Empirical Linkage Discovery in Bi-Objective Optimization

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

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

Renato Tinós
Dep. of Computing and Mathematics
University of São Paulo
Ribeirão Preto, Brazil
rtinos@ffclrp.usp.br

Abstract

In complex problems, variable subsets may have a joint non-monotonical influence on function value. Therefore, variation operators in many single-objective (SO) state-of-the-art optimizers leverage such dependent variable sets to improve effectiveness and efficiency. In multi-objective optimization (MO), we optimize multiple objective functions and each may have different dependencies. Thus, choosing relevant dependencies to improve a solution is challenging in MO. To overcome this difficulty, we can transform the MO problem into a set of SO problems (that may be infinite) and discover the dependencies for each SO problem separately. However, dependency discovery is expensive, even for a single problem. Performing it for each SO problem separately seems unacceptable. Moreover, the dependencies of the scalarized problem are neither necessarily a subset nor a superset of the dependencies of all objective functions, making choosing appropriate dependencies impossible. Two variables dependent in all objective functions can be independent in their scalarization, and two variables independent in all objective functions can be dependent in their scalarization. Therefore, we propose the bi-objective non-monotonicity check (BONM). BONM is a linkage learning technique that uses only a single check to discover weight vector ranges for which a given variable pair is dependent concerning the non-monotonicity check. Limiting our proposition only to scalarization using weight vectors may deteriorate its applicability for the MO problems with concave Pareto fronts. Nevertheless, it enables using gray-box-dedicated operators in the black-box setting for MO problems. Finally, the proposed parameter-less optimizer that employs BONM significantly outperforms the competing state-of-the-art MO optimizers.

CCS Concepts

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

Keywords

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

ACM Reference Format:
Michal W. Przewozniczek, Marcin M. Komarnicki, and Renato Tinós. 2025. Empirical Linkage Discovery in Bi-Objective Optimization. In *Foundations of Genetic Algorithms XVIII (FOGA '25), August 27–29, 2025, Leiden, Netherlands.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3729878.3746625

1 Introduction

If variables have a joint non-monotonical influence on function value, they should be processed together in variation operations that aim at improving a solution at hand. Therefore, such variables are considered dependent. A *problem structure* created by a network of such dependencies may be complicated, making choosing an appropriate variation mask challenging. Nevertheless, the exact solvers seem to always leverage their effectiveness and efficiency by using the knowledge about problem structure [34]. Gray-box optimizers can use the problem structure known *a priori* to support variation masks, which are the key element of highly effective operators [32, 33].

In black-box optimization, to use dependencies we must discover them first. Statistical Linkage Learning (SLL) predicts dependencies using statistical analysis of variable value frequencies in the population. SLL may lead to excellent results [2, 9, 12] and can be considered as cost efficient. The disadvantage of SLL is that it may suffer from *false linkage*, i.e., reporting two independent variables as dependent [25]. False linkage can significantly deteriorate the effectiveness of SLL-using optimizers [23]. Empirical Linkage Learning (ELL) performs direct checks (e.g., the non-linearity [32] or the non-monotonicity check [19]) for variable dependency. ELL is proven to report only the true dependencies and never reports false linkage [27]. The disadvantage of ELL was its high computational cost [25]. However, the recently proposed ELL techniques significantly mitigate this disadvantage [21, 32], which allowed using gray-box operators in black-box optimization and obtaining a significant effectiveness increase [29].

In MO optimization problems (in binary domains), we consider m objective functions $F(\boldsymbol{x}) = (f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \dots, f_m(\boldsymbol{x}))$, where $\boldsymbol{x} = (x_1, x_2, \dots, x_n)$ is a binary vector of size n (we assume that each objective function is maximized). Solution $\boldsymbol{a}$ *dominates* $\boldsymbol{b}$ if and only if $f_i(\boldsymbol{a}) \geq f_i(\boldsymbol{b})\ \forall i \in \{1, 2, \dots, m\}$ and $F(\boldsymbol{a}) \neq F(\boldsymbol{b})$. A set of all non-dominated solutions (Pareto-optimal solutions) is a Pareto-optimal set $\mathcal{P}_S$. A set of objective value vectors of all Pareto-optimal solutions is a Pareto-optimal front $\mathcal{P}_F$. For some problems, the size of $\mathcal{P}_F$ may be large (or even infinite if the domain is continuous). Therefore, in MO, we usually expect to find a resulting Pareto front (PF) that is a good approximation of $\mathcal{P}_F$ [16].

We can use scalarization to obtain an SO problem from an MO one. We can use weight vectors for this purpose: $\tilde{f}(\boldsymbol{x}) = \sum_{i=1}^{m} \omega_i f_i(\boldsymbol{x})$, where $\omega_i \in \mathbb{R}, 0 \leq \omega_i \leq 1$, and $\sum_{i=1}^{m} \omega_i = 1$. Let

$\tilde{x}$ be an optimal solution to $\tilde{f}$. Then, $\tilde{x}$ is a part of $\mathcal{P}_S$ and $f(\tilde{x})$ is a part of $\mathcal{P}_F$. Thus, for any MO problem with a convex $\mathcal{P}_F$, we can define a set of its scalarizations $C = \{\tilde{f}_1(\boldsymbol{x}), \tilde{f}_2(\boldsymbol{x}), ..., \tilde{f}_C(\boldsymbol{x})\}$ such that the objective value vectors of optimal solutions of functions in C define $\mathcal{P}_F$ or its good approximation. Thus, finding a convex $\mathcal{P}_F$ or its good approximation can be interpreted as finding the optimal solutions to functions in C. Such a task can be equivalent to solving many SO problems. In this work, we propose a linkage learning technique that is particularly useful for problems with convex $\mathcal{P}_F$. However, the proposed BONM and an optimizer using it are applicable to solving other bi-objective problems as well.

In this work, we show that for the considered non-monotonical dependencies, the structure of each $\tilde{f}_c(\boldsymbol{x})$ (where $c \in \{1, ..., C\}$) can be different, and the set of its dependencies is neither necessarily the subset nor the superset of dependencies that occur in objective functions $f_i(\boldsymbol{x})$. Indeed, two variables that are independent in all objective functions, can be non-monotonically dependent in some $\tilde{f}_c(\boldsymbol{x})$. Consequently, two variables dependent in every objective function can be independent in some $\tilde{f}_c(\boldsymbol{x})$.

Above, we show that solving an MO problem (in some cases) can be equivalent to solving a set of SO problems. Thus, using the knowledge about their problem structure shall significantly improve the effectiveness of solving each of them. Therefore, the general motivations behind this work are as follows:

M1. Consider an MO problem as a set of SO problems obtained by using a weighted sum (even if this may be unfavourable for solving some MO problems or finding some parts of their $\mathcal{P}_F$). We chose weighted sum because it does not raise additional dependencies, which may be the case for other scalarization types (this issue is thoroughly analyzed in the latter part of this work).

M2. Propose a linkage learning technique that, using a single check, discovers the weighted sum ranges for which a given pair of variables is dependent.

M3. Use the knowledge about discovered weighted sum ranges for which variable pairs are dependent to efficiently obtain a set of variable dependencies for a given weight vector that defines an SO problem.

M4. For a given SO problem (obtained from MO by using a weight vector), use the knowledge about variable dependencies to search for high-quality solutions for this SO problem.

In black-box optimization, even if we know the functions in C (which we do not), we do not know their structures and decomposing each function separately is impossible due to computational cost. Therefore, (following motivation M2), we propose the bi-objective non-monotonicity check (BONM), a linkage learning technique that, with a single check, can discover the weight ranges (not necessarily continuous) for which a given pair of variables is dependent. Thus, it allows of the effective and efficient use of ELL in MO, which is the key novelty proposed by this work. Using BONM, we propose classifying the pairs of dependent variables depending on the distribution of the weight ranges for which a given dependency occurs. This classification improves understanding of which mechanisms are suitable for solving a given MO problem.

The rest of this paper is organized as follows. The related work is presented in the next section. The third section describes the proposed linkage learning (LL) technique, the proposed classification of variable dependency distribution, and the proposed optimizer. Section 4 presents the experiment results. The last section concludes this work and identifies the most promising future research directions.

2 Related Work

2.1 Variable dependency concepts

SLL techniques analyze the frequencies of gene values in the Genetic Algorithm (GA) population to discover dependencies between variables. Mutual Information [15] is commonly used to measure dependency strength. The Dependency Structure Matrix (DSM) is constructed to represent the network of dependencies. Using DSM, we can obtain clusters of dependent genes and use them as masks in the optimization process (usually in various mixing operators). Linkage Trees (LT) [31] are commonly used for this purpose, but other options exist, too [12].

The process of LT construction is as follows. First, we create the leaves by assigning a single leaf to each variable (gene). Then, using DSM, we join the most dependent nodes until creating a single node (root) containing all variables. The nodes can be interpreted as clusters of dependent variables of increasing size. Thus, they are used as masks for variation operators. More information about LTs and their construction can be found in [9, 27, 31].

Optimal Mixing (OM) [30] is a variation operator that is an alternative to standard crossover. In OM, we use an LT node (a cluster of dependent genes) to replace gene values in the *source* individual with the genes from the *donor* individual. If this operation deteriorates source's fitness, then it is reverted or preserved otherwise. Thus, OM includes crossover and selection functionality. Indeed, in Linkage Tree Gene-pool Optimal Mixing Evolutionary Algorithm (LT-GOMEA) [2] OM replaces crossover and selection. No mutation is used. To avoid specifying the population size, LT-GOMEA uses the population-sizing scheme [11]. Thus, it creates many subpopulations of increasing size which are created during the run (each new subpopulation doubles the size of the largest one created that far). LT-GOMEA is parameterless. Its versions were shown to be effective in many different domains [4, 20].

The order of LT nodes considered by OM is random [31]. However, some optimizers consider the shorter nodes first [9]. Additionally, some works show that preserving or rejecting source modification if fitness does not change may be the key to solving some problems [26].

SLL does not cause any additional Fitness Function Evaluations (FFE). Its main disadvantage is that it does not decompose well some types of problems [25] and may report false linkage. Using linkage models of low quality may significantly deteriorate the effectiveness of SLL-using optimizers [23].

In SO, the non-linearity check [18] discovers a dependency between variables x_g and x_h if there exists $\boldsymbol{x}$, for which the following condition holds:

$$f(\boldsymbol{x}) + f(\boldsymbol{x}^{g,h}) \neq f(\boldsymbol{x}^{g}) + f(\boldsymbol{x}^{h}) \quad (1)$$

where $(\boldsymbol{x}^{i})$ is $\boldsymbol{x}$ with the i-th gene flipped, and $\boldsymbol{x}^{i,j}$ is $\boldsymbol{x}$ with the i-th and j-th genes flipped.

Consider, $f_{onemax}(\boldsymbol{x}) = u(\boldsymbol{x})$, where $u(\boldsymbol{x})$ is sum of binary values in $\boldsymbol{x}$ (so called *unitation*). In f_{onemax}, we can greedily optimize each gene separately, and we are guaranteed to find the optimal solution no matter what the starting solution is. Thus, it is intuitive to consider all variables in f_{onemax} as independent.

Indeed, the non-linearity check will not discover any variable dependencies in f_{onemax}. However, for $f_{sqOnemax}(x) = (u(x))^2$, the non-linearity check will discover all variables as dependent on each other, which is counter-intuitive because a greedy optimizer will find the optimal solution to $f_{sqOnemax}$ in the same way as for f_{onemax}. Thus, from the optimization point of view, the variables in $f_{sqOnemax}$ are independent as well.

The above example shows that for some problems, the non-linearity check may become oversensitive and discover dependencies that are irrelevant to the optimization process. Using such irrelevant dependencies to construct the variation masks may deteriorate their quality [13, 19].

The non-monotonicity check [19] ignores the irrelevant dependencies discovered by the non-linearity check. It finds x_g and x_h dependent if at least one of the following conditions is true [21, 24]:
C1. $f(\mathbf{x}) < f(\mathbf{x^g})$ & $f(\mathbf{x^h}) \geq f(\mathbf{x^{g,h}})$
C2. $f(\mathbf{x}) = f(\mathbf{x^g})$ & $f(\mathbf{x^h}) \neq f(\mathbf{x^{g,h}})$
C3. $f(\mathbf{x}) > f(\mathbf{x^g})$ & $f(\mathbf{x^h}) \leq f(\mathbf{x^{g,h}})$
C4. $f(\mathbf{x^h}) < f(\mathbf{x^{g,h}})$ & $f(\mathbf{x}) \geq f(\mathbf{x^g})$
C5. $f(\mathbf{x^h}) = f(\mathbf{x^{g,h}})$ & $f(\mathbf{x}) \neq f(\mathbf{x^g})$
C6. $f(\mathbf{x^h}) > f(\mathbf{x^{g,h}})$ & $f(\mathbf{x}) \leq f(\mathbf{x^g})$

Note that the non-monotonicity check will discover dependencies neither for f_{onemax} nor $f_{sqOnemax}$, which is an expected result.

The recent research concerning the optimization of real-world SO problems confirms that using the non-monotonicity check is favourable over using the non-linearity check [28]. **Therefore, in this work, unless stated otherwise, we consider the non-monotonical dependencies whenever we refer to the *dependency* term.**

2.2 Partition Crossover and Missing Linkage Detection

Partition Crossover (PX) [33] is a gray-box operator and considers Variable Interaction Graph (VIG) that stores information about pairwise variable dependencies, e.g., it can store all non-monotonically dependent variable pairs (although in gray-box-related works, it is typical to consider non-linear dependencies [34, 35]). In gray-box, VIG is user-supported and is guaranteed to contain only and all existing dependencies. For parent individuals $\boldsymbol{x}_{p1}$ and $\boldsymbol{x}_{p2}$, PX removes all VIG vertices that refer to variables equal in $\boldsymbol{x}_{p1}$ and $\boldsymbol{x}_{p2}$. Then, it creates the clusters of variables connected by the remaining VIG. Using a PX mask m_{px}, we can create $\boldsymbol{x}'_{p1} \leftarrow \boldsymbol{x}_{p1} + m_{px}(\boldsymbol{x}_{p2})$, where $\boldsymbol{x}'_{p1}$ is a copy of $\boldsymbol{x}_{p1}$ with the genes from $\boldsymbol{x}_{p2}$ marked by m_{px}. Analogously, we can obtain $\boldsymbol{x}'_{p2}$. Fitness relations between $\boldsymbol{x}_{p1}$, $\boldsymbol{x}_{p2}$, $\boldsymbol{x}'_{p1}$ and $\boldsymbol{x}'_{p2}$ will be as follows [29]:
$(f(\boldsymbol{x}'_{p1}) < f(\boldsymbol{x}_{p1})$ **and** $f(\boldsymbol{x}'_{p2}) > f(\boldsymbol{x}_{p2}))$ **OR**
$(f(\boldsymbol{x}'_{p1}) > f(\boldsymbol{x}_{p1})$ **and** $f(\boldsymbol{x}'_{p2}) < f(\boldsymbol{x}_{p2}))$ **OR**
$(f(\boldsymbol{x}'_{p1}) = f(\boldsymbol{x}_{p1})$ **and** $f(\boldsymbol{x}'_{p2}) = f(\boldsymbol{x}_{p2}))$

ELL techniques employing the non-monotonicity check are proven to report only true dependencies [27]. We can use their output to create empirical VIG (eVIG) [32]. Similarly to gray-box VIG, eVIG contains only the true dependencies but can also miss some of them. In black-box optimization, we can obtain linkage using ELL, but executing the non-monotonicity check for every variable pair is expensive and does not have to be necessary. The missing linkage detection mitigates both these disadvantages [29]. It uses PX in black-box by replacing VIG with eVIG. After obtaining offspring, it checks if their fitness meets the conditions listed above. If these conditions are not met, then a missing linkage between at least one gene from the PX mask and the rest of the genotype is detected. Thus, the non-monotonicity check can be executed only if some linkage is missing and for the limited number of pairs to check.

2.3 Optimizers using population pyramid

Parameter-less Population Pyramid (P3) [9] employs the aforementioned SLL, LTs and OM. Its main novelty is the new way of population management that resembles a pyramid. P3 uses many subpopulations, each denoted as a *level*. Each level maintains its separate DSM and LT. In each iteration, P3 creates a new individual (denoted as *climber*). Initially, the climber is optimized by the First Improvement Hill Climber (FIHC). FIHC is a greedy optimizer that flips genes in a random order, the modification is preserved if it improves fitness. In each FIHC iteration, every gene is flipped once. The iterations are executed until at least one gene was modified in the prior iteration. A copy of the FIHC-optimized climber is added to the first pyramid level. Then, the climber is mixed with subsequent pyramid levels using OM. If mixing with a given level improves the climber, then its improved copy is added to the higher pyramid level (a new level is created, if necessary). Thus, the higher the pyramid level, the higher the number of improvements a given individual has received. P3 starts its procedure with an empty pyramid.

P3 is parameterless, which is important for practice. It uses many LTs in parallel (each for a level), which is important for effective optimization of *overlapping* problems [25]. In overlapping problems, many variables are indirectly dependent (if variable A depends on B, and B depends on C, then A is indirectly dependent on C). Such problems are hard to solve, and many real-world problems belong to this class [5].

P3 employs SLL that may report false linkage, which, in turn, can deteriorate the effectiveness and efficiency of an optimizer [25]. Therefore, replacing SLL with ELL may improve it. In [27], ELL is introduced into P3 and LT-GOMEA making it effective for some problems that are hard to decompose for SLL but deteriorating their effectiveness for many overlapping problems. Therefore, the Dark Gray-box Genetic Algorithm (DGGA) employs a different idea [29]. DGGA employs the same procedure as P3, but SLL and OM are replaced with ELL, which is obtained using the missing linkage detection procedure and PX. DGGA uses the non-monotonicity check to ensure the discovery of only true dependencies, i.e., avoid obtaining false linkage. Missing linkage detection assures that linkage is discovered only when necessary. PX was shown useful for solving overlapping problems [33]. All these features, combined in one optimizer, make it highly effective [29]. Finally, DGGA can be considered P3, with SLL and OM replaced by ELL and PX.

2.4 Linkage learning optimizers in multi-objective optimization

Using SLL in MO seems challenging. In SO, all individuals solve the same optimized function. Thus, the statistical analysis may reveal the predicted dependencies within the same function. However, MO can be considered as a set of SO problems. Each may have a different structure, and the population of individuals should search for high-quality results for all these SO problems. Performing statistical analysis over the population of individuals that solve many problems which may have significantly different structures is not reasonable. Therefore, the Multi-Objective Gene-pool Optimal Mixing Evolutionary Algorithm (MO-GOMEA) [16], in each iteration, clusters the population concerning their representation in the objective space. Then, it performs SLL for each such subpopulation separately and mixes individuals within the subpopulation using OM. Each such subpopulation is evolved separately and is supposed to converge to a different part of PF.

Multi-Objective Parameter-less Population Pyramid (MO-P3) [22] uses a different strategy. In each iteration, MO-P3 randomly chooses a weight vector and optimizes a climber in the SO manner concerning this vector. MO-P3 uses the pyramid-like population management inherited from P3 [9]. Thus, it is a multi-population optimizer that maintains a separate decomposition model for each level. The intuition is that some of these models will be useful to optimize a given PF part.

3 Non-monotonicity Checking in Bi-objective Optimization

The experimental verification of DGGA shows that it is highly competitive [29]. However, the non-monotonicity check is limited to SO. Therefore, in the first subsection, we propose the bi-objective non-monotonicity check (BONM). BONM discovers the weight ranges for which a given pair of variables is dependent. Therefore, in the second subsection, we analyze variable dependency distributions obtained from BONM and propose a classification of these distributions. Some of these distributions denoted as the *middle* may be considered counter-intuitive. In Section 3.3, we propose a new optimizer joining BONM, the missing linkage detection, and PX in the MO-P3-based framework [9, 22].

3.1 Proposed bi-objective non-monotonicity check

Any MO problem may be considered a set of SO problems that can be obtained using scalarization. Each of these SO problems may have its separate problem structure, i.e., the set of variable dependencies. BONM obtains information about all underlying structures of all SO problems (that are a result of MO scalarization), even if their number is infinite. To this end we define the scalarization function (for bi-objective problems): $\tilde{f_w}(\boldsymbol{x}) = (1-w)f_1(\boldsymbol{x}) + wf_2(\boldsymbol{x})$, where $w \in [0,1]$. Thus, if $w < 0.5$, then the first objective influences the scalarized fitness more, while if $w > 0.5$, then the situation is the opposite. We compute four objective vectors (in an analogy to four fitness values in the non-monotonicity check) $F(\boldsymbol{x})$, $F(\boldsymbol{x^g})$, $F(\boldsymbol{x^h})$, and $F(\boldsymbol{x^{g,h}})$. By solving a set of linear equations, we can compute for which ranges of w each one of the conditions C1-C6 hold. Thus, we obtain ranges of w for which x_g and x_h are dependent with respect to solution $\boldsymbol{x}$.

Let us consider the *Trap3-Inverse Trap3* problem [16, 22] defined using standard deceptive function and its inverse version:

$$dec(u) = \begin{cases} k-1-u & \text{if } u < k \\ k & \text{if } u = k \end{cases} \quad dec_{inv}(u) = \begin{cases} u-1 & \text{if } u > 0 \\ k & \text{if } u = 0 \end{cases} \tag{2}$$

where u is the sum of gene values, so-called *unitation* [6], and k is the deceptive function size. In the example we consider $n = 9$, $k = 3$, $f_1(\boldsymbol{x})$ is a concatenation of three dec functions and $f_2(\boldsymbol{x})$ is a concatenation of three dec_{inv} functions. The first function for both objectives is defined on the first three bits, the second on bits 4-6, and the third on the last three bits. We consider $\boldsymbol{x} = 010\ 111\ 000$, and check the dependency between genes $g = 4$ and $h = 6$. Then, $F(\boldsymbol{x}) = (1{+}3{+}0 = 4, 0{+}2{+}3 = 5)$, $F(\boldsymbol{x^4}) = (1,4)$, $F(\boldsymbol{x^6}) = (1,4)$, and $F(\boldsymbol{x^{4,6}}) = (2,3)$. Thus, for the given $\boldsymbol{x}$, genes 4 and 6 are dependent for $w \in [0, 0.5]$. No dependency is discovered for $w > 0.5$. However, if for the same problem, we would use BONM for $\boldsymbol{x} = 110\ 000\ 110$, then the dependency between x_4 and x_6 would be discovered for $w \in [0.5, 1]$. This example shows that after testing the two solutions, linkage discovered by BONM will show that x_4 and x_6 are dependent for all values of w.

The above example concerns *missing linkage*. As the non-monotonicity check may be unable to discover the true dependency for a given individual $\boldsymbol{x}$ in SO, BONM may be unable to discover the true dependency for some w ranges in MO. Therefore, in BONM, we store and summarize the discovered dependency ranges for each pair of genes.

Note that scalarization types other than the weighted sum may be inappropriate for BONM. To show this phenomenon, let us analyze the following example. We will consider the Zeromax-Onemax problem [16, 22] defined as $f_{onemax}(u) = u$; $f_{zeromax}(u) = n - u$. In the Zeromax-Onemax problem, all the variables are independent, and all solutions are a part of $\mathcal{P}_S$. In this example, we use Tchebychev scalarization in the context of maximization of the bi-objective problem, defined as $\tilde{f_{wT}}(\boldsymbol{x}) = \min\{(1-w)(f_1(\boldsymbol{x}) - f_1^*), w(f_2(\boldsymbol{x}) - f_2^*)\}$ where f^* is the reference solution.

We consider the Zeromax-Onemax instance with $n = 4$, $f^* = (0,0)$, $\boldsymbol{x} = 1100$, and $w = 0.5$. We check the dependency between genes $g = 2$ and $h = 3$. We get, $\tilde{f_{wT}}(\boldsymbol{x}) = \min\{0.5(2-0) = 1, 0.5(2-0) = 1\} = 1$, $\tilde{f_{wT}}(\boldsymbol{x^2}) = 0.5$, $\tilde{f_{wT}}(\boldsymbol{x^3}) = 0.5$, and $\tilde{f_{wT}}(\boldsymbol{x^{2,3}}) = 1$. In such a case, the non-monotonicity check will report the dependency between x_2 and x_3 because $\tilde{f_{wT}}(\boldsymbol{x}) > \tilde{f_{wT}}(\boldsymbol{x^2})$ & $f_{wT}(\boldsymbol{x^3}) < \tilde{f_{wT}}(\boldsymbol{x^{2,3}})$. In the other words – if $x_2 = 1$, then the fitness is higher for $x_3 = 0$, but if $x_2 = 0$, then the fitness is higher for $x_3 = 1$, i.e., changing the value of x_2 influences the value of x_3 that refers to the higher fitness. Thus, x_2 and x_3 are dependent.

The above example shows that the Tchebyshev scalarization may create dependencies that arise from scalarization rather than the considered objective functions. We will get similar observations for the Penalized Boundary Intersection (PBI) [36]. The above phenomenon was also mentioned in the context of SO in [10], where the multi-structured problems are identified. The multi-structure problems occur when two problems with different structures are combined in a non-linear manner. Such problems may occur when the optimized SO function uses a min or max function or the Tchebycheff scalarization. In BONM, we wish to avoid discovering the dependencies that originate from scalarization. Therefore, BONM uses the weighted sum that does not suffer from the above flaw.

3.2 Variable dependency distributions in Multi-Objective Problems

In the previous subsection, we analyzed an example in which a pair of genes is dependent for any value of w. We will denote such a variable dependency distribution as *complete*. However, there are other possibilities. Let us consider an example, where $n = 6$, f_1 is a concatenation of two order-3 *dec* functions defined on genes 1-3 and 4-6, f_2 is a *Zeromax* function. Consider $g = 2$, $h = 3$, and $\boldsymbol{x} = 111\,111$. In this example, in the context of f_1, genes in groups 1-3 and 4-6 depend on each other. However, in the context of f_2, there are no dependencies between genes. BONM will discover that x_2 and x_3 are dependent for $w \in [0, 0.75]$. If a given variable pair is dependent for $w = 0$ and is not dependent for $w = 1$, then we denote such type of variable dependency distribution as *left*. In the opposite situation, we call such a distribution the *right* one.

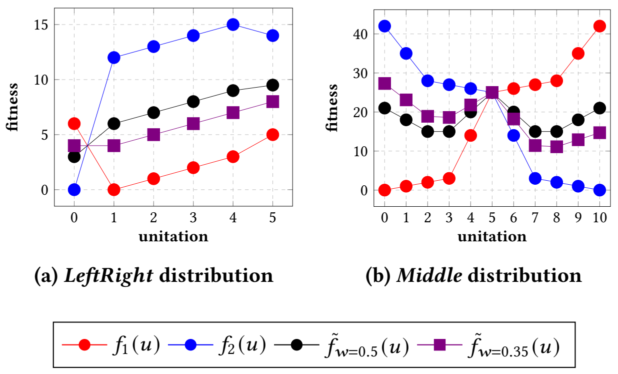


Figure 1: Variable dependency distribution examples

It seems intuitive that if a pair of variables is dependent for the first objective but not for the second, then, with the increase of w, the dependency will disappear (*left* distribution). However, less intuitive types of variable dependency distributions in MO problems are also possible. In Figure 1a, we present two functions of unitation. For both functions, the variables are dependent on each other. Additionally, f_1 is deceptive. However, if we consider the linear combination of these functions, the variables will be dependent only if $w \leq 0.3(3)$ or $w \geq 0.6(6)$. If two variables are dependent for $w = 0$ and $w = 1$, and there exists at least one value of $w \in (0, 1)$, for which these variables are not dependent, then we will call such a variable dependency distribution *LeftRight*.

Finally, the last type of variable dependency distribution considered in this work occurs if both objectives are trivial to solve, but their combination is hard. Figure 1b shows an example of functions f_1 and f_2 that meet such conditions. Each objective can be solved with a greedy algorithm, i.e., it is as trivial to solve as *Onemax* or *Zeromax*. However, the combination of both objectives may be hard to solve. $\tilde{f}_{w=0.35}$ has a curvature similar to bimodal deceptive functions that are hard to solve for SLL-using optimizers [25], with 254 local optima, from which only one is global. The variable dependency distributions where two variables are independent for $w = 0$ and $w = 1$ but dependent for at least one $w \in (0, 1)$, we will denote as *middle*.

In the general case, the number of separate ranges in which two variables are dependent may be large for each variable pair. In the results section, we show that depending on the problem, different types of variable dependency distributions may occur in an MO problem. We also show that the *middle* distributions may occur as a majority type in some real-world problem instances.

3.3 BONM-using optimizer

Here, we describe the details of the proposed Bi-Objective Dark Gray Algorithm (BO-DGGA). BO-DGGA employs the following main mechanisms: BONM, missing linkage detection [29], and PX [33] joined in the MO-P3-based framework [9].

Pseudocode 1 presents the main procedure of BO-DGGA. At the beginning of every iteration, as in MO-P3 [22], we randomly choose w (line 5). Then, we filter those dependencies whose ranges contain w and construct the empirical VIG (line 6). As in P3 [9] and MO-P3, in each iteration, we create *climber* individual and optimize it with FIHC (line 7). After these operations, *climber* is mixed with the subsequent pyramid levels (line 10).

The mixing operation is performed in the same way as in DGGA [29]. First, we create the PX masks for *climber* and every individual in the population (lines 21-26). Each mask includes information for which donor it was created (line 25). Then, the masks are shuffled (line 27) and sorted by length (as in P3 and MO-P3, shorter go first, line 28). Finally, for each mask, we copy the genes from the donor to *climber* (line 30). If the *climber* is improved, then its improved version is returned. If mixing decreases the fitness of *climber*, then, using the same mask, we perform mixing in the opposite direction, i.e., from the climber to the donor (line 35). If this operation does not improve donor's fitness, then we have detected the missing linkage, and we perform linkage discovery (line 37). Such missing linkage detection is the same as in [29]. For each pair of genes selected by missing linkage detection, we perform BONM. If *climber*'s fitness remains the same after mixing, then such a mask is stored in the *slideMasks* list (line 39) and the genotype changes are reverted. Finally, if *clilmber* was not improved, we randomly choose one mask from the *slidingMasks* list and use it for mixing randomly chosen genes from this mask (line 42). If this operation improves the *climber*, then we return its improved version. Otherwise, *MixWithPop* returns *climber* without any modification.

If mixing with a given pyramid level improves the *climber*, then its improved version is added to the next pyramid level (line 12). After climbing *climber* is used as a donor for the individuals in the *bestInds* set (line 16). Mixing is performed in the same way with the difference that instead of using $\tilde{f}_w$, the domination relation is considered. Similarly, *bestInds* are used as donors for *climber* (line 17). Finally, *climber* is added to the *bestInds* set, and all dominated individuals are removed from *bestInds*. If two individuals in the

Pseudocode 1 BO-DGGA general procedure

```
bestInds ← empty;
levels ← levels+ CreateEmptyPop();
depRanges ← empty
while ¬StopConditin do
    w ← GetRandom(0,1);
    eVIG ← CreateVIG(depRanges, w);
    climber ← FIHC(CreateRandomInd(), $\tilde{f}_w$);
    levels[0] ← levels[0] + newInd;
    for each lev ∈ levels do
        climberImpr ← MixWithPop(climber, lev, eVIG, $\tilde{f}_w$);
        if fitness(climberImpr) > fitness(climber) then
            levels[lev + 1] ← levels[lev + 1] + climberImpr;
        climber ← climberImpr;
    climbPop ← CreateEmptyPop() + climber;
    for each best ∈ bestInds do
        best ← MixWithPopDom(best, climbPop, eVIG, $\tilde{f}_w$);
    climber ← MixWithPopDom(climber, bestInds, eVIG, $\tilde{f}_w$);

    bestInds ← bestInds + climber;
    bestInds ← ExcludeDominated(bestInds);

function MixWithPop(climber, pop, eVIG, $\tilde{f}_w$)
    pxMasks ← empty;
    for each ind ∈ pop do
        for each rareMask ∈ GetPXMasks(climber, ind) do
            newMask.mask ← rareMask;
            newMask.donor ← ind;
            pxMasks ← pxMasks + newMask;
    pxMasks ← Shuffle(pxMasks);
    pxMasks ← SortByLen(pxMasks);
    for each m ∈ pxMasks do
        newClimber ← Mix(climber, m.donor, m.mask);
        if $\tilde{f}_w$(newClimber > $\tilde{f}_w$(climber) then
            return newClimber;
        else
            if $\tilde{f}_w$(newClimber) < $\tilde{f}_w$(climber)) then
                modDonor ← Mix(m.donor, climber, m.mask);
                if $\tilde{f}_w$(modDonor) ≤ $\tilde{f}_w$(m.donor)) then
                    depRanges ← depRanges + DiscLL(mask);
            else
                slideMasks ← slideMasks + mask;
                climber ← newClimber;
    if size(slideMasks) > 0 then
        newClimber ← MixMaskUnif(climber, m.donor, m.mask);

        if $\tilde{f}_w$(newClimber > $\tilde{f}_w$(climber) then
            return newClimber;
    return climber;
```

bestInds are the same in the objective space, then one of them (randomly chosen) is removed too.

In BO-DGGA, we use PX crossover to effectively mix individuals [29, 33]. Thus, an empirical VIG is created using BONM. Repeating performing BONM for every variable pair for various individuals would be computationally expensive [27]. Therefore, we use the missing linkage detection procedure [29] to execute BONM only when necessary and only for those variable pairs that have the potential to be dependent. We maintain the *bestInds* set isolated from the population to improve the convergence of the method but do not destroy the population diversity. Finally, for mixing operations involving *bestInds*, we use the domination relation instead of $\tilde{f}_w$ to increase the probability of investigating new regions of PF.

4 Results

The objective of the experiments was to answer the following research questions.

RQ1. ***Does replacing SLL and OM with BONM and PX improve the effectiveness?*** Therefore, we compare BO-DGGA with MO-P3 (see Section 3.3 for justification).

RQ2. ***Do variables with the* middle *dependency distributions occur in real-world problems?*** Therefore, we consider the Uncapacitated Facility Location Problem (UFLP) [17].

RQ3. ***Is constructing an empirical VIG for a given*** w ***superior to considering a joint set of dependencies arising from both objectives?*** Therefore, we compare BO-DGGA with its variation named BO-DGGA-Sum. The BO-DGGA-Sum and BO-DGGA procedures are the same except for two differences. First, BO-DGGA-Sum discovers the dependencies for each objective separately (the decomposition costs are the same as in BONM). Second, in BO-DGGA-Sum, two genes are considered dependent if a dependency is found for any of the two objectives.

RQ4. ***Is BO-DGGA effective in solving problems with* middle *variable dependency distributions?*** Therefore, we propose artificial MO problems with various characteristics of the middle variable dependency distributions. Using these problems, we compare BO-DGGA with other optimizers, including MO-P3 and MO-GOMEA.

RQ5. ***Can we consider BO-DGGA as generally effective?*** Therefore, we consider a wide set of benchmarks commonly used in MO research concerning binary search spaces [16, 22]. On this basis, we compare BO-DGGA with the aforementioned MO-P3 and MO-GOMEA. Additionally, to support a better view, the competing method set (in all experiments) is supplemented with MOEA/D [36] and NSGA-II [8].

4.1 Problem details

4.1.1 Uncapacitated Facility Location Problem (UFLP). UFLP is related to various real-world location problems [14] and is defined as follows. We are given m customers that can be served by n potential facilities. The service cost of serving the j-th customer ($j \in J = \{1, 2, ..., m\}$) by the i-th facility ($i \in I = \{1, 2, ..., n\}$) is $c_{ij} \geq 0$. The cost of setting up a facility in location i has a fixed cost $f_i > 0$. The standard version of this problem is an SO one [17] – we wish to find the minimum summarized cost of facilities placement and the assignment of customers to facilities (each customer is assigned to exactly one facility, the one that raises the lowest cost). In fact, these are two separate objectives defined as follows.

$$f_{sc}(x) = \sum_{i \in I} x_i f_i; \quad f_{ac}(x) = \sum_{j \in J} \min\{c_{i,j} | x_i > 0\}; \tag{3}$$

$$\text{s.t.} \sum_{i \in I} x_i \geq 1 \tag{4}$$

The first objective (f_{sc}) is to minimize the instant cost of setting up the facilities. The second objective (f_{ac}) is to minimize the cost of customers' assignments to facilities. The linear combination of f_{sc} and f_{ac} is NP-hard [14]. We consider 40 different UFLP instances [1]. Note that UFLP is known under different names.

Table 1: Artificial functions of unitation for obtaining various *middle* variable dependency distributions

unitation	0	1	2	3	4	5	6	7	8	9	10
SideSharpDec	0	1	2	3	8	10	11	12	13	18	30
SideFlat	0	1	2	3	4	5	6	7	38	39	40
MiddleSharpDec	0	1	2	3	14	25	26	27	28	35	42
MiddleSharp	0	1	2	3	8	13	14	15	16	17	18
MiddleFlat	0	1	2	3	5	6	7	8	9	10	11

4.1.2 Artificial bi-objective problems with the middle variable dependency distributions. Table 1 presents the definitions of five artificial functions employed to construct bi-objective problems with the *middle* variable dependency distributions. Each function is trivial to solve with a greedy optimizer and does not raise any dependencies. However, their linear combinations with their inverse versions do not share this feature. Figure 2 shows that depending on w, the characteristics of some of them may be found similar to bimodal deceptive functions (see the next subsection). Thus, they can be considered as hard to solve.

4.1.3 Standard benchmarks. In the MO-related literature, the concatenations of standard deceptive functions and their inverse versions are frequently considered objectives (e.g., *Trap3-Inverse Trap3*, for $dec(u)$ and $k = 3$). This work extends this set by step [9], bimodal [7], and noised bimodal [25] deceptive functions.

Step deceptive functions introduce plateaus into standard deceptive functions (Formula 2) and are defined as follows.

$$step\_trap(u) = \left\lfloor \frac{(k-s)(\mod s) + dec(u)}{s} \right\rfloor \tag{5}$$

As in [9], we use $s = 2$. We consider $k = 7$ and $k = 11$, which refer to standard deceptive functions for $k = 3$ and $k = 5$. Therefore, these problems are denoted as *stepDec3* and *stepDec5*, respectively.

We use the bimodal deceptive functions for $k = 6$ and $k = 10$ with a single global optimum defined as follows.

$$bimodal\_trapS(u) = \begin{cases} k/2 - |u - k/2| - 1 & , u \neq k \wedge u \neq 0 \\ k/2 & , u = 0 \\ k/2 + 1 & , u = k \end{cases} \tag{6}$$

Finally, we use the noised bimodal function [25] of order 10 with a single optimum. Its values are presented in Table 2.

In the artificial benchmark set we considered the Leading Ones Trailing Zeros (LOTZ) [16, 22] defined as follows.

$$f_{LO}(x) = \sum_{i=1}^{l-1} \prod_{j=1}^{i} x_j; \quad f_{TZ}(x) = \sum_{i=1}^{l-1} \prod_{j=i}^{l-1} (1 - x_j) \tag{7}$$

Table 2: Bimodal noised deceptive function of unitation ($k = 10$) with a single global optimum

unitation	0	1	2	3	4	5	6	7	8	9	10
function value	4	0	2	1	3	2	3	1	2	0	5

When considering the *Leading Ones* function (f_{LO}), we optimize the number of subsequent ones at the beginning of a genotype. Oppositely, for the *Trailing Zeroes* we wish to increase the number of subsequent zeroes at the end of it.

The multi-objective MAXCUT problem was considered in the same version as in [16, 22]. We used the same instances generated with the approach proposed in [3]. MAXCUT is defined as follows.

Let $G = (V, E)$ be a weighted undirected graph, where $V = (v_1, v_2, \ldots, v_n)$ is a set of n vertices and E is the set of edges (v_i, v_j). Each edge is associated weight $w_{i,j}$. The objective is divide n vertices into two disjoint subsets A and B ($A = V\ B$) and maximize the total weight of edges (v_i, v_j) having $v_i \in A$ and $v_j \in B$. To make the problem bi-objective, we consider two different weight sets.

In the binary representation, each x_i decides if $v_i \in A$ ($x_i = 0$) or $v_i \in B$ ($x_i = 1$). We consider the same problem instances as in [16, 22] and the same reference sets that are equal to $\mathcal{P}_F$ or are its approximation.

The last considered problem was the multi-objective knapsack, which involves n items and m knapsacks (the capacity of s-th knapsack equals c_s). Depending on knapsack s, each item i is associated with weight $w_{i,s}$ and profit $p_{i,s}$. Each item i is either selected for all knapsacks or not selected at all. Solutions are represented as binary strings of size n. A solution is feasible if the summarized weight of all selected items does not exceed the capacity of any knapsack. The objective is to maximize the profit for every considered knapsack.

$$\max_x (f_1(x), f_2(x), \ldots, f_m(x)), \tag{8}$$

$$\text{s.t.} \sum_{i=1}^{n} w_{i,s} x_i \leq c_s \tag{9}$$

where $f_s(x) = \sum_{i=1}^{n} p_{i,s} x_i$ for $s = 1, 2, \ldots, m$ (we consider $m = 2$).

We consider the same problem implementation, instances, and reference sets that are equal to or approximate $\mathcal{P}_F$ as in [16, 22]. During solution evaluation, solutions that violate the constraint are repaired first using the mechanism proposed in [37], i.e., selected items are removed one by one until the constraint for all knapsacks is satisfied. The items with the lowest profit/weight ratio are removed first.

4.2 Experiment setup

The competing method set includes MO-P3, MO-GOMEA, NSGA-II and MOEA/D. Including MO-P3 allows verifying if exchanging SLL and OM for BONM and PX is favourable. MO-P3 and MO-GOMEA allow comparison of BO-DGGA with state-of-the-art optimizers. Finally, NSGA-II and MOEA/D support the comparison with the well-known baseline.

The cost of constructing empirical VIG in BO-DGGA is negligible because, for a given w, for each variable pair, it is enough to check whether w belongs to the dependency ranges discovered for a given pair. The costs of constructing DSM and LTs in MO-P3 and

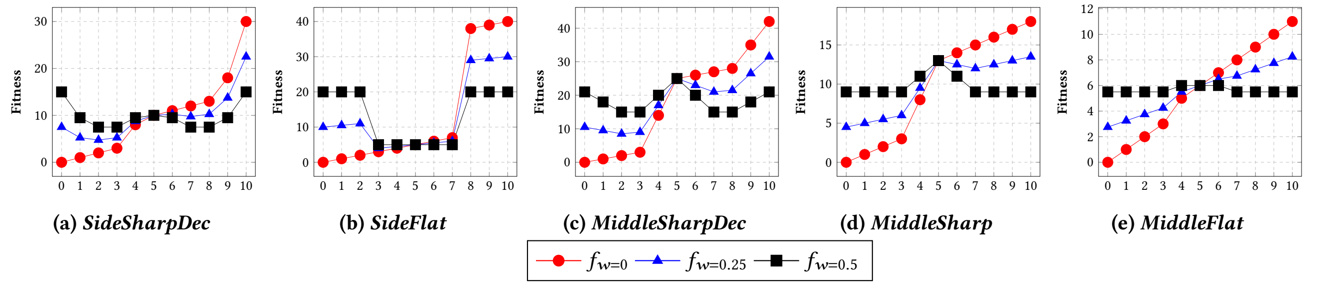


Figure 2: Considered MO problems with the *middle* variable dependency distributions (X axis - problem size)

MO-GOMEA are negligible as well [9]. Therefore, we adopt the FFE-based stop condition. Same as in [16, 22], the computation budget is $25 \cdot 10^6$ FFE.

As a primary quality measure, we use the Inverted Generational Distance (IGD) defined as

$$D_{\mathcal{P}_F \to \mathcal{S}}(\mathcal{S}) = \frac{1}{|\mathcal{P}_F|} \sum_{f^0 \in \mathcal{P}_F} \min_{x \in \mathcal{S}} \{d(f(x), f^0)\}, \tag{10}$$

where $\mathcal{P}_F$ is the Pareto-optimal front, $\mathcal{S}$ is the final front proposed by the optimizer and $d(\cdot,\cdot)$ is the Euclidean distance.

The additional measure was the percentage of $\mathcal{P}_F$ points found. For the problems for which $\mathcal{P}_F$ is not known, we construct pseudo-optimal PF by using all fronts proposed by all optimizers in all runs for a given instance. Such a procedure was also used in [22]. For each instance, we performed 20 runs. If necessary, we use the Wilcoxon signed-rank test to check the statistical significance of the results differences (5% significance level).

All the optimizers were implemented in C++ and joined in one programmistic project, the full source codes and the settings files of all the experiments are available on Github [1]. We have used the original source code of MO-GOMEA [2], MO-P3 [3], MOEA/D[4] and NSGA-II [5]. In NSGA-II, we use bit-flipping mutation with probability $1/n$, uniform crossover (it makes NSGA-II independent of the gene order) probability is 0.9. The results reported in [22] show that for a similar benchmark set, the population size of 400 is a better choice than the the smaller ones for NSGA-II and MOEA/D. In this work, we extend this choice and execute NSGA-II and MOEA/D for 400 and 800 individuals. As a final result for each optimizer we report the better the considered two. The same or similar settings of NSGA-II and MOEA/D can be found in [16, 22].

The experiments concerning UFLP show the effectiveness of BO-DGGA in solving complex real-world problem. Additionally, they show that variable pairs with the *middle* dependency distribution may occur in real-world instances. The effectiveness of the proposed mechanisms and the competing optimizers for problems containing only the *middle* variable dependency distributions is verified using the artificial functions proposed in Section 4.1.2. Finally, the overall BO-DGGA effectiveness is verified using the standard benchamrks presented in Section 4.1.3.

[1]https://github.com/przewooz/moDLED
[2]the source codes used in [16], supported by the authors
[3]https://github.com/przewooz/moP3 (including the P3 implementation from [9])
[4]https://github.com/ZhenkunWang
[5]http://www.iitk.ac.in/kangal/codes.shtml

Table 3: Variable dependency distribution types and BONM costs (median values)

	n	Ep*	Q	C	L	R	LR	M	Cost
Cap111- -Cap134	50	66 69	N/A	0	0	0	0	100	2E+5 3E+5
CapA	100	100	N/A	0	97	0	0	3	2E+6
CapB	100	100	N/A	0	64	0	0	36	2E+6
CapC	100	100	N/A	0	56	0	0	44	3E+6
MidFlt	400	2	100	0	0	0	0	100	3E+6
MidSh	400	2	100	0	0	0	0	100	4E+6
MidShD	400	2	100	0	0	0	0	100	5E+6
SidFl	400	2	100	0	0	0	0	100	4E+6
SidShD	400	2	100	0	0	0	0	100	8E+6
0M-1M	400	0	N/A	N/A	N/A	N/A	N/A	N/A	0E+0
LOTZ	400	100	100	0	0	100	0	0	2E+5
Dec.8	400	2	100	100	0	0	0	0	3E+6
St.3	399	2	100	15	0	0	85	0	1E+6
St.5	396	3	100	1	0	0	99	0	1E+6
Bm.6	402	1	100	100	0	0	0	0	7E+6
Bm.10	90	10	100	100	0	0	0	0	3E+5
BmN.10	60	15	100	100	0	0	0	0	7E+4
MaxCut	50	100	N/A	53	13	13	18	2	2E+4
Knap	750	77	N/A	58	18	15	6	3	8E+6

*Ep - epistasis [%]; Q-quality; Var. dep. distributions [%]: C-*complete*, L-*left*, R-*right*, LR-*LeftRight*, M-*middle*; FFE cost of BONM

For *Trap-Inverse Trap* problems, we consider standard ($k = 8$), step ($k = 7$ and $k = 11$ that extend standard deceptive functions with $k = 3$ and $k = 5$), bimodal ($k = 6$ and $k = 10$), and noised bimodal deceptive functions with a single global optimum ($k = 10$) defined in Section 4.1.3. The inverse versions of considered functions are computed as $f_{inv}(u) = f(k-u)$.

In the SO scenarios, standard deceptive functions were shown to be easy to decompose by SLL, while ELL usually decomposes such problems at a significantly higher cost [23]. Bimodal functions used in this work have one global optimum and $\binom{k}{k/2} + 1$ local optima. They are relatively hard for SLL to decompose, while ELL techniques should decompose them quickly [27]. Bimodal noised functions have even more local optima, and the problems built using them seem more challenging to decompose and solve.

The *Zeromax-Onemax* problem can be considered an easy-to-solve baseline. LOTZ should be solved effectively and efficiently by optimizers using local search. Oppositely, the bi-objective MAXCUT and Knapsack problems can be considered as hard to solve.

4.3 Main results

4.3.1 Variable dependency distributions analysis. In Table 3, we present the percentage of variable dependency distribution types discovered by BONM and its FFE-based cost. These results were obtained in BO-DGGA runs. Whenever it applies, we consider the largest problem instance considered in the subsequent sections. For instances Cap111-Cap134, we report min/max values because the results for these instances were similar. The *epistasis* (*Ep*) is a percentage of variable pairs that were found dependent for any w. Thus, $Ep = 0$ for *Zeromax-Onemax* means no dependencies, and $Ep = 1$ for *CapA* means that every gene is dependent on all others for at least one value of w. Quality is a percentage of discovered dependencies. We report it for problems for which we know the true dependencies.

Note that for real-world instances, the true dependencies remain unknown. In such a case, BONM (same as non-linearity and non-monotonicity checks for SO) can not discover that two variables are independent for any range of w. To do so, one would have to perform BONM for every available $\boldsymbol{x}$, which is not doable due to the practical reasons [24]. Therefore, the high percentage of discovered dependencies for the artificial problem instances (100% in all cases) is a strong premise that for real-world instances, the percentage of discovered dependencies is high as well. However, it is not a proof.

For Cap111-Cap134 (UFLP instances), all variable dependency distributions are of the *middle* type and $Ep < 0.7$. Such results show that the middle distributions may be the only ones to occur in some real-world instances. For CapA, CapB, and CapC, all variable pairs depend on each other for at least some w values. The percentage of the middle variable dependency distributions remains high but is not a majority. This observation shows the significant differences in the nature of Cap111-Cap134 and CapA-CapC instances.

No dependencies were detected for *Zeromax-Onemax*, and no BONM costs were paid because no missing linkage was detected. For LOTZ, all genes were found to be dependent. The decomposition cost was low compared to the overall budget (less than 1%). In all of the runs, the discovered variable dependency distribution *Right*. For trap-inverse trap problems using standard deceptive functions, bimodal, and bimodal noised functions, all variable dependency distributions were correctly discovered as *complete*.

For step deceptive problems, all distributions should be discovered as *complete*, too, but most of them were of the *LeftRight* type. However, these *LeftRight* distributions covered over 99% of the available w range. Thus, most probably, $\mathcal{P}_F$ was found before the dependency ranges were supplemented to become *complete*. For MAXCUT and Knapsack, all variable dependency distribution types were discovered but the percentage of the *middle* ones is negligible.

The above results show that the percentage of the *middle* variable dependency distributions may be high in some real-world instances. However, in the considered MO-dedicated benchmarks, the percentage of such dependency distributions is no more than negligible. Thus, proposing artificial benchmarks with a high percentage of the *middle* distributions is justified and needed. For such problems (proposed in Section 4.1.2), BONM discovered all dependent variables and all variable dependency distributions were correctly classified as the *middle* ones.

The BONM costs reported in Table 3 show that the *missing linkage* detection mechanisms efficiently limit ELL costs in MO. The highest median cost was approximately $8 \cdot 10^6$, which is less than 33% of the computational budget. For many problems, the BONM cost was significantly lower and was enough to discover all dependencies.

4.3.2 UFLP-based comparison. In Table 4, we report the results for those UFLP problem instances for which at least one method from the triple BO-DGGA, MO-P3, and MO-GOMEA could not find the $\mathcal{P}_F$. For Cap111-Cap134 instances, MO-DGGA, MO-P3, and MOEA/D find $\mathcal{P}_F$ in all of the runs. None of the considered optimizers could find $\mathcal{P}_F$ for CapA-CapC instances. However, for these three instances, the best results (for both considered PF quality measures) were reported by MO-GOMEA, while BO-DGGA was the second-best optimizer for these instances (for CapA and CapB, these differences were statistically significant). To summarize, MO-GOMEA outperforms BO-DGGA for UFLP instances with a large $\mathcal{P}_F$ size and low percentage of the *middle* variable dependency distributions. It is likely that for these instances, $\mathcal{P}_F$ is non-convex and, therefore, they are harder to solve BO-DGGA. Additionally, the instances with a high percentage of the *middle* distributions seem hard to solve for MO-GOMEA.

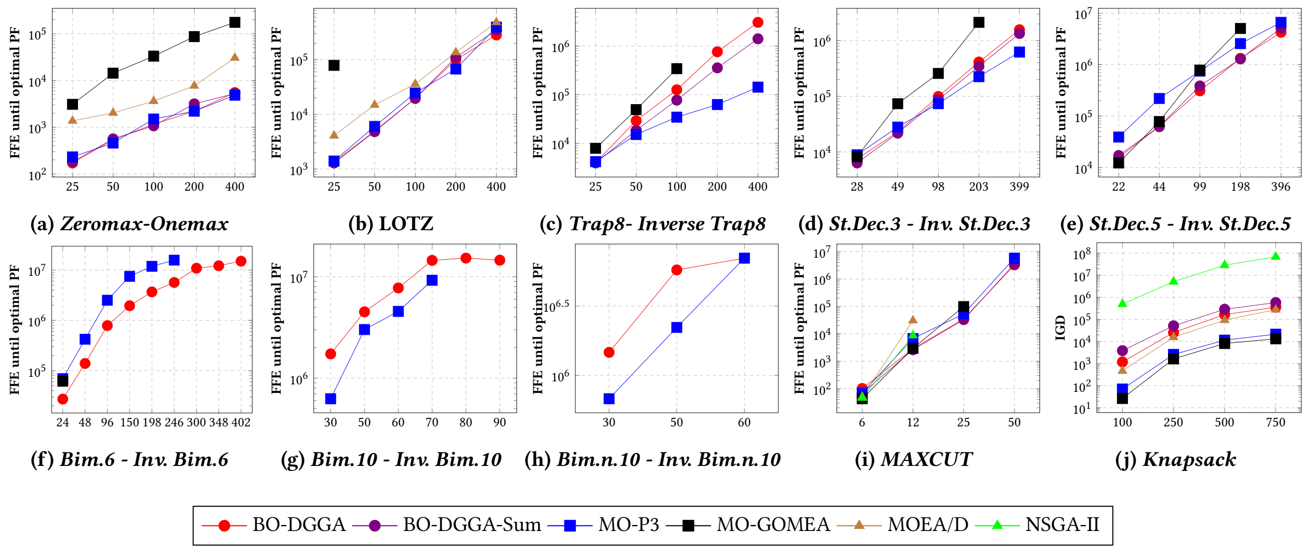
BO-DGGA performed equally well as MO-P3 for Cap111-Cap134 instances. For CapA-CapC instances, BO-DGGA outperforms MO-P3 significantly. Finally, BO-DGGA outperforms BO-DGGA-Sum for every UFLP instance reported in Table 4. Note that NSGA-II was unable to find any part of $\mathcal{P}_F$ in any run.

4.3.3 Result quality for artificial problems with the middle variable dependency distributions. Table 5 shows the results obtained for the artificial problems with the *middle* variable dependency distributions. We report the median percentage of the $\mathcal{P}_F$ points found, IGD, the percentage of computation budget at which the best PF was found for the 400-bit instances. Additionally, in the last column, we report the largest n, for which a given optimizer has found $\mathcal{P}_F$ in all runs. As presented, only BO-DGGA has found $\mathcal{P}_F$ in all runs for all considered problems. As expected, BO-DGGA-Sum was ineffective because it discovers and uses linkage for separate objectives (for these problems, objectives do not have any dependencies). Surprisingly, MO-P3 and MO-GOMEA were ineffective. The reasonable explanation is that using SLL, they could not discover the dependencies between variables, which deteriorated their performance. Note that the computation budget was enough to converge for both optimizers – MO-P3 and MO-GOMEA were finding resulting PF before consuming 1% of the computational budget. The second most effective optimizer was MOEA/D, which was faster than BO-DGGA in finding $\mathcal{P}_F$. However, MOEA/D could not solve the 400-bit *Side-Flat* instance in any of the runs. It seems that the landscape of this instance was too difficult to handle without considering variable dependencies.

4.3.4 Effectiveness comparison for standard benchmarks. Figure 3 presents the scalability analysis of the considered optimizers on a typical MO-dedicated benchmark set. We present only the results with a 100% success rate (i.e., $\mathcal{P}_F$ was found in all 20 runs). Except for MAXCUT and Knapsack, in these experiments, all the variable dependency distributions are *complete*, i.e., all dependencies are true for any value of w. Thus, BONM was not advantageous over the decomposition strategy used by BO-DGGA-Sum. Therefore, the quality of the results reported by BO-DGGA and BO-DGGA-Sum is close in all cases. For *Zeromax-Onemax* and LOTZ (Fig. 3a

Table 4: Main results for UFLP problem (the percentage of $|\mathcal{P}_F|$ found and IGD)

		BO-DGGA		BO-DGGA-Sum		MO-P3		MO-GOMEA		MOEA/D		NSGA-II	
Testcase	$\|\mathcal{P}_F\|$	$\mathcal{P}_F$[%]	IGD	$\mathcal{P}_F$[%]	IGD	$\mathcal{P}_F$[%]	IGD	$\mathcal{P}_F$[%]	IGD	$\mathcal{P}_F$ [%]	IGD	$\mathcal{P}_F$[%]	IGD
Cap111	50	**100**	0.0E+00	98	2.0E+02	**100**	0.0E+00	40	7.2E+06	**100**	0.0E+00	0	1.5E+09
Cap112	50	**100**	0.0E+00	95	4.9E+02	**100**	0.0E+00	40	7.3E+06	**100**	0.0E+00	0	2.7E+09
Cap113	50	**100**	0.0E+00	94	7.2E+02	**100**	0.0E+00	40	7.6E+06	**100**	0.0E+00	0	3.9E+09
Cap114	50	**100**	0.0E+00	95	4.9E+02	**100**	0.0E+00	42	7.0E+06	**100**	0.0E+00	0	6.6E+09
Cap121	50	**100**	0.0E+00	98	1.1E+01	**100**	0.0E+00	42	7.4E+06	**100**	0.0E+00	0	1.5E+09
Cap122	50	**100**	0.0E+00	97	2.4E+02	**100**	0.0E+00	42	7.1E+06	**100**	0.0E+00	0	2.9E+09
Cap123	50	**100**	0.0E+00	96	4.8E+02	**100**	0.0E+00	40	7.0E+06	**100**	0.0E+00	0	4.5E+09
Cap124	50	**100**	0.0E+00	96	3.6E+02	**100**	0.0E+00	42	6.8E+06	**100**	0.0E+00	0	6.7E+09
Cap131	50	**100**	0.0E+00	98	2.4E+02	**100**	0.0E+00	42	7.3E+06	**100**	0.0E+00	0	1.6E+09
Cap132	50	**100**	0.0E+00	94	4.8E+02	**100**	0.0E+00	40	6.5E+06	**100**	0.0E+00	0	2.7E+09
Cap133	50	**100**	0.0E+00	94	4.8E+02	**100**	0.0E+00	42	7.5E+06	**100**	0.0E+00	0	4.2E+09
Cap134	50	**100**	0.0E+00	96	3.6E+02	**100**	0.0E+00	42	7.2E+06	**100**	0.0E+00	0	6.7E+09
CapA	1910	49	1.0E+10	7	8.1E+13	22	4.7E+10	**71**	1.0E+09	24	2.9E+10	0	5.2E+32
CapB	1994	61	8.8E+08	6	8.0E+10	29	5.0E+09	**77**	1.7E+08	37	1.3E+09	0	5.0E+32
CapC	2257	67	2.6E+08	8	6.0E+10	37	9.5E+08	**76**	1.4E+08	39	6.1E+08	0	4.4E+32



(a) *Zeromax-Onemax* (b) LOTZ (c) *Trap8- Inverse Trap8* (d) *St.Dec.3 - Inv. St.Dec.3* (e) *St.Dec.5 - Inv. St.Dec.5*

(f) *Bim.6 - Inv. Bim.6* (g) *Bim.10 - Inv. Bim.10* (h) *Bim.n.10 - Inv. Bim.n.10* (i) *MAXCUT* (j) *Knapsack*

Figure 3: Scalability analysis for standard benchmarks (X axis - problem size)

and 3b), the best optimizers were both BO-DGGA versions, MO-P3 and MOEA/D. MO-GOMEA performed significantly worse (especially for LOTZ, which is caused by the lack of local search in its procedure). NSGA-II was unable to repeatedly find $\mathcal{P}_F$ for any of the considered cases. Similar results were presented in [22]. For *Trap-InvTrap* using the standard deceptive function (Fig. 3c), MO-P3 scaled significantly better than the other methods, which is expected because the SLL technique it employs is highly efficient and precise in decomposing such problems [23]. However, both BO-DGGA versions were the second most effective optimizers. MO-GOMEA, although it uses the same SLL technique as MO-P3, was not as effective because it could not find the whole $\mathcal{P}_F$, which is related to the optimizer procedure rather than to the decomposition technique it employs. MOEA/D and NSGA-II do not use any decomposition. Therefore, they could not find $\mathcal{P}_F$ in any of the runs. Such results are coherent with [22].

The step deceptive functions are slightly harder to decompose for SLL than standard deceptive functions. Therefore, the scalability of MO-P3 and both BO-DGGA versions become similar (Fig. 3d and 3e) when compared to the problem using standard deceptive function. The fourth-best method is MO-GOMEA, while MOEA/D and NSGA-II were unsuccessful in all of the runs. Bimodal deceptive functions are hard to decompose for SLL-based techniques. Therefore, for problems based on bimodal-6 and bimodal-10 functions, BO-DGGA scales significantly better than MO-P3 (Fig. 3f and 3g). No other of the considered optimizers could repeatedly find $\mathcal{P}_F$ for these problems. Finally, the problem based on the bimodal noised function seems harder to decompose and solve than the bimodal-based problems (Fig. 3h). Only BO-DGGA and MO-P3 solve this problem (but only for the limited lengths).

Table 5: Main results for artificial problems with the *middle* variable dependency distributions

	$\mathcal{P}_F$ [%]	IGD	LastUp [%]	n_{max}
	SideSharpDec	$n = 400$	$\|\mathcal{P}_F\| = 401$	
MO-DGGA	**100**	**0.00E+00**	36	**400**
MO-DGGA-Sum	0	2.43E+05	<1	30
MO-P3	0	2.12E+05	<1	100
MO-GOMEA	0	3.18E+05	<1	N/A
MOEA/D	**100**	**0.00E+00**	**<0.01**	**400**
NSGA-II	0	1.87E+05	95	N/A
	SideFlat	$n = 400$	$\|\mathcal{P}_F\| = 1601$	
MO-DGGA	**100**	**0.00E+00**	**70**	**400**
MO-DGGA-Sum	1	3.60E+05	<1	N/A
MO-P3	2	5.12E+05	<1	N/A
MO-GOMEA	0	8.80E+05	<1	N/A
MOEA/D	56	2.38E+01	97	30
NSGA-II	0	6.14E+05	46	N/A
	MiddleSharpDec	$n = 400$	$\|\mathcal{P}_F\| = 401$	
MO-DGGA	**100**	**0.00E+00**	0.32	**400**
MO-DGGA-Sum	1	4.11E+05	<1	N/A
MO-P3	1	4.71E+05	<1	N/A
MO-GOMEA	0	4.16E+05	<1	N/A
MOEA/D	**100**	**0.00E+00**	**<1**	**400**
NSGA-II	0	1.43E+05	89	N/A
	MiddleSharp	$n = 400$	$\|\mathcal{P}_F\| = 401$	
MO-DGGA	**100**	**0.00E+00**	27	**400**
MO-DGGA-Sum	1	7.84E+04	<1	N/A
MO-P3	1	9.09E+04	<1	N/A
MO-GOMEA	0	6.96E+04	<1	N/A
MOEA/D	**100**	**0.00E+00**	**<0.01**	**400**
NSGA-II	0	2.29E+04	<1	N/A
	MiddleFlat	$n = 400$	$\|\mathcal{P}_F\| = 401$	
MO-DGGA	**100**	**0.00E+00**	0.36	**400**
MO-DGGA-Sum	0	1.31E+04	<1	N/A
MO-P3	1	2.05E+04	<1	N/A
MO-GOMEA	0	2.58E+04	<1	N/A
MOEA/D	**100**	**0.00E+00**	**<1**	**400**
NSGA-II	0	9.54E+03	<1	N/A

For MAXCUT (Fig. 3i), the optimizers form three groups that scale similarly. BO-DGGA and MO-P3 are the first group of optimizers that solved 50-bit instances in all runs. MO-GOMEA is a single member of the second group and solves 25-bit instances. Finally, MOEA/D and NSGA-II are in the third group that can find $\mathcal{P}_F$ in all of the runs only for 12-bit instances.

For bi-objective Knapsack, none of the optimizers can find $\mathcal{P}_F$. Therefore, we compare IGD (Fig. 3j). MO-GOMEA and MO-P3 obtain the best results. The BO-DGGA versions and MOEA/D form the second group. Finally, NSGA-II is significantly worse than the other optimizers. These results are similar to those reported in [22].

4.4 Results discussion summary

The results analysis answers the research questions stated at the beginning of this section. The experiment results show that BO-DGGA outperforms MO-P3 significantly. Thus, replacing SLL and OM with BONM and PX may improve the effectiveness of a MO-dedicated optimizer (RQ1). In Section 4.3.1, we show that it is likely that in instances of real-world MO problems, a large percentage of variable pairs can be characterized by the *middle* variable dependency distribution (RQ2). For problems with such variable dependency distributions, BO-DGGA outperforms BO-DGGA-Sum and other considered optimizers, which shows that constructing an empirical VIG for a given w is superior to considering a joint set of dependencies occurring in objective functions (RQ3) and that BO-DGGA is effective in solving such problems (RQ4). Finally, the comparison based on standard benchmarks, two state-of-the-art optimizers and two well-known MO-dedicated optimizers shows that BO-DGGA is effective in solving various bi-objective problems (RQ5).

5 Conclusions

In this work, we propose BONM, a new linkage learning technique dedicated to bi-objective optimization. BONM discovers the dependency for ranges of multi-objective problem scalarizations at once, which allows for obtaining (and comparing) dependency structures for any scalarization. Thus, each scalarization does not have to be decomposed separately. Missing linkage detection assures that the BONM costs remain relatively low. Thanks to the advantages inherited from the non-monotonicity check, BONM can decompose additive and non-additive problems and always discovers only true dependencies.

We analyze the types of variable dependency distributions that may occur in bi-objective optimization. We show that even if two variables are dependent for both objectives, their linear combination is not guaranteed to be dependent (the *LeftRight* distributions). Moreover, we show examples of trivial objectives, which combinations may be NP-hard (some of the UFLP instances) and are characterized by the *middle* dependency distributions. The proposed analysis led to formulating the new benchmark problems.

Finally, we propose BO-DGGA and show that exchanging SLL and OM by BONM and PX may lead to excellent results in bi-objective optimization. For the considered benchmarks set, BO-DGGA outperforms the competing state-of-the-art optimizers for most of the considered problems.

The main directions of future research are as follows. BONM should be extended to handle problems with a larger number of objectives. Using the ideas proposed in [21], the cost of BONM should be reduced. In this work, we employ PX. However, other gray-box SO-dedicated operators can be considered to improve MO optimization, e.g., the perturbation masks employed by Iterated Local Search [32]. Finally, using the propositions from [13], it is possible to adjust BONM-based mechanisms to continuous search spaces. Finally, introducing mechanisms for handling problems with non-convex $\mathcal{P}_F$ to BO-DGGA seems highly promising.

Acknowledgements

This work was supported by the Polish National Science Centre (NCN) under Grant 2020/38/E/ST6/00370. This work was partially supported in Brazil by São Paulo Research Foundation - FAPESP (under grants #2024/15430-5 and #2024/08485-8)

References

[1] J. E. Beasley. 1990. OR-Library: Distributing Test Problems by Electronic Mail. *The Journal of the Operational Research Society* 41, 11 (1990), 1069–1072.

[2] 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.

[3] 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* (Amsterdam, The Netherlands) *(GECCO '13)*. ACM, New York, NY, USA, 359–366. doi:10.1145/2463372.2463420

[4] 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* (Cancún, Mexico) *(GECCO '20)*. Association for Computing Machinery, New York, NY, USA, 603–611. doi:10.1145/3377930.3390225

[5] Wei-Ming Chen, Chung-Yu Shao, Po-Chun Hsu, and Tian-Li Yu. 2012. A test problem with adjustable degrees of overlap and conflict among subproblems. In *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation* (Philadelphia, Pennsylvania, USA) *(GECCO '12)*. Association for Computing Machinery, New York, NY, USA, 257–264. doi:10.1145/2330163.2330201

[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] Kalyanmoy Deb, Jeffrey Horn, and David E. Goldberg. 1993. Multimodal Deceptive Functions. *Complex Systems* 7, 2 (1993).

[8] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6, 2 (April 2002), 182–197. doi:10.1109/4235.996017

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

[10] 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, New York, NY, USA, 703–711. doi:10.1145/3512290.3528828

[11] 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* (Orlando, Florida) *(GECCO'99)*. 258–265.

[12] 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.

[13] Marcin Michal Komarnicki, Michal Witold Przewozniczek, Halina Kwasnicka, and Krzysztof Walkowiak. 2022. Incremental Recursive Ranking Grouping for Large Scale Global Optimization. *IEEE Transactions on Evolutionary Computation* (2022), 1–1. doi:10.1109/TEVC.2022.3216968

[14] Jakob Krarup and Peter Mark Pruzan. 1983. The simple plant location problem: Survey and synthesis. *European Journal of Operational Research* 12, 1 (1983), 36–81. doi:10.1016/0377-2217(83)90181-9

[15] S. Kullback and R. A. Leibler. 1951. On Information and Sufficiency. *Ann. Math. Statist.* 22, 1 (03 1951), 79–86. doi:10.1214/aoms/1177729694

[16] 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. doi:10.1016/j.swevo.2018.02.005

[17] Ehsan Monabbati and Hossein Taghizadeh Kakhki. 2015. On a class of subadditive duals for the uncapacitated facility location problem. *Appl. Math. Comput.* 251 (2015), 118–131. doi:10.1016/j.amc.2014.10.072

[18] M. Munetomo and D.E. Goldberg. 1999. A genetic algorithm using linkage identification by nonlinearity check. In *IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No.99CH37028)*, Vol. 1. 595–600 vol.1. doi:10.1109/ICSMC.1999.814159

[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. doi:10.1162/evco.1999.7.4.377

[20] Chantal Olieman, Anton Bouter, and Peter A. N. Bosman. 2020. Fitness-based Linkage Learning in the Real-Valued Gene-pool Optimal Mixing Evolutionary Algorithm. *IEEE Transactions on Evolutionary Computation* (2020). doi:10.1109/TEVC.2020.3039698

[21] Michal W. Przewozniczek, , Renato Tinós, and Marcin M. Komarnicki. 2023 (in press). 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, New York, NY, USA.

[22] Michal Witold Przewozniczek, Piotr Dziurzanski, Shuai Zhao, and Leandro Soares Indrusiak. 2021. Multi-Objective parameter-less population pyramid for solving industrial process planning problems. *Swarm and Evolutionary Computation* 60 (2021), 100773. doi:10.1016/j.swevo.2020.100773

[23] 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* (Cancún, Mexico) *(GECCO '20)*. Association for Computing Machinery, New York, NY, USA, 742–750.

[24] 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. doi:10.1109/TEVC.2024.3496193

[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. 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* (Lisbon, Portugal) *(GECCO '23)*. ACM, 955–962.

[27] 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* (Lille, France) *(GECCO '21)*. Association for Computing Machinery, New York, NY, USA, 609–617. doi:10.1145/3449639.3459333

[28] Michal W. Przewozniczek, Renato Tinós, Francisco Chicano, Jakub Nalepa, Bogdan Ruszczak, and Agata Wijata. 2025 (in press). On Revealing the Hidden Problem Structure in Real-World and Theoretical Problems Using Walsh Coefficient Influence. In *Proceedings of the Genetic and Evolutionary Computation Conference* (Malaga, Spain) *(GECCO '25)*. Association for Computing Machinery, New York, NY, USA. doi:10.48550/arXiv.2504.13949

[29] 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* (Boston, Massachusetts) *(GECCO '22)*. Association for Computing Machinery, New York, NY, USA, 269–277. doi:10.1145/3512290.3528734

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

[31] 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.

[32] 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* (Boston, Massachusetts) *(GECCO '22)*. Association for Computing Machinery, New York, NY, USA, 296–304. doi:10.1145/3512290.3528716

[33] 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* (Aberystwyth, United Kingdom) *(FOGA '15)*. Association for Computing Machinery, New York, NY, USA, 137–149. doi:10.1145/2725494.2725497

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

[35] 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. doi:10.1162/EVCO_a_00184

[36] Q. Zhang and H. Li. 2007. MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition. *IEEE Transactions on Evolutionary Computation* 11, 6 (Dec 2007), 712–731. doi:10.1109/TEVC.2007.892759

[37] E. Zitzler and L. Thiele. 1999. Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. *IEEE Transactions on Evolutionary Computation* 3, 4 (Nov 1999), 257–271. doi:10.1109/4235.797969