONs for configuration 19_8_4, namely for experiments 0 (figure 8.3a) and 8, the global optimum was found, but not that often. These LONs both show one big funnel where the global optimum is the sink and there are a lot of big attractors that have edges towards the global optimum. These attractors belong to this funnel, as can be seen in figure 8.3a. The hypothesis is that the global optimum should be easier to find when it belongs to the main funnel and it should also be easier to find when it is widely connected with big attractors. Both these requirements seem to comply, but it still seems to have a hard time finding the global optimum. Therefore the experiments were repeated, to see if these new results show disagreements with the previous results. The tables of Figure 8.4, show the number of sinks, successes and vertices for both attempts. The results show that the number of successes and vertices are similar for most experiments except for some. Namely, the successes between both attempts differ quite a lot for experiments 0, 2 and 8. And all three of these LONs show features that seem characteristic of easily solvable problems. The LONs all look similar in structure (global optimum belongs to the main funnel, and is connected to big attractors), the number of vertices is about equal and the prominent funnels have about the same size. The LONs also contain the same attractors with similar strengths. The difference, however, for the mentioned experiments, is the number of times that the global optimum has been found. To summarize these three experiments should be easy to solve, and the results show that they are easy to solve for one of the attempts. This indicates that LT-GOMEA failed in one of the attempts to learn the correct building blocks. Experiment 1 shows a high number of successes on the first try and experiments 0 and 8 show more successes on the second try.
It is noteworthy that the number of vertices varies significantly between all the different problems of this configuration. But the number of vertices for the same problems is similar for both attempts. This indicates that repeating the same problems results in approximately the same number of vertices. There is no relationship found between the number of successes and the number of vertices within problems generated by the same input parameters. For example, on the second try, experiment 3 has 326 successes and 181 vertices and experiment 8 has 456 successes and 905 vertices. They both are successful but vary a lot in the number of vertices. The cases that were not successful for both attempts (experiments 2, 4, 6 and 9), also vary in vertex count. These non-successful

cases have in common that the global optimum if it has been found is not part of the main funnel, this is different from experiments 0, 2 and 8.



(a) (b)

Figure 8.3: Different LONs for experiment 0 and Configuration 19_8_4, within the Deceptive-trap non-neutral codomain

19_8_4	Sinks	Sucesses	Vertices	19_8_4	Sinks	Sucesses	Vertices
exp0	6	29	191	exp0	7	118	219
exp1	3	244	222	exp1	4	137	200
exp2	8	0	315	exp2	16	0	322
exp3	5	310	156	exp3	6	326	181
exp4	42	13	882	exp4	34	2	841
exp5	10	222	442	exp5	11	210	463
exp6	15	1	278	exp6	13	0	253
exp7	8	252	278	exp7	4	303	304
exp8	21	72	887	exp8	28	456	905
exp9	35	1	1605	exp9	29	3	1533

Figure 8.4: The experiments are done twice for the same instances of configuration 19_8_4.

For overlap higher than 3, there are experiments that have an exceptionally high number of vertices, and those experiments are not successful. For configuration 19_8_4 this is experiment 9 and for configuration 25_8_5 these are experiments 1, 3 and 5 and for configuration 37_8_6 this is experiment 7.
Not all unsuccessful experiments have this exceptionally high number of vertices. For instance, experiment 2 of configuration 25_8_5, where the LON is shown in figure 8.5. The blue funnel contains the global optimum. There is a small region containing the blue funnel which is fully disconnected from the rest of the LON. Most of the vertices are part of the brown funnel. It is hard to observe in the representation of figure 8.5, but all the other small funnels, are also fully disconnected from the brown funnel. The LON shows that it is hard to escape from the large region of attraction located in the brown funnel. And it is necessary to escape this region to find the global optimum, and therefore this

LON clearly shows the trouble that the search is having for finding the global optimum. This phenomenon where the global optimum is not connected to the biggest region of attraction can be observed in lots of failure cases. So the two cases that often resolve in failure are: one where the number of vertices is exceptionally high and the other where the global optimum is badly connected. For experiments 1,3 and 5 of configuration 25_8_5 the number of vertices seems to be exceptionally high and for experiments 0 and 2, the global optimum is not connected to the big regions of attraction.

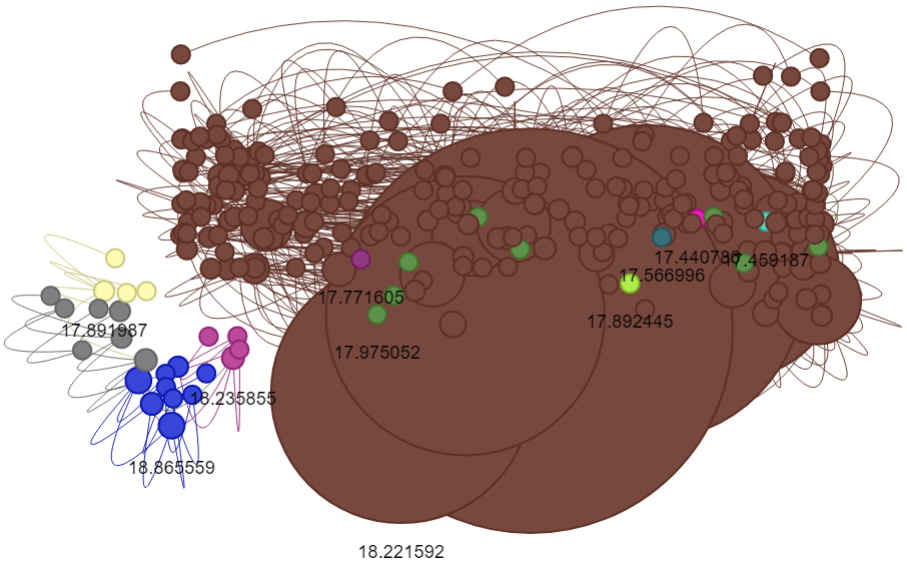
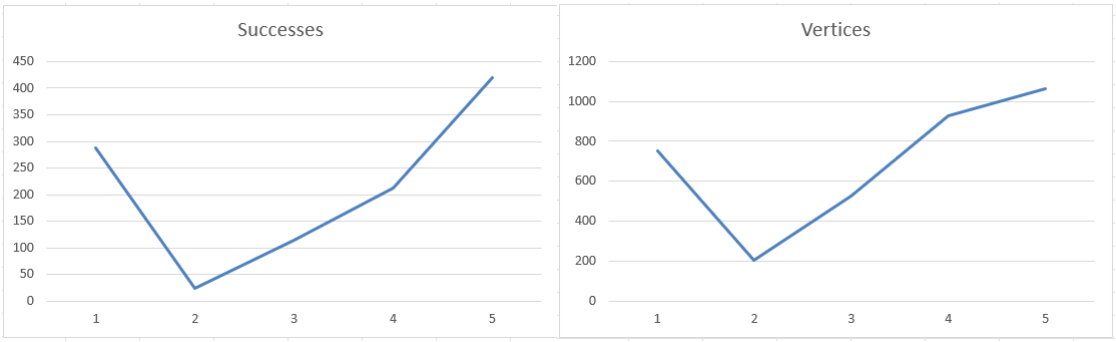


Figure 8.5: Experiment 2 of configuration 25_8_5. The blue funnel contains the global optimum.

For almost all experiments of configuration 37_8_6, except for experiment 7, the global optimum is the biggest attractor (second biggest for experiment 8) and the funnel of the global optimum is also the biggest funnel. This also reflects the high number of successes which are achieved by LT-GOMEA for this configuration. Experiment 9 also shows a second funnel which contains large attractors, this is also reflected by the fewer amount of successes.
As can be observed in figure 8.6, the average number of successes shows the same trend as the average number of vertices. The successes also follow the same trend as for the implementation of LT-GOMEA from the previous experiment within the deceptive-trap non-neutral codomain. The correlation between successes and vertices cannot be recognized within the instances of the same configuration, because an exceptionally high number of vertices is often associated with a low number of successes. There are both plenty of successful cases where the vertices are on the higher and lower end within that configuration.



(a) Average number of successes for all configurations

(b) Average number of vertices for all configurations

Figure 8.6: Average number of successes and vertices for the configurations in the deceptive-trap non-neutral codomain for LT-GOMEA where the building blocks are learned from a hill-climbed population.

10_8_0	Sinks	Sucesses	Vertices
exp0	11	416	826
exp1	11	226	746
exp2	11	461	819
exp3	9	266	742
exp4	11	7	560
exp5	8	442	809
exp6	6	422	810
exp7	11	217	747
exp8	10	195	682
exp9	8	232	765

13_8_2	Sinks	Sucesses	Vertices
exp0	13	49	246
exp1	11	10	223
exp2	14	0	152
exp3	13	52	208
exp4	9	107	159
exp5	11	0	113
exp6	11	1	164
exp7	13	17	164
exp8	9	0	169
exp9	15	1	463

19_8_4	Sinks	Sucesses	Vertices
exp0	6	29	191
exp1	3	244	222
exp2	8	0	315
exp3	5	310	156
exp4	42	13	882
exp5	10	222	442
exp6	15	1	278
exp7	8	252	278
exp8	21	72	887
exp9	35	1	1605

25_8_5	Sinks	Sucesses	Vertices
exp0	29	3	736
exp1	31	0	1121
exp2	10	9	299
exp3	82	12	3725
exp4	6	524	450
exp5	16	0	1138
exp6	15	497	424
exp7	10	230	348
exp8	12	397	748
exp9	5	443	287

37_8_6	Sinks	Sucesses	Vertices
exp0	15	503	707
exp1	27	526	1039
exp2	28	543	1221
exp3	14	511	751
exp4	2	562	383
exp5	32	502	1413
exp6	12	554	965
exp7	51	12	1995
exp8	23	320	1020
exp9	21	171	1133

Figure 8.7: Number of sinks, successes and vertices for each experiment for each configuration of the deceptive-trap non-neutral codomain.

Experiments are also done for the random codomain and the same configurations are used. The building blocks are once again learned from hill-climbed solutions and algorithm 4 is used for creating the LON. The results are shown in the tables of figure 8.8. Only configuration 10_8_0 shows a high number of successes. Configuration 13_8_2 shows some successes but configurations with higher overlap hardly show successes. The number of vertices decreases for an increase in the overlap. The LONs are not shown because of the high number of vertices, and it is hard to say something convenient about the LONs when the search fails that often. For an overlap of 0, it is obvious to see that the search did not struggle to find the global optimum and the global optimum has been found in

most of the runs, and only one sink exists.

10_8_0	Sinks	Sucesses	Vertices
exp0	1	600	12994
exp1	1	600	10251
exp2	1	600	11314
exp3	1	600	10945
exp4	1	600	11860
exp5	1	473	12571
exp6	1	600	12618
exp7	1	598	12126
exp8	1	600	12262
exp9	1	600	11757

13_8_2	Sinks	Sucesses	Vertices
exp0	241	4	11718
exp1	221	3	11491
exp2	132	96	12042
exp3	248	0	10549
exp4	255	4	11190
exp5	315	13	10968
exp6	167	25	11172
exp7	253	17	10588
exp8	323	1	11797
exp9	176	14	10826

19_8_4	Sinks	Sucesses	Vertices
exp0	481	7	7974
exp1	408	2	8544
exp2	488	3	8165
exp3	470	0	8157
exp4	364	5	8788
exp5	485	2	8169
exp6	425	4	9045
exp7	522	2	7240
exp8	294	21	9237
exp9	375	0	9140

25_8_5	Sinks	Sucesses	Vertices
exp0	408	8	7252
exp1	465	1	6646
exp2	448	2	7525
exp3	301	12	8422
exp4	423	2	7881
exp5	428	3	7622
exp6	383	12	7858
exp7	512	3	7880
exp8	446	0	8478
exp9	472	5	6951

37_8_6	Sinks	Sucesses	Vertices
exp0	514	0	5209
exp1	462	2	6827
exp2	344	18	7480
exp3	464	5	6138
exp4	551	0	6442
exp5	481	3	5270
exp6	466	0	6604
exp7	535	1	5641
exp8	429	5	6848
exp9	259	6	7930

Figure 8.8: Number of sinks, successes and vertices for each experiment for each configuration for the random codomain.

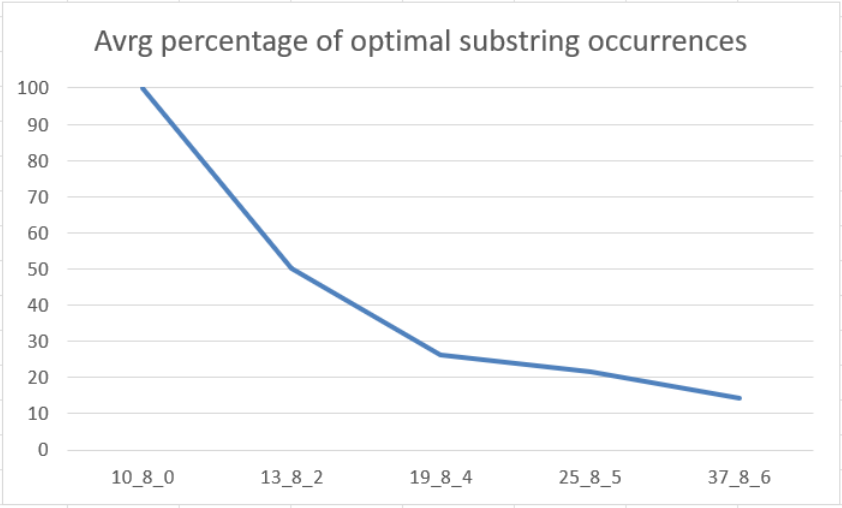


Figure 8.9: Average percentage of optimal subfunction substring occurrences over 10 instances per input parameter configuration.

8.3 Discussion

In this experiment, an implementation of LT-GOMEA is used, where the building blocks are learned once over a population of hill-climbed solutions. The CliqueTreeMK parameters of the previous experiment are used. The population size has to be substantially larger, than for the previous LT-GOMEA implementation, to get successful runs. This makes sense because in the previous LT-GOMEA implementation, the building blocks are learned for each generation. In this experiment, the building blocks are learned once

so the population has to contain enough information to learn the right building blocks instantly and the population has to contain solutions that have optimal values for the blocks. The LONs for this implementation are considerably more complete than the LONs of the previous implementation because all the solutions that are considered and make an improvement are represented in the LON.

8.3.1 Increasing overlap for the deceptive-trap non-neutral codomain

For the deceptive-trap non-neutral codomain, the trend of average successes is similar for both LT-GOMEA implementations (figure 7.2 and 8.6a). Expectantly the complete LONs of this implementation can give more insight into regular LT-GOMEA, since the trend is similar. The experiments of configuration 19_8_4 are performed twice, to see if they produce different LONs and successes for the same instances. Figure 8.6 shows that the number of vertices and successes show a connection, whereas a higher number of vertices shows more successes. This is not the case for instances of the same configuration. The expectation is that there are more Local Optima when the overlap increases, because the number of subfunctions increases. For the deceptive-trap codomain with 0 overlap, there are only 2 subfunction configurations that occur in the population, because the hill-climber, either climbs to the deceptive attractor or to the optimal subfunction configuration. When the overlap increases, there are more local optima that the local search can end up, so there will be a more diverse population. Still, the number of vertices in figure 8.6b shows that there are fewer solutions considered for an overlap of 2 than for an overlap of 0. LT-GOMEA finds it more difficult to learn the building blocks from a diverse population because diversity causes more subfunction substrings to be present and this makes the blocks less distinguishable. The expectation is that if there is less overlap, then the global optimum will have more subfunction substrings which are optimal, meaning the global optimum contains more subfunction substrings that have the highest fitness. However, when there is more overlap, it becomes less likely that the optimal substrings will be present in the global optimum. This is because when a bit overlaps for multiple subfunctions, there is an increased probability that the optimal subfunction configurations will disagree for that bit, which can cause at least one of the subfunctions to settle for a suboptimal substring. Note that suboptimal substrings can be present in the global optimum, because the global optimum does not necessarily only exist out of the subfunction substrings with highest fitness.

Optimal Subfunction Substring Occurences

The expectation about the optimal subfunction substring occurrences is confirmed by analyzing how often the optimal substring occurs in the global optimum for each CliqueTreeMk configuration. For an overlap of 0 each subfunction has the optimal subfunction substring, because if there is no overlap, there is no interference of overlapping subfunctions, and each subfunction will therefore have the optimal substring in the global optimum. In this case of 0 overlap, this is for 10 subfunctions. For an overlap higher than 2, the optimal subfunction substrings within the global optimum occur an average of 5 times (out of 10 instances per CliqueTreeMk configuration) in the global optimum. But note that with higher overlap there are far more subfunctions in total, so if the number of optimal substrings stays at 5, then the percentage of optimal substrings within all the subfunctions decreases vastly for higher overlap. This can be observed in figure 8.9. The

increase in local optima with higher overlap also makes the optimal substrings more accessible and therefore less deceptive. Also, the variance in difficulty for cases of the same CliqueTreeMk configuration (same overlap) show a relationship. When the number of optimal substrings within the global optimum is exceptionally high, the problem becomes more difficult and if it is exceptionally low the problem is easier. Some instances for the configuration of 37_8_6 contain 1 optimal substring within the global optimum but it also happens that the global optimum contains 8 optimal substrings. For the instances where the number of optimal subfunction substring occurrences is closer to the average, no relationship in difficulty is found. It is believed that this is caused by not only the number of optimal substrings, but also how the overlapping subfunctions aid the search process for finding the optimal substrings. If the correct overlapping subfunction substrings, meaning the subfunction substrings within the global optimum are relatively easy to find for the search algorithm, this will make it so, that the optimal substring is also easier to find. The expectancy is that this is especially the case when the overlapping subfunction substrings are the deceptive attractor substrings. Also, when problems are hard for higher overlap, and there are almost no optimal subfunction occurences, then it is believed that the deceptiveness is not produced by solely the deceptive attractor substring, but more by the increase in overlap and local optima that randomly generate more deceptiveness. Because the deceptive attractor substring cannot be deceptive anymore if there are almost no optimal substrings present in the global optimum. So when there are difficult instances with higher overlap, this is probably not only caused by the intended deceptiveness of the deceptive-trap codomain.

Explanation of LT-GOMEA performance on neutral codomain

Lt-GOMEA is still able to perform similarly to the first experiment while now the building blocks are only learned from hill-climbed solutions. The increase in local optima for higher overlap is not so much, that an evolving population is needed to learn the right building blocks. The increase in local optima can also be a benefit for LT-GOMEA, by smaller changes that are needed to move between them. The decrease in optimal subfunction substrings in the global optima decreases the deceptiveness of the problem. When there is less overlap, the population is less diverse and this is advantageous for learning the building blocks. On the other hand, when there is more overlap, the problem becomes less deceptive and moving between LOs becomes easier. The hypothesis is that the dip in performance for an overlap of 2 is a result of the two aforementioned effects, where lower overlap makes it easier to recognize the building blocks and higher overlap reduces the deceptiveness of the problem. For an overlap of 2, the building blocks are harder to learn than for an overlap of 0, and the problem is still quite deceptive, which probably causes the difficulty for LT-GOMEA.
When the overlap is increasing, linkage learning seems to play a diminishing role because of the more diverse population, however, it is thought that not only linkage learning is important for finding the right subfunction substrings belonging to the global optimum, but also plays an important role in preserving the subfunction substrings that are found. This is why for the random codomain and higher overlap, where it is harder to learn the building blocks because of a diverse population, linkage learning still proves to be necessary.

Difficult Problems

There are two different cases that can be distinguished for difficult problems, and the LONs are a useful tool for recognizing these cases. As shown in figure 8.3, there are instances where LT-GOMEA does not perform well, but the LON reveals that the global optimum is connected to many large attractors, and that the attractors and global optimum are part of the biggest funnel within the LON. In these cases, LT-GOMEA may have failed to learn the right building blocks, but by initializing the population and learning the building blocks again, it may be possible for LT-GOMEA to learn the correct building blocks and improve its performance. These instances may appear to be difficult at first, but the LT-GOMEA together with the population may simply have failed to extract the right information. This is why rerunning LT-GOMEA can be necessary for these cases and can ultimately result in a wide range of successes. Another case exists where LT-GOMEA consistently performs poorly (for instance the problem of figure 8.5), and it is believed that some cases are simply harder due to the high level of deceptiveness and the presence of more optimal subfunction substrings in the global optimum.

8.3.2 Increasing overlap for the random codomain and comparison with deceptive-trap codomain

The data from table 8.8 indicates that the performance of LT-GOMEA drops significantly when the overlap increases for the random codomain. This is likely due to the fact that there are so many local optima for an increase in overlap, that the initial population is simply too diverse to effectively learn the building blocks used for identifying the subfunction orderings present in the global optimum. This can explain the sharp drop in performance as the overlap increases. It also indicates that it is necessary for LT-GOMEA in the random codomain, to improve the population and update the learning blocks. Linkage learning cannot be utilized well enough when the population is too diverse and improving a population is therefore necessary to reduce diversity for problems in the random codomain. This can be the reason why in the previous experiment LT-GOMEA performed much better for higher overlap in the random codomain. For the deceptive codomain, LT-GOMEA can perform well with a population of local optima. This is in line with expectations because the decptive-trap codomain is so defined that there are fewer local optima within the substrings of the subfunctions, namely the optimal substring and the deceptive attractor substring. For the random codomain these can be many more and this explains the sharper increase in local optima for higher overlap for the random codomain. This can also explain the difference in vertices found for the two codomains, with the random codomain having a relatively larger number of vertices.

8.3.3 Usefulness of LONs

In addition to the benefits of LONs mentioned in section 7.3.3, the LONs generated by this implementation provide additional information. They include all of the solutions that made an improvement. Also, the edges are a direct result of altering a particular solution. This results in a more complete and detailed LON that can be used to better understand the struggles of LT-GOMEA or the difficulties of the generated problem. The funnels that contain the global optimum, large attractors, or large basins are particularly effective for analyzing difficulties because they provide the most comprehensive overview of the search in 20 runs. For example, figure 8.5 shows a case where the funnels in the

LON can be useful for identifying challenges faced by the algorithm. The blue funnel contains the global optimum, which is poorly connected to almost every other node in the LON. In contrast, the large attractor nodes in the brown funnel are well connected and may disrupt the search for the global optimum. By analyzing these attractors, it can gain insights into what makes them so attractive, such as the number of deceptive subfunction substrings they contain and the differences between their subfunction substrings and those of the global optimum. Additionally, the number of vertices in the LON can be used to estimate the efficiency of the search algorithm in exploring the search space.

Chapter 9

Experiment: Increasing overlap for ILS

It is interesting to see whether ILS and LT-GOMEA experience the same difficulties when the overlap changes for instances within the used codomains. This can provide insights into whether the challenges are caused by the Mk Landscape or by limitations of the search algorithm. Iterated Local Search (ILS) does not employ linkage learning, so if LT-GOMEA is able to solve the problem easily while ILS struggles, this may suggest the importance of linkage learning or highlight other weaknesses of ILS in comparison to LT-GOMEA. When both LT-GOMEA and ILS face challenges with the same overlap, it is possible that the cause is the same for both algorithms. This shared cause may make it easier to identify the reason for these difficulties.

9.1 Setup

The experiments are conducted on smaller instances compared to those used for LT-GOMEA because ILS performs significantly worse than LT-GOMEA. The CliqueTreeMk configurations are chosen in a way that allows ILS to perform reasonably well, but noticeable differences in performance can be observed for different levels of overlap. Similar to the other experiments, 10 different instances are generated for each CliqueTreeMk configuration. For all configurations a branching of 1 is used. The following table shows the used configurations.

N	M	k	o
30	5	6	0
31	6	6	1
30	7	6	2
30	9	6	3
30	13	6	4
30	25	6	5

The search algorithm and LON creation methods used in the experiments are described in Chapter 4. The experiments are conducted on the random and deceptive-trap non-neutral codomains. The mutation size is chosen such that for the deceptive-trap codomain, with 0 overlap, it is possible for a subfunction that has converged to the deceptive attractor to converge to the optimal substring if the correct bits are mutated. If this is not the case,

the mutations will not change the solution because each solution is hill-climbed and no mutation will ever lead to a different subfunction substring if it has already converged to the attractor substring (for 0 overlap). This is only possible if the bits are mutated in a way that results in a substring with one bit difference relative to the optimal substring. The subfunction size is 6, and the hamming distance between the attractor and optimal substring is 6. In order to reach a substring that has a hamming distance of 1 from the optimal substring, a mutation size of at least 5 must be used. The chosen mutation size is 5 because if the mutation size is too large, there is a higher probability that the mutations will also affect optimal subfunctions. When the difference between the global optimum and the current solution is small, it may be difficult to make the necessary small changes when the mutation size is too large.
For each instance, ILS is repeated for 1000 runs. A run is terminated if there is no improvement after 100 mutation attempts (after mutation solutions are optimized with BILS). The final solution of a run, the solution that did not improve after 100 mutations becomes an attractor.

9.2 Results

The tables and graphs in Figure 9.1 and 9.2 show that the different codomains have a different relationship between overlap, average success rate and average number of vertices. In the deceptive-trap non-neutral codomain, an increase in overlap is accompanied by an increase in both successes and vertices. The configuration with 0 overlap (5_6_0) does not result in any successes for any instances. And the configuration with the highest success rate (25_6_5) shows no instances with low success rates (under 100). All other configurations, however, include instances with low success rates. For the random codomain, the average success rate is similar across all configurations, but the distribution of success rates among instances within each configuration varies. Configurations with higher overlap tend to have more stable success rates, while those with lower overlap exhibit greater fluctuations, for example, in configuration (5_6_0), there is an instance with 997 successes and another with 0 successes. The average number of vertices decreases as overlap increases.

5_6_0	Sinks	Sucesses	Vertices
exp0	16	0	23
exp1	16	0	23
exp2	16	0	23
exp3	15	0	23
exp4	19	0	25
exp5	17	0	24
exp6	20	0	26
exp7	15	0	23
exp8	20	0	26
exp9	17	0	23

6_6_1	Sinks	Sucesses	Vertices
exp0	6	6	19
exp1	7	10	28
exp2	9	1	23
exp3	7	0	30
exp4	7	2	28
exp5	7	13	25
exp6	7	0	35
exp7	6	6	29
exp8	8	0	25
exp9	10	3	22

7_6_2	Sinks	Sucesses	Vertices
exp0	4	37	37
exp1	5	3	41
exp2	3	61	40
exp3	7	5	36
exp4	5	0	39
exp5	3	44	17
exp6	2	45	29
exp7	6	3	59
exp8	2	7	32
exp9	2	155	47

9_6_3	Sinks	Sucesses	Vertices
exp0	3	2	80
exp1	2	38	51
exp2	4	154	62
exp3	2	154	54
exp4	2	97	72
exp5	2	108	51
exp6	3	32	57
exp7	2	84	55
exp8	2	19	55
exp9	2	255	57

13_6_4	Sinks	Sucesses	Vertices
exp0	2	87	61
exp1	2	297	84
exp2	1	612	77
exp3	3	491	95
exp4	1	23	88
exp5	2	25	54
exp6	1	983	101
exp7	1	588	43
exp8	2	752	80
exp9	2	78	83

25_6_5	Sinks	Sucesses	Vertices
exp0	1	710	173
exp1	1	634	134
exp2	1	486	69
exp3	1	943	153
exp4	1	824	117
exp5	1	247	177
exp6	1	685	223
exp7	1	986	131
exp8	1	678	107
exp9	2	588	112

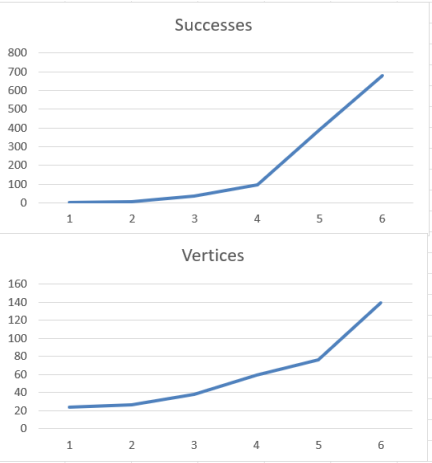


Figure 9.1: Number of sinks, successes and vertices for each experiment for each configuration for the deceptive-trap non-neutral codomain. The configurations are given with three parameters: (M=number of subfunctions,k= subfunction length, o = overlap)

5_6_0	Sinks	Sucesses	Vertices	6_6_1	Sinks	Sucesses	Vertices	7_6_2	Sinks	Sucesses	Vertices
exp0	1	665	2720	exp0	1	958	2559	exp0	2	414	1793
exp1	1	136	2734	exp1	2	281	2084	exp1	1	0	1989
exp2	1	776	2081	exp2	2	139	2216	exp2	2	780	1958
exp3	1	669	2767	exp3	2	198	2559	exp3	1	96	1700
exp4	1	0	1567	exp4	2	364	2524	exp4	2	254	2102
exp5	1	76	2740	exp5	1	799	2312	exp5	1	112	1909
exp6	1	659	2694	exp6	1	0	2138	exp6	1	259	2127
exp7	1	997	2217	exp7	1	465	2358	exp7	3	853	2045
exp8	1	48	2184	exp8	2	75	2535	exp8	1	433	2611
exp9	1	16	2656	exp9	2	332	2222	exp9	1	538	2069
9_6_3	Sinks	Sucesses	Vertices	13_6_4	Sinks	Sucesses	Vertices	25_6_5	Sinks	Sucesses	Vertices
exp0	2	54	1364	exp0	3	237	1201	exp0	1	423	607
exp1	1	759	1760	exp1	3	364	1894	exp1	3	381	1007
exp2	1	393	1587	exp2	1	429	1077	exp2	3	582	1365
exp3	3	294	1764	exp3	1	446	1428	exp3	2	241	986
exp4	2	440	1723	exp4	3	245	1453	exp4	1	390	1102
exp5	3	67	1748	exp5	3	522	1364	exp5	1	363	1053
exp6	3	0	1370	exp6	3	531	1535	exp6	2	584	1050
exp7	1	416	1285	exp7	2	340	1204	exp7	1	215	1139
exp8	3	304	1136	exp8	2	475	1505	exp8	2	870	1518
exp9	1	495	1483	exp9	4	148	1069	exp9	1	484	833

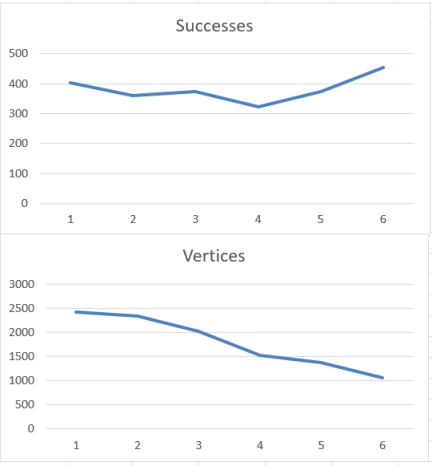


Figure 9.2: Number of sinks, successes and vertices for each experiment for each configuration for the random codomain. The configurations are given with three parameters: (M=number of subfunctions,k= subfunction length, o = overlap)

9.3 Discussion

9.3.1 Deceptive-trap non-neutral codomain

For configuration (5_6_0), it is unlikely that there will be any successes. This is because every subfunction in the global optimum has the optimal substring. In order to change one subfunction, all swaps of a mutation must occur exactly in one subfunction. For a subfunction to start with the correct substring, the subfunction must before BILS differ from the optimum subfunction substring by at most 1 bit. There are a total of $2^k = 2^6 = 64$ subfunction substrings, and the subfunctions that differ from the optimum by at most 1 bit are: $1 + 6 = 7$ (optimum + substrings with hamming distance of 1) substrings. Therefore, the probability that one particular subfunction in the solution starts with the optimal subfunction substring is $p = \frac{7}{64}$. To start with an initial solution where all subfunctions have the optimal substring after local search, the probability would be $\frac{7}{64}^M \approx \frac{1}{64000}$ when M = 5.
The probability of swapping the correct bits for a subfunction that is converged to the deceptive attractor substring is $p = \frac{6}{30} * \frac{5}{29} * \frac{4}{28} * \frac{3}{27} * \frac{2}{26} = \frac{1}{25000}$. These low probabilities make it so that ILS never succeeded to find the global optimum. Initially, it was believed that an increase of LOs by increasing the overlap, would make it more difficult for ILS to find the global optimal because there would be more LOs to consider and a higher chance of getting far from the global optimum. However, it has also there are also more local optima with higher overlap and therefore more local optima between the initial solution and the global optimum. This means that smaller adjustments are needed to improve a solution, which makes it less likely that ILS will get stuck. Despite having a small number of local optima for an overlap of 0, the problem is still difficult for ILS to solve in the configuration (5_6_0) because there are only two possible subfunction substrings when a solution is optimized by local search, and the number of different bits between the local optima is large. This shows that fewer local optima do not necessarily mean that the problem is easier to solve, and linkage learning proves to be neccesary. Additionally, the results from the last experiment showed that with an increase in overlap, there are fewer subfunctions with deceptive substrings in the global optimum, which suggests that the

problem becomes easier to solve especially for BILS.

In summary, the increase in local optima for this codomain for these configurations appears to be advantageous because smaller steps are needed to explore the search space. The increase in local optima is also reflected by the increase in vertices for higher overlap. Additionally, the subfunctions become less deceptive, making it easier for ILS to navigate through the search space in the direction of the global optimum.

9.3.2 Random Codomain

It is surprising that the success rates for the random codomain are similar for each CliqueTreeMk configuration. However, it can be observed that for lower overlap, there is more variation in the success rate. This is likely because, as with the deceptive-trap codomain, ILS struggles with low overlap if the random codomain creates deceptive substrings, and this could make it almost impossible for ILS to find the global optimum for lower overlap. The deceptiveness of higher overlap seems to decrease because local optima are closer together, making the substrings within the global optimum better reachable. This decrease in deceptiveness can be seen in the LONs, as there are fewer instances of large attractors (with higher strength than the global optimum) present for higher overlap in the random codomain. It is worth noting that deceptive substrings can be formed randomly in the random codomain.

It was initially thought that the LOs would increase significantly with higher overlap, making it harder to find the global optimum in the random codomain. However, the number of vertices actually decreases with higher overlap, which is surprising. This may be because more improvements are made through BILS which reduces the number of times that mutations are made, and not all of these improvements are visible as new vertices. Only fully optimized solutions (when BILS find an LO after mutation) are represented as new nodes if they are an improvement on the previous node. Therefore, it is believed that most improvements are made during BILS when overlap increases, as local optima are closer together and local search can do more work.

It is known that ILS performs poorly on larger problems, so the trend of success for an increase in overlap for these configurations does not necessarily apply to all problems made by CliqueTreeMk.

Chapter 10

Summary

To understand how the conclusions of this thesis were reached, it is helpful to examine the chronological order in which important discoveries were made. The first experiment (see Chapter 7) provided good insights, and it became clear that higher overlap did not necessarily lead to increased search difficulty for LT-GOMEA. In fact, after initial decreases in performance, all codomains showed an increase in success at an overlap of 2 for deceptive codomains and an overlap of 3 for the random codomain. This raised the question of what caused this dip and subsequent increase in performance. The first drawn conclusion was that for all codomains, an increase in overlap increased the number of local optima and decreased the hamming distance between local optima, which eased the search process. This may be an important factor, but it also raises the question of why the problems generated with the same input parameters showed different levels of success, and why the LONs could vary significantly in terms of the number of large attractors, large funnels, and access to the global optimum and for the neutral deceptive-trap codomain variation in plateau sizes. Is the performance of LT-GOMEA just inconsistent or is there a clear difference between the problem difficulty? It was also concluded that LT-GOMEA performed better for higher overlap on the deceptive-trap codomain than on the random codomain because the deceptive substrings decrease diversity within the population, allowing the building blocks to be more easily identified. This increases the probability of selecting the building blocks out of solutions that have subfunction substrings as in the global optimum. Not only does the deceptiveness reduce diversity, but the way subfunction fitnesses are constructed also results in fewer local optima for each subfunction substring in the deceptive codomains compared to the random codomain, which can have many more local optima. This can lead to increased diversity within the population and can make linkage learning even harder.

Because the LONs for population-based LT-GOMEA have some limitations, it was also interesting to examine the LONs for the non-population-based LT-GOMEA. The LONs show more consistent behaviour when a problem is solved repeatedly for two LONs, making them more reliable for drawing conclusions. For the deceptive-trap non-neutral codomain, the performance was very similar to experiment one, which led to the suggestion that it could be for the same reasons. Sometimes, LT-GOMEA was unsuccessful when only learning building blocks from an initial population of optimized solutions, while the LON indicated that the problem did not appear to be particularly difficult. In these cases, the LON often showed a global optimum with good connectivity to large attractors and the global optimum was positioned in a large funnel. Repeating the initialization of the population and re-learning the building blocks often resulted in better performance.

But outside these cases, there also were cases that just seemed difficult compared to problems generated with the same input parameters. This partly solved the question of experiment one, it is not necessarily the performance of LT-GOMEA that causes these difficulty differences, but rather that some problems with the same input parameters are just harder. This finding led to the following thought: "Deceptive problems are considered difficult because the deceptive attractor subfunction substrings mislead the search and prevent the discovery of the optimal subfunction substrings. But what if the optimal subfunction substrings are not present, then the problem becomes less deceptive." Therefore, for all input parameter configurations, the percentage of subfunctions in the global optimum with the optimal subfunction substrings was calculated. As overlap increased, the average percentage per input parameter decreased, explaining the decrease in deceptiveness. However, there was also a high variation in the percentages of problems generated by the same input parameters (except for an overlap of 0), which may help to explain the differences in difficulty for instances generated with the same overlap. This notion of fewer optimal substrings in the global optimum also led to the believe that there are more substrings present that are not either the deceptive or optimal substring. For the deceptive neutral codomain this is thought to cause the increase in global optima and size of plateaus, because there are many substrings that are not the deceptive or optimal substring, and have equal fitness. If these are present in the global optimum, this can cause many different substring configurations to produce the same optimal fitness. Non-population-based LT-GOMEA where the building blocks are learned from hill-climbed solutions performed poorly on the random codomain, for all overlaps more than 0. The large number of local optima on the random codomain makes the initial population too diverse to learn the correct building blocks from. The population needs time to evolve for becoming less diverse, which aids in recognizing the blocks and preserving them. This explains the differences between the random and deceptive codomain in experiment 2. The number of local optima is far larger for the random codomain. Although it is believed that deceptiveness plays a smaller role for the random codomain, this does not mean that deceptive subfunction substrings are non-existing.

The dip in performance (seen in experiment 1) for both codomains must now be explained. ILS also showed improvement on the deceptive-trap codomain with increasing overlap. For lower overlap, the problem is too deceptive for ILS to solve. For LT-GOMEA, this is not an issue because it uses linkage learning to determine which building blocks are important. The low diversity within the population at lower overlap allows linkage learning to easily identify the blocks and select the few correct subfunction orderings present in the population. However, as overlap increases, linkage learning becomes more difficult due to the increase in diversity. At the same time, the deceptiveness decreases due to the decreased number of optimal subfunction substrings at higher overlap, and the increased number of subfunctions causes local optima to be closer together. These counteracting forces are believed to contribute to the dip in performance seen in experiments 1 and 2 for the deceptive-trap codomain. There is a clear difference in performance between experiment 1 and 2 on the random codomain. In experiment 1, performance improved after the initial dip, but this did not occur in experiment 2. The increase in local optima with increasing overlap probably leads to a population that is too diverse when it consists only of hill-climbed solutions, therefore it is necessary to have an evolving population. The dip in performance seen in experiment 1 for the random codomain is quite different as for the deceptive codomain, the dip is at an overlap of 3 instead of an overlap of 2, and the performance does not increase as much as for the deceptive codomain, and the dip in

performance is less for the random codomain.
As seen in experiment 2, the number of local optima is much larger for the random codomain, which may explain the limited increase in performance. Nonetheless, an increase in local optima can still be advantageous because it makes it easier to move between the local optima. Similar to the deceptive codomain, it is thought that linkage learning is important for lower overlap but not as crucial because the local optima are closer together. This is believed to influence the dip occurring at a higher overlap than for the deceptive codomain.

10.1 Conclusions

Most of the conclusions are mentioned in the summary, but in this section they will be addressed according to the research questions.

10.1.1 Increasing overlap

We want to find out how the overlap influences the problem difficulty for LT-GOMEA for the problems generated by CliqueTreeMk. This is highly dependent on the used codomain. We decided to use the random and deceptive-trap codomain. These two codomains will be compared. We used an additional implementation of LT-GOMEA where the building blocks are only learned from a hill-climbed population. The insights gained from using this implementation helped us to answer the following research question:
How does the amount of overlap in TD MK Landscapes affect the performance of LT-GOMEA within the random and deceptive-trap codomain?
This research question will also be answered:
How does LT-GOMEA perform when the building blocks are learned from a population of hill-climbed solutions?

For the deceptive codomain, both implementations of LT-GOMEA perform well at an overlap of 0, but there is a dip in performance at an overlap of 2. As overlap increases beyond 2, performance improves again. It is believed that for lower overlap, linkage learning is the primary factor in problem-solving. As overlap increases, local optima are closer together and it becomes easier for the search to move between them. However, linkage learning becomes more difficult due to increased diversity within the population. The ratio of optimal subfunction substrings to the total number of subfunctions decreases with increasing overlap, reducing the deceptiveness of the problems. The dip in performance is thought to be caused by the counteracting effects, namely, a decreasing utilization of linkage learning for an increase in overlap, making the problem more difficult, and decreasing deceptiveness and closer local optima for an increase in overlap, making the problem easier. These counteracting forces are believed to explain the dip in performance.
For the random codomain, the standard implementation of LT-GOMEA also shows a dip in performance. This dip occurs at an overlap of 3 and is not as severe as for the deceptive codomain. Additionally, the performance does not improve as much after the dip. When LT-GOMEA is learned on hill-climbed solutions, only an overlap of 0 performs well. This is likely due to the high diversity of the initial population, which makes it difficult to learn the right building blocks. The population needs to evolve in order to learn and preserve the necessary blocks. The bad performance for higher overlap with a non-evolving population, also suggests that the number of local optima for the random

codomain is larger than that of the deceptive codomain. This may be contributing to the limited improvement in performance after the dip, as the search space is large and may be a limiting factor in the search. However, the increase in overlap also results in closer optima, which can improve performance to some extent. There seems to be a balance between these two factors. For standard LT-GOMEA, linkage learning with lower overlap is less important for the random codomain, as for the deceptive codomain, this is believed to cause the dip to be at a higher overlap for the random codomain.

10.1.2 Usefulness of LONs

To better understand the challenges faced by LT-GOMEA when solving problems generated by CliqueTreeMk, LONs are used to provide insights into the search process. The method for creating LONs for population-based search algorithms differs from LONs for simple search algorithms, as it is not just one improving solution, but the entire population that contributes to finding the global optimum. The creation of LONs for the LT-GOMEA implementation used in the second experiment was inspired by the work of Ochoa et al. [13]. However, creating LONs for standard LT-GOMEA required some changes. In the following research question, we will discuss the usefulness of the LONs, particularly those from experiment 2: *Can LONs effectively capture the difficulties encountered by LT-GOMEA?*
The LONs were useful for understanding why the search may have gotten stuck. By analyzing large regions of attraction or attractor nodes, it was possible to determine whether they were helping or hindering the search process. When the regions of attraction were located within the same funnel as the global optimum, it often facilitated the search process, when they are in a different funnel, the problem becomes more difficult. The LONs were particularly useful in identifying differences in difficulty among instances generated with the same input parameters. Some LONs had significantly more nodes or large attractors compared to the other instances. This helped to support the theory that problems generated with the same input parameters can differ significantly from deceptiveness and that problems within the deceptive codomain become less deceptive with an increase in overlap.
To generate the LON, it is necessary to keep track of all solutions used during the search. This allows for the collection of useful statistics, such as the number of considered solutions, the frequency of which a solution is traversed, and the differences in blocks and bits between attractors/solutions with the global optimum. These statistics can help to provide a deeper understanding of the search process of LT-GOMEA.

The last research question mainly issued the LONs from the second experiment. However, the LONs from the first experiment were constructed a little differently, which leads to some disadvantages in comparison. The answer to this question will mainly consist of these disadvantages, as the advantages were already discussed in the previous question. *Can LONs, as used in this study, provide useful insights into the search processes of population-based algorithms?* The LON only includes nodes that are the best solution(s) within a generation. Attractors that belong to these best solutions are often the ones that cause LT-GOMEA to converge, and therefore have a significant impact on whether LT-GOMEA can find the global optimum or not. If LT-GOMEA fails, this can be seen in the LON. However, the LON is incomplete as it does not include every solution that LT-GOMEA improves. These missing nodes can sometimes be essential for

finding the global optimum, and they are not shown in the LON. They greatly impact the search process and therefore the edges and funnels. Therefore, when looking at the edges and funnels, it is important to keep in mind that they have a different meaning for LT-GOMEA compared to ILS and the other LT-GOMEA implementation. The LONs are still useful by understanding which attractors disrupt or enhance the search and in what generation. They also show how frequently the global optimum has been found in comparison to other nodes.

10.2 Future work

10.2.1 Improving CliqueTreeMk as Benchmark Function

When comparing ILS to LT-GOMEA, there are significant differences in performance, especially when the overlap is low for the deceptive-trap codomain. Linkage learning appears to be necessary for finding the global optimum in these cases. The CliqueTreeMk problem generator is therefore a good benchmark for testing evolutionary algorithms on the ability of linkage learning. Several interesting findings are made, including that instances generated with the same input parameters can vary greatly in deceptiveness. The number of optimal subfunction substrings in the global optimum can be an indicator of which instances are more or less difficult for problems generated with the same input parameters. It would be useful if the benchmark function could also provide information on the occurrence of these optimal subfunction substrings in the global optimum. It would also be interesting to investigate when and to what degree subfunction substrings from the random codomain are deceptive. This could help to provide a measure of deceptiveness for a problem within the random codomain and helps to indicate the difficulty of problems with the same input parameters.

10.2.2 Broader Analysis

There are many input parameter configurations for which the performance of LT-GOMEA can yet be analyzed. The subfunction size, which increases overlap (when the problem length is kept equal), contributes to the ruggedness and difficulty of Nk landscapes. This was tested briefly for TD Mk Landscapes. The difficulty increased, but it is not fully understood why. Additionally, the effect of branching on performance was quickly tested and no significant changes were observed, but this needs to be investigated more thoroughly. It would also be interesting to study the effect of different problem lengths and see if the increase in overlap still matches our findings.

10.2.3 LON analysis

For the use of LONs it is always interesting to know which metrics are important for estimating the difficulty of problems. And for the population-based LONs, it would be interesting to see if there is a way to visualize funnels that can give more concrete information as it does for ILS and the second implementation of LT-GOMEA.
This thesis focused on occurrences and strength of deceptive attractors in order to analyze the difficulty of certain problems. However, further research is needed to fully understand what makes these attractors so deceptive and thus the problem more difficult to solve. One possible indicator of difficulty within the deceptive-trap codomain is the number of

optimal subfunction substrings. However, there are still many problems with the same input parameters and number of optimal substrings that vary greatly in difficulty, so this is not the only indicator. Therefore, it would be useful to analyze the solutions of deceptive attractors and compare them to the global optimum. This could reveal which blocks differ and why it is difficult to move from a deceptive attractor to the global optimum. It may be helpful to start with smaller problems, as they may be easier to analyze and identify the reasons for their deceptiveness.

Bibliography

[1] Peter AN Bosman, Ngoc Hoang Luong, and Dirk Thierens. Expanding from discrete cartesian to permutation gene-pool optimal mixing evolutionary algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, pages 637–644, 2016.

[2] Peter AN Bosman and Dirk Thierens. More concise and robust linkage learning by filtering and combining linkage hierarchies. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pages 359–366, 2013.

[3] Francisco Chicano, Darrell Whitley, Gabriela Ochoa, and Renato Tinós. Optimizing one million variable nk landscapes by hybridizing deterministic recombination and local search. In *Proceedings of the genetic and evolutionary computation conference*, pages 753–760, 2017.

[4] Felipe A Csaszar. A note on how nk landscapes work. *Journal of Organization Design*, 7(1):1–6, 2018.

[5] Ilan Gronau and Shlomo Moran. Optimal implementations of upgma and other common clustering algorithms. *Information Processing Letters*, 104(6):205–210, 2007.

[6] Sebastian Herrmann, Gabriela Ochoa, and Franz Rothlauf. Communities of local optima as funnels in fitness landscapes. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, pages 325–331, 2016.

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

[8] Stuart A Kauffman and Edward D Weinberger. The nk model of rugged fitness landscapes and its application to maturation of the immune response. *Journal of theoretical biology*, 141(2):211–245, 1989.

[9] Rui Li, Michael Emmerich, Jeroen Eggermont, Ernst GP Bovenkamp, Thomas Bäck, Jouke Dijkstra, and Johan HC Reiber. Mixed-integer nk landscapes. In *Parallel Problem Solving from Nature-PPSN IX*, pages 42–51. Springer, 2006.

[10] Gabriela Ochoa, Katherine M Malan, and Christian Blum. Search trajectory networks of population-based algorithms in continuous spaces. In *International Conference on the Applications of Evolutionary Computation (Part of EvoStar)*, pages 70–85. Springer, 2020.

[11] Gabriela Ochoa, Marco Tomassini, Sebástien Vérel, and Christian Darabos. A study of nk landscapes' basins and local optima networks. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 555–562, 2008.

[12] Gabriela Ochoa and Nadarajen Veerapen. Additional dimensions to the study of funnels in combinatorial landscapes. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, pages 373–380, 2016.

[13] Gabriela Ochoa and Nadarajen Veerapen. Mapping the global structure of tsp fitness landscapes. *Journal of Heuristics*, 24(3):265–294, 2018.

[14] Gabriela Ochoa, Sébastien Verel, Fabio Daolio, and Marco Tomassini. Local optima networks: A new model of combinatorial fitness landscapes. In *Recent advances in the theory and application of fitness landscapes*, pages 233–262. Springer, 2014.

[15] Dirk Thierens and Peter AN Bosman. Model-based evolutionary algorithms part 2: Linkage tree genetic algorithm, slides of lecture from the course evolutionary computing uu. 2022.

[16] Dirk Thierens and Tobias van Driessel. A benchmark generator of tree decomposition mk landscapes. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 229–230, 2021.

[17] TR van Driessel. Performance and effectiveness of linkage tree gene-pool optimal mixing evolutionary algorithm on tree decomposition mk landscapes. Master's thesis, 2021.

[18] Nadarajen Veerapen, Gabriela Ochoa, Renato Tinós, and Darrell Whitley. Tunnelling crossover networks for the asymmetric tsp. In *International Conference on Parallel Problem Solving from Nature*, pages 994–1003. Springer, 2016.

[19] Sébastien Verel, Fabio Daolio, Gabriela Ochoa, and Marco Tomassini. Local optima networks with escape edges. In *International Conference on Artificial Evolution (Evolution Artificielle)*, pages 49–60. Springer, 2011.

[20] Sébastien Verel, Gabriela Ochoa, and Marco Tomassini. Local optima networks of nk landscapes with neutrality. *IEEE Transactions on Evolutionary Computation*, 15(6):783–797, 2010.

[21] Pradnya A Vikhar. Evolutionary algorithms: A critical review and its future prospects. In *2016 International conference on global trends in signal processing, information computing and communication (ICGTSPICC)*, pages 261–265. IEEE, 2016.

[22] Edward D Weinberger et al. Np completeness of kauffman's nk model, a tuneably rugged fitness landscape. *Santa Fe Institute Technical Reports*, 1996.

[23] 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*, pages 927–934, 2015.

[24] L Darrell Whitley, Francisco Chicano, and Brian W Goldman. Gray box optimization for mk landscapes (nk landscapes and max-ksat). *Evolutionary computation*, 24(3):491–519, 2016.