Conditional Direct Empirical Linkage Discovery for Solving Multi-Structured Problems

Michal W. Przewozniczek
Wroclaw Univ. of Science and Techn.
Wroclaw, Poland
michal.przewozniczek@pwr.edu.pl

Peter A.N. Bosman
Centrum Wiskunde & Informatica
Amsterdam, The Netherlands
peter.bosman@cwi.nl

Anton Bouter
Centrum Wiskunde & Informatica
Amsterdam, The Netherlands
anton.bouter@cwi.nl

Arthur Guijt
Centrum Wiskunde & Informatica
Amsterdam, The Netherlands
arthur.guijt@cwi.nl

Marcin M. Komarnicki
Wroclaw Univ. of Science and Techn.
Wroclaw, Poland
marcin.komarnicki@pwr.edu.pl

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

Abstract

Many state-of-the-art Genetic Algorithms (GAs) use information about variable dependencies to construct masks for variation operators and, in turn, improve their effectiveness and efficiency. In the black-box setting, the dependency structure model is not known and must be discovered as a part of the optimization process. The precision of this model may be decisive for the effectiveness of the GAs using it. This work considers the recently identified multi-structured problems that arise when two or more problems with a different structure (i.e., different variable dependencies) are combined in a non-linear manner. Such problems are hard to solve because, usually, it is not enough to know all the dependencies to solve them effectively. To do so, one must know which dependencies are a part of which substructure, i.e., the dependencies between dependencies. Finally, an optimizer must detect which substructure is valid for the solution at hand. Statistical Linkage Learning (SLL) was proposed to decompose multi-structure problems. However, SLL may report false dependencies, which can deteriorate the search. Therefore, we propose the Conditional Direct Empirical Linkage Discovery (cDLED) technique to decompose multi-structured problems. cDLED guarantees to report only true dependencies. Using cDLED, we propose detecting which problem substructure refers to the given solution. Using these two mechanisms, we propose an optimizer that is highly competitive with other state-of-the-art GAs. We consider single-objective optimization, but our propositions can also be useful in multi- and many-objective optimization. Additionally, we propose a more general formal representation of multi-structured problems.

CCS Concepts

• **Computing methodologies** → **Artificial intelligence**; **Discrete space search**; **Randomized search**; • **Theory of computation** → **Evolutionary algorithms**; • **Mathematics of computing** → **Evolutionary algorithms**.

FOGA '25, Leiden, Netherlands

ACM ISBN 979-8-4007-1859-5/2025/08
https://doi.org/10.1145/3729878.3746624

Keywords

Linkage Learning, Model Building, Empirical linkage learning, Genetic Algorithms, Multi-objective optimization

ACM Reference Format:
Michal W. Przewozniczek, Peter A.N. Bosman, Anton Bouter, Arthur Guijt, Marcin M. Komarnicki, and Dirk Thierens. 2025. Conditional Direct Empirical Linkage Discovery for Solving Multi-Structured Problems. In *Foundations of Genetic Algorithms XVIII (FOGA '25), August 27–29, 2025, Leiden, Netherlands.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3729878.3746624

1 Introduction

In multi- and many-objective problems, the objective functions frequently have different sets of dependent variables. If these objectives are scalarized, e.g., using the Tchebycheff scalarization, we obtain a single-objective problem with two or more underlying problem structures. A similar situation may occur if the function to be optimized contains a min or max function. In general, the multi-structure phenomenon occurs when two or more problems are combined into one in a non-linear manner [11].

Many state-of-the-art GAs discover variable dependencies to model the problem structure [10, 25, 33, 38]. Frequently, this information is used to create masks for various mixing operators [1, 30, 34, 37]. The quality of the problem structure model and the way it is used are frequently decisive for the optimizer's effectiveness [22]. However, in problems with many underlying substructures, even knowing all variable dependencies may not be helpful. To propose *appropriate* mixing masks, we need to know which dependencies belong to which substructure and which substructure is appropriate for processing a given individual.

The notion of a multi-structured problem was introduced recently, and the Linkage Kernel Gene-pool Optimal Mixing Evolutionary Algorithm (LK-GOMEA) was proposed to handle such problems [11]. The key intuition behind LK-GOMEA is to use locality to determine which dependencies are appropriate for mixing a given solution. LK-GOMEA uses SLL to discover local dependencies. LK-GOMEA was shown to be effective for various multi-structured problems. However, SLL may report false dependencies, deteriorating the effectiveness and efficiency of the optimization process. Empirical Linkage Learning (ELL) techniques were proven to discover only the true dependencies [30], but their computational cost is frequently higher than SLL [25]. Therefore, we propose an ELL

Table 1: Dependency Structure Matrix (DSM) example

	1	2	3	4	5
1	X	0.81	0.22	0.76	0.09
2	0.81	X	0.11	0.68	0.11
3	0.22	0.11	X	0.55	0.88
4	0.76	0.68	0.55	X	0.22
5	0.09	0.11	0.88	0.22	X

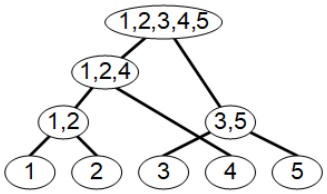

Figure 1: Linkage Tree for DSM presented in Table 1

technique, namely cDLED, and an efficient way of its use. Using cDLED, we propose a mating restrictions mechanism to mix only those individuals who refer to the same substructure. Finally, we propose an optimizer using the two aforementioned mechanisms, namely the cDLED Parameter-less Population Pyramid for Multi-Structure Problems (cDLED-P3M). cDLED-P3M is inspired by the Parameter-less Population Pyramid (P3) [10]. P3 was shown to be effective in solving complex problems [25, 30]. The experiments performed on the large problem set confirm that cDLED-P3M is highly competitive with other state-of-the-art optimizers.

The remainder of this paper is organized as follows. The next section presents the related work. In Section 3, we present the details of the considered multi-structured problems. In the fourth section, we describe the details of the proposed cDLED and cDLED-P3M. In Section 5, we present the results of the performed experiments and their analysis. Finally, the last section summarizes this work and identifies promising further research steps.

2 Related Work

2.1 Statistical Linkage Learning

SLL is a core mechanism of many effective GAs [4, 10, 14, 15, 20, 33]. It aims to discover the dependencies by applying statistical analysis to the gene value frequencies in the GA population. Frequently, Mutual information (MI) is used [17] to measure the dependency between two genes, X and Y:

$$I(X;Y) = \sum_{x \in X} \sum_{y \in Y} p(x,y) \log_2 \frac{p(x,y)}{p(x)p(y)}. \quad (1)$$

In SLL-using GAs, the MI values are stored in the Dependency Structure Matrix (DSM) [14].

DSM is used to create clusters of dependent genes. These clusters may then be used by mixing [35] or other operators [14]. Frequently, Linkage Trees (LT) are used to cluster DSM [10, 30, 35]. LT is created as follows. We create a single leaf for each gene. Then, we join the pair clusters that are the most dependent on each other (according to DSM). To measure the dependency strength when at least one cluster contains more than one gene, usually, we compute the average DSM values for all pairs of genes from two clusters. The most dependent clusters are joined until the cluster that contains all the genes is created (the root of the created LT).

The example of DSM for a 5-gene problem is presented in Table 1. Figure 1 presents the LT that results from this DSM. First, we join nodes {3} and {5} because the the dependency strength is the highest for these variables. The dependency strength between node {3,5} and node {1} is $(0.22 + 0.09)/2 = 0.15$. For nodes {2} and {4}, it is 0.11 and 0.38, respectively. Therefore, we join nodes {1} and {2}. The rest of the tree is created in the same manner.

2.2 SLL-using Genetic Algorithms

Optimal Mixing (OM) is an operator that uses clusters of dependent variables [10, 30, 32, 34]. In OM, we try to improve the *source* individual using the *donor* and a mask (e.g., an LT node). We replace the genes in the source individual with the genes from the *donor* marked by a mask. If the fitness does not decrease, then the modification is preserved. Otherwise, it is rejected. Note that recent research shows that for some problems, other strategies of the source individual modification may be useful [28].

During OM, each individual in the population is considered a source. For each source, we consider all LT nodes as masks (except for the root). The donor is selected randomly from the population, but it must differ from the source by at least one gene marked by a mask. The considered mask order may be random [35], or use the shorter masks first [10].

Gene-pool Optimal Mixing Evolutionary Algorithm (GOMEA) [34], which succeeded Linkage Tree Genetic Algorithm (LTGA) [33], is a GA that replaces crossover and mutation operators with OM. Usually, GOMEA uses LTs. In such a case, we denote it LT-GOMEA. At the start of each generation of LT-GOMEA, a DSM is estimated from the population and used to construct an LT.

LT-GOMEA is parameter-less and does not require to manually set the population size. To this end, LT-GOMEA is frequently combined with a population-sizing scheme [1, 12]. When using this scheme, LT-GOMEA starts with a single population of size two. After four generations of a population of a given size, one iteration of the population with a doubled size is executed (and created if necessary). Thus, LT-GOMEA maintains many populations of different sizes. To limit their number, some of the populations may be considered useless. If, for a given population of a given size, a larger population with a higher average fitness exists, then the given population and all populations of a smaller size are terminated. Each of these populations evolve independently, except for the Forced Improvements (FI) phase. FI is executed if a source individual is not changed during OM, or the best-found individual for the whole population was not changed for a given number of generations. In FI, OM for the given source individual is executed again, but the current best-found individual acts as a donor.

LT-GOMEA was originally proposed for binary [34] and permutation-based problems [1]. It is parameter-less and was shown to be effective in solving hard problems of various types [1, 25]. In LT-GOMEA, SLL is used to determine linkages and build the LT, but the core idea is to represent linkages effectively in a linkage tree.

The linkages could also be detected using ELL techniques [23, 30] or even introduced manually (i.e., by analyzing the targeted problem).

Parameter-less Population Pyramid (P3) [10], same as LT-GOMEA, uses OM and LTs. However, instead of a classical population vector, P3 uses subpopulations identified as *levels*. In each P3 iteration, a new individual is randomly created, optimized by the First Improvement Hillclimber (FIHC) [10, 25], and added to the lowest level of the pyramid. Then, the new individual climbs up the pyramid by mixing with other individuals on subsequent pyramid levels, using DSM and the resulting LT specifically learned for the individuals at that level. If such mixing leads improves the climbing individual, then it is added to the next level, after which mixing at that level proceeds. Thus, the higher the pyramid level is, the better individuals it should contain. P3 was shown to be effective, especially in solving problems with overlaps [10, 25, 30]. It was also successfully applied to permutation-based problems [40].

2.3 Gray-box Optimization

One of the differences between black- and gray-box optimization [38, 39] is that in gray-box, we know the exact gene dependencies. Thus, no linkage learning is needed to discover them. To explain the meaning of gene dependency, we will use the *Walsh decomposition* [13]. Using it, we can define any pseudo-Boolean function as:

$$f(\boldsymbol{x}) = \sum_{i=0}^{2^n-1} w_i \psi_i(\boldsymbol{x}) \tag{2}$$

where $w_i \in \mathbb{R}$ is the ith Walsh coefficient, $\psi_i(\mathbf{x}) = (-1)^{\mathbf{i}^T\mathbf{x}}$ generates a sign, $\mathbf{i} \in \mathbb{B}^n$ is the binary representation of index i, and $\boldsymbol{x} = \{x_1, x_2, \ldots, x_n\}$ is a binary solution vector of size n.

We consider x_g and x_h as *directly* dependent if there exists at least one nonzero Walsh coefficient w_i in the Walsh decomposition of $f(\boldsymbol{x})$ such that the gth and hth elements of $\mathbf{i}$ equal one [36].

We can also represent $f(\boldsymbol{x})$ in the additive form:

$$f(\boldsymbol{x}) = \sum_j f_j(\boldsymbol{x}_{\boldsymbol{I}_j}), \tag{3}$$

where I_j are subsets of $\{1, ..., n\}$ that are not necessarily disjoint. If x_g and x_h are directly dependent, then there exists at least one subfunction that takes x_g and x_h as arguments [23].

Subsets I_j do not have to be disjoint. If so, then two or more subfunctions f_j may share some variables (their argument sets share at least one argument). If so, then these two subfunctions *overlap*. The overlapping problems are particularly hard to solve because the overlapping subfunctions may contradict each other, i.e., the values of the shared arguments for which two subfunctions reach their optimal values may contradict each other.

Variable Interaction Graph (VIG) represents the dependencies between variables. If two variables are directly dependent, then the appropriate VIG entry equals 1 or 0 otherwise. Table 2 presents the exemplary VIG. This VIG refers to the overlapping function that, in the additive form, can be represented as $f(\boldsymbol{x}) = f_1(x_1, x_2, x_3, x_4) + f_2(x_4, x_5, x_6, x_7) + f_3(x_7, x_8, x_9, x_{10})$.

The main difference between DSM and VIG is that VIG reports all and only the direct dependencies. It does not report any false (non-existing) dependencies. In contrary, DSM presents predictions (on the base of statistics) that given pairs of variables are more or less dependent on each other. DSM does not allow to tell direct from *indirect* dependencies (e.g., x_3 and x_5 are not directly dependent, but they are indirectly dependent because they both depend on x_4).

Table 2: The example of dependencies represented by a Variable Interaction Graph (VIG)

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	X	**1**	**1**	**1**	0	0	0	0	0	0
2	**1**	X	**1**	**1**	0	0	0	0	0	0
3	**1**	**1**	X	**1**	0	0	0	0	0	0
4	**1**	**1**	**1**	X	**1**	**1**	**1**	0	0	0
5	0	0	0	**1**	X	**1**	**1**	0	0	0
6	0	0	0	**1**	**1**	X	**1**	0	0	0
7	0	0	0	**1**	**1**	**1**	X	**1**	**1**	**1**
8	0	0	0	0	0	0	**1**	X	**1**	**1**
9	0	0	0	0	0	0	**1**	**1**	X	**1**
10	0	0	0	0	0	0	**1**	**1**	**1**	X

The objective of gray-box optimization is to improve the effectiveness and efficiency of the optimization process. Efficiency can be improved by using *partial evaluations* [38], i.e., recomputing only a part of $f_j(\boldsymbol{x}_{\boldsymbol{I}_j})$ to obtain the exact value of $f(\boldsymbol{x})$. Except for some surrogate-based research trying to apply partial evaluations to black-box optimization [7], this phenomenon is restricted to gray-box. Therefore, it is out of the scope of this work.

Many gray-box-like operators employ VIG to improve the optimizer effectiveness. For instance, we can use VIG to propose perturbation masks in Iterated Local Search [36]. A more sophisticated proposition is the Partition Crossover (PX) operator [5, 31, 37]. In PX, we compare the genotypes of two mixed individuals to create the mixing masks by clustering all differing directly dependent genes. For instance, consider two individuals, $\boldsymbol{x}_a$ = 0001001000 and $\boldsymbol{x}_b$ = 1111110111 and the VIG presented in Table 2. For such a pair of individuals, two masks will be considered in PX: $m_1\{1, 2, 3\}$ and $m_2 = \{5, 6, 7, 8, 9, 10\}$. If we consider m_2, then we will obtain two offspring individuals: $\boldsymbol{x}_{a,2}$ = 0001**110111** and $\boldsymbol{x}_{b,2}$ = 111**001000**. Above, we have shown that for the VIG from Table 2, we can represent $f(\boldsymbol{x})$ as a sum of three subfunctions. Note that $\boldsymbol{x}_{a,2}$ has the same argument values of f_1 as $\boldsymbol{x}_a$ and the same arguments of f_2 and f_3 as $\boldsymbol{x}_b$. The situation is analogous for $\boldsymbol{x}_{b,2}$. Thus, we can say that although subfunctions may share some variables, using VIG and PX, we obtain offspring individuals that inherit the subfunctions arguments from one or the other parent. This feature is highly useful for the optimization of functions with overlaps [37]. The features of PX presented above were also used to detect the missing linkage in black-box optimization [31].

Above, we analyze the dependencies in the additive problems. However, the problems can be non-additively decomposable [16]. Consider $f_u(\boldsymbol{x}) = u(\boldsymbol{x})^2$, where $u(\boldsymbol{x})$ is the number of 1's in $\boldsymbol{x}$ (*unitation*). For f_u, Walsh's decomposition will mark all genes as dependent on each other. However, in f_u, each variable can be optimized separately. Thus, we can state that f_u is non-additively decomposable. The next subsection presents linkage learning techniques that can discover dependencies for additively and non-additively decomposable problems. Moreover, these techniques deliver information about dependencies whose quality is close to VIG.

2.4 Empirical Linkage Learning

SLL-using GAs are effective and capable of solving problems that are difficult to handle for optimizers that do not use mechanisms of problem structure discovery [2, 10, 14, 35]. The main disadvantage of SLL is that these techniques are statistical and may report *false linkage*, i.e., two independent genes may be considered dependent. False linkage may cause GAs using linkage learning to be ineffective [25, 30]. The recently proposed ELL techniques (ELL) do not suffer from the above flaw. They are proven to discover only the true dependencies at the cost of using more function evaluations [25, 30]. ELL techniques analyze how changes to the genotype influence local optimization results. It has been shown that using ELL may lead to high-quality results and solving problems that are difficult to solve using SLL [25, 30, 32].

Direct Empirical Linkage Discovery (DLED) [30] is an ELL technique that discovers only the direct dependencies. DLED was initially proposed for binary search spaces but was adjusted to solve permutation-based [29] and non-binary discrete problems [26]. Recently, a linkage learning technique using monotonicity checking that refers to DLED intuitions was proposed to decompose problems in continuous domains [16]. Although the DLED cost is high, it was successfully introduced to P3 and LT-GOMEA [30].

In the binary search space, the DLED dependency check is equivalent to the non-monotonicity check [19] and works as follows. Given individual $\boldsymbol{x} \in \mathbb{B}^n$, to check if genes g and h are dependent, DLED creates the individuals $\boldsymbol{x}^g$, $\boldsymbol{x}^h$, and $\boldsymbol{x}^{g,h}$ obtained from individual $\boldsymbol{x}$ by flipping genes g, h or both of them, respectively. Genes g and h are considered to be dependent if at least one of the following conditions holds [31, 32]:

- $f(\boldsymbol{x}) < f(\boldsymbol{x}^g) \;\;\&\;\; f(\boldsymbol{x}^h) \geq f(\boldsymbol{x}^{g,h})$ OR
- $f(\boldsymbol{x}) = f(\boldsymbol{x}^g) \;\;\&\;\; f(\boldsymbol{x}^h) \neq f(\boldsymbol{x}^{g,h})$ OR
- $f(\boldsymbol{x}) > f(\boldsymbol{x}^g) \;\;\&\;\; f(\boldsymbol{x}^h) \leq f(\boldsymbol{x}^{g,h})$ OR
- $f(\boldsymbol{x}) < f(\boldsymbol{x}^h) \;\;\&\;\; f(\boldsymbol{x}^g) \geq f(\boldsymbol{x}^{g,h})$ OR
- $f(\boldsymbol{x}) = f(\boldsymbol{x}^h) \;\;\&\;\; f(\boldsymbol{x}^g) \neq f(\boldsymbol{x}^{g,h})$ OR
- $f(\boldsymbol{x}) > f(\boldsymbol{x}^h) \;\;\&\;\; f(\boldsymbol{x}^g) \leq f(\boldsymbol{x}^{g,h})$

To summarize, DLED finds that genes g and h of individual $\boldsymbol{x}$ are dependent if the modification of gene g influences the fitness relation for gene h or oppositely. DLED discovers *only* direct dependencies [30], i.e., if DLED discovers the dependency between two variables, say x_g and x_h, it means that there exists at least one non-zero Walsh coefficient that refers to these variables. It also means that if $f(\boldsymbol{x})$ is represented in the additive form, then there exists at least one subfunction $f_j(\boldsymbol{x}_{I_j})$ such that $g, h \in I_j$.

DLED-based techniques can decompose non-additively separable problems, which may be crucial while solving some real-world problems [16]. Recently, new techniques were proposed to limit the computation cost raised by DLED. This includes the missing linkage detection [31] and computing DLED as a side effect of local search [32, 36]. However, since these techniques do not apply to the optimizer proposed in this work, we do not present their details.

2.5 Hybrid Linkage Learning

Despite ELL advantages, SLL-using optimizers remain state-of-the-art for many optimization problems. For instance, P3 frequently outperforms ELL-using optimizers in solving the Ising Spin Glasses (ISG) problem [25, 30]. On the other hand, ELL may be one of the keys to solving many problems effectively [25]. Therefore, joining the advantages of SLL and ELL seems to be justified.

Empirical Linkage Learning for Permutation-based Problems (pbELL) is a DLED-like ELL technique dedicated to the decomposition of permutation-based problems [29]. pbELL is used to construct the pbELL-DSM. In pbELL-DSM, the strength of dependency between two variables is measured by the number of dependency detections made by pbELL. pbELL-DSM can be hybridized with a standard SLL-based DSM (see Section 2.1) by adding these two matrices and producing HLL-DSM. If the values in SLL-DSM are normalized to the $(0; 1)$ range, then the dependencies reported by pbELL-DSM may be considered dominating because pbELL-DSM counts the number of pbELL detections. Thus, if pbELL discovers the dependency between x_g and x_h at least once, then the value referring to the g, h dependency in pbELL-DSM will be higher than the value referring to this dependency in SLL-DSM. Thus, we can state that in HLL, SLL is used to differentiate the strength of dependencies reported by pbELL-DSM.

pbELL was introduced to two state-of-the-art optimizers, namely LT-GOMEA [1] and Parameter-less Population Pyramid for Permutation-based Problems (P4) [40]. P4 may be considered as P3 dedicated to solving permutation-based problems. Originally, LT-GOMEA and P4 use SLL. As presented in [29], LT-GOMEA and P4 with HLL obtain significantly better results than their versions using only SLL or only pbELL. For those problem classes where SLL-using optimizer versions reported higher quality results than ELL-using versions, HLL-using performed equally to SLL-using ones. In the other cases, HLL-using versions reported results of quality equal to ELL. In one case, HLL-using versions reported results of higher quality than SLL- and ELL-using versions. These observations show the potential and robustness behind using HLL.

HLL was also employed recently in LT-GOMEA and P3 to solve problems in binary domains. Both optimizers employed the First Improvement Hill Climber with Linkage Learning (FIHCwLL) [32] to perform DLED-based linkage learning at a low cost. Linkage discovery performed by FIHCwLL was used to construct empirical VIG (eVIG) [32, 36]. The difference between eVIG and pbELL-DSM is that eVIG does not count the number of dependency detections between any pair of variables. It only reports if the dependency was found at least once or not. Since FIHCwLL uses the DLED-based variable dependency check, it can only discover the existing dependencies. Thus, eVIG may only miss reporting some dependencies (if they were not discovered by the DLED check) when compared to VIG. However, the results presented in [32] show that eVIG is usually equal to or close to VIG.

In [32], eVIG and SLL-DSM were hybridized in the same manner as in [29], i.e., they were added. The difference is that eVIG is a matrix that contains only $\{0; 1\}$ values. The results were similar to the results reported in [29]. Considering 14 problem classes, the HLL-using versions of LT-GOMEA and P3 were always reporting results of better or equal quality to all other considered versions of these optimizers (except one problem class, for which the original LT-GOMEA outperformed LT-GOMEA using HLL). Thus, using HLL may increase the robustness and effectiveness of GAs.

2.6 Best- and Worst-of-Traps Problems

Multi-structured problems contain multiple linkage substructures, such that different solutions may have locally different linkage [11]. In effect, many dependencies between variables are conditional, potentially on some hidden variable. The Best-of-Traps problem is defined as follows [11]. First, given a permutation π_{fn}, optimum s^*_{fn} as well as block length k for each of the fns substructures, the concatenated permuted trap function is defined as follows.

$$f^{\text{CPT}}_{\text{fn}}(\boldsymbol{x}, \pi_{\text{fn}}, \boldsymbol{x}^*_{\text{fn}}) = \sum_{b=0}^{n/k} \text{DT}\left(\sum_{i=bk}^{bk+k-1} \boldsymbol{x}[\pi_{\text{fn}}[i]] = \boldsymbol{x}^*_{\text{fn}}[\pi_{\text{fn}}[i]] \right) \quad (4)$$

where equality is to be interpreted as 1 and inequality as 0, and DT is the deceptive trap function [6]:

$$DT(u) = \begin{cases} k & \text{if } u = k \\ k - u - 1 & \text{otherwise} \end{cases} \quad (5)$$

where u is *unitation* (to recall, the sum of binary gene values) and k is the function size.

The Best-of-Traps (BOT) and Worst-of-Traps (WOT) are defined as:

$$f_{\text{BOT}}(\boldsymbol{x}) = \max_{\text{fn}=0}^{\text{fns}-1} f^{\text{CPT}}_{\text{fn}}(\boldsymbol{x}, \pi_{\text{fn}}, \boldsymbol{x}^*_{\text{fn}}) \quad (6)$$

$$f_{\text{WOT}}(\boldsymbol{x}) = \min_{\text{fn}=0}^{\text{fns}-1} f^{\text{CPT}}_{\text{fn}}(\boldsymbol{x}, \pi_{\text{fn}}, \boldsymbol{x}^*_{\text{fn}}) \quad (7)$$

Multi-structured problems occur when two or more problems with different structures are combined in a non-linear manner, for example, using min or max operator. Tchebycheff scalarization [41], as used in multi-objective optimization, is an example of such a combination, and is used as a scalarization method that allows both the convex and concave portions of the Pareto front are to be found.

The results presented in [11] show that SLL-based LT-GOMEA [8] fails to solve multi-structured problems reliably. Therefore, the Linkage Kernel GOMEA (LK-GOMEA) is proposed. In LK-GOMEA, for each solution, a linkage model is learned over a local neighborhood. The neighborhood is determined by taking the $\lceil\sqrt{P}\rceil$ solutions that are the nearest based on Hamming distance, followed by making them symmetric, i.e. if b is in the neighborhood of a, then we ensure a is also in the neighborhood of b. As the model is learned over a local neighborhood, only locally relevant variable dependencies are represented in the model. While learning a model for each solution is expensive, this allows LK-GOMEA to solve certain multi-structured problems in a scalable manner.

2.7 Conditional Linkage Learning

When OM is applied to (a subset of) variables within a solution, the resulting fitness of this solution is naturally determined by the genotypes of both the parent solution and the donor. However, different solutions will not necessarily benefit from mating with the same donor solution, i.e., they may have different preferred mating partners. One possible cause for this is the presence of conditional linkage, which is particularly the case in multi-structured problems.

The foundations of conditional linkage are related to various Estimation of Distribution Algorithms (EDAs) that use Bayesian Networks as a linkage model [21]. Conditional linkage models were first used together with OM in the Real-Valued GOMEA (RV-GOMEA) [3]. When using such linkage models for variation using OM, the sample distribution of the variables to which variation is applied can be dependent on the values of a number of different problem variables to which variation is not applied, thereby modelling a conditional dependency. Similarly, in discrete optimization, where the individuals frequently govern the underlying probability distribution in the population, the conditional linkage may be exploited by using mating restrictions, i.e., not simply selecting a random donor from the population but making the donor selection dependent on (a subset of) the genotype of the parent.

3 Best- and Worst-of-Functions Problems

Inspired by [11], we propose the Best-of-Functions (BOF) and Worst-of-Functions (WOF) problems. BOF and WOF are multi-structured problems and are more general than the Best-of-traps and Worst-Of-Traps problems. BOF is defined as follows.

$$f_{\text{BOF}}(\boldsymbol{x}) = \max_{fn=0}^{fns-1} f_{fn}\left(\boldsymbol{y}_{x,fn}\left(\boldsymbol{x}, \pi_{fn}, m_{fn}, size(f_{fn})\right)\right) \quad (8)$$

where

$$\boldsymbol{y}_{x,fn}(\boldsymbol{x}, \pi, m, s) = \Big[\boldsymbol{x}\big[\pi[0]\big] \oplus m[0], \boldsymbol{x}\big[\pi[1]\big] \oplus m[1], ..., , \boldsymbol{x}\big[\pi[s-1]\big] \oplus m[s-1]\Big] \quad (9)$$

where $\boldsymbol{x}$ is a solution vector, π_{fn} is a vector for redefining an order of variables in $\boldsymbol{x}$ for the *fn*th function, m_{fn} is a mask for the *fn*th function, $\oplus$ is a bitwise XOR, and $size(f_{fn})$ returns a number of input variables for the *fn*th function. In the latter part of this work, each function f_{fn}, we denote as the *base function*.

To summarize, in BOF, we consider *fns* base functions (problems) denoted as f_{fn}. The value of $f_{BOF}(\boldsymbol{x})$ is the highest value returned by all considered f_{fn} for a given $\boldsymbol{x}$. The value of each f_{fn} is computed using the order of variables π_{fn} and mask m_{fn}. Each f_{fn} may have a different order of variables and a different mask.

The above definition allows using f_{fn} functions of different sizes. The size of BOF is $size(f_{\text{BOF}}) = \max_{fn=0}^{fns-1} size(f_{fn})$. Therefore, if $size(f_{fn}) < size(f_{\text{BOF}})$, then $size(f_{\text{BOF}}) - size(f_{fn})$ variables will be ignored during the computation of the f_{fn} value. Therefore, $\boldsymbol{y}$ is a solution vector after reordering and using a mask. For each f_{fn}, $\boldsymbol{y}$ may be different and have different sizes.

In BOF, for a given $\boldsymbol{x}$, we denote as *dominating* those f_{fn}s that fulfil the following condition:

$$dom(\boldsymbol{x}, fn) = \begin{cases} 1 & \text{if } f_{\text{BOF}}(\boldsymbol{x}) = f_{fn}\left(\boldsymbol{y}_{x,fn}\left(\boldsymbol{x}, \pi_{fn}, m_{fn}, size(f_{fn})\right)\right) \\ 0 & \text{otherwise} \end{cases} \quad (10)$$

For any $\boldsymbol{x}$, the number of dominating f_{fn}s is always in the range of $[1; fns]$.

We define the Worst-of-Functions (WOF) problems as

$$f_{\text{WOF}}(\boldsymbol{x}) = \min_{fn=0}^{fns-1} f_{fn}\left(\boldsymbol{y}_{x,fn}\left(\boldsymbol{x}, \pi_{fn}, m_{fn}, size(f_{fn})\right)\right) \quad (11)$$

Table 3: Exemplary individuals and a BOF problem

	1	2	3	4	5	6	7	8	9	10	11	12	
$\boldsymbol{x}_0$	1	1	1	1	0	1	0	1	0	0	0	0	$f_{\text{BOF}} = 5$
$\boldsymbol{y}_{x_0,0}$	1	1	1	1	0	1	0	1	0	0	0	0	$f_0 = 2+1+2$
$\boldsymbol{y}_{x_0,1}$	1	1	0	1	0	1	0	0	0	0	1	1	$f_1 = 0+0+1$
$\boldsymbol{x}_1$	1	0	0	0	0	0	1	1	1	1	0	1	$f_{\text{BOF}} = 5$
$\boldsymbol{y}_{x_1,0}$	1	0	0	0	0	0	1	1	1	1	0	1	$f_0 = 0+1+0$
$\boldsymbol{y}_{x_1,1}$	0	0	0	0	1	1	1	1	0	1	1	0	$f_1 = 2+2+1$
$\boldsymbol{x}_2$	1	0	1	0	1	1	1	1	0	1	0	1	$f_{\text{BOF}} = 4$
$\boldsymbol{y}_{x_2,0}$	1	0	1	0	1	1	1	1	0	1	0	1	$f_0 = 1+2+1$
$\boldsymbol{y}_{x_2,1}$	1	0	1	1	1	1	0	1	0	1	1	0	$f_1 = 0+0+1$

BOT problems are hard to solve, and the effectiveness of the state-of-the-art optimizers applied to these problems may be significantly limited [11]. WOT problems seem even harder to solve. Similar intuitions apply to BOF and WOF, which are more general than BOT and WOT. Therefore, in the latter part of this work, we focus on analyzing BOF while proposing the optimizers dedicated to solving the WOF problems we will address in future work.

4 Conditional Direct Linkage Discovery for Solving Multi-Structured Problems

This section presents the proposed cDLED (dedicated to decomposing the multi-structured problems) and cDLED-P3M.

4.1 Motivations

Let us analyze the following example. We use bimodal functions of order k [9] defined as

$$bimodal\_trap(u) = \begin{cases} k/2 - |u - k/2| - 1 & , u \neq k \wedge u \neq 0 \\ k/2 & , u = k \vee u = 0 \end{cases} \quad (12)$$

where u is *unitation* (the sum of binary gene values) and k is the function size.

We consider the BOF instance where $fns = 2$ and f_0, f_1 are the concatenations of three bimodal-4 functions. π_0 is the same as the gene order, while π_1 is shifted by two genes, i.e., $\pi_1 = \{3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 1, 2\}$. The considered masks are defined as $\forall_{0 \leq i < f_{ns}} m_0[i] = 0$.

We consider three individuals presented in Table 3. Since f_0 and f_1 are both built from the same bimodal-4 functions but located in different positions, for each individual, we report $\boldsymbol{y}_{x,0}$, $\boldsymbol{y}_{x,1}$, the values of f_0, f_1, and the value of f_{BOF}. The gene variables referring to the first, second and third bimodal-4 function are marked in red, blue, and green, respectively. For individuals $\boldsymbol{x}_0$ and $\boldsymbol{x}_2$, f_0 is the dominating base function, while for $\boldsymbol{x}_1$, it is f_1. If we wish to optimize the value of f_0, then we should mix those individuals, in which f_0 is the dominating base function. Moreover, in the mixing operation, we should only use the dependencies arising from f_0. If we use gene dependencies arising from f_1 that are not common with the dependencies arising from f_0, then such dependencies may be considered as *false linkage* (from the f_0 point of view) and may significantly deteriorate the quality of the evolutionary search [22]. For instance, according to f_0, three groups of genes are dependent: 1-4, 5-8, and 9-12. In OM, if $\boldsymbol{x}_0$ is the source individual, $\boldsymbol{x}_2$ is the donor, and as the mixing mask, we use the group of genes 5-8, then we will obtain $\boldsymbol{x}'_0 = 1111\ 1111\ 0000$, for which $f_0(\boldsymbol{x}'_0) = 6$. However, if in the same operation, we will use any group of dependent genes that arise from f_1 (3-6, 7-10, and {1,2,11,12}), then for each group used as a mixing mask, we will obtain $\boldsymbol{x}_0^{3-6} = 11\boldsymbol{10}\ \boldsymbol{11}01\ 0000$ ($f_0(\boldsymbol{x}_0^{3-6}) = 2$), $\boldsymbol{x}_0^{7-10} = 1111\ 01\boldsymbol{11}\ \boldsymbol{01}00$ ($f_0(\boldsymbol{x}_0^{7-10}) = 2$), or $\boldsymbol{x}_0^{\{1,2,11,12\}} = \boldsymbol{10}11\ 0101\ 00\boldsymbol{01}$ ($f_0(\boldsymbol{x}_0^{\{1,2,11,12\}}) = 1$).

Table 4: Dependencies discovered by DLED for individuals $\boldsymbol{x}_0$ and $\boldsymbol{x}_2$ from Table 3

	1	2	3	4	5	6	7	8	9	10	11	12
1	X	**1**	**1**	**1**	0	0	0	0	0	0	0	0
2	**1**	X	**1**	**1**	0	0	0	0	0	0	0	0
3	**1**	**1**	X	**1**	0	0	0	0	0	0	0	0
4	**1**	**1**	**1**	X	0	0	0	0	0	0	0	0
5	0	0	0	0	X	**1**	**1**	**1**	0	0	0	0
6	0	0	0	0	**1**	X	**1**	**1**	0	0	0	0
7	0	0	0	0	**1**	**1**	X	**1**	0	0	0	0
8	0	0	0	0	**1**	**1**	**1**	X	0	0	0	0
9	0	0	0	0	0	0	0	0	X	**1**	**1**	**1**
10	0	0	0	0	0	0	0	0	**1**	X	**1**	**1**
11	0	0	0	0	0	0	0	0	**1**	**1**	X	**1**
12	0	0	0	0	0	0	0	0	**1**	**1**	**1**	X

The above examples show that the knowledge about dependencies arising from f_1 may be useless for optimizing f_0, which seems intuitive. A similar situation may occur if we use groups of dependent genes arising from f_0 but individual $\boldsymbol{x}_1$ as a donor. Based on the above example, we formulate the motivations behind the proposed cDLED:

M1. As a base of cDLED, we wish to use DLED because it is proven to discover only the existing gene dependencies.

M2. We wish to mix only individuals with the same (preferably single) dominating base function.

M3. We wish to use only those dependencies that arise from the same base function that is the dominating one in both mixed individuals.

M4. The complete DLED-based decomposition of a single individual requires performing dependency checks for $n(n-1)/2$ gene pairs and may be computationally expensive. Thus, we wish to perform the complete DLED decomposition only for selected individuals.

4.2 DLED-based Conditional Decomposition and Mating Restrictions

Following the motivations presented in the previous section, in cDLED, we will perform the complete DLED decomposition only for chosen individuals. The complete set of dependencies obtained with DLED from a single individual will be denoted as *Ind-Linkage*. In this way, we can obtain many Ind-linkages because DLED may discover different dependencies for different individuals. Moreover, in BOF problems, DLED will usually discover the dependencies only for the dominating base function because other base functions do not influence the value of BOF. For instance, the results of complete DLED-based dependency discovery for individuals $\boldsymbol{x}_0$ and $\boldsymbol{x}_2$ from Table 3 are presented in Table 4 (their Ind-linkages are the same). The same results for individual $\boldsymbol{x}_1$ are presented in Table 5.

Table 5: Dependencies discovered by DLED for individual x_1 from Table 3

	1	2	3	4	5	6	7	8	9	10	11	12
1	X	**1**	0	0	0	0	0	0	0	0	**1**	**1**
2	**1**	X	0	0	0	0	0	0	0	0	**1**	**1**
3	0	0	X	**1**	**1**	**1**	0	0	0	0	0	0
4	0	0	**1**	X	**1**	**1**	0	0	0	0	0	0
5	0	0	**1**	**1**	X	**1**	0	0	0	0	0	0
6	0	0	**1**	**1**	**1**	X	0	0	0	0	0	0
7	0	0	0	0	0	0	X	**1**	**1**	**1**	0	0
8	0	0	0	0	0	0	**1**	X	**1**	**1**	0	0
9	0	0	0	0	0	0	**1**	**1**	X	**1**	0	0
10	0	0	0	0	0	0	**1**	**1**	**1**	X	0	0
11	**1**	**1**	0	0	0	0	0	0	0	0	X	**1**
12	**1**	**1**	0	0	0	0	0	0	0	0	**1**	X

Table 6: Exemplary population with probed dependencies

Ind	Genotype	g_{ind}	dependencies for g_{ind}
x_{p0}	1111 0110 1100*	1	2,3,4
x_{p1}	00 1111 0101 11	5	3,4,6
x_{p2}	1001 1001 0000	10	9,11,12
x_{p3}	10 0110 1111 01	10	7,8,9

**the genotypes are formatted depending on which base function (f_0 or f_1) is the dominating one*

If we are to improve individual x_{impr} with OM, then we need to identify the dependencies that are appropriate for improving this individual. However, we cannot perform the complete DLED-based decomposition for every individual we wish to improve. Therefore, we randomly choose a single gene g_{ind} and check its dependencies with the rest of the genotype using x_{impr}. Then, we choose only those Ind-linkages that cover all dependencies discovered for the g_{ind} gene. Using the filtered Ind-linkages, we construct an *empirical VIG* [31] (eVIG, a dependency matrix containing only 0s and 1s, where 1 indicates that at least one of the filtered Ind-linkages considers the dependency between a given pair of genes). eVIG generated using the procedure described above, shall differ depending on the base functions that are dominating for the considered x_{impr}. Thus, it may be considered a *conditional eVIG* because it contains only a subset of discovered variable dependencies that is (or should be) appropriate for those base functions of BOF that fulfil the $dom(x_{impr}, fn) = 1$ condition.

After constructing a conditional eVIG dedicated to a given x_{impr}, we should choose the donor individuals that will be appropriate for x_{impr}. Therefore, for each individual, we store the results of DLED made for gene g_{ind} (note that g_{ind} may be different for each individual). Then, as mating individuals, we choose only those whose dependencies for gene g_{ind} are fully covered by the conditional eVIG constructed for x_{impr}.

Let us consider the same BOF instance as in the previous example. We assume that Ind-linkages were discovered for individuals x_0, x_1 and x_2. Ind-linkages discovered for x_0 and x_2 are the same and are presented in Table 4, while Ind-linkage for x_1 is presented in Table 5. We consider a population of 4 individuals. Their genotypes, g_{ind} selected for each of them, and the dependencies discovered for each g_{ind} are presented in Table 6. We wish to mix x_{p3} for which $g_{x_{p3}} = 10$, and $g_{x_{p3}}$ was found dependent on genes 7, 8, and 9. These dependencies are covered only by the Ind-linkage presented in Table 5. Ind-linkages for x_0 and x_2 presented in Table 4 do not contain dependencies between gene pairs (7;10) and (8;10). Therefore, they are rejected. Thus, the conditional eVIG will be the same as the Ind-linkage presented in Table 5. After determining conditional eVIG, we need to choose the appropriate candidates for donors. The only individual (except x_{p3}) whose dependencies found for g_{ind} are fully covered by the constructed conditional eVIG is x_{p1}. Thus, it will be the only donor candidate chosen from the considered population. x_{p0} and x_{p2} are rejected because the conditional eVIG constructed for x_{p3} (equivalent to the matrix presented in Table 5) does not contain the dependency for pairs (1;3), (1;4) and (10;11), (10;12), respectively.

In this section, we have proposed cDLED based on filtering the dependencies found by DLED. To limit its costs, it probes the problem dependency structure. Using cDLED, we have proposed a mating restrictions mechanism. In the next section, we propose the optimizer utilizing all the proposed mechanisms.

4.3 cDLED-based Parameter-less Population Pyramid for Multi-Structure Problems

Pseudocode 1 cDLED-P3M general procedure

```
1:  IndLinkSet ← empty; bests ← empty;
2:  levels ← levels+ CreateEmptyPop();
3:  while ¬StopCondition do
4:      startBest ← GetHighestFitInd(bests);
5:      newInd ← FIHC(CreateRandomInd());
6:      g_newInd ← rand(1, n);
7:      newInd.probeLink ← DLED(newInd, g_newInd);
8:      if AnyNewDepend(newInd.probeLink, IndLinkSet) then
9:          IndLinkSet ← IndLinkSet+ DLED(newInd, [1; n])
10:     empVIG ← empty;
11:     for each IndLink ∈ IndLinkSet do
12:         if DependCovered(newInd.probeLink, IndLink) then
13:             empVIG ← empVIG + Ind;
14:     levels[0] ← levels[0] + newInd;
15:     for each lev ∈ levels do
16:         filtInds ← filterByProbe(lev, empVIG);
17:         DSM ← CreateDSM(filtInds);
18:         combDSM ← empVIG + DSM;
19:         clusters ← GetLTClusters(combDSM);
20:         imprInd ← OM(newInd, clusters, filtInds);
21:         if imprInd ≠ newInd then
22:             levels[lev + 1] ← levels[lev + 1] + imprInd;
23:             lastSuccClusters ← clusters;
24:         newInd ← imprInd;
25:     filtBests ← filterByProbe(bests, empVIG);
26:     if size(filtBests)=0 then
27:         bests ← bests + newInd;
28:     else
29:         for each best ∈ filtBests do
30:             if fitness(best) < fitness(newInd) then
31:                 newInd ← OM(newInd, lastSuccClusters, best);
32:             else
33:                 best ← OM(best, lastSuccClusters, newInd);
34:         bests ← RemoveInds(bests, filtBests);
35:         bests ← bests+ GetBestInd(filtBests + newInd);
36:     curBest ← GetHighestFitInd(bests);
37:     if fitness(startBest)<fitness(curBest) then
38:         IndLinkSet ← IndLinkSet+ DLED(curBest, [1; n])
```

cDLED-P3M, similarly to P3 [10], uses a pyramid-like structure to handle the population and is parameter-less. Its general procedure is presented in Pseudocode 1. In each iteration, a new individual *newInd* is randomly created and optimized by FIHC (line 5). The

underlying structure of *newInd* is probed for a single, randomly chosen gene g_{newInd} (lines 6-7). If probing the dependencies of gene g_{newInd} has discovered any new dependency that was not known before, then the Ind-linkage of *newInd* is obtained and added to the Ind-linkages list (line 9). Before climbing, Ind-linkages are filtered based on probed dependencies of *newInd* and a conditional eVIG for *newInd* is constructed (lines 10-13). Then, *newInd* climbs up the pyramid as in P3 (lines 15-24) with the following differences. First, on each level, we consider only those individuals whose probed dependencies are fully covered by the conditional eVIG (if there are no such individuals, then we consider them all). For the filtered individuals, we create an SLL-based DSM (lines 16-17). Finally, we add the conditional eVIG and DSM and use such a matrix to create clusters for OM. The idea behind this step is the same as in HLL [29, 32] – we use ELL-based linkage as a dominating one and SLL-based linkage to differentiate the dependencies found by ELL.

After climbing, we filter the set of best-found individuals (using the same procedure as while searching for mating individuals), *bests*, and create *filtBests* (line 25). Each of *filtBests* is then mixed with *newInd* using the last set of clusters that were a part of the last successful OM (lines 29-33). We may assume that the individuals in *filtBests* and *newInd* optimize the same base function. In cDLED-P3M, we wish to preserve a single best individual for each base function. Thus, we remove all *filtBests* from *bests* (line 34) and insert into *bests*, the best individual from the set of individuals created by *filtBests* and *newInd* (line 35). The results analysis shows that the number of *bests* in all of the cDLED-P3M runs quickly becomes equal to *fns*. Thus, it increases the pressure on improving these best-found individuals without violating the diversity of the pyramid (which would happen if the best-found solutions for two or more base functions were told from each other). Finally, similarly to P3-DLED and LT-GOMEA-DLED [30], if the best-found fitness has increased, then we obtain an Ind-linkage for an individual related to this best-found fitness (line 38).

5 Results

5.1 Experiments setup

To create BOF problems as base functions, we consider typical problems frequently employed in GA-related research concerning single-objective optimization. The set of base problems includes standard deceptive and bimodal functions concatenations (formulas (5) and (12), respectively) of order 5 and 6, respectively, cycling traps (overlapping by 1 bit), NK-fitness landscapes, Ising Spin Glasses and Max3Sat with $ClauseRatio = 4.27$. For each base problem, we consider the following number of base functions: $fns = 1, 2, 4, 6, 8$. For each problem, we execute 20 runs with different, randomly chosen masks and the permutations π.

To show that cDLED and the mating restrictions mechanism based on cDLED are the keys to cDLED-P3M effectiveness, we compare it with P3M. P3M is the same optimizer but without cDLED and all mechanisms that refer to it, i.e., the construction of conditional eVIG dedicated to *newInd* and the mating restrictions mechanisms to filter the donor individuals. Finally, cDLED-P3M is compared with LK-GOMEA [11]. Finally, to show that multi-structured problems are hard to solve without dedicated mechanisms, the competing optimizer set is supplemented by the binary version of LT-GOMEA [1, 25], the predecessor of LK-GOMEA.

LK-GOMEA spends a significant part of its computational budget on constructing linkage models for its individuals. Therefore, fitness function evaluations (FFE) may not be a fair stop condition [24, 27]. Thus, we use the time-based stop condition set to four hours. To assure the fairness of the comparison, all optimizers were joined in the single programmistic project and shared the implementation of all considered problems. All optimizers were coded in C++, and for all of them, we have used the source codes provided by their authors[1]. All the experiments were conducted on PowerEdge R7425 Dell server AMD Epyc 7601 2.2GHz 256GB RAM. The number of computation processes was one less than the number of physical processor cores. A similar experimental protocol was used in [25, 30, 31]. The complete source code is available on GitHub[2].

5.2 Main results

The scalability analysis of all the considered optimizers is presented in Figures 2 and 3. We also present the generalized comparison of cDLED-P3M with competing optimizers (Table 7). The ranking is made in the following way. If, for a given problem, optimizer A solves test cases of larger problem size (i.e. is successful in at least 80% of the runs) than optimizer B, then it is considered a better one. If two optimizers solve a given problem for the same problem size and it was the largest problem size considered, then we perform a Sign test (significance level of 5%) to check if the differences in median time until finding the optimal solution are statistically significant. Depending on the statistical test result, one optimizer can be found to be better than the other, or both optimizers can be equal. Finally, if two optimizers solve a given problem for the same maximal problem size but do not solve this problem for larger considered sizes, then both optimizers are considered equal.

As explained in [22], the problems consisting of standard deceptive functions (Formula (5)) are suitable to solve by SLL-using optimizers. Therefore, we may expect that for the BOF problems using the deceptive-5 function (concatenation and a cycling trap), LK-GOMEA will scale significantly better than cDLED-P3M. However, as presented in Figures 2a-2e, cDLED-P3M scales similarly to LK-GOMEA for all considered *fns*. It is statistically faster for *fn=1*, *fn=2*, and slower for *fn=6*, *fn=8* (Table 7). LT-GOMEA is competitive to cDLED-P3M and LK-GOMEA for a lower number of *fns*, but starting from *fn=4*, it scales significantly worse. A similar situation takes place for the cycling trap using the deceptive-5 function (Figures 2f-2j), where the curvatures referring to LK-GOMEA and cDLED-P3M are even more similar, and LT-GOMEA is unable to scale equally well starting up from *fn=4*.

The situation is significantly different for the concatenation and a cycling trap using the bimodal-6 deceptive function (Figures 2k-2o and 2p-2t, respectively). As shown in [22, 25, 30], such problems may be hard to solve for SLL-using optimizers. cDLED-P3M uses ELL hybridized with SLL, and, as expected, for both of these problems, it scales significantly better than both GOMEAs (although for bimodal-6 and bimodal-6 cycling trap with *fn=1*, it is statistically

[1] https://github.com/8uurg/Multistructured-Problems-LK-GOMEA
[2] https://github.com/przewooz/conditionalDLED

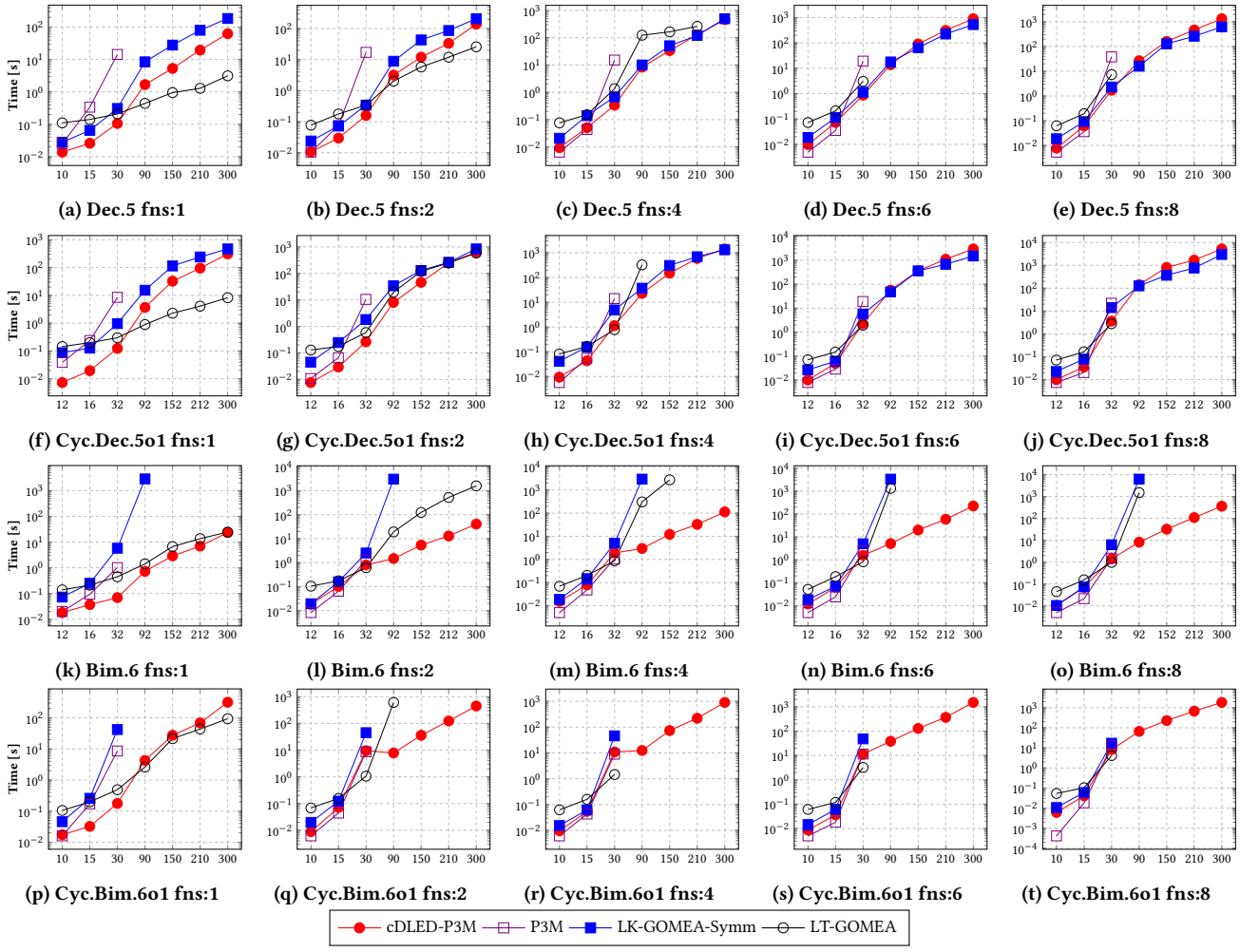

Figure 2: Time scalability for BOF using deceptive functions (at least 80% successful runs, x-axis – the number of bits)

Table 7: cDLED-P3M vs competing optimizers – a generalized comparison (ranking described at the beginning of Section 5.2)

	cDLED-P3M vs LK-GOMEA			cDLED-P3M vs P3M			cDLED-P3M vs LT-GOMEA		
	Better	**equal**	**worse**	**Better**	**equal**	**worse**	**Better**	**equal**	**worse**
Dec.5	2	1	2	**5**	0	0	3	0	2
Cyc-Dec.5 o1	1	3	1	**5**	0	0	3	1	1
Bim.6	**5**	0	0	**5**	0	0	**4**	1	0
Cyc-Bim.6 o1	**5**	0	0	**5**	0	0	**4**	1	0
NK-Land.	**5**	0	0	**5**	0	0	**4**	0	1
ISG	0	1	4	**5**	0	0	0	0	**5**
Max3Sat	0	1	4	0	1	4	0	1	4
	18	6	11	**30**	1	4	**18**	4	13

equal to LT-GOMEA). For all problems using deceptive functions, the P3M baseline scales significantly worse than cDLED-P3M.

In Figure 3, we present the scalability for the BOF problems using well-known overlapping problems. For NK-fitness landscapes (Figures 3a-3e), cDLED-P3M scales significantly better than all other optimizers (except for $fn = 1$, for which LT-GOMEA is faster). For ISG-using BOFs (Figures 3f-3j), LT-GOMEA is significantly better than LK-GOMEA and cDLED-P3M. The latter two optimizers scale

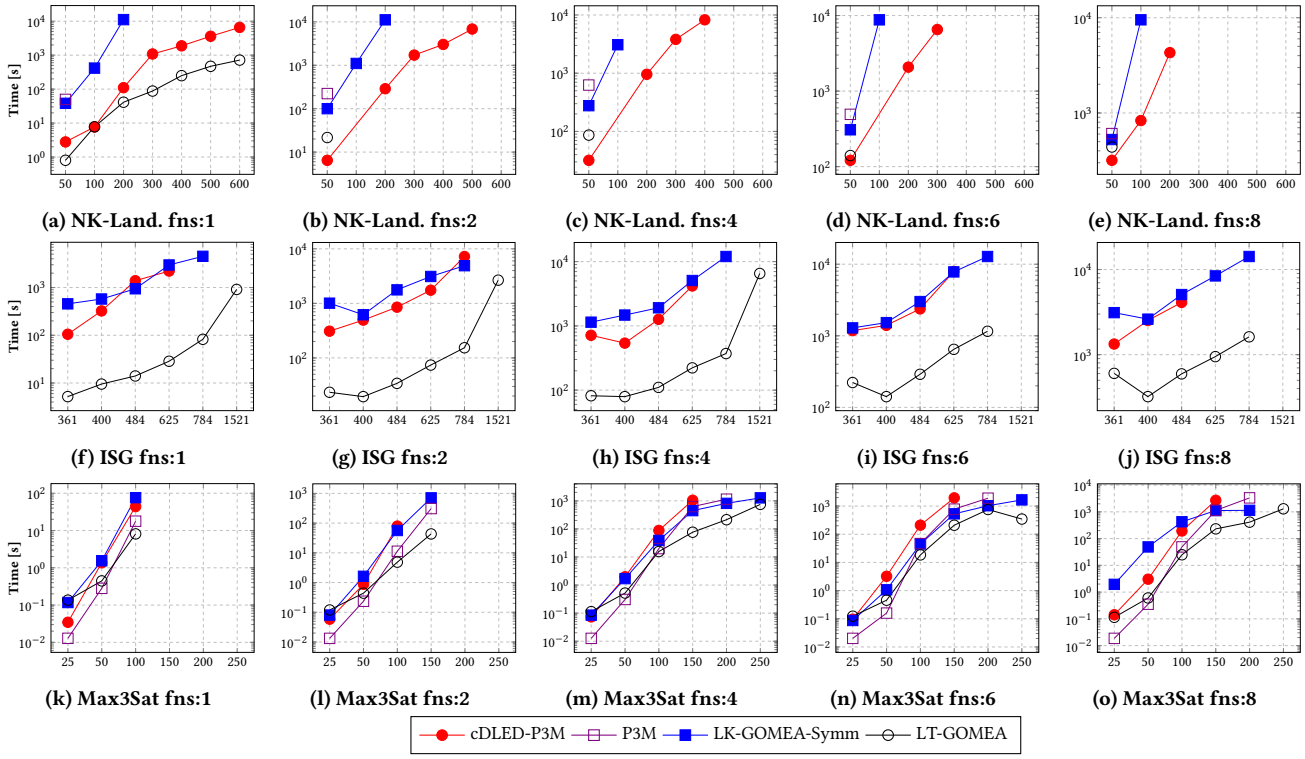

Figure 3: Time scalability for BOF using overlapping problems (at least 80% successful runs, x-axis – the number of bits)

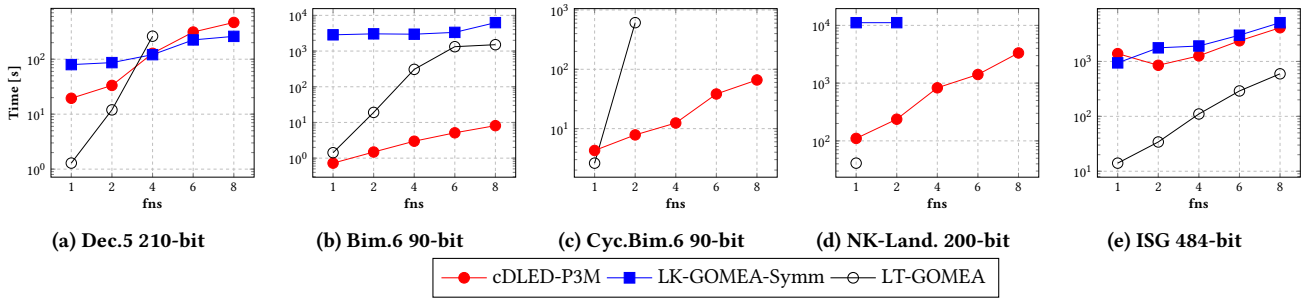

Figure 4: The influence of *fns* on the time-based scalability

similarly, although LK-GOMEA solves slightly larger instances for all fn except for $fn = 2$, for which both optimizers are equal. For Max3Sat, all optimizers (including P3M) scale similarly. However, LT-GOMEA is the fastest and scales slightly better for large *fns*. The situation for BOFs using ISG and Max3Sat is surprising because LT-GOMEA does not use any substructure-modelling mechanisms (the same as P3M). The explanation of this phenomenon seems to be as follows. Two instances of ISG or Max3Sat may have the same parameters (e.g., length and dependent variable pairs percentage, etc.) but some of these instances may be significantly easier to optimize than others. Thus, it is likely that LT-GOMEA and P3M, naturally, focus on optimizing only one base function. The one that is easy to solve. This explanation is supported by the results of P3M that outperforms cDLED-P3M for some of the *fns* (see Table 7).

As presented in Table 7, the proposed cDLED-P3M is highly competitive to the competing optimizers. The lowest advantage cDLED-P3M has over LT-GOMEA. However, this is caused by the fact that for all considered problems, we include their $fn = 1$ into the comparison. For such test cases, the substructure modelling mechanisms included in cDLED-P3M (and in LK-GOMEA) are only an additional cost. Thus, such a test case set prefers LT-GOMEA. If we consider only test cases with $fn \geq 2$, then cDLED-P3M would be better than LT-GOMEA for 18 problems, equal for 2 and worse for 8, which seems significantly more convincing.

The comparison between cDLED-P3M and LK-GOMEA shows that there are some problem types suitable for a given optimizer. cDLED-P3M performs significantly better for all problems using NK-landscapes and the Bimodal-6 function. Oppositely, LK-GOMEA

Table 8: General linkage costs

	DLED cost		DLED quality	
	Min	**Max**	**Min**	**Max**
Dec.5	0.52	**0.78**	0.66	0.79
Cyc.Dec.5 o1	0.21	0.48	0.54	0.75
Bim.6	0.49	0.74	0.68	0.75
Cyc.Bim.6 o1	**0.09**	0.29	0.52	0.73
NK-land.	0.43	**0.63**	0.56	0.89
ISG	**0.05**	**0.61**	0.91	**0.99**
Max3Sat	**0.12**	0.48	**0.37**	0.72

is a better choice for BOF based on ISG and Max3Sat instances. Finally, for problems using the Dec-5 function, the performance of both optimizers is similar (although cDLED-P3M seems faster for lower fn values and slower for the higher ones).

5.3 ELL decomposition costs of cDLED-P3M

The core of cDLED-P3M is based on DLED-based problem structure decomposition. ELL techniques are expensive [25, 30]. Therefore, we shall investigate the overall cost of DLED decomposition and its quality. As a DLED cost, we consider the percentage of the total run time spent on DLED. As a quality measure, we consider a percentage of discovered true dependencies. Such a measure is sufficient because DLED never reports false linkage [30]. In Table 8, we report the DLED costs for the largest problem instances for which cDLED-P3M was successful (at least 80% successful runs).

For problems based on deceptive functions, the DLED cost is relatively high (up to even 78%), and its quality is in the range of 50% up to 79%. cDLED-P3M scales relatively well for all these problems. Thus, a reasonable conclusion is that discovering and using even less than 80% of the dependencies is enough to solve these problems effectively. The situation is similar for BOF problems using NK-landscapes – the DLED cost is about 50%, and the quality of ELL linkage that is sufficient to find the optimal is in the range of 50-90%. For ISG-based problems, the high DLED-based linkage quality is obtained in all of the successful runs - it is always higher than 90%. However, the costs are in the range from 5% up to even 61%. Thus, it seems that ISG-based problems are relatively easy to decompose, but finding an optimal solution, in many runs, is dependent on the fitness landscape of the particular test case. The observation seems to be supported by Figure 4e – for $fn = 2$ cDLED-P3M finds an optimal solution faster than for $fn = 1$. The reason is that if there are two ISG structures, then one of them may have an easier-to-solve landscape. Thus, solving the "easier" substructure allows finding the optimal solution faster than in the $fn = 1$ case, where we encounter only one structure that may be difficult to solve. Thus, on the one hand, increasing the number of substructures makes the problem harder to solve, but on the other hand, if at least one of these substructures is easier to solve than the other, then the problem may become easier to solve. This conclusion is also supported by cDLED-P3M results for Max3Sat – Figures 3k and 3n show that cDLED-P3M may solve larger instances when there are more substructures. Finally, for Max3Sat-based BOF instances, we can see that in many runs, the SLL-based decomposition was enough for cDLED-P3M to solve these instances (the lowest minimal quality enough to solve a test case), which is coherent with the conclusion that LT-GOMEA can solve multi-structured Max3Sat problem well even for large fn.

Table 8 shows that the cDLED costs are high in many runs. Recently, new propositions were formulated to overcome this problem and reduce the costs of the DLED-based techniques family to very low values [31, 32, 36]. However, none of these techniques is directly usable in cDLED. These techniques can be divided into two groups. The first group [31] focuses on discovering a given dependency only once, while cDLED needs to detect dependencies between dependencies. Thus, it must rediscover known dependencies as well. The second group [32, 36] computes DLED-based decomposition as a side-effect of a local optimizer. Thus, the DLED decomposition is computed before an individual becomes locally optimal and each modification made by a local optimizer may refer to a different base function. In cDLED, we wish to increase the probability that a given set of dependencies refers to the same base function. Therefore, the promising future work direction is to propose new mechanisms for lowering cDLED costs without violating its assumptions.

6 Conclusions

In this paper, we propose cDLED-P3M that employs ELL- and SLL-based techniques and is highly competitive to other state-of-the-art GA-based optimizers while solving multi-structured problems. The core of the proposed optimizer is based on two mechanisms, i.e., cDLED and the mating restrictions mechanism of the donor individuals. Both mechanisms seem highly suitable for multi- and many-objective optimization, where these kinds of mechanisms are frequently used but in a much simpler form. For instance, mating restrictions in MOEA/D [41] and population clustering in Multi-Objective GOMEA [18] consider various neighbourhood forms rather than analyzing an underlying problem structure. Using these mechanisms based on the discovered conditional dependencies may be a step towards proposing more effective optimizers dedicated to multi- and many-objective optimization.

Another promising future work direction is proposing the ELL cost reduction mechanisms dedicated to cDLED-P3M. Similarly, the hybridization of LK-GOMEA and cDLED-P3M may lead to high-quality results. However, the main objective of future work will be to propose an optimizer that aims to solve WOF problems that seem even harder to handle than BOF.

Acknowledgements

This work was supported by the Polish National Science Centre (NCN) under Grant 2020/38/E/ST6/00370 and through the project "DAEDALUS - Distributed and Automated Evolutionary Deep Architecture Learning with Unprecedented Scalability" with project number "NWO-18373" of the research programme "NWO-Open Technology Programme" which is (partly) financed by the "NWO-Dutch Research Council (NWO)". Other financial contributions as part of this project have been provided by "CElekta-Elekta AB" and "CORTECLC-Ortec Logiqcare B.V."

References

[1] Peter A.N. Bosman, Ngoc Hoang Luong, and Dirk Thierens. 2016. Expanding from Discrete Cartesian to Permutation Gene-pool Optimal Mixing Evolutionary Algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference*

2016 (GECCO '16). ACM, 637–644.

[2] Peter A.N. Bosman and Dirk Thierens. 2013. More Concise and Robust Linkage Learning by Filtering and Combining Linkage Hierarchies. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation (GECCO '13)*. ACM, 359–366.

[3] Anton Bouter, Stefanus C. Maree, Tanja Alderliesten, and Peter A. N. Bosman. 2020. Leveraging Conditional Linkage Models in Gray-Box Optimization with the Real-Valued Gene-Pool Optimal Mixing Evolutionary Algorithm. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference (GECCO '20)*. Association for Computing Machinery, 603–611.

[4] Ping-Lin Chen, Chun-Jen Peng, Chang-Yi Lu, and Tian-Li Yu. 2017. Two-edge Graphical Linkage Model for DSMGA-II. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '17)*. ACM, 745–752.

[5] Wenxiang Chen, Darrell Whitley, Renato Tinós, and Francisco Chicano. 2018. Tunneling between Plateaus: Improving on a State-of-the-Art MAXSAT Solver Using Partition Crossover. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '18)*. Association for Computing Machinery, 921–928.

[6] Kalyanmoy Deb and David E. Goldberg. 1993. Sufficient Conditions for Deceptive and Easy Binary Functions. *Ann. Math. Artif. Intell.* 10, 4 (1993), 385–408.

[7] Arkadiy Dushatskiy, Tanja Alderliesten, and Peter A. N. Bosman. 2021. A Novel Approach to Designing Surrogate-Assisted Genetic Algorithms by Combining Efficient Learning of Walsh Coefficients and Dependencies. *ACM Trans. Evol. Learn. Optim.* 1, 2, Article 5 (jul 2021), 23 pages.

[8] Arkadiy Dushatskiy, Marco Virgolin, Anton Bouter, Dirk Thierens, and Peter A. N. Bosman. 2021. Parameterless Gene-pool Optimal Mixing Evolutionary Algorithms. (Sept. 2021). arXiv:2109.05259 http://arxiv.org/abs/2109.05259

[9] David E. Goldberg, Kalyanmoy Deb, and Jeffrey Horn. 1992. Massive Multimodality, Deception, and Genetic Algorithms. In *Parallel Problem Solving from Nature 2, PPSN-II*, Reinhard Männer and Bernard Manderick (Eds.). Elsevier, 37–48.

[10] Brian W. Goldman and William F. Punch. 2014. Parameter-less Population Pyramid. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation (GECCO '14)*. ACM, 785–792.

[11] Arthur Guijt, Dirk Thierens, Tanja Alderliesten, and Peter A. N. Bosman. 2022. Solving Multi-Structured Problems by Introducing Linkage Kernels into GOMEA. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '22)*. Association for Computing Machinery, 703–711.

[12] Georges R. Harik and Fernando G. Lobo. 1999. A Parameter-less Genetic Algorithm. In *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation - Volume 1 (GECCO'99)*. 258–265.

[13] R. B. Heckendorn. 2002. Embedded Landscapes. *Evolutionary Computation* 10, 4 (2002), 345–369.

[14] Shih-Huan Hsu and Tian-Li Yu. 2015. Optimization by Pairwise Linkage Detection, Incremental Linkage Set, and Restricted / Back Mixing: DSMGA-II. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation (GECCO '15)*. ACM, 519–526.

[15] Marcin M. Komarnicki, Michal W. Przewozniczek, and Tomasz M. Durda. 2020. Comparative Mixing for DSMGA-II. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference (GECCO '20)*. Association for Computing Machinery, 708–716.

[16] Marcin Michal Komarnicki, Michal Witold Przewozniczek, Halina Kwasnicka, and Krzysztof Walkowiak. 2023. Incremental Recursive Ranking Grouping for Large-Scale Global Optimization. *IEEE Transactions on Evolutionary Computation* 27, 5 (2023), 1498–1513.

[17] S. Kullback and R. A. Leibler. 1951. On Information and Sufficiency. *Ann. Math. Statist.* 22, 1 (03 1951), 79–86.

[18] Ngoc Hoang Luong, Han La Poutré, and Peter A.N. Bosman. 2018. Multi-objective Gene-pool Optimal Mixing Evolutionary Algorithm with the Interleaved Multi-start Scheme. *Swarm and Evolutionary Computation* 40 (2018), 238 – 254.

[19] Masaharu Munetomo and David E. Goldberg. 1999. Linkage identification by non-monotonicity detection for overlapping functions. *Evol. Comput.* 7, 4 (dec 1999), 377–398.

[20] Kalia Orphanou, Dirk Thierens, and Peter A. N. Bosman. 2018. Learning Bayesian Network Structures with GOMEA. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '18)*. Association for Computing Machinery, 1007–1014.

[21] Martin Pelikan and David E. Goldberg. 2006. *Hierarchical Bayesian Optimization Algorithm*. Springer Berlin Heidelberg, 63–90.

[22] Michal W. Przewozniczek, Bartosz Frej, and Marcin M. Komarnicki. 2020. On Measuring and Improving the Quality of Linkage Learning in Modern Evolutionary Algorithms Applied to Solve Partially Additively Separable Problems. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference (GECCO '20)*. Association for Computing Machinery, 742–750.

[23] Michal Witold Przewozniczek, Bartosz Frej, and Marcin Michal Komarnicki. 2025. From Direct to Directional Variable Dependencies—Nonsymmetrical Dependencies Discovery in Real-World and Theoretical Problems. *IEEE Transactions on Evolutionary Computation* 29, 2 (2025), 490–504.

[24] Michal W. Przewozniczek and Marcin M. Komarnicki. 2018. The Influence of Fitness Caching on Modern Evolutionary Methods and Fair Computation Load Measurement. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO '18)*. ACM, 241–242.

[25] Michal W. Przewozniczek and Marcin M. Komarnicki. 2020. Empirical Linkage Learning. *IEEE Transactions on Evolutionary Computation* 24, 6 (Dec 2020), 1097–1111.

[26] Michal Witold Przewozniczek and Marcin Michal Komarnicki. 2021. Empirical problem decomposition — the key to the evolutionary effectiveness in solving a large-scale non-binary discrete real-world problem. *Applied Soft Computing* 113 (2021), 107864.

[27] Michal W. Przewozniczek and Marcin M. Komarnicki. 2021. Fitness Caching - From a Minor Mechanism to Major Consequences in Modern Evolutionary Computation. In *2021 IEEE Congress on Evolutionary Computation (CEC)*. 1785–1791.

[28] Michal Witold Przewozniczek and Marcin Michal Komarnicki. 2023. To Slide or Not to Slide? Moving along Fitness Levels and Preserving the Gene Subsets Diversity in Modern Evolutionary Computation. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '23)*. ACM, 955–962.

[29] Michal W. Przewozniczek, Marcin M. Komarnicki, Peter A. N. Bosman, Dirk Thierens, Bartosz Frej, and Ngoc Hoang Luong. 2021. Hybrid Linkage Learning for Permutation Optimization with Gene-Pool Optimal Mixing Evolutionary Algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO '21)*. Association for Computing Machinery, 1442–1450.

[30] Michal W. Przewozniczek, Marcin M. Komarnicki, and Bartosz Frej. 2021. Direct Linkage Discovery with Empirical Linkage Learning. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '21)*. Association for Computing Machinery, 609–617.

[31] Michal W. Przewozniczek, Renato Tinós, Bartosz Frej, and Marcin M. Komarnicki. 2022. On Turning Black - into Dark Gray-Optimization with the Direct Empirical Linkage Discovery and Partition Crossover. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '22)*. Association for Computing Machinery, 269–277.

[32] Michal Witold Przewozniczek, Renato Tinós, and Marcin Michal Komarnicki. 2023. First Improvement Hill Climber with Linkage Learning – on Introducing Dark Gray-Box Optimization into Statistical Linkage Learning Genetic Algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '23)*. Association for Computing Machinery, 946–954.

[33] Dirk Thierens. 2010. The Linkage Tree Genetic Algorithm. In *Parallel Problem Solving from Nature, PPSN XI: 11th International Conference, Proceedings, Part I*. 264–273.

[34] Dirk Thierens and Peter A.N. Bosman. 2011. Optimal Mixing Evolutionary Algorithms. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation (GECCO '11)*. ACM, 617–624.

[35] Dirk Thierens and Peter A.N. Bosman. 2013. Hierarchical Problem Solving with the Linkage Tree Genetic Algorithm. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation (GECCO '13)*. ACM, 877–884.

[36] Renato Tinós, Michal W. Przewozniczek, and Darrell Whitley. 2022. Iterated Local Search with Perturbation Based on Variables Interaction for Pseudo-Boolean Optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '22)*. Association for Computing Machinery, 296–304.

[37] Renato Tinós, Darrell Whitley, and Francisco Chicano. 2015. Partition Crossover for Pseudo-Boolean Optimization. In *Proceedings of the 2015 ACM Conference on Foundations of Genetic Algorithms XIII (FOGA '15)*. Association for Computing Machinery, 137–149.

[38] D. Whitley. 2019. Next generation genetic algorithms: a user's guide and tutorial. In *Handbook of Metaheuristics*. Springer, 245–274.

[39] L. Darrell Whitley, Francisco Chicano, and Brian W. Goldman. 2016. Gray Box Optimization for Mk Landscapes Nk Landscapes and Max-Ksat. *Evol. Comput.* 24, 3 (Sept. 2016), 491–519.

[40] Szymon Wozniak, Michal W. Przewozniczek, and Marcin M. Komarnicki. 2020. Parameter-Less Population Pyramid for Permutation-Based Problems. In *Parallel Problem Solving from Nature – PPSN XVI*, Thomas Bäck, Mike Preuss, André Deutz, Hao Wang, Carola Doerr, Michael Emmerich, and Heike Trautmann (Eds.). Springer International Publishing, 418–430.

[41] Qingfu Zhang and Hui Li. 2007. MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition. *IEEE Transactions on Evolutionary Computation* 11, 6 (Dec. 2007), 712–731.