MIHEA in Action: A Reproduction Study and Evaluation on Learning Bayesian Networks from Data

Ruben Nair

TU Delft Delft University of Technology

MIHEA in Action: A Reproduction Study and Evaluation on Learning Bayesian Networks from Data

by

Ruben Nair

Student number: 4953207

to obtain the degree of Master of Science
in Computer Science, track Software Technology, at the Delft University of Technology,
Faculty Electrical Engineering, Mathematics and Computer Science. To be defended publicly on Thursday
the 25th of April, 2024

Thesis committee:

Prof. dr. Peter A.N. Bosman	TU Delft, CWI, supervisor
Dr. A. Panichella	TU Delft
Anton Bouter, PhD.	CWI
Damy Ha, M.Sc.	LUMC

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft Delft University of Technology

CWI Research institute for mathematics & computer science in the Netherlands

Preface

This thesis marks the end of my time as a student at Delft University of Technology. It has been a challenging journey for me, requiring two changes in project direction. I could not have done this without my supervisors. I want to thank my daily supervisors, Anton Bouter and Damy Ha, for their time, support and encouragement through this process, helping me see the bigger picture whenever I got tunnel-visioned on a problem. I also want to thank my supervisor Peter Bosman for his flexibility and support, and for being open-minded and comforting during some of the difficult times. Furthermore, I want to thank Annibale Panichella for taking the time to join my thesis committee.

Doing my Master's thesis in Peter's PhD group (the Evolutionary Intelligence group at CWI) has been a great experience. I really enjoyed getting to know everyone and what they were working on, both during work (breaks) and in some activities outside of research. They introduced me to indoor bouldering, which has been a hobby of mine ever since. For that, I want to thank the whole group.

Finally, I want to thank my family for their love and support throughout this process.

Ruben Nair
Delft, April 2024

Contents

1 Introduction

In the world of optimization, mixed-integer problems, i.e. problems where both discrete and continuous problem variables are optimized, pose unique challenges. These problems can be found in various domains. To name a few examples, training Deep Neural Networks in computer science [1], portfolio optimization in finance [2], optimizing logistics in the automotive industry [3], and creating treatment plans in healthcare [4] can all encounter problems of a mixed-integer nature.

Efficiently solving mixed-integer optimization problems is crucial for optimizing real-world systems and processes, and often proves to be complex due to the structure of these problems and the interplay between variables in the discrete and continuous domains. Over the past couple of decades, Evolutionary Algorithms (EAs) have gained popularity as powerful optimization techniques capable of handling such complex, nonlinear problems. Unlike traditional mathematical optimization methods, EAs are stochastic, population-based algorithms inspired by the principles of natural selection and genetics. These principles were first developed into EAs in several domains around the 1960s and early 70s [5, 6]. In the discrete domain, Genetic Algorithms (GAs) were created. GAs primarily rely on fundamental principles of crossover and mutation, representing solutions as strings of binary digits and applying genetic operators to improve candidate solutions iteratively [5, 7, 8]. In the continuous domain, the Evolution Strategy (ES) was developed. ES solely uses mutation as a variation operator, by sampling from a Gaussian distribution and adding those sampled values to the continuous variables. However, as optimization problems grew in complexity and diversity, these basic approaches faced limitations in scalability and efficiency. To overcome these challenges, model-based EAs were developed, which leveraged probabilistic models to guide the search process more intelligently [9]. Model-based EAs integrate machine learning techniques and probabilistic models to capture and exploit the underlying structure of the optimization problem, leading to enhanced exploitation of the search space.
One of the first EAs to combine discrete and continuous model-based EAs into a single framework suitable for mixed-integer problems is the Mixed-Integer Hybrid Evolutionary Algorithm (MIHEA) [10]. By using an alternating approach between discrete and continuous optimization, MIHEA is better at solving problems with cross-domain dependencies than other existing mixed-integer EAs [10], like the Mixed Integer Evolution Strategies (MIES) algorithm [11]. Despite its potential, MIHEA remains relatively underutilized in research endeavours. A possible explanation for its lack of usage in research could be that the source code has never been released. This thesis seeks to bridge this gap in research by first reproducing MIHEA, followed by applying the algorithm to a novel context: structure learning of Bayesian Networks (BNs).

BNs are probabilistic graphical models that capture probabilistic relationships between random variables, offering a transparent and more interpretable framework for probabilistic reasoning and decision-making. This makes BNs well-suited for the development of explainable AI systems. These models are already widely used in various fields, including medicine [12, 13, 14], finance [15], and ecology [16]. Learning the structure and parameters of BNs from data is essential for making accurate predictions and informed decisions in complex systems. Before the data can be converted into a BN, any columns containing continuous values must first undergo discretization, as BNs are designed to handle discrete data. This discretization of the data can be done simultaneously with learning the structure of the network. Discretizing the continuous nodes in the network solely based on data from the random variables is commonly done by optimizing discrete variables that represent the number of bins a node is split into, and subsequently determining the bin boundaries based on some policy [17]. This discrete formulation of the problem can

be solved using various discrete optimization techniques. However, discretization of the continuous values can also be done using continuous variables and a continuous optimization algorithm. This could give more control over some properties of the bins like their width and placement. Combining the discrete network structure optimization and the continuous node discretization optimization brings this optimization problem into the mixed-integer domain.
Research exploring the application of EAs in the context of mixed-integer Bayesian network structure learning together with node discretization is relatively scarce; most of the currently used approaches perform optimizations focused on one of these aspects alone [18], or do not make use of EAs [19, 20]. The closest comparison would be the recent work done in [21], where the discrete model-based EA known as the Gene-Pool Optimal Mixing Evolutionary Algorithm (GOMEA) is used for both learning the network structure and learning the discretizations. Their algorithm is shown to match or outperform state-of-the-art results in learning randomly generated ground truth networks. However, their method does not use model-based EAs for the continuous aspect of the problem, learning discretizations for nodes representing continuous data. MIHEA does use a model-based EA for continuous optimization, which could lead to improved performance on this task.

This thesis explores the intersection of mixed-integer optimization using model-based EAs for both the discrete and continuous parts of BN structure learning and node discretization. Specifically, we investigate how MIHEA performs when applied to learn BN structures and discretizations from data. This investigation is guided by the following research questions:

How does MIHEA perform when applied to BN structure learning?

- What representation(s) of solutions for structure learning of BNs work well with MIHEA?
- How does MIHEA's performance scale with respect to problem sizes on BN structure learning?
- How does MIHEA compare to the state-of-the-art DBN-GOMEA method in BN structure learning in the MI setting?

The rest of this thesis is structured as follows. Chapter 2 explains relevant background information and introduces terminologies used throughout the rest of this work. After Chapter 2, the reproduction of the MIHEA code is investigated and discussed in Chapter 3. This is followed by Chapter 4, which describes the application of MIHEA to the structure learning of BNs. This chapter also presents the methodology and results of the experiments used to answer the research questions. Finally, the results are discussed in Chapter 5, and the thesis is concluded in Chapter 6, which also highlights avenues for future work.

2

Background

This chapter delves into the background for this thesis, by describing single-objective optimization problems, black-box optimization and mixed-integer domains. After that, the general workings of EAs are discussed. The explanation about EAs focuses on GOMEA [22], an EA that applies to problems with discrete values, and an EA for real-valued problems is introduced, called the incremental adapted maximum-likelihood gaussian model iterated density-estimation algorithm (iAMaLGaM) [23].

2.1. Optimization problems

An optimization problem involves decision variables, an objective function mapping variables to a real value and constraints imposed on the variables. Mathematically, it can be formulated as:

$$\begin{aligned} \text{minimize} \quad & f(x) \\ \text{subject to} \quad & g_i(x) \leq 0, \qquad i = 1, 2, \ldots, p \\ & h_j(x) = 0, \qquad j = 1, 2, \ldots, q \\ \text{where} \quad & x \in \Omega \end{aligned} \tag{2.1}$$

where Ω denotes the set of all possible solutions to the problem and $f : \Omega \to \mathbb{R}$ is the objective function. Furthermore, $g_i(x)$ are inequality constraint functions and $h_j(x)$ are equality constraint functions. If no constraint functions exist (i.e. $p = q = 0$), as is the case for all problems examined in this thesis, the problem is unconstrained. The set of all possible solutions for which $f(x), g_i(x)$ and $h_j(x)$ are defined is called the domain of the problem. Any solution that satisfies all the constraints is called a feasible solution. The other solutions in the domain are called infeasible. A solution $x \in \Omega$ consists of n real-valued numbers, each of which is called a decision variable. The optimal solution to the problem x^* is defined as the best feasible solution, i.e. the solution that satisfies all constraints and $f(x^*) \leq f(x)$ for all x in the feasible domain of the problem. The goal of solving an optimization problem is thus to find the closest approximation to the optimal solution.

Optimization problems can generally be divided into three categories based on the information available during the solving process: white-box, black-box and grey-box optimization.

White-Box Optimization In white-box optimization, the optimizer has direct access to the objective function $f(x)$ and the constraint functions $g_i(x), h_j(x)$. This knowledge can be exploited during the problem-solving process. Because of this, the most efficient solvers often solve such problems analytically or by applying specific techniques that are tailored towards the problem.

Black-Box Optimization In black-box optimization, it is assumed that nothing about the problem is known. A solver is only given a 'black box' that can take an input x and return the corresponding objective value (and feasibility, if constraints exist). This lack of information about the problem means that the adaptability of a solver becomes important to efficiently and effectively find a good solution.

Grey-Box Optimization Grey-box optimization falls somewhere between the previous two categories of optimization. If a problem is categorised as a grey-box problem, some information about the problem is known, or some approximations can be made. This means that white-box approaches are often too specific and therefore not effective or applicable, while the black-box approaches can be too general and don't effectively utilize the (limited) available knowledge.

Furthermore, optimization problems can be classified into three categories depending on the characteristics of the search space Ω: discrete, continuous and mixed-integer problems. In discrete optimization, decision variables can only take on distinct, separate values. To give an example, the travelling salesman problem is a discrete optimization problem where each decision variable represents a city to be visited.

In continuous optimization, decision variables can take any real value within a specified range. Continuous search spaces are infinite by definition since there always exists a number between two values, no matter how close those two values are to each other. In practice, continuous search spaces are limited by the machine precision of the used variables. This search space is still big; assuming that the continuous variables are encoded as doubles in C++, there are about $2^{62} \approx 4.6 \cdot 10^{18}$ possible values between 0.0 and 1.0 alone. An example of a continuous optimization problem is the optimization of weights and biases of a machine learning model.

In mixed-integer optimization, the problem involves a combination of both discrete and continuous decision variables. For example, the facility location problem is a mixed-integer optimization problem. The location of a facility can be modelled with real-valued decision variables, while the type or presence of a facility can be represented by a discrete value [24, 25].

2.2. Evolutionary Algorithms

EAs are optimization algorithms inspired by the principles of natural evolution. This inspiration is reflected in the commonly used terminology. For example, solutions to an optimization problem are often referred to as individuals, solution sets are called populations and the objective value $f(x)$ is called the fitness function. EAs are particularly well-suited for solving complex optimization problems, including black-box scenarios where the objective function and constraints are not explicitly known [26]. The general workings of EAs involve the following key components:

Initialization A population of initial solutions is randomly generated to form the initial population.

Selection Individuals in the population that will contribute to the next generation are selected based on their fitness. Generally, better-performing individuals have a higher chance of being selected. This form of selection mimics the natural selection procedure that is common in nature. Various selection mechanisms, such as tournament selection or a fitness-based selection that selects the top τ percent of the population, can be employed to guide the evolution process. By limiting the selection to a small group with above average fitness, a pressure towards finding better solutions is imposed, also known as the selection pressure. It is also more likely that components that positively contribute to the fitness of an individual, so-called building blocks, are preserved. Additionally, it's worth noting that while the focus is often on selecting better-performing individuals, selection mechanisms could also deliberately select slightly worse solutions to facilitate exploration, thereby ensuring a diverse exploration of the search space.

Variation The individuals that were selected for the next generation undergo variation. There are two common types of variation. Some EAs utilize only one type, while others EAs employ both. The first type of variation is called crossover. During crossover, individuals are grouped in pairs and parts of their genetic information are exchanged to create new offspring. The idea is to combine beneficial traits from different individuals to potentially produce better solutions. The other type of variation is called mutation. During mutation, random changes are introduced into the genetic information of the offspring, promoting diversity in the population. Mutation helps explore new regions of the solution space that might contain better solutions. The main purpose of the variation stage is to explore the search space, looking for solutions with improved fitness.

The iterative process of selection and variation mimics the evolutionary process observed in nature, gradually improving the population over successive generations. This process continues until it reaches

some termination criterion. This could be by exceeding a budget in the amount of generations, evaluations or time spent. Another common termination criterion is if a population has converged, i.e. the best solution no longer changes.

2.2.1. Gene-pool Optimal Mixing Evolutionary Algorithm

The Gene-pool Optimal Mixing Evolutionary Algorithm (GOMEA) is an EA designed to optimize problems consisting of discrete variables [22]. It aims to identify and exploit dependencies between variables. This can be done in several different ways; for this thesis, we utilized the most common version used in literature called the *Linkage Tree* model [27]. GOMEA has demonstrated robustness and efficiency in the realm of single-objective optimization, surpassing both classic Genetic Algorithms and Estimation of Distribution Algorithms [22]. This success prompted further investigation into its applicability across other optimization domains. Over the past decade, researchers have extended GOMEA's capabilities to multi-objective optimization [28], real-valued optimization [29], and genetic programming [30], yielding promising results in each domain.

Family of Subsets

An important concept used in GOMEA is called the Family of Subsets (FOS) [22]. A FOS is a collection of subsets of a main set S. Mathematically, it can be described as a subset of $\mathcal{P}(S)$, the powerset of S. In GOMEA, this set S contains all indices to variables in a solution. The subsets of S in the FOS represent sets of variables whose values are correlated in some way. This FOS is used during the crossover step; variables belonging to the same subset are crossed over together. This allows the algorithm to navigate more efficiently through the search space, reducing the number of evaluations used.

A common way to create the FOS with GOMEA is by using linkage learning to identify sets of correlated variables in the current population. This is achieved by building a hierarchical cluster, known as the *linkage tree*, using a proximity distance measure. The distance measure is based on the correlation between variables or groups of variables. The hierarchical clustering algorithm used is agglomerative and proceeds in a bottom-up manner. Each problem variable starts as its own cluster. The algorithm then recursively joins the closest clusters until only one cluster remains. The resulting linkage tree comprises a total of $2l_d - 1$ clusters, where l_d is the number of discrete variables. The collection of all these clusters is also known as the *Family of Subsets* (FOS). An example of a linkage tree and its corresponding FOS for a solution with five variables is shown in Figure 2.1.

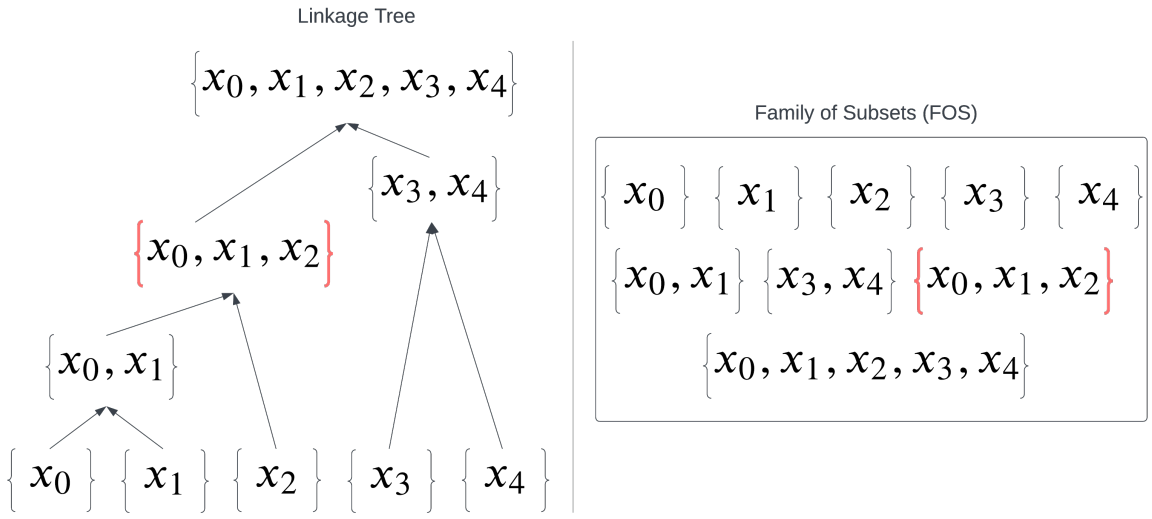


Figure 2.1: An example of a linkage tree built with hierarchical clustering, and the corresponding FOS

Hierarchical Clustering Using Mutual Information

A pivotal aspect of GOMEA's linkage learning is that the hierarchical clustering algorithm requires a suitable distance measure. In this thesis, the distance measure used for linkage learning is based on mutual information, a metric that gauges the shared information between random variables.

Mutual information (I) quantifies the information shared between random variables. It builds on the concept of entropy (H), which is defined as $H(X_k) = -\sum_i p_i(X_k) \log\, p_i(X_k)$, where X_k is a discrete random variable and $p(X_k)$ is the corresponding probability mass function. Using that formula, the mutual information between l discrete random variables is defined as $I(X_1, \ldots, X_l) = \sum_{k=1}^{l} H(X_k) - H(X_1, \ldots, X_l)$. Mutual information is well-suited as a similarity measure for GOMEA. However, to be able to apply it to the hierarchical clustering algorithm, it has to be converted to a distance measure. The distance measure used for this is called variation of information ($d(X_i, X_j)$), calculated as the difference between joint entropy ($H(X_i, X_j)$) and mutual information ($I(X_i, X_j)$):

$$d(X_i, X_j) = H(X_i, X_j) - I(X_i, X_j) = H(X_i|X_j) + H(X_j|X_i)$$

Since the clusters in GOMEA can have different sizes, the distance measure is normalized by dividing it by the total information represented by entropy (H):

$$D(X_i, X_j) = \frac{d(X_i, X_j)}{H(X_i, X_j)} = 2 - \frac{H(X_i) + H(X_j)}{H(X_i, X_j)}$$

Gene-pool Optimal Mixing

In GOMEA, Gene-pool optimal mixing (GOM) is the crossover operator. During crossover, GOMEA loops over the population. For each individual in the population, it also loops over all subsets in the FOS in random order. For each of these subsets, a random solution in the population is chosen as the donor. The crossover is performed by adding the variables of the donor that are in the indices described by the current FOS subset. This newly created offspring is then evaluated. Only if the fitness is equal or better, it will replace individual i in the offspring.

To illustrate a step of the GOM process, take a solution 01010 and a FOS subset $\{x_0, x_1, x_2\}$ (highlighted in Figure 2.1 in red). If GOMEA then applies crossover to this solution with the donor solution 11100, the values at indices 0, 1, and 2 are copied from the donor, resulting in the offspring solution 11110. If solution 11110 has a fitness score at least as good as the fitness score of 01010, it will replace that solution in the offspring population.

2.2.2. Incremental Adapted Maximum-Likelihood Gaussian Model Iterated Density-Estimation

The Incremental Adapted Maximum-Likelihood Gaussian Model Iterated Density-Estimation (iAMaLGaM-ID$\mathbb{E}$A), or iAMaLGaM for short, is a real-valued Estimation of Distribution Algorithm (EDA) in the family of EAs [23]. It is designed to tackle problems that can be represented with continuous variables. Like other EDAs, it works by modelling a probability distribution over the solution space based on a selected group of best-performing individuals in the population, which is then sampled to generate new solutions as offspring for the next generation. This concept alone however has several pitfalls that can result in poor performance. Primarily, basic EDAs are prone to premature convergence due to the exponential rate at which the variance decreases [31, 32, 33]. Furthermore, the usage of maximum likelihood estimates (MLE) can result in inefficient sampling on slope-like regions of the search space, where the direction of descent is an important factor for finding better solutions. The iAMaLGaM algorithm has applied two techniques to reduce these inefficiencies, which are called adaptive variance scaling (AVS) and anticipated mean shift (AMS) respectively.

Adaptive Variance Scaling

In AVS, a variance multiplier c^{AVS} is maintained. When sampling a new solution, the covariance matrix $\hat{\Sigma}$ is multiplied element-wise with this multiplier. If the best fitness value improves and the corresponding solution lies far from the population's mean (by more than 1 standard deviation), the current variance allows for progress. In that case, increasing the variance may lead to further improvement for the next generation. Since the selection of the next generation is based on a subset of the best-performing individuals in the

population, the selection tends to decrease the variance. To counteract this effect, the multiplier c^{AVS} is scaled by some value $\eta^{INC} > 1$ when such an improvement is found. However, if no improvement was found in a generation or the improvements are close to the mean, the exploration range is likely too large and the variance should decrease, facilitated by multiplying c^{AVS} with $\eta^{DEC} \in [0, 1]$. These factors are symmetrically related, since $\eta^{DEC} = \frac{1}{\eta^{INC}}$. Furthermore, to ensure that the variance is sufficiently enlarged to prevent premature convergence, $c^{AVS} \geq 1$ is enforced until a consecutive number of generations with no improvements occurs. This number is called the maximum no-improvement stretch and is usually set to $25 + l_c$.

Anticipated Mean Shift

On a slope-like region of the solution space, the direction of the descent is more useful for finding better solutions than the MLE. This direction can be extracted by taking the difference of the means of subsequent generations:

$$\hat{\mu}^{shift}(t) = \hat{\mu}(t) - \hat{\mu}(t-1)$$

Where $\hat{\mu}^{shift}(t)$ denotes the mean shift for generation t. This shift can then be used to move a fraction α of all solutions in that direction by adding $\hat{\mu}^{shift}(t)$, multiplied by some fraction δ, to them. This process causes the direction of the distribution to align with the direction of improvement, illustrated by the red circle in Figure 2.2.

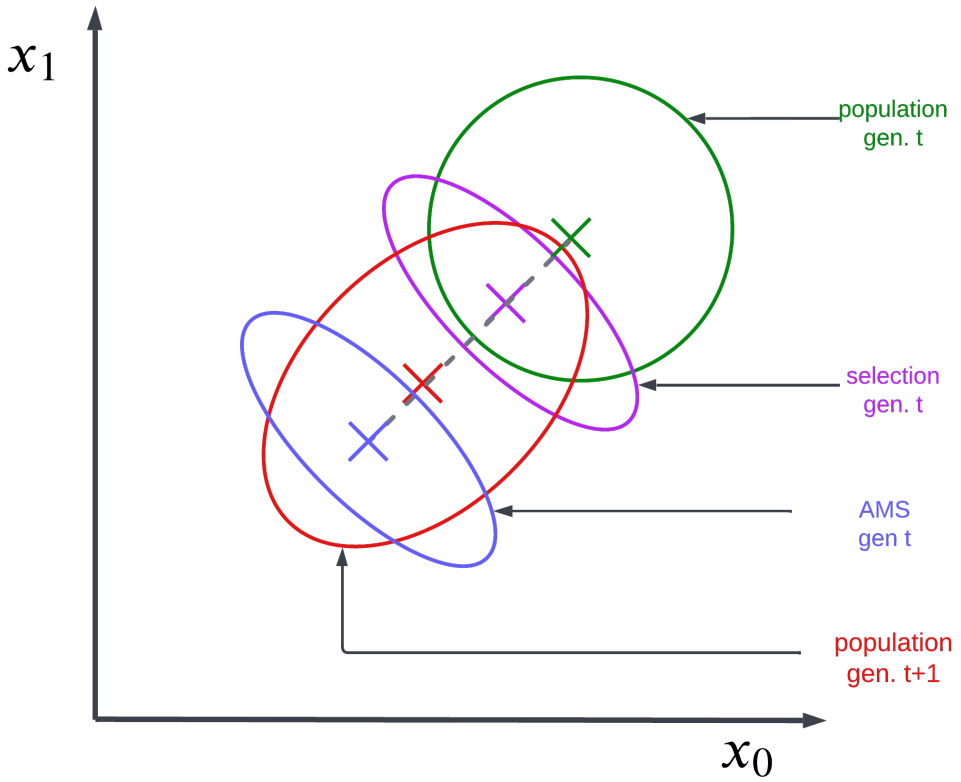


Figure 2.2: Simplified illustration of iAMaLGaM for two consecutive generations of a 2-variable minimization problem ($\min x_0 + x_1$).

All combined

The iAMaLGaM algorithm combines the EDA process with AVS and AMS, and generally works as shown in Figure 2.2:

Initially, a random population of solutions is generated (green circle), with the option to specify an initialization range for problem variables if known. Each solution in the initial population randomly selects variables within this range, and their fitness is evaluated using the evaluation function. A selection process, usually targeting the top 35 percent of solutions based on fitness, identifies a subset (purple circle) used to estimate parameters for updating the distribution. The new distribution is created by applying the AVS and AMS

principles (blue circle), which is then utilized to generate an entirely new population of solutions (red circle). The next generation of solutions is created by copying the best solution of the current generation and sampling the model to fill the rest of the population.

3

Reproduction study

The source code for MIHEA, as presented in [10], has not been made publicly available. Consequently, to conduct further analysis, the reproduction of MIHEA was initiated based on the descriptions provided in the paper. During this reproduction, certain inconsistencies were identified, prompting a more in-depth investigation. This chapter delves into these inconsistencies by reproducing some of the experiments presented in the original paper.

3.1. Original code

In MIHEA, a solution X consists of l_d discrete variables and l_c continuous variables with a total length $l = l_d + l_c$. This solution can be written as:

$$X = X_d X_c = [d_0 \dots d_{l_d-1}][c_0 \dots c_{l_c-1}]$$

where $d_i \in \mathbb{N}$ and $c_i \in \mathbb{R}$. MIHEA builds and learns models of the discrete and continuous subspaces separately. This could lead to premature convergence in either of the models, especially in problems with cross-domain dependencies of variables. MIHEA addresses this issue by balancing the ratio of learning the discrete and continuous models, based on the number of evaluations used to generate offspring with each model. For a population of size n, the generation of the continuous part of the offspring with iAMaLGaM requires n evaluations. The discrete part requires more evaluations. The linkage tree used by GOMEA creates a FOS containing $2l_d - 1$ subsets of variables. For each of these subsets, up to n evaluations are used, since each individual in the population is crossed over with a random donor based on the variables of the current FOS subset. This gives an upper bound of $n(2l_d - 1)$ evaluations used for each generation of GOMEA. To keep the number of evaluations between model updates consistent, MIHEA learns the continuous model after the whole population is updated once, i.e. when one FOS element has been used by GOMEA. The discrete model is only updated once all FOS elements have been used.

Following the textual description provided in [10], the algorithm works as follows: First, a population is randomly initialized, based on initialization ranges for each variable. Then, selection and variance are repeated until some termination criterion is met. As described above, first the discrete model is learned. It then loops over the learned FOS elements. For each element, the continuous model is learned based on the top $\tau = 0.35$ fraction of solutions in the population. Finally, the algorithm loops over the population and generates offspring for the discrete and continuous parts of the solutions, following the procedures of GOMEA and iAMaLGaM respectively. Note that the offspring generation for the discrete part is slightly different from the GOMEA procedure, since the loops over the population and FOS elements are reversed in MIHEA to achieve the balance between discrete and continuous model updates.

The pseudocode of MIHEA that was given alongside the textual description can be found in Algorithm 1, 2 and 3. Algorithm 1 outlines the general flow of MIHEA, while Algorithm 2 and 3 show how the continuous and discrete parts of solutions are generated, respectively. In the pseudocode, $\mathcal{P}_i$ represents solution i in the population, and $\mathcal{X}_i$ refers to solution i in the offspring.

There seem to be two main issues with the pseudocode presented in Algorithm 1.

Firstly, the issue with the usage of indices i and j in the main loop of the algorithm. Based on lines 11 and 12, variable i is used as the solution index in the population, and variable j as the index to the FOS

subset. However, these variables are swapped in their initialization on lines 7 and 10. The loop on line 7 also has an off-by-one error. Since the FOS contains $2l_d - 1$ elements, the last FOS index should be $2l_d - 2$ instead of $2l_d - 1$.

Secondly, the update of the population to the new offspring population on line 13 seems to be incorrectly indented. If the population update is done in this manner, then the truncation selection for iAMaLGaM on line 8 will always select the same solutions for an entire iteration of the FOS and not benefit from the potential improvements found in the offspring in between. Furthermore, the offspring solutions will be overwritten for each step of the loop on line 7. As a result, the population is only updated to the offspring solutions generated during the processing of the last FOS element, rendering the offspring solutions made for all other $2l_d - 2$ FOS elements useless.

These two issues are fixed in Algorithm 4, which will be referred to as the 'interpreted' MIHEA version.

Algorithm 1 Mixed-Integer Hybrid EA (original)

1: **for** $i \in \{0, 1, \ldots, n-1\}$ **do**
2: $\quad \mathcal{P}_i \leftarrow$ CreateRandomSolution$()$
3: $\quad$ EvaluateFitness$(\mathcal{P}_i)$
4:
5: **while** $\neg$ TerminationCriterionSatisfied **do**
6: $\quad$ LearnDiscreteModel$(\mathcal{P})$
7: $\quad$ **for** $i \in \{0, 1, \ldots, 2l_d - 1\}$ **do**
8: $\quad\quad \mathcal{S} \leftarrow$ TruncationSelection$(\mathcal{P}, \tau)$
9: $\quad\quad$ LearnContinuousModel$(\mathcal{S})$
10: $\quad\quad$ **for** $j \in \{0, 1, \ldots, n-1\}$ **do**
11: $\quad\quad\quad \mathcal{X}_i \leftarrow$ GenerateContinuousPart$(\mathcal{P}_i)$
12: $\quad\quad\quad \mathcal{X}_i \leftarrow$ GenerateDiscretePart$(j, \mathcal{X}_i, \mathcal{P})$
13: $\quad \mathcal{P} \leftarrow \mathcal{X}$
14:

Algorithm 2 GenerateContinuousPart$(\mathcal{P}_i)$ (original)

1: $\mathcal{X}_c \leftarrow$ SampleContinuousModel$()$
2: $\mathcal{X} \leftarrow \mathcal{X}_c \cup \mathcal{P}_{i_d}$
3: EvaluateFitness$(\mathcal{X})$
4: **return** $\mathcal{X}$

Algorithm 3 GenerateDiscretePart$(j, \mathcal{X}_i, \mathcal{P})$ (original)

1: $\mathcal{X}_{prev} \leftarrow \mathcal{X}_i$
2: $donor \leftarrow$ GetRandomSol$(\mathcal{P})$
3: $\mathcal{X}_d \leftarrow$ CopySubset$(j, donor, \mathcal{X}_i)$
4: $\mathcal{X} \leftarrow \mathcal{X}_{i_c} \cup \mathcal{X}_d$
5: EvaluateFitness$(\mathcal{X})$
6: **if** $fitness(\mathcal{X}) \geq fitness(\mathcal{X}_{prev})$ **then**
7: $\quad$ **return** $\mathcal{X}$
8: **else**
9: $\quad$ **return** $\mathcal{X}_{prev}$

Algorithm 4 Mixed-Integer Hybrid EA (interpreted)

```
1:  for i ∈ {0, 1, ..., n − 1} do
2:      P_i ← CreateRandomSolution()
3:      EvaluateFitness(P_i)
4:
5:  while ¬ TerminationCriterionSatisfied do
6:      LearnDiscreteModel(P)
7:      for j ∈ {0, 1, ..., 2l_d − 2} do
8:          S ← TruncationSelection(P, τ)
9:          LearnContinuousModel(S)
10:         for i ∈ {0, 1, ..., n − 1} do
11:             X_i ← GenerateContinuousPart(P_i)
12:             X_i ← GenerateDiscretePart(j, X_i, P)
13:         P ← X
14:
```

3.2. Modified code

The interpreted code still contains a potential issue. The textual description does not mention a separate offspring population, which would suggest that newly generated solutions are directly stored in the original population. This can also be seen in the pseudocode; after generating new offspring solutions on lines 11 and 12, they immediately replace the solutions in the population on line 13.

In GOMEA, the donors for crossover are taken from the population. Essentially, the original population can be seen as a model of distributions for the discrete variables. By modifying the population directly during the creation of offspring, this model is changed and distributions will be skewed in favour of better-performing solutions. This increases the selection pressure and is therefore prone to premature convergence.

Utilizing an offspring population during the generation of new solutions, that is only copied to the population when all FOS elements have been processed, can combat this cause of premature convergence. This remedy has been introduced in Algorithm 5, 6 and 7, which will be referred to as the ‘modified’ MIHEA version.

The changes in the general flow of ‘modified’ MIHEA can be seen in Algorithm 5. Here, ‘modified’ MIHEA initializes a separate offspring population on line 7, and uses that offspring population for selection on line 9 and generation of the continuous part of offspring solutions on line 12. The population is then updated after the FOS loop has been completed, on line 14.

Algorithm 6 is slightly different from the original generation of the continuous part in Algorithm 2. It uses the offspring solution’s discrete variables instead of taking them from the population solution, removing line 2 from the original version. Similarly, line 4 from the original generation of discrete variables (Algorithm 3) is removed in the ‘modified’ MIHEA version (Algorithm 7). Due to the addition of the separate offspring population, $\mathcal{X}_i$ now already contains the updated continuous variables, making line 4 redundant.

Algorithm 5 Mixed-Integer Hybrid EA (modified)

for $i \in \{0, 1, \ldots, n-1\}$ **do**
    $\mathcal{P}_i \leftarrow$ CreateRandomSolution()
    EvaluateFitness($\mathcal{P}_i$)

while $\neg$ TerminationCriterionSatisfied **do**
    LearnDiscreteModel($\mathcal{P}$)
    $\mathcal{X} \leftarrow \mathcal{P}$
    for $j \in \{0, 1, \ldots, 2l_d - 2\}$ **do**
        $\mathcal{S} \leftarrow$ TruncationSelection($\mathcal{X}, \tau$)
        LearnContinuousModel($\mathcal{S}$)
        for $i \in \{0, 1, \ldots, n-1\}$ **do**
            $\mathcal{X}_i \leftarrow$ GenerateContinuousPart($\mathcal{X}_i$)
            $\mathcal{X}_i \leftarrow$ GenerateDiscretePart($j, \mathcal{X}_i, \mathcal{P}$)
    $\mathcal{P} \leftarrow \mathcal{X}$

Algorithm 6 GenerateContinuousPart($\mathcal{X}_i$) (modified)

$\mathcal{X}_{i_c} \leftarrow$ SampleContinuousModel()

EvaluateFitness($\mathcal{X}_i$)
return $\mathcal{X}_i$

Algorithm 7 GenerateDiscretePart($j, \mathcal{X}_i, \mathcal{P}$) (modified)

$\mathcal{X}_{prev} \leftarrow \mathcal{X}_i$
$donor \leftarrow$ GetRandomSol($\mathcal{P}$)
$\mathcal{X}_{i_d} \leftarrow$ CopySubset($j, donor, \mathcal{X}_i$)

EvaluateFitness($\mathcal{X}_i$)
if $fitness(\mathcal{X}_i) \geq fitness(\mathcal{X}_{prev})$ **then**
    return $\mathcal{X}_i$
else
    return $\mathcal{X}_{prev}$

3.3. Benchmark problems

To test which version of MIHEA is most similar to the original paper version, the scalability experiments from [10] are recreated. These scalability experiments are based on five benchmark functions, that are introduced in this section. All these functions are to be minimized and have an optimal fitness value of 0.

3.3.1. F1-F4

The first four benchmark functions consist of combinations of well-known discrete and continuous problems shown in Table 3.1, mixing them to bring them to the mixed-integer domain. The sphere function simply squares each continuous variable and sums them together, meaning that all variables are fully independent of each other. The rotated ellipsoid introduces dependencies between the continuous variables. It is similar to the sphere function, since the continuous variables are also squared and summed together. However, each problem dimension is scaled with a different value. The resulting ellipsoid is rotated by 45 degrees, by multiplying with a rotation matrix **R**. Furthermore, two discrete binary functions are used. Firstly, the onemax function represents the sum of all decision variables. Similarly to the sphere function, the problem variables are fully independent. Secondly, the deceptive trap function is used. This function breaks the variables up into groups of k, with $k = 5$ in this case. It is deceptive, since the fitness value for each such group decreases as the number of variables set to 1 decreases. However, the optimal value for a group is achieved when all variables are set to 1. The benchmark functions F1-F4 then consist of the combinations of onemax or deceptive trap for the discrete part, and sphere or rotated ellipsoid for the continuous part, as shown in Table 3.2.

Function name	Domain	Definition
Sphere	Continuous	$F_{sphere}(X_c) = \sum_{i=0}^{l_c-1} c_i^2$
Rotated Ellipsoid	Continuous	$F_{R.Ellip.}(X_c) = F_{Ellip.}(\mathbf{R} \cdot X_c)$, where $F_{Ellip.}(X_c) = \sum_{i=0}^{l_c-1} 10^{6 \cdot i/(l_c-1)} \cdot c_i^2$
Onemax	Discrete	$F_{Onemax}(X_d) = \sum_{i=0}^{l_d-1} 1 - d_i$
Deceptive Trap	Discrete	$F_{DT5}(X_d) = \sum_{i=0}^{l_d/k-1} f_{Trap-k}^{sub}(\sum_{j=ki}^{ki+k-1} d_j)$, where $f_{Trap-k}^{sub} = \begin{cases} 0 & \text{: if } u = k \\ 1 - (k-1-u)/k & \text{: otherwise} \end{cases}$

Table 3.1: Overview of discrete and continuous functions that are used in mixed-integer functions

ID	Function name	Definition
F1	OnemaxSphere	$F_1(X_d, X_c) = F_{Onemax}(X_d) + F_{Sphere}(X_c)$
F2	OnemaxEllipsoid	$F_2(X_d, X_c) = F_{Onemax}(X_d) + F_{R.Ellip.}(X_c)$
F3	DT5Sphere	$F_3(X_d, X_c) = F_{DT5}(X_d) + F_{Sphere}(X_c)$
F4	DT5Ellipsoid	$F_4(X_d, X_c) = F_{DT5}(X_d) + F_{R.Ellip.}(X_c)$

Table 3.2: Overview of the mixed-integer functions used in [10] for benchmark experiments

3.3.2. F5

For the first four benchmark functions, problem variables are either independent or there are dependencies between variables within their respective domains (discrete or continuous). Benchmark function F5 introduces cross-domain dependencies, i.e. dependencies between the discrete and continuous variables. It combines the F_{DT5} function with the rotated ellipsoid in an additively decomposable manner; the function can be split into sub-functions consisting of groups of k discrete and k continuous variables. For each subset of k discrete variables, there are 2^k different possible combinations possible. Each of these combinations is linked to a rotated ellipsoid with a different centre. In particular, for this function, there is a list of 2^k different groups of k random numbers in the range $[-5, 5]$, specifying the k-dimensional centre of the ellipsoid. A subset of k discrete, binary numbers can then be converted to decimal and used as an index in the list of random numbers. The corresponding k numbers are then used as the centre of the rotated ellipsoid for the continuous variables corresponding to the discrete numbers that were used as

index.

Benchmark function F5 reuses the rotated ellipsoid function, but with a few modifications, notably that it is not centred around 0 and that it is grouped per k variables. The function F5 is defined as:

$$F_5(X_d, X_c) = \sum_{i=0}^{0.5l/k-1} (1 + 10^a f_{sub}^{trap}(\sum_{j=ki}^{ki+k-1} d_j)) \cdot (1 + f_{R.Ellip.}^{sub}([d_i \dots d_{i+k-1}], [c_i \dots c_{i+k-1}]))$$

With $k = 5$, meaning that $[d_i \dots d_{i+k-1}]$ is a group of 5 discrete variables and $[c_i \dots c_{i+k-1}]$ is a group of 5 continuous variables. The a value in the equation functions as a scaling factor, determining the contribution of the discrete trap function to the total fitness.

3.4. Experiment setup

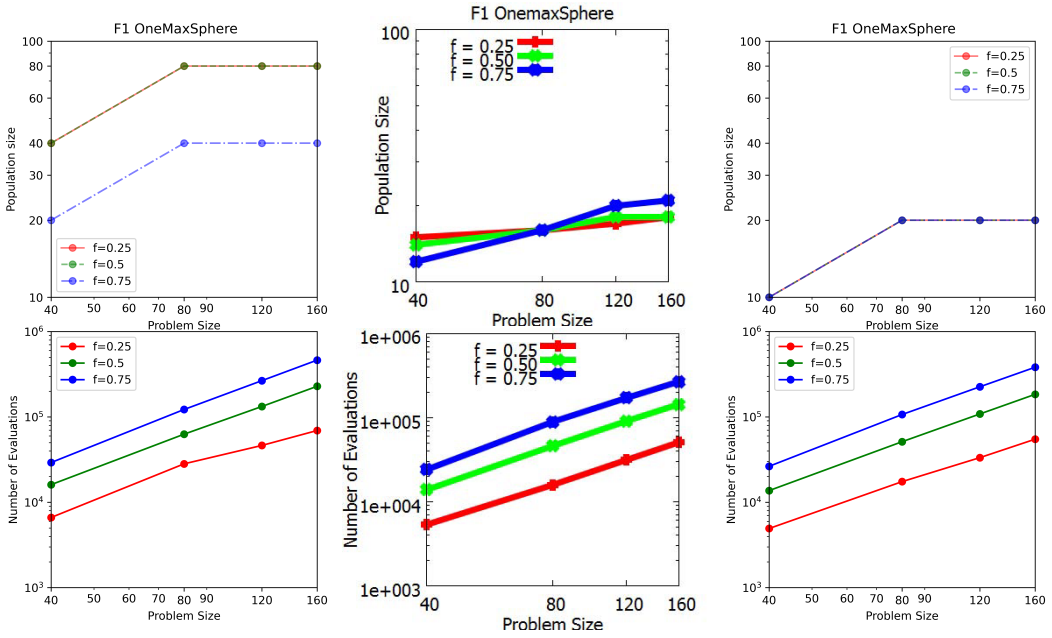
The scaling experiments with the interpreted and modified MIHEA code were set up the same way as in [10]. For F1-F4, four different problem sizes were tested: 40, 80, 120 and 160 variables total. The following proportions of $f \in \{0.25, 0.5, 0.75\}$ were tested, where f represents the fraction of continuous variables in the problem, which is calculated as $f = \frac{l_c}{l_c + l_d}$. The objective of these experiments is to determine the minimum population size at which MIHEA reliably solves the problem, defined as achieving a solution with a fitness value of 10^{-10} or lower in at least 29 out of the 30 runs. The base population sizes were estimated from the original results, and scaled by a factor of 2 if more than 1 run did not reach the desired fitness target. These estimations are not as accurate as the original results. Therefore, it is expected that the reproduction results obtained are less accurate compared to those reported in the original paper.

For F5, three problem sizes are tested: 20, 40 and 60. Since F5 requires the number of discrete and continuous variables to be equal, $f = 0.5$ is always used.

3.5. Results

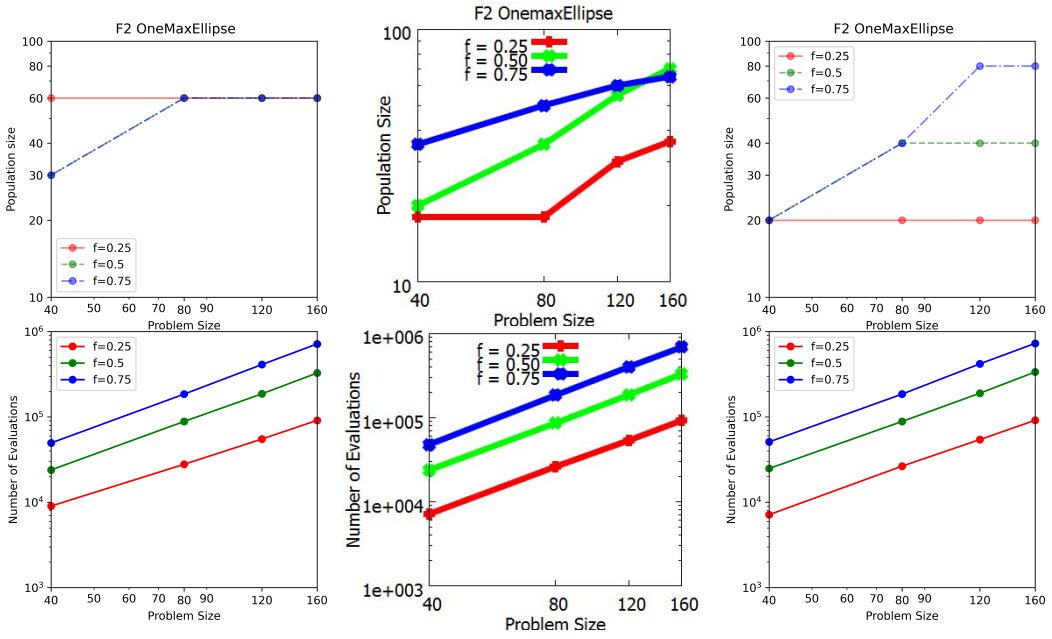
3.5.1. F1-F4

The results on the first four benchmark functions are shown in Figures 3.1, 3.2, 3.3 and 3.4 for F1 to F4 respectively. For all these functions, the interpreted code requires significantly larger population sizes to consistently solve the problems. This is consistent with the idea that directly modifying the population during the generation of new solutions for GOMEA results in premature convergence. On the other hand, the modified code matches the original plots closely, both in terms of population sizes required and average evaluations used.



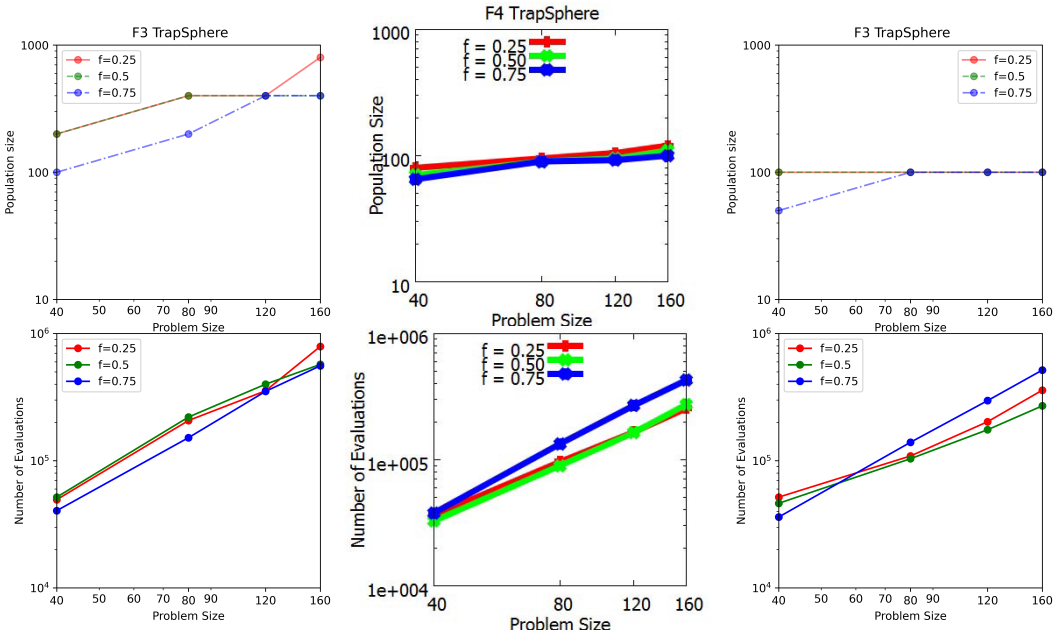
(a) Interpreted code **(b)** Original plots from [10] **(c)** Modified code

Figure 3.1: Scalability comparison between interpreted code, original plots and modified code on F1.



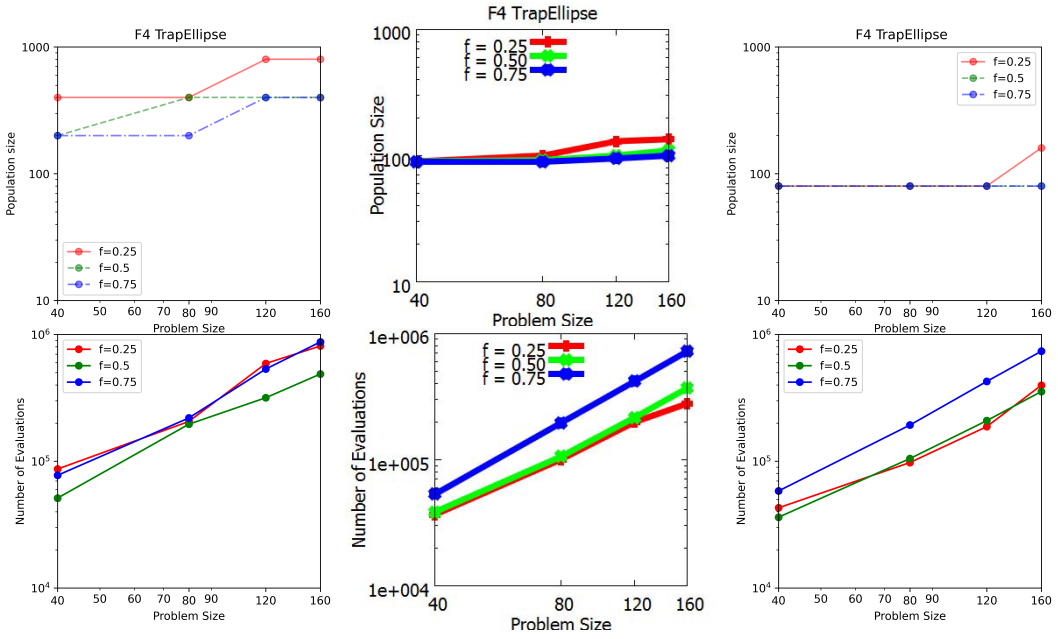
(a) Interpreted code **(b)** Original plots from [10] **(c)** Modified code

Figure 3.2: Scalability comparison between interpreted code, original plots and modified code on F2.



(a) Interpreted code **(b)** Original plots from [10] **(c)** Modified code

Figure 3.3: Scalability comparison between interpreted code, original plots and modified code on F3.



(a) Interpreted code **(b)** Original plots from [10] **(c)** Modified code

Figure 3.4: Scalability comparison between interpreted code, original plots and modified code on F4.

3.5.2. F5

For the F5 function, the value of the scaling factor a is important for the difficulty of optimizing the function. The scaling term 10^a in front of the discrete trap function balances the magnitude of its values with the rotated ellipsoid function, where the continuous variables are multiplied with the factor $10^{6 \cdot i/(l_c - 1)}$. A higher a value means that the discrete trap aspect of the fitness function contributes more to the total fitness of a solution. As a result, finding a suboptimal combination of discrete variables will have a significantly worse fitness score, making it easier to find the optimum for the discrete variables.
In [10], results on F5 are shown with a set to three values: $1.15, 2.75$ and 3.00. However, when this experiment was replicated with the interpreted and modified MIHEAs, none of the problems were solved at least 29 out of 30 times. Therefore, a slightly different experiment was performed, to find the smallest a values for which the algorithms can consistently solve the problem for the given problem sizes (20, 40, 60 variables). To find these values, the granularity of the search was gradually increased in sections between points where problems with one a-value were consistently solved, and problems with the other a-value were not. Five different population sizes were tested, starting with 150 and increasing exponentially up to 2400.

The results of this are shown in Figure 3.5 and 3.6, for the original and modified MIHEA versions respectively. For the original MIHEA, all problem sizes were also tested with $a = 4.5$, where only the smallest problem size of 20 was solved at least 29 times. Similarly, the modified MIHEA was tested with $a = 4.0$ and only had the smallest problem size get solved at least 29 times as well.

The original MIHEA shows very inconsistent results as a gets smaller. Regardless, the smallest a-values that were still consistently solved are bigger than the smallest a-values that the modified MIHEA can solve. Furthermore, the results in Figure 3.6 exhibit less variance. Especially for the bigger problem sizes 40 and 60, they show that a has a tipping point, which for the modified MIHEA lies around 4.30. As a decreases near this tipping point, the required population size to consistently solve the problem grows quickly. and past the tipping point, it makes the problem almost impossible to solve consistently with MIHEA.

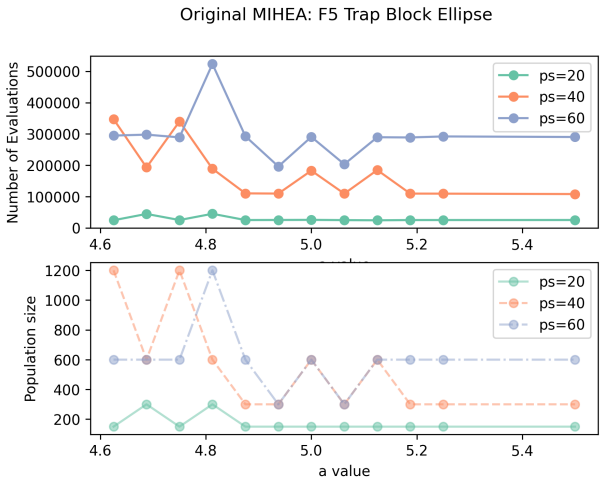


Figure 3.5: Original MIHEA: smallest population sizes (and corresponding number of evaluations) required to consistently solve F5, for different values of scaling factor a. Here, 'ps' stands for problem size, indicating the total number of discrete and continuous variables.

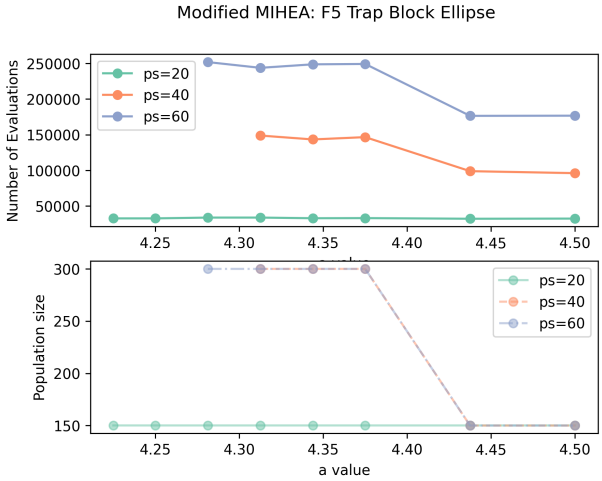


Figure 3.6: Modified MIHEA: smallest population sizes (and corresponding number of evaluations) required to consistently solve F5, for different values of scaling factor a. Here, 'ps' stands for problem size, indicating the total number of discrete and continuous variables. Note the different range of a compared to Figure 3.5.

3.6. Conclusion

Based on the results of the recreation of the scaling benchmark experiments from [10], the modified MIHEA version appears to most closely match the original results. Therefore, this version also is the closest match to the actual implementation used in that paper, even though this is not directly in line with the provided descriptions. The rest of this thesis will use only this modified version of MIHEA, and simply refer to it as MIHEA. The source code of both versions is publicly available[1].

[1] `https://github.com/RubenNair/mihea`

4

Bayesian Network Structure Learning

In the previous chapter, the most likely match to the original MIHEA code was established. In this chapter, MIHEA is applied to the structure learning of BNs with continuous nodes from data. Several approaches to represent solutions for this application are introduced and tested on small BNs with one continuous node. The best of these approaches are applied to larger BNs to investigate their scalability and compare them to the state-of-the-art.

4.1. Bayesian Networks

As outlined in the introduction, BNs [34] are models that capture probabilistic relationships between variables. BNs can be represented as directed acyclic graphs (DAGs) where each node represents a random variable, and the edges between nodes represent their conditional probabilistic dependencies. Commonly, BNs are used to model causal dependencies. In that case, an edge going from node 1 to node 2 implies that the value of node 2 conditionally depends on the value of node 1.
Many real-world datasets in various domains, like medicine [12, 13, 14] and finance [15], exhibit conditional dependencies between variables. In these cases, modelling the data using BNs can give better insight into relationships between the variables [35], facilitating a deeper understanding of the processes that created the data.

A simple example of the network structure of a BN with 5 nodes is shown in Figure 4.1. In this example, there are three nodes (A_1, A_2 and A_4) that do not depend on any other node. The A_3 node has three parents, meaning that its value is conditionally dependent on the values of A_1, A_2 and A_4. The last node, A_5, only has A_3 as a parent.

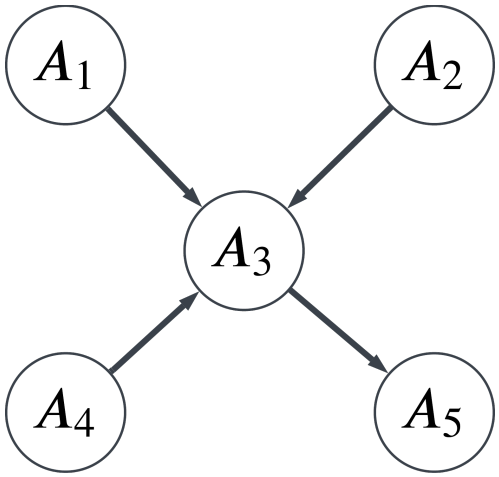


Figure 4.1: Example of the network structure of a BN with 5 nodes

A more formal definition of BNs starts with a set of variables $A = [A_1, \ldots, A_N]$, where N is the total number of nodes in the network. Then, the BN consists of a network structure indicating the dependencies

between the nodes, and a set of local conditional probability distributions $\Pr(A_i|Pa(A_i))$ for each node, where Pa indicates the parents of a node as defined in the network structure. The total joint probability distribution that follows from these local conditional probability distributions is then defined as:

$$\Pr(A_1, \ldots, A_N) = \prod_{i=1}^{N} \Pr(A_i|Pa(A_i))$$

For Figure 4.1, this joint probability distribution would be:

$$\Pr(A_1, A_2, A_3, A_4, A_5) = \Pr(A_1) \cdot \Pr(A_2) \cdot \Pr(A_3|A_1, A_2, A_4) \cdot \Pr(A_4) \cdot \Pr(A_5|A_3)$$

Learning the structure of a BN can be described as follows: Given some set of training data $D = d_0, \ldots, d_{n-1}$, where each $d \in \mathbb{R}^N$ represents an assignment of values for each of the nodes in the network, the aim is to find a network that best describes the data.
Learning the structure of BNs from data is an NP-hard problem [36]. It can become even more difficult if continuous variables are present in the data. Datasets that are good candidates to be modelled by a BN can consist of a mix of discrete and continuous variables, while most BNs are designed to work with discrete variables. In previous works, this issue is addressed by discretizing the continuous nodes into bins, or by assuming their data follows a Gaussian distribution [37]. In many real-world datasets, this Gaussian distribution assumption of the continuous nodes does not hold, making discretizations of the continuous nodes the preferred option for general BN structure learning methods. Since the optimal discretizations of the continuous nodes for a BN are dependent on the conditional dependencies between nodes in the network, the discretization process and network structure learning should be optimized at the same time.

The first time GOMEA was applied to structure learning of BNs was in [38], creating a method called BN-GOMEA. In [38], the problem of learning the structure of BNs based on discrete data is investigated. They converted the network structure into a fixed-length string representation, where each variable can take three values indicating the presence or absence and direction of an edge in the network. To optimize the structure, they used the uniform joint distribution likelihood-equivalence Bayesian Dirichlet (BDeu) score. The effectiveness of BN-GOMEA was compared to several different approaches from the literature. The other approaches consisted of different EAs, like variations of standard GAs and crossover operators. BN-GOMEA was also compared against some state-of-the-art non-EA methods, like the Ordering-Based Search [39], Sparse Candidate [40] and the Max-Min Hill-Climbing [41] algorithms. On almost all of the four datasets used in their experiments, BN-GOMEA was shown to outperform the other approaches, indicating that GOMEA is an effective technique for learning BNs from discrete data.

Very recently, BN-GOMEA was extended to apply to learning the structure of BNs from continuous data [21], resulting in the DBN-GOMEA method. For this method, the solution representation is modified by adding a variable for each continuous node in the network. This variable controls how many bins the data of the continuous node is divided into. To perform the discretization of the continuous nodes, a discretization policy is applied. Several different policies are used, like creating bins of equal width or with an equal frequency of data values. The fitness function used in [21] is based on the Bayesian Information Criterion [42], and modified to apply to continuous data. This fitness function is referred to as the Density function and will be explained in more detail in Section 4.2.
In their experiments, DBN-GOMEA is compared to a state-of-the-art method for this problem that combines the Bayesian discretization method with a structure learning algorithm, referred to as LDBN [20]. Their results show that DBN-GOMEA achieves similar or better results in retrieving randomly generated BNs, showing that GOMEA also performs well in the mixed-integer case of learning BNS.

Based on the positive results of previous works demonstrating the effectiveness of GOMEA-based methods for structure learning of BNs, MIHEA appears to be a promising approach for learning the structure of BNs from continuous data. To apply MIHEA to this problem, the Density function will be used as the fitness function, similar to [21].

4.2. Density fitness function

The Density function, as introduced in [21], is a fitness function designed to work on the structure learning of BNs when there are continuous values in the data that need to be discretized. This fitness function assumes that the data in each bin of discretized nodes is uniformly distributed. Then, optimizing the discretizations can be done by maximizing their densities. Maximizing the log of the densities is equivalent to maximizing the densities themselves and gives more numerical stability, resulting in the fitness function presented in Equation 4.1:

$$LL(D,G) = \prod_{i=1}^{n} \log\left(f_{\text{density}}(d_i, G)\right) \tag{4.1}$$

In Equation 4.1, D is a set of size n, consisting of training samples $d \in \mathbb{R}^N$, and G represents the network structure and discretizations of a solution. Before calculating the densities, the data is normalized to make the densities invariant to the data ranges. The f_{density} function calculates the density of a bin per node. For each node in the network and each unique combination of parent values for that node, the probability of each value the node can take given its parent's values is calculated. This value is then converted to a density by dividing it by the normalized width of the bin in which the data value lies. For discrete nodes, this width equals 1 divided by the number of possible values for that node.

If the LL is used as a fitness function by itself, solutions with complex discretizations can have higher fitness values. The solution that maximises the LL function on the data will likely use the maximum allowed number of bins for each continuous node to maximize the densities. To balance the quality of a model's fit to the data and its complexity, a penalty term is introduced. The penalty term for the Density function is taken from the BIC score [43] and is calculated as shown in Equation 4.2:

$$C(G) = \sum_{i=1}^{N} |Pa(X_i)| \cdot (|X_i| - 1) \cdot \log\left(\frac{n}{2}\right) \tag{4.2}$$

In Equation 4.2, $|Pa(X_i)|$ represents the number of parent discretizations, and $|X_i|$ is the number of discretizations of a node X_i.

Combining these two functions results in the Density fitness function, as displayed in Equation 4.3:

$$\text{Density}(D,G) = LL(D,G) - C(G) \tag{4.3}$$

4.3. Solution representation

An important choice for applying MIHEA to learning the structure of BNs is how the discrete and continuous variables are used to represent solutions. This section discusses the representation of the discrete variables, and introduces and tests several options for using the continuous variables in the solution representation.

The GOMEA and iAMaLGaM algorithms both work on fixed-length (l_d for GOMEA, l_c for iAMaLGaM) strings of variables. The discrete variables in a solution represent the structure of a BN, more specifically the connections present in the network. The representation used for the discrete variables is equivalent to those used in [38] and [21]. Each variable can take one of three values: *0* if there is no edge between nodes A_i and A_j, *1* if there is an edge pointing from node A_i to A_j and *2* if there is an edge pointing from node A_j to node A_i. Here, $i \in 1, \ldots, N$ and $j \in i+1, \ldots, N$, with N being the number of nodes in the BN. Each variable in this encoding represents the existence and direction of an edge in the network. As a result, the length of the network as discrete variables (l_d) can be calculated as:

$$l_d = \sum_{i=1}^{N}(N-i) = \frac{1}{2}N(N-1)$$

The structure of the BN in Figure 4.1, according to this method, is then encoded as 0100100210.
A caveat of this representation is that it can have cycles in the network, which are not allowed in a BN. As a remedy, a repair mechanism is added. Whenever an updated solution is evaluated, the repair mechanism checks if any cycles are present by traversing the graph with a depth-first search. This search is performed

by starting in each node in the network once, prioritizing nodes without parents. When the depth-first search encounters an edge that completes a cycle, that edge is removed.

The representation of the continuous variables is less straightforward. The continuous variables are used to discretize the continuous nodes into bins, by storing the boundary values. However, the optimal number of bins that the continuous nodes can be split into is not known beforehand. Therefore, an encoding is needed that can deal with varying numbers of bins in a fixed-length array of variables. Furthermore, for iAMaLGaM to create a good model of the distribution of the variables, variables in the same location across different solutions in the population should represent the same bin. If the continuous variables directly represent the boundary values, maintaining that link between variables and bins becomes complex.

To address these challenges, the continuous variables and boundaries have been split up into separate sets. Each continuous variable now represents the *width* of a bin, either with respect to the range of data values for a continuous node, or concerning the number of data samples that will fall into the bin. For each continuous node, the total bin width that can be used by a solution is 1, which spans the whole data or sample range. The continuous variables are converted to boundaries in a separate array until the sum of the converted continuous variables exceeds 1. Various methods of representing the bin widths with the continuous variables and converting them to boundaries have been created and tested. They are explained in more detail in Sections 4.3.2 to 4.3.7.

4.3.1. Benchmark problems

In order to test the effectiveness of the different continuous variable representations and to investigate how they behave, all methods are tested on two small networks. The first network consists of a singular, continuous node. The probability distribution for the values of this node has been randomly divided into 5 bins, meaning that the ground truth solution for this network has 4 boundaries. Since there is only one node, there is no network structure to learn and there are no discrete variables ($l_d = 0$). This network will be referred to as 'random node'.

The second network is taken from [20]. In [20], a simple network is introduced to demonstrate the sensitivity of two approaches to learning discrete BNs with continuous data. This network consists of 3 nodes; two discrete nodes and one continuous node. The network structure is shown in Figure 4.2. In this figure, the nodes P_1 and P_2 represent the discrete nodes and $\mathcal{X}$ represents the continuous node. The discrete nodes are binary and have equal probabilities of being 0 or 1. The continuous node can take values between 1.0 and 7.5. The probability distributions for this node $\mathcal{X}$ are shown in Figure 4.3. This figure shows that the optimal discretization of this node has the boundaries: $\{2.0, 3.0, 4.0, 4.5, 5.0, 6.0\}$, meaning that the ground truth solution for this network has 6 boundaries.

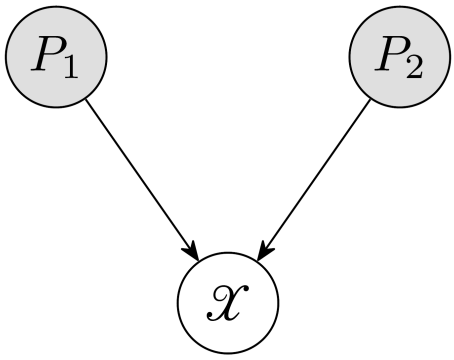


Figure 4.2: Network structure of the 'simple network' introduced in [20]

Similarly to the approach in [20], this network is only used to test the discretization process. Therefore, the network structure is set and not learned during optimization. The benefit of using the simple network for testing compared to the random node network is that it has dependencies between nodes in the graph, which means that the penalty factor of the Density function is larger and therefore punishes the complexity of the solutions more. This network will be referred to as the 'simple network'.

All the solution representation ideas are experimented with on both of these networks. For the simple network, 2000 data samples were generated. For the random node, 20.000 data samples were generated.

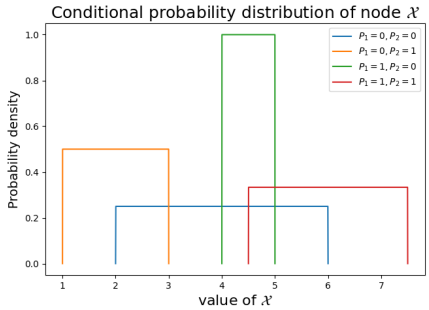


Figure 4.3: Conditional probability distribution for node $\mathcal{X}$ in the simple network

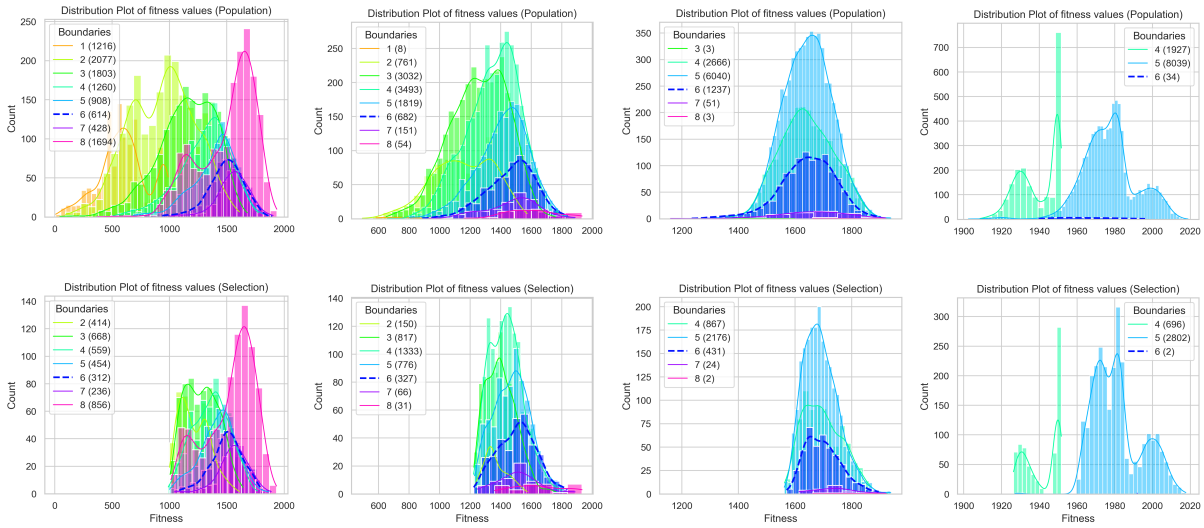
These high sample sizes were chosen to minimize the impact of overfitting on the data. All methods were tested on both of these datasets by running each algorithm 10 times with a population size of 1000. For this data, the ground truth solution of the random node network has a fitness score of 1528.51, and the ground truth solution of the simple network has a fitness score of 2023.66. The maximum allowed number of discretizations per continuous node is set to 9.
For each generation, the amount of boundaries and the fitness of each solution is tracked, for both the population and the selection. The data of a few of these generations is then highlighted for each method to show their effectiveness and how these distributions change during the optimization process.

4.3.2. simple representation (MIHEA+s)

The first solution representation method, when combined with MIHEA, will be referred to as the simple representation (or MIHEA+s). In this method, the continuous variables are kept in the domain $[0, 1]$. Each variable represents the width of a bin, relative to the data range for that node. For example, if the data for a continuous node lies in the range $[10, 20]$, a solution with continuous variables $[0.1, 0.3, 0.2, 0.4]$ would have boundaries placed at $[11, 14, 16]$. These three boundaries represent four bins: $[10, 11], (11, 14], (14, 16], (16, 20]$.

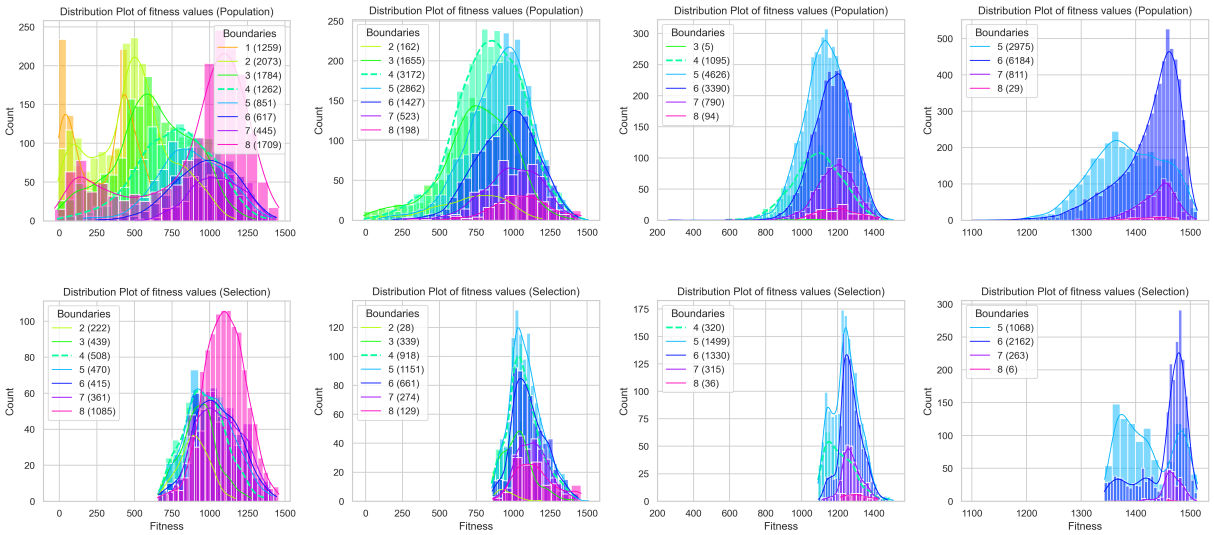
The continuous variables are randomly initialized, with a small modification. If all variables are initialized between 0 and 0.99, the probability of getting solutions with a larger amount of boundaries decreases quickly. To make solutions with higher amounts of boundaries in the initial population more likely, the upper bound of initialization for the continuous variables was linearly scaled over the population from 0.99 down to (1 / maximum number of boundaries). In other words, the upper bound of initialization of the continuous variables of a solution was calculated as $\frac{1}{B_{max}} + \frac{i}{n} \cdot (0.99 - \frac{1}{B_{max}})$, where B_{max} is the maximum number of boundaries, i is the index of this solution in the population and n is the total population size.

The results of the experiment are shown in Figure 4.4 and Figure 4.5 for the simple network and random node, respectively. These figures show histograms of the population (top) and selection (bottom) in various generations of the optimization process, separated into groups of solutions that represent the same amounts of boundaries. The distributions with an equal amount of boundaries as the ground-truth solution are highlighted with a dashed line. Both of these graphs show similar results. The initial population consists of more solutions with less than 4 boundaries, and also more solutions with 8 boundaries. The selections made from that initial population are more equally spread among the different amounts of boundaries, but relatively more solutions with 8 boundaries. Over the next generations, the graphs show that the proportion of solutions in the population with the same amount of boundaries as the ground truth (illustrated with the dashed line) slightly increases at first, but eventually becomes very small for the simple network and zero for the random node. For the simple network in Figure 4.4, it mostly finds solutions with one or two fewer boundaries than the ground truth. For the random node in Figure 4.5, the opposite effect happens and solutions with one or two more boundaries than the ground truth dominate the population. The latter effect is likely caused by the fact that the random node's penalty factor is small due to a lack of dependencies. This results in the complexity of a discretization being punished less and thus favouring solutions that have extra boundaries added.



(a) Generation 0 **(b)** Generation 1 **(c)** Generation 14 **(d)** Generation 50

Figure 4.4: Distribution of fitness in the population (top) and selection (bottom) of various generations, grouped by number of boundaries, using MIHEA+s on the simple network

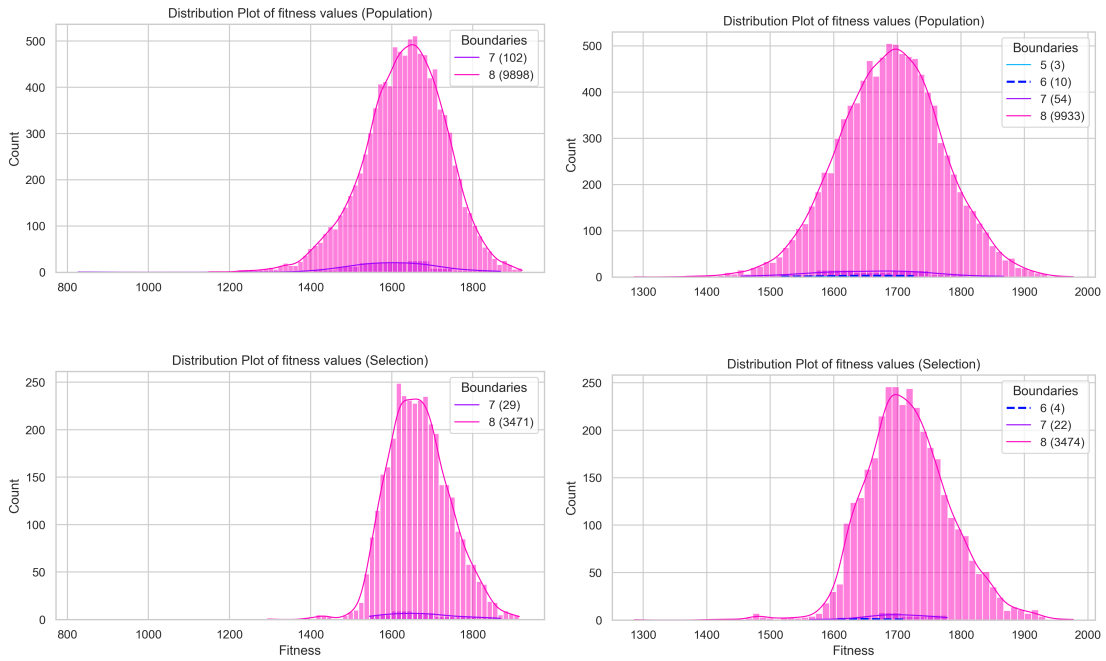


(a) Generation 0 **(b)** Generation 1 **(c)** Generation 8 **(d)** Generation 20

Figure 4.5: Distribution of fitness in the population (top) and selection (bottom) of various generations, grouped by number of boundaries, using MIHEA+s on the random node network

4.3.3. simple representation and normalization (MIHEA+s-n)

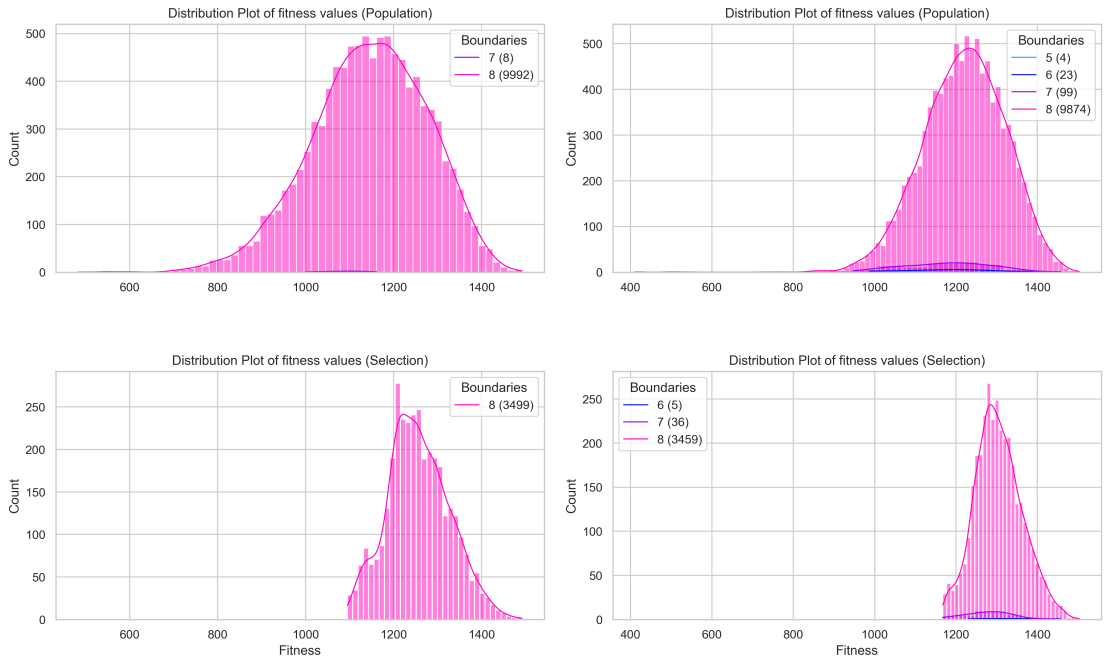
The second solution representation approach is based on the simple representation, with an extra step where variables are normalized. Whenever the continuous variables are initialized at the start or changed by iAMaLGaM during optimization, they are normalized, such that the variables for each continuous node sum to 1. As a result, most of the solutions in the population will have the maximum number of boundaries. The idea behind this method is that the normalization process can induce the 'collapse' of unnecessary bins. If a bin width becomes much smaller than the other bin widths of a continuous node, the normalization will make it even smaller until it reaches zero. Since the algorithm only adds a bin if it has a width bigger than zero, this would allow the algorithm to create solutions with fewer boundaries than the maximum.



(a) Generation 0 **(b)** Generation 2

Figure 4.6: Distribution of fitness in the population (top) and selection (bottom) of various generations, grouped by number of boundaries, using MIHEA+s-n on the simple network

The results, shown in Figure 4.6 and Figure 4.7, confirm that collapses of bins happen with the normalization process, since there are solutions in the population of generation 2 with as few as 5 boundaries. However, the collapses happen too infrequently to steer the optimization towards those solutions. The few solutions with 5 boundaries do not have high enough fitness values to be selected by iAMaLGaM.



(a) Generation 0 **(b)** Generation 2

Figure 4.7: Distribution of fitness in the population (top) and selection (bottom) of various generations, grouped by number of boundaries, using MIHEA+s-n on the random node network

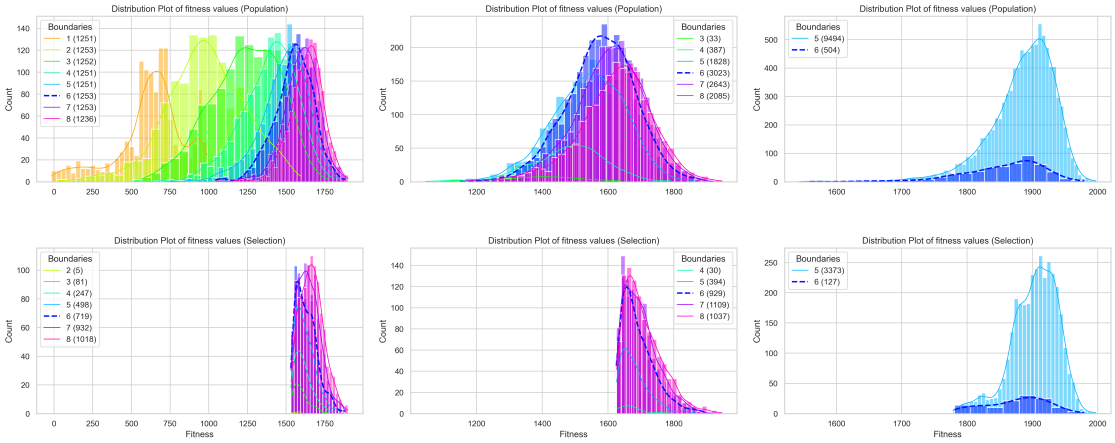
4.3.4. simple representation with forced bin spread and sample-based boundaries (MIHEA+s-bs)

The third solution representation approach is also based on the simple representation, but has two modifications.

Firstly, this approach starts with a population where the distribution of the number of boundaries per solution is equal. This reduces the bias for any number of boundaries from the start. The equal spread is achieved by predetermining for each solution in the population how many boundaries it should have. Then, all continuous variables are initialized between 0 and 0.99. If a solution should have b boundaries, the first b variables for each continuous node are normalized.

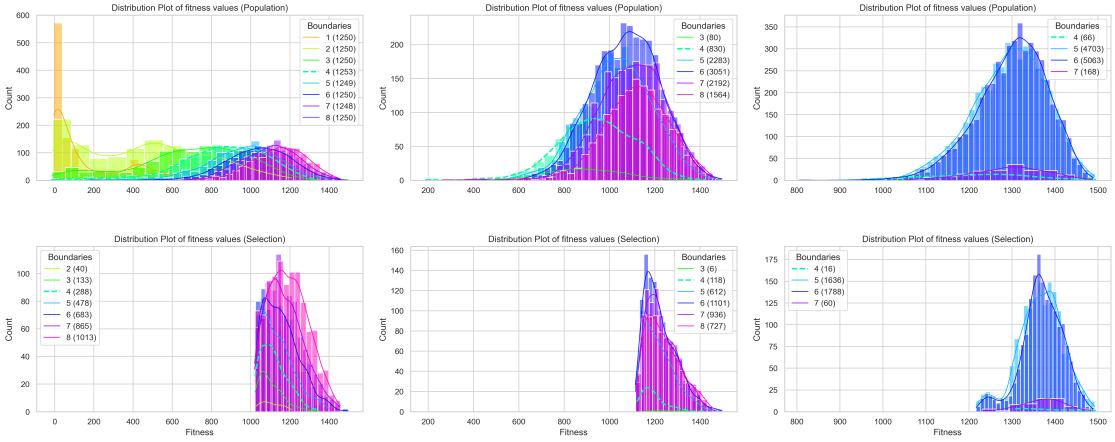
Secondly, this approach determines the location of its boundaries based on the number of samples in each bin, instead of basing it on the ranges of values in the data. This change significantly reduces the search space for discretizations of the continuous variables, at the cost of being more dependent on the number of samples available in the data and having less precision in the placement of boundaries compared to MIHEA+s. For each continuous node in the data, the algorithm keeps track of a list of indices to the samples, sorted on that node's data values. Then, the continuous variables in a solution are converted to boundaries by multiplying their value with the total number of samples in the data. The integers surrounding that value are used as indices to the sorted list of data samples for the corresponding continuous node. The boundary is then placed in the midpoint between the data values of the two selected samples. In this approach, bins that would have no data samples in them are also not created; continuous variables with values lower than 1 divided by the number of samples are ignored. Furthermore, only the first variables that sum to 1 are converted to boundaries. The last data sample does not have a neighbour to determine the midpoint between, so this last boundary is instead set equal to the value of that last data sample.

The results of the experiment using this approach are shown in Figure 4.8 and Figure 4.9. In generation 0 of both of these figures, the initial population shows an equal spread of the number of boundaries. Then, in generation 1 of both figures, the populations have already been steered away from solutions with only 1 or 2 boundaries. Eventually, similarly to the results of MIHEA+s, the distributions of both populations get



(a) Generation 0 **(b)** Generation 1 **(c)** Generation 30

Figure 4.8: Distribution of fitness in the population (top) and selection (bottom) of various generations, grouped by number of boundaries, using MIHEA+s-bs on the simple network



(a) Generation 0 **(b)** Generation 1 **(c)** Generation 11

Figure 4.9: Distribution of fitness in the population (top) and selection (bottom) of various generations, grouped by number of boundaries, using MIHEA+s-bs on the random node network

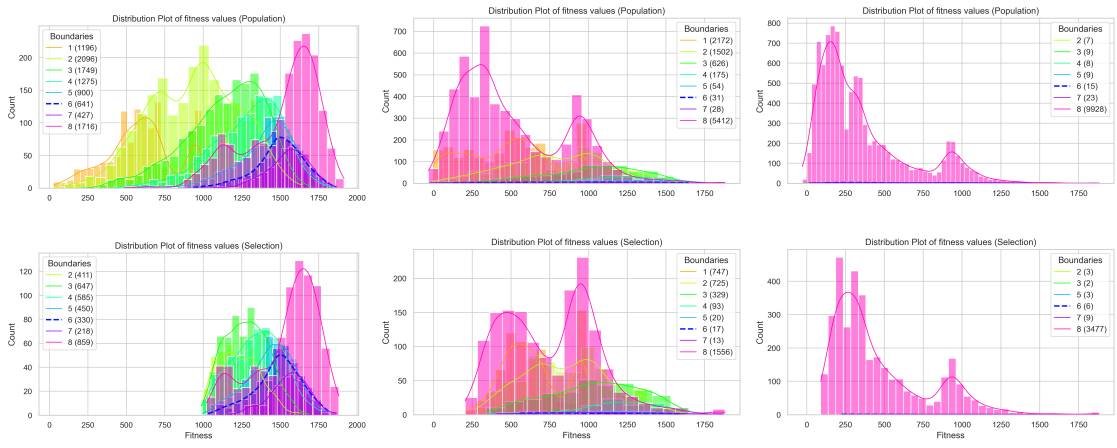
dominated by solutions with 5 boundaries. Compared to the respective ground truth solutions, this is one boundary too few for the simple network and one boundary too many for the random node.

4.3.5. inverse variables representation (MIHEA+s-t)

The fourth solution representation approach differs from the simple representation when it comes to the domain of the continuous variables. In the previous approaches, the solutions with the same number of boundaries as the ground truth get selected, and therefore sampled, less often. A possible explanation for this phenomenon lies in the domain of the variables and the method for converting the variables to boundaries. To create a solution with two boundaries for a continuous node, the average value of the first two variables corresponding to that node has to be around 0.5. To get three boundaries, the average of the first three variables needs to be around 0.33. This pattern continues for achieving higher amounts of boundaries, requiring the average of the first x variables to be around $\frac{1}{x}$ to get a solution with x boundaries. This could make it harder to end up with solutions with a high number of boundaries.

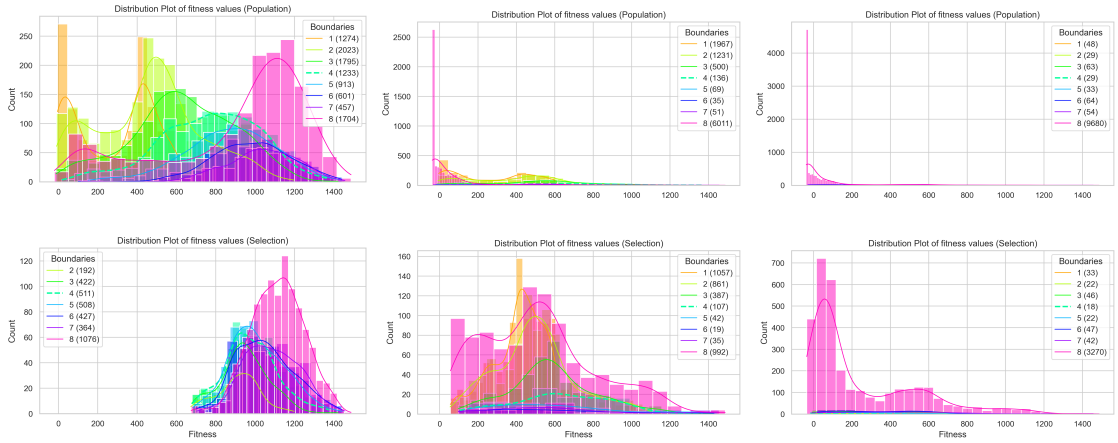
To test if this problem causes the previous approaches to fail in converging to solutions with the same number of boundaries as the ground truth, the MIHEA+s-t approach was created. In this approach, the continuous variables are still initialized as values $x \in [0, 0.99]$, and are also converted to boundaries in this domain. However, when the continuous model is updated and sampled to create new variables, the variables are represented as $\frac{1}{x}$, meaning that their domain is $[1.01, \infty)$. With this method, the average value of the variables should be around x to create a solution with x boundaries, avoiding the exponential decrease.

The results of the experiments with this method are shown in Figure 4.10 and Figure 4.11. For both networks, the approach does seem more likely to create solutions with more boundaries. But after only 2 generations, the population has already almost converged to a population consisting of solutions with the maximum number of boundaries. With this approach, the values of the continuous variables explode and go far past the maximum number of boundaries, resulting in many narrow bins being created. Many of these solutions are worse than their parents, which can be seen in the graphs by the general shift towards the left side over the first two generations. This approach is also unable to converge to solutions with the same number of boundaries as the ground truth.



(a) Generation 0 **(b)** Generation 1 **(c)** Generation 2

Figure 4.10: Distribution of fitness in the population (top) and selection (bottom) of various generations, grouped by number of boundaries, using MIHEA+s-t on the simple network



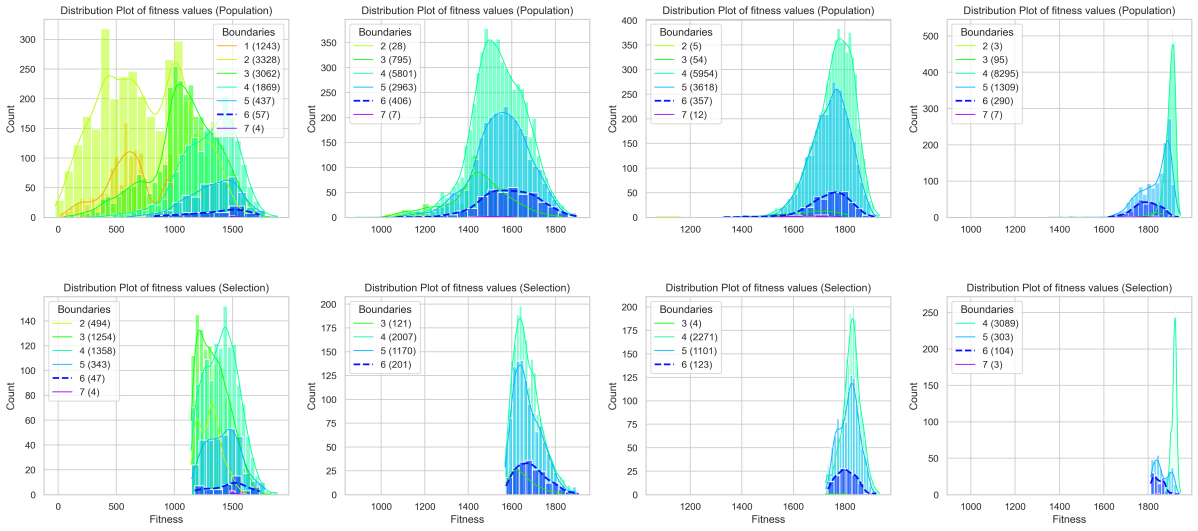
(a) Generation 0 **(b)** Generation 1 **(c)** Generation 2

Figure 4.11: Distribution of fitness in the population (top) and selection (bottom) of various generations, grouped by number of boundaries, using MIHEA+s-t on the random node network

4.3.6. extra variable representation (MIHEA+s-ev)

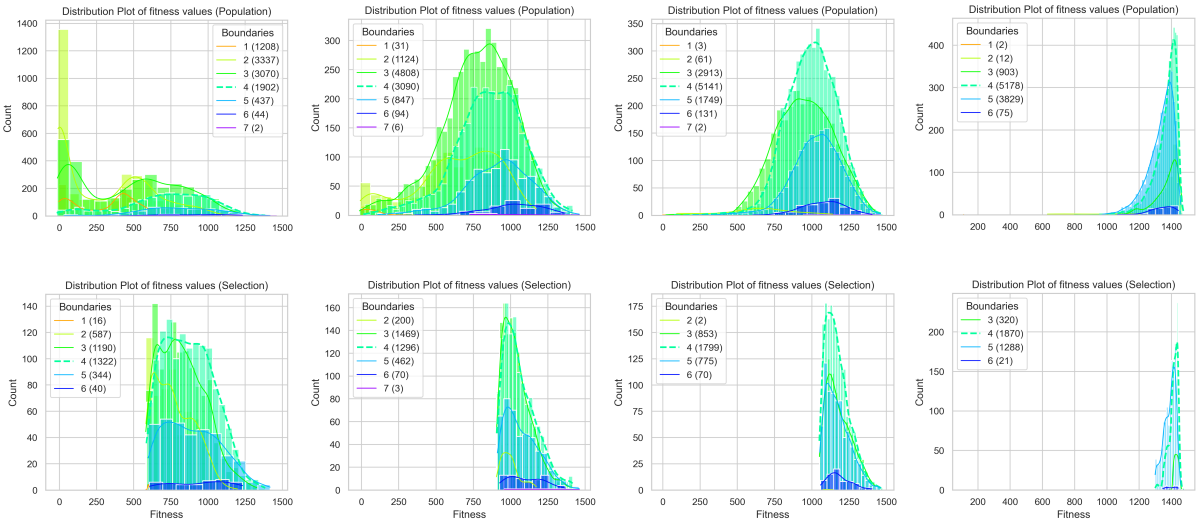
The fifth solution representation approach is also based on the simple representation, with an added continuous variable in the solutions for each continuous node in the network. In all previous approaches, the number of boundaries in a solution was only implicitly modelled, based on how many variables together sum to a total of 1. The idea of this approach is to give more control to the continuous model during optimization by explicitly modelling the number of boundaries in a solution. This extra variable has a different domain than the other variables: it can take any real value between 2 and the maximum number of bins. When the variables are converted to boundaries, these extra variables are rounded to integers. They are used as limits to how many continuous variables are converted to boundaries. This number is only used as the upper bound to how many boundaries can be made for a solution; if the sum of the continuous variables converted to boundaries surpasses 1 before this limit is reached, the remaining variables are still ignored and the solution will have less boundaries than the value of the extra variable.

The results of the experiments for this approach are shown in Figure 4.12 and Figure 4.13. The results in Figure 4.12 show that on the simple network, this approach tends to converge to solutions with fewer boundaries than the ground truth, with a large peak in the number of solutions with 4 boundaries. Nevertheless, in generation 17, there are still solutions with 6 boundaries, the same number of boundaries as the ground truth. This can be attributed to the fact that at least one of the 10 runs converged to a solution with 6 boundaries, closely approximating the ground truth solution. For the random node, Figure 4.13 shows that the approach also finds a lot of solutions with 4 boundaries, which is the same as the ground truth solution for this network.



(a) Generation 0 **(b)** Generation 5 **(c)** Generation 12 **(d)** Generation 17

Figure 4.12: Distribution of fitness in the population (top) and selection (bottom) of various generations, grouped by number of boundaries, using MIHEA+s-ev on the simple network



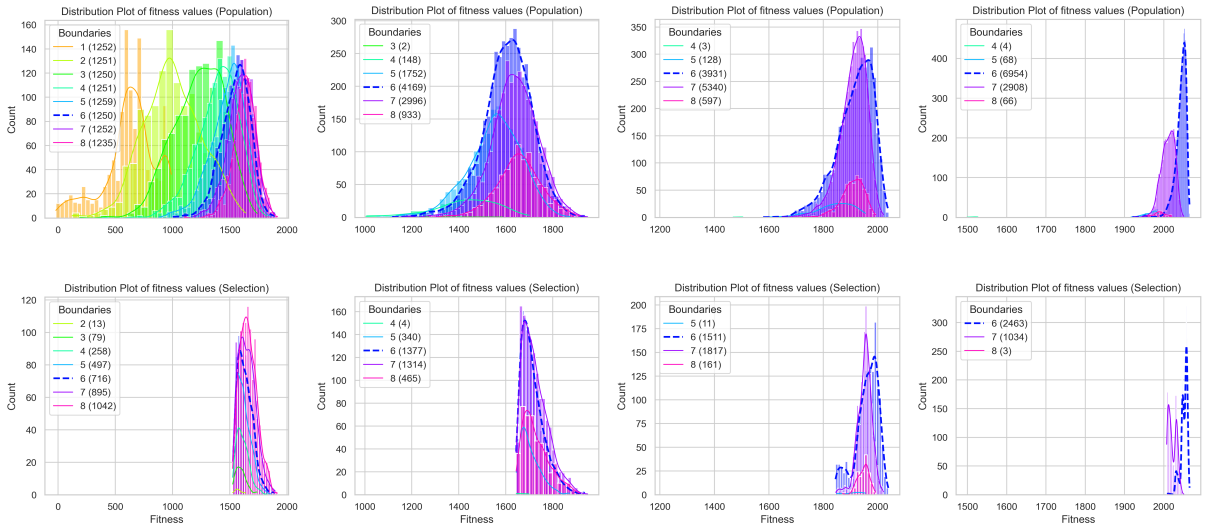
(a) Generation 0 **(b)** Generation 2 **(c)** Generation 5 **(d)** Generation 19

Figure 4.13: Distribution of fitness in the population (top) and selection (bottom) of various generations, grouped by number of boundaries, using MIHEA+s-ev on the random node network

4.3.7. extra variable representation, forced bin spread and sample-based boundaries (MIHEA+s-ev-bs)

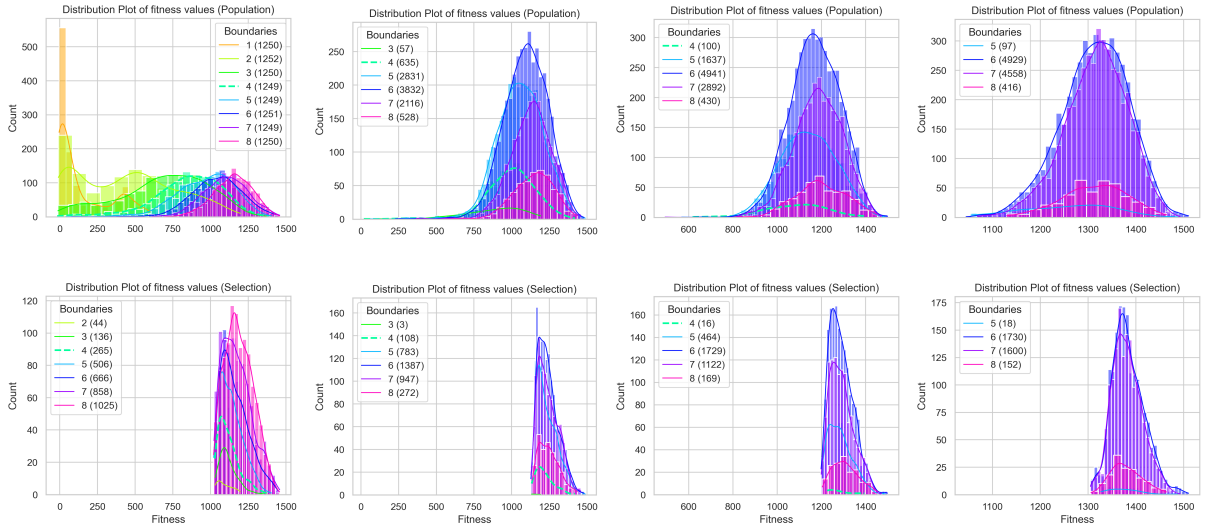
The final solution representation approach that was created is a combination of the MIHEA+s-ev and MIHEA+s-bs approaches. It takes the simple representation with forced bin spread and sample-based boundaries, and adds an extra variable per continuous node for each solution, that models the number of boundaries. As a result, this approach has a smaller search space than the simple representation, and also gives more control to the continuous model about the number of boundaries by explicitly modelling the maximum number of boundaries allowed for a solution.

The results of the experiments with this approach are shown in Figure 4.14 and Figure 4.15. On these



(a) Generation 0 **(b)** Generation 2 **(c)** Generation 15 **(d)** Generation 25

Figure 4.14: Distribution of fitness in the population (top) and selection (bottom) of various generations, grouped by number of boundaries, using MIHEA+s-ev-bs on the simple network



(a) Generation 0 **(b)** Generation 2 **(c)** Generation 5 **(d)** Generation 10

Figure 4.15: Distribution of fitness in the population (top) and selection (bottom) of various generations, grouped by number of boundaries, using MIHEA+s-ev-bs on the random node network

networks, this approach tends to converge to solutions with a larger amount of boundaries than MIHEA+s: solutions with 6 or 7 boundaries attribute the largest share of the populations. On the simple network, Figure 4.14 shows that most populations converge to solutions with 6 boundaries, which is the same as the ground truth solution. For the random node, Figure 4.15 shows that solutions with the same number of boundaries as the ground truth solution disappear from the populations within 10 generations. Once again, this is likely caused by the penalty factor being too low to punish the complexity of solutions, resulting in extra boundaries that do not exist in the ground truth solution.

4.3.8. Conclusion

During the experiment outlined at the end of Section 4.3, the elitist of each run was also tracked, with the best results presented in Table 4.1. This table shows that the MIHEA+s-ev approach was able to find the best solutions in terms of fitness value and exactly match the ground truth solution's number of boundaries for both tested networks. The MIHEA+s-ev-bs approach also finds the correct number of boundaries for the simple network, with a similar fitness value as MIHEA+s-ev.
For both networks, several methods are able to find solutions with better fitness scores than the ground truth solution. This can be attributed to overfitting to the samples, since the optimal placement of boundaries can differ from the ground truth network based on the generated samples.
Due to time limitations, only a few of the proposed approaches can be used to experiment with on larger BNs. Based on the results from Table 4.1 and the results from the previous subsections, the MIHEA+s-ev and MIHEA+s-ev-bs approaches seem the most promising and will be used in further experimentation. Furthermore, since all approaches have been based on MIHEA+s, and it also had relatively good results, MIHEA+s will also be used in further experimentation.

	simple network		random node	
	fitness	#boundaries	fitness	#boundaries
MIHEA+s	2027.15	5	1530.77	5
MIHEA+s-n	2055.95	7	1521.27	7
MIHEA+s-bs	2026.84	5	1530.76	5
MIHEA+s-t	1955.49	8	1517.57	8
MIHEA+s-ev	**2068.36**	**6**	**1532.26**	**4**
MIHEA+s-ev-bs	2066.60	**6**	1527.30	5
Ground truth	2023.66	**6**	1528.51	**4**

Table 4.1: Overview of fitness values and number of boundaries of the best elitist found during 10 runs on the simple network and the random node. The best values of each column are in bold.

4.4. Experiments

In this experiments section, multiple experiments are introduced and executed to address the research questions posed in this study. Subsequently, the results obtained from these experiments are presented and analyzed.

4.4.1. Network generation

The process of network generation in this thesis is the same as in [21]. The BNs for the experiments are created using the network generator algorithm proposed in [44]. This algorithm generates random DAGs, limited to at most 40% of the possible edges and at most 6 parents per node to reduce complexity and allow for faster evaluations.
The nodes of the networks are randomly assigned to be discrete or continuous, with the conditions that 10% of the nodes are discrete, and the minimum number of discrete nodes for a network is one. These conditions imply that for all tested networks in this thesis, there is exactly one discrete node. Both the discrete and continuous nodes are discretized; these discretizations range between 2 and 6 values. The possible values that a discrete node can take are then determined by taking all values from 1 to the discretization range. Subsequently, a discrete probability table is generated for each random variable, mapping the possible parent values to the probability of a specific discretization value. In this thesis, probability tables are created using three methods: Equal Width (EW), Equal Frequency (EF), or random (RAND) probability distributions. In EF probability distributions, all values have the same probability of being sampled. For EW and RAND distributions, these tables are randomly generated.

These generated probability tables can be sampled to create data for discrete nodes. For continuous nodes, an extra step is required to turn the discrete probability tables into continuous probability distributions. For each discrete value a continuous node can take, there is an associated data range that specifies the possible continuous values. These data ranges border each other, but do not overlap. When a value is

sampled from the discrete probability table, it is converted to a continuous value by uniformly sampling from the associated data range. For EW probability distributions, these data ranges all have the same width. For the other two distributions, the ranges are randomly set.

4.4.2. Metrics

In the experiments, several metrics are employed to evaluate various aspects of the solutions obtained. Firstly, the Density fitness score is used, as this metric serves as the optimization objective for the algorithms. The Density fitness score is dependent on the number of nodes in the network and the sample size of the data. As a result, the fitness values of ground truth solutions on a small network can be significantly smaller than the fitness values of ground truth solutions on bigger networks. Furthermore, the Density score by itself does not give much insight into how close a solution is to the ground truth. Therefore, the results will present the difference between the Density of the ground truth solution and the Density of elitist solutions created by the algorithms.

The metric used for determining the quality of the discretizations is the Kullback-Leibler (KL) divergence. The KL divergence with respect to the ground truth solution provides insight into the quality of discretizations of continuous nodes by quantifying the difference between their probability distributions. A lower KL divergence indicates that the discretized nodes closely approximate the distribution of the ground truth, suggesting higher-quality discretizations. For all experiments, the KL divergence is calculated based on a set of 50.000 test samples.

The quality of the network structures is assessed using two metrics, accuracy and sensitivity. The accuracy indicates how many edges a solution has that are also present in the ground truth solution, expressed as a fraction of the total number of possible edges in the network (see Equation 4.4). Note that for this metric, the direction of edges is disregarded. In other words, the true positives (TP) is the sum of variables in a solution with a non-zero value, that are non-zero in the corresponding ground truth variable as well. The true negatives (TN) is the sum of variables in a solution with a value of 0, where the corresponding ground truth variable has a value of 0 as well. The number of edges in the network (l_{total}) is equal to the number of discrete variables in a solution.

$$\text{Accuracy} = \frac{TP + TN}{l_{\text{total}}} \tag{4.4}$$

The sensitivity indicates how many of the existing edges in the ground truth solution were also present in a solution. This metric also disregards the direction of edges. The sensitivity is calculated by dividing the number of TPs by the number of edges that are present in the ground truth network (see Equation 4.5).

$$\text{Sensitivity} = \frac{TP}{l_{\text{edges}}} \tag{4.5}$$

Finally, to measure the scalability of the algorithms, the total execution time and number of evaluations are used as metrics.

4.4.3. Experiment setup

Three experiments have been set up to investigate the performance and scalability of MIHEA, and to compare it to the state-of-the-art approach called DBN-GOMEA [21]. the discrete DBN-GOMEA method uses the same encoding and optimization algorithm for the discrete optimization part as MIHEA, but has an added discrete variable for each continuous node in the network. This extra variable represents how many bins each continuous node is discretized into. As a result, DBN-GOMEA uses fewer variables than the MIHEA approaches. This makes the search space for DBN-GOMEA smaller than the MIHEA approaches, likely resulting in a faster optimization process. As a downside, DBN-GOMEA has less control over the placements of boundaries for discretized nodes compared to MIHEA.
In these experiments, MIHEA is benchmarked against the two best-performing versions of DBN-GOMEA: DBN-GOMEA-EW and DBN-GOMEA-EF [21], employing EW and EF discretization policies, respectively.

In the three experiments, the scalability is explored by having either a varying number of samples in the data on a network with a fixed number of nodes, or by fixing the sample size and varying the number

of nodes in the network. Furthermore, the three types of network generation methods used for data discretizations (EW, EF and RAND) are explored. These aspects have been combined into a total of three experiments. Each experiment uses a different data generation method. Furthermore, one experiment is dedicated to testing the scalability with respect to the number of data samples, while the other two experiments vary in the number of nodes in the network. For the experiment with a varying number of data samples, 15 different ground truth networks consisting of 4 nodes are generated for each sample size. The other two experiments, investigating the scalability with respect to the number of nodes in the network, are based on 30 different ground truth networks per node size, for which 4000 samples are generated.

For all experiments in this thesis, the maximum number of discretizations has been set to 9, meaning that solutions can have at most 8 boundaries. The experiments are run on a server with 4 AMD Opteron 6386 SE CPUs, totalling 64 cores, and with 256 GB of memory.

4.5. Results

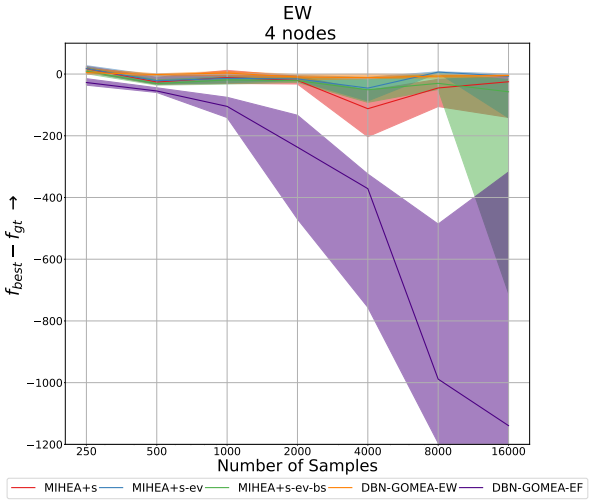
Density

The Density results of the three experiments are shown in Figure 4.16. For the first experiment (Figure 4.16a), the DBN-GOMEA-EW algorithm results in the best fitness values, outperforming all other methods especially when the sample sizes get larger. On data generated with the EW method, this is expected. For DBN-GOMEA-EW to find a solution that closely approximates the ground truth solution, it only needs to find the correct number of bins for each continuous node. Suppose those numbers correspond to the number of bins per continuous node in the ground truth solution. In that case, the discretization method will place the boundaries on the same locations as in the ground truth. The only reason why other algorithms might have better fitness values is due to overfitting the data. Especially if the sample size is lower, the optimal solution is likely different from the ground truth solution, since the combination of samples could have higher densities if the boundaries are placed differently.
The MIHEA methods perform well in this experiment, getting similar results to DBN-GOMEA-EW. As sample sizes get larger, the MIHEA methods outperform DBN-GOMEA-EF as well.

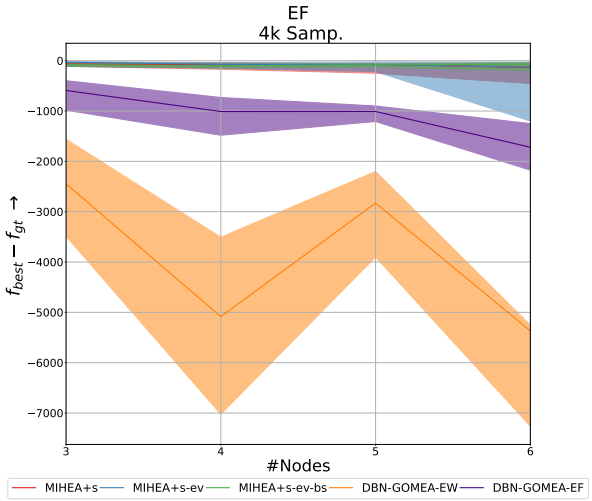
In Figure 4.16b, where the number of nodes in the network is scaled and the data is generated with EF discretizations, the MIHEA methods outperform both DBN-GOMEA methods in Density scores. All MIHEA methods find solutions with Density scores very close to the ground truth.
In this experiment, DBN-GOMEA-EF cannot generate solutions close to the ground truth. The fact that the data consists of samples of the ground truth network means it does not perfectly follow EF distributions. As a result, using an EF discretization policy can generate different combinations of boundaries for the data, compared to the ground truth. It is expected that in this case, DBN-GOMEA-EF will get closer to the ground truth network as the number of samples in the data increases. DBN-GOMEA-EW scores the lowest in this metric. Since the boundaries in the generated networks are no longer equidistant from each other, using an EW discretization policy will not set the boundaries at the best values, resulting in a relatively low Density score.

For the last experiment (Figure 4.16c), all methods are closer in performance but the MIHEA versions still outperform the DBN-GOMEA methods. Relative to the experiment using EF data, DBN-GOMEA-EW performed significantly better on the RAND data. However, compared to the other tested methods, it still scores the lowest.

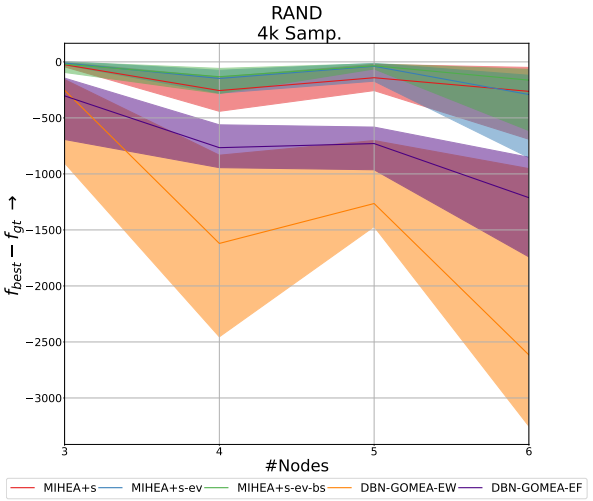
Overall, these results show that on a network generated with EF or RAND discretizations, MIHEA performs the best in optimizing the Density fitness function. If the data contains a low amount of samples, only DBN-GOMEA-EW can achieve higher scores, given that the ground truth network's boundaries are equidistant from each other.



(a) networks with EW discretizations, 4 nodes and varying sample sizes



(b) networks with EF discretizations, 4.000 samples and varying amounts of nodes



(c) networks with RAND discretizations, 4.000 samples and varying amounts of nodes

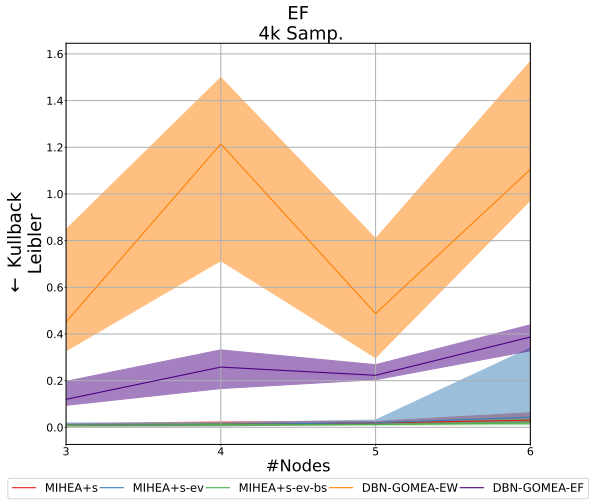
Figure 4.16: Plots of $f_{best} - f_{ground\ truth}$ of the three chosen MIHEA versions and DBN-GOMEA for the three experiments. The coloured lines indicate the median values, while the shaded areas cover the interquartile range. The arrows on the y-axis indicate the direction of improvement.

KL divergence

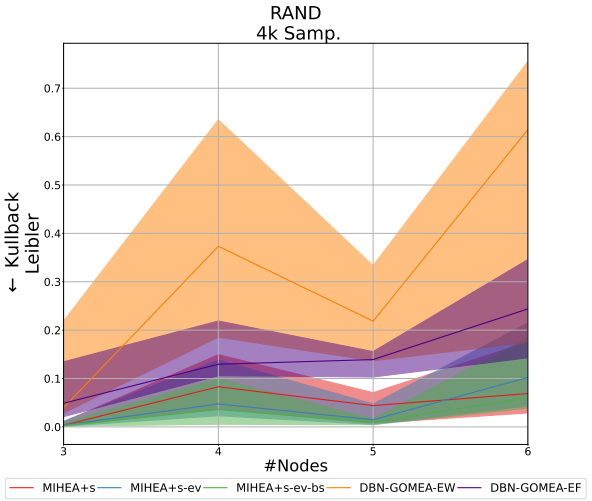
The resulting KL divergences with respect to the ground truth network from the three experiments are shown in Figure 4.17. For the first experiment, Figure 4.17a shows that for all tested sample sizes, DBN-GOMEA-EW found the best discretizations on EW data. On small sample sizes, the MIHEA versions and DBN-GOMEA-EF show similar performance, but as the sample sizes grow the performance of all MIHEA versions increases faster than DBN-GOMEA-EF, ending up with a similar KL divergence as DBN-GOMEA-EW around 8000 samples and bigger.

The results of the other two experiments are shown in Figure 4.17b and Figure 4.17c. These results are almost identical to the results of those experiments on the Density scores. In Figure 4.17b, where the number of nodes in the network is scaled and the data is generated with EF discretizations, the MIHEA methods outperform both DBN-GOMEA methods in KL scores. All MIHEA methods find solutions with KL scores very close to 0. The DBN-GOMEA-EF method cannot find solutions with a KL score close to the ground truth. Again, this is caused by the data consisting of a small finite number of samples of the ground

(a) networks with EW discretizations, 4 nodes and different sample sizes



(b) networks with EF discretizations, 4.000 samples and varying amounts of nodes



(c) networks with RAND discretizations, 4.000 samples and varying amounts of nodes

Figure 4.17: Measure of quality of the discretizations: plots of the KL divergence to the ground truth solution for the three chosen MIHEA versions and two versions of DBN-GOMEA for the three experiments. The coloured lines indicate the median values, while the shaded areas cover the interquartile range. The arrows on the y-axis indicate the direction of improvement.

truth network, resulting in distributions that are not exactly EF. As the number of samples increases, the KL score of the DBN-GOMEA-EF method is expected to get closer to 0. DBN-GOMEA-EW has the worst KL scores in this experiment. Since the boundaries in the generated networks are not equidistant from each other, meaning that using an EW discretization policy will not set the boundaries at the best values.

In the last experiment (Figure 4.17c), all methods are closer in KL scores but the MIHEA versions still outperform both DBN-GOMEA methods. DBN-GOMEA-EW has better KL scores on this RAND data than on the EF data, but is still outperformed by all other tested methods.

To investigate if the best-performing algorithms for each experiment have significantly better results than the others in the KL divergence metric, statistical testing has been employed. For these experiments, the algorithm with the lowest mean for each sample size or amount of nodes was tested pairwise with all other methods, using a Mann-Whitney U test with Bonferroni correction. Here, the correction is 60 (28 + 16

+ 16 total tests), meaning that the adjusted p-value is $\frac{0.05}{60} \approx 0.000833$. The results of these tests are shown in Table 4.2, 4.3 and 4.4. As expected, for the experiment on EW data, DBN-GOMEA-EW is significantly better than all other methods. For the other two experiments, MIHEA+s-ev-bs has the lowest mean values for all network sizes, but there is no significant difference in results between the different MIHEA versions. Only on the largest tested network with 6 nodes, MIHEA+s-ev performs statistically worse than the other two MIHEA versions, for both experiments.

From the KL divergence results, similar conclusions can be drawn as from the Density results. On a network generated with EF or RAND discretizations, MIHEA is most likely to generate good discretizations. In these experiments, the KL divergence scores of all MIHEA versions were only second to DBN-GOMEA-EW on networks generated with EW data. Besides that, there is no significant difference in KL divergence scores between the three different MIHEA versions.

Number of samples	MIHEA+s	MIHEA+s-ev	MIHEA+s-ev-bs	DBN-GOMEA-EW	DBN-GOMEA-EF
200	0.175 ± 0.050	0.155 ± 0.040	0.169 ± 0.049	**0.078** ± 0.033	0.180 ± 0.040
300	0.174 ± 0.068	0.189 ± 0.072	0.209 ± 0.070	**0.067** ± 0.043	0.220 ± 0.062
1000	0.087 ± 0.065	0.077 ± 0.043	0.083 ± 0.043	**0.008** ± 0.009	0.126 ± 0.022
2000	0.035 ± 0.012	0.040 ± 0.022	0.038 ± 0.020	**0.007** ± 0.008	0.134 ± 0.061
4000	0.043 ± 0.021	0.027 ± 0.020	0.027 ± 0.020	**0.004** ± 0.004	0.075 ± 0.030
8000	0.011 ± 0.025	0.004 ± 0.007	0.004 ± 0.007	**0.001** ± 0.001	0.081 ± 0.040
16000	0.008 ± 0.014	0.007 ± 0.012	0.019 ± 0.024	**0.0003** ± 0.0003	0.065 ± 0.060

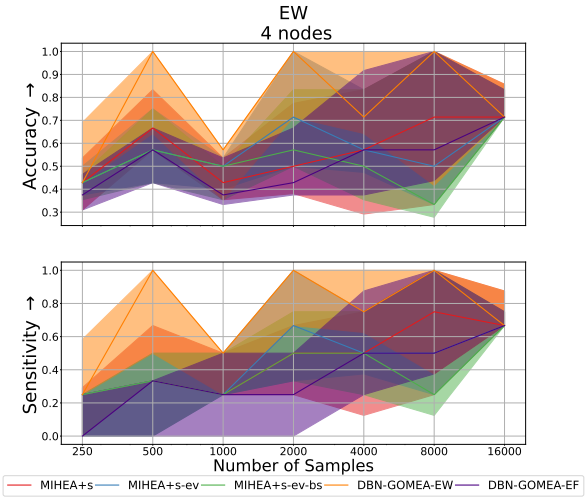
Table 4.2: Average KL divergences and standard deviations compared to the ground truth networks on the tested EW networks for different sample sizes. The bold values indicate the best results for that row, and any other value that is not statistically different from it.

Number of nodes	MIHEA+s	MIHEA+s-ev	MIHEA+s-ev-bs	DBN-GOMEA-EW	DBN-GOMEA-EF
3	**0.013** ± 0.012	**0.015** ± 0.013	**0.009** ± 0.010	0.548 ± 0.334	0.165 ± 0.108
4	**0.047** ± 0.093	**0.047** ± 0.106	**0.014** ± 0.013	1.087 ± 0.450	0.259 ± 0.109
5	**0.080** ± 0.149	**0.102** ± 0.175	**0.013** ± 0.005	0.574 ± 0.309	0.243 ± 0.059
6	**0.090** ± 0.118	0.156 ± 0.180	**0.052** ± 0.087	1.169 ± 0.378	0.387 ± 0.111

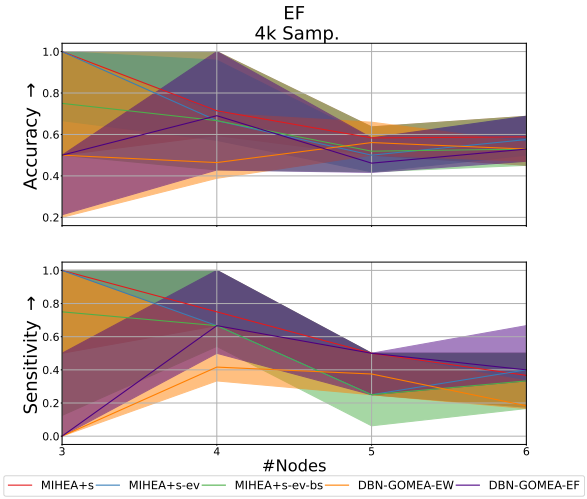
Table 4.3: Average KL divergences and standard deviations compared to the ground truth networks on the tested EF networks for different amounts of nodes. The bold values indicate the best results for that row, and any other value that is not statistically different from it.

Number of nodes	MIHEA+s	MIHEA+s-ev	MIHEA+s-ev-bs	DBN-GOMEA-EW	DBN-GOMEA-EF
3	**0.010** ± 0.016	**0.020** ± 0.041	**0.013** ± 0.030	0.100 ± 0.098	**0.086** ± 0.086
4	**0.092** ± 0.063	**0.078** ± 0.065	**0.059** ± 0.057	0.405 ± 0.210	0.160 ± 0.073
5	**0.044** ± 0.040	**0.040** ± 0.049	**0.020** ± 0.027	0.308 ± 0.256	0.134 ± 0.033
6	**0.119** ± 0.119	0.141 ± 0.122	**0.104** ± 0.098	0.578 ± 0.361	0.256 ± 0.118

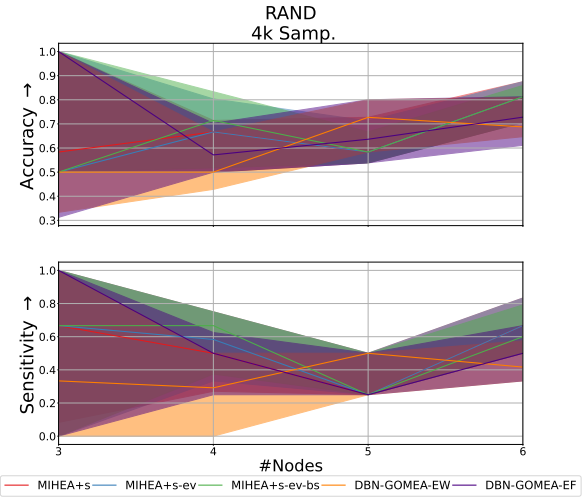
Table 4.4: Average KL divergences and standard deviations compared to the ground truth networks on the tested RAND networks for different amounts of nodes. The bold values indicate the best results for that row, and any other value that is not statistically different from it.



(a) networks with EW discretizations, 4 nodes and different sample sizes



(b) networks with EF discretizations, 4.000 samples and varying amounts of nodes



(c) networks with RAND discretizations, 4.000 samples and varying amounts of nodes

Figure 4.18: Measures of quality of network structures: plots of accuracy and sensitivity of the three chosen MIHEA versions and DBN-GOMEA for the three experiments. The coloured lines indicate the median values, while the shaded areas cover the interquartile range. The arrows on the y-axis indicate the direction of improvement.

Accuracy and Sensitivity

The results of the accuracy and sensitivity metrics for the three experiments are presented in Figure 4.18. For the first experiment (Figure 4.18a), DBN-GOMEA-EW has the best network structures, getting the same structure as the ground truth in some runs with 500 samples, and with sample sizes larger than or equal to 2000. The other algorithms give similar results, with the highest median reaching an accuracy of 0.7 and a sensitivity of 0.75 In general, there does seem to be a trend for all algorithms that the accuracy and sensitivity of some runs increase as the sample sizes increase, since the shaded areas get closer to 1 for the larger sample sizes.

On the experiment with EF data (Figure 4.18b), both MIHEA+s and MIHEA+s-ev perform well for both accuracy and sensitivity on networks with only 3 nodes, reaching a median of 1.0 on both. However, as the amount of nodes increases, all algorithms show similar performance, reaching around 0.6 accuracy and 0.4 sensitivity.

On the experiment with RAND data (Figure 4.18c), DBN-GOMEA-EF starts out performing well, with median accuracy and sensitivity of 1 on networks with 3 nodes. But here as well, the performance of all algorithms becomes similar and reaches around 0.7 accuracy and 0.5 sensitivity.

In general, DBN-GOMEA-EW still performs best on EW data as expected. Besides that, the quality of the learned network structures is similar for all algorithms, despite the good performance of the MIHEA versions on the Density metric. This can be attributed to the way MIHEA alternates the discrete and continuous algorithms, combined with the tested problem sizes. For all the tested networks, the number of continuous variables of a solution is larger than the number of discrete variables. As a result, optimizing the continuous part of the problem is more difficult and takes more time to find good solutions. By the time the continuous part of a solution has found a good approximation to the boundaries of the ground truth solution, the discrete part of the population has likely already converged to a certain network structure, or is close to convergence. That network structure has then been optimized based on earlier found discretizations, which are not yet close to the ground truth and thus are not likely to approximate the ground truth network structure well. If early convergence of the discrete population occurs, the discrete part will no longer have solutions with most of the other possible values per variable. Since GOMEA does not have a mutation operator, if these values are no longer present in the population, they cannot be found again during optimization. As a result, after finding good discretizations, the possible improvements in network structure that can still be found are limited.

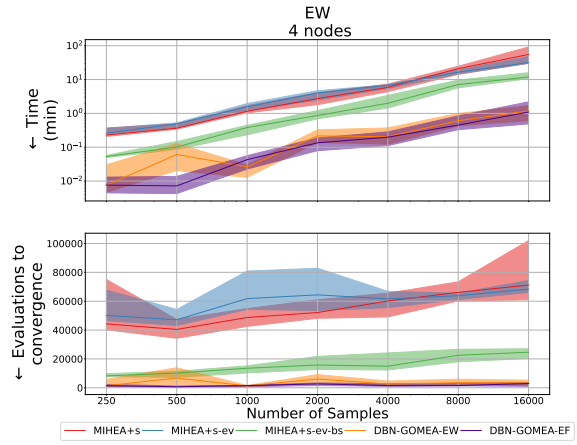
Execution time and Evaluations

The execution times and evaluations used for the three experiments are shown in Figure 4.19. The execution time graphs are log-log plots. For the first experiment (Figure 4.19a), the execution time of all algorithms scales polynomially with respect to the number of samples in the data. The algorithms separate into three groups; both versions of DBN-GOMEA are significantly faster than the other algorithms. The slowest algorithms are MIHEA+s and MIHEA+s-ev. The execution time of the last algorithm, MIHEA+s-ev-bs, is right in the middle of the other two groups. On lower sample sizes, it trends closer to the execution times of the DBN-GOMEA versions, while on larger sample sizes, it is closer to the other MIHEA versions. The number of evaluations used remains relatively constant for all tested sample sizes. This implies that the algorithms need approximately the same amount of generations to converge to a solution. The increase in sample sizes only makes the evaluation process slower by a polynomial factor, resulting in an overall polynomial scaling in time.

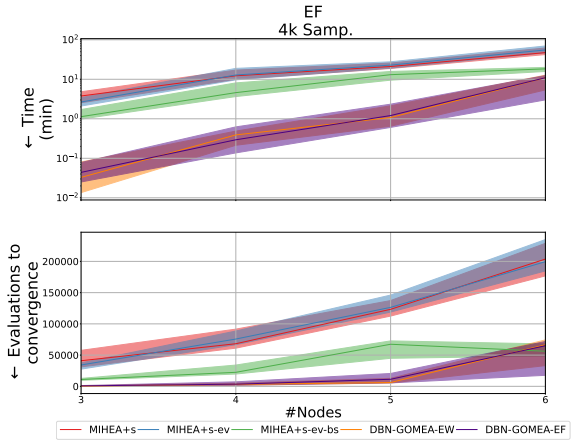
The execution times and evaluations of the experiments that vary the number of nodes in the network, Figure 4.19b and Figure 4.19c, are very similar. Like in the first experiment, the algorithms separate into three groups: the DBN-GOMEA versions together, then MIHEA+s-ev-bs, and finally the MIHEA+s and MIHEA+s-ev versions. Besides MIHEA+s-ev-bs, the algorithms also seem to scale polynomially with respect to the number of nodes in the network. However, the trends of these groups are slightly different. in the node scaling experiments, MIHEA+s-ev-bs has execution times similar to the other MIHEA versions. But as the number of nodes increases, the execution times for MIHEA+s-ev-bs do not increase as fast. On networks with 6 nodes, it is almost as fast as the DBN-GOMEA algorithms. This can also be attributed to the way the continuous variables are converted to boundaries.

The increase in nodes in the network makes it harder for MIHEA+s and MIHEA+s-ev to converge to a solution, as depicted by the increase in evaluations made to reach the elitists.

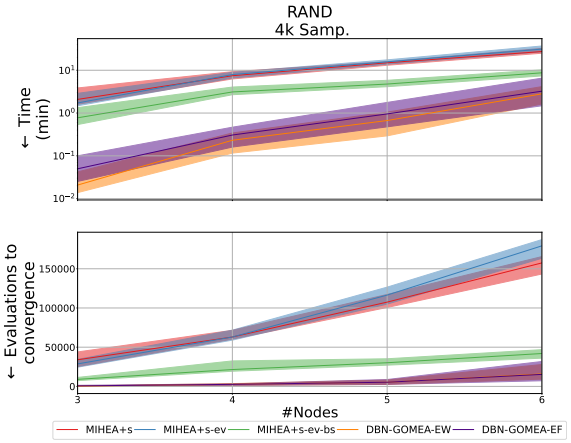
Overall, MIHEA scores significantly lower in performance than DBN-GOMEA. The time it takes to converge to a solution is much higher, and it also requires significantly more evaluations. The only exception is MIHEA+s-ev-bs, which comes close to DBN-GOMEA's execution times on networks with 6 nodes. Extrapolating from these results, it is possible that on networks with more than 6 nodes and limited data samples, MIHEA+s-ev-bs can match or outperform both DBN-GOMEA versions in terms of execution time, but this would require further testing to verify.



(a) networks with EW discretizations, 4 nodes and different sample sizes



(b) networks with EF discretizations, 4.000 samples and varying amounts of nodes



(c) networks with RAND discretizations, 4.000 samples and varying amounts of nodes

Figure 4.19: Plots of the time and number of evaluations required to find the best solution for the three chosen MIHEA versions and DBN-GOMEA for the three experiments. The colored lines indicate the median values, while the shaded areas cover the interquartile range. The arrows on the y-axis indicate the direction of improvement.

5

Discussion

The results in the previous chapter have shown that MIHEA obtains similar or better results compared to the state-of-the-art DBN-GOMEA approaches on discretizing the continuous nodes, while having similar quality in learned network structures. In this chapter, the limitations of the experiments are presented and discussed.

Limited problem sizes One notable limitation of the experiments is the restricted range of network sizes tested, which ranged from only 3 to 6 nodes. While these network sizes were sufficient for initial exploration and scalability expectations, they may not fully capture the complexities of larger networks. Extending the analysis to include larger network sizes could provide a more comprehensive understanding of MIHEA's performance across a broader spectrum of problem instances. Additionally, investigating the scalability of MIHEA on larger networks could uncover potential challenges and opportunities for improvement in handling those network structures.

Artificial networks Tying into the previous point, the experiments performed in this work are based on randomly generated networks. Before applying MIHEA to real-world problems, it is essential to validate its performance on real-world data. While randomly generated networks are suited for the purpose of this work by providing a controlled environment for algorithmic testing, they may not fully capture the intricacies and characteristics present in real-world networks.

Solution representations A critical aspect of applying MIHEA to the structure learning of Bayesian Networks (BNs) is the choice of solution representations, specifically regarding the discretization of continuous nodes. In this study, considerable efforts were devoted to designing solution representations that capture the essential characteristics of finding good discretizations. These representations were developed based on initial intuitions about the problem domain and refined through iterative testing and tweaking. However, it's important to acknowledge that the tested solution representations may not be optimal for the problem at hand. There is no guarantee that the selected representations fully capture the underlying ideas for good discretizations, or effectively encode the dependencies between variables. Therefore, further exploration and experimentation with different solution representations are warranted.

Lack of representation While this study focused on benchmarking MIHEA against the state-of-the-art algorithm DBN-GOMEA for the structure learning of BNs, it's important to acknowledge the absence of comparison with alternative approaches. Specifically, numerous approaches exist that do not rely on EAs and are capable of learning BNs from data containing continuous values that were not included in the comparative analysis. For example, the work in [20] uses a Bayesian method for discretizing and a K2 structure learning method to optimize. Some algorithms solve the problem by taking a constraint-based approach, instead of a score-based approach [18]. While the primary objective was to assess MIHEA's performance relative to DBN-GOMEA, a comprehensive evaluation would entail comparing it against a broader spectrum of methods, including those not based on EAs.

6

Conclusion and Future Work

6.1. Conclusion

In this thesis, the Mixed-Integer Hybrid Evolutionary Algorithm (MIHEA) was investigated. The description of the code for MIHEA in [10] was shown to be inconsistent with the experiment results, prompting a reproduction study resulting in a version of the code that more accurately fits the results. MIHEA was then applied to the structure learning of BNs from data with continuous values. This problem was divided into a discrete and continuous part, expressed by discrete and continuous variables respectively. The discrete variables represent the network structure, encoding the presence and direction of edges between nodes in the network. The continuous variables are used to encode the discretizations of continuous nodes. Several different encodings of the discretizations of continuous variables were created. The three best-performing representations were used for the experiments designed to answer the research questions. These experiments have shown that MIHEA matches or outperforms the state-of-the-art DBN-GOMEA approach on the discretizations of continuous nodes, while achieving similar results in quality for the learned network structures. Furthermore, the scalability of MIHEA is dependent on the solution representation. Out of the three tested solution representations, MIHEA+s-ev-bs shows similar performance to the other two approaches in terms of discretizations and network structure quality, while having significantly lower execution time and required evaluations. This representation uses an extra continuous variable per continuous node in the network that represents the number of bins a solution is allowed to use for that node. The boundaries between bins are placed based on how many data samples fit into the bin widths specified by the other continuous variables. The scalability of MIHEA+s-ev-bs is more dependent on sample size, and on constant sample size, it can achieve similar results to DBN-GOMEA.

In summary, the findings of this thesis indicate that mixed-integer EAs are a promising method when it comes to the problem of learning the structure of BNs from data containing continuous values.

6.2. Future work

Building upon the findings in this thesis, several promising directions for further experiments or improvements have arisen.

Separated covariance matrices The current version of MIHEA is limited to fixed-length encodings for the continuous variables. For the structure learning of BNs, this results in redundant variables and therefore wasted computations. A possible way to mitigate this issue is to separate the covariance matrices into blocks, creating a different covariance matrix for each continuous solution length. Not only does this allow for variable-length continuous encodings, but the covariance matrices are also specifically applied to a certain discretization size. A drawback of this approach is that each covariance matrix is based on a smaller set of solutions, which could increase the required population sizes to find good solutions. Future work could investigate whether this trade-off gives positive results in terms of performance.

Local search The DBN-GOMEA methods used in this thesis apply local search during optimization. For each created solution, the discrete variables are changed one by one to all other values (0, 1 or 2) to see how that modified network structure affects the fitness score. The change to the discrete variables is kept if the fitness value improves. Local search could counteract the effects of the early convergence

of the discrete part of the population, resulting in improved network structures while retaining the quality of discretizations. Adding local search to MIHEA has not been tested in this work since evaluations are expensive to calculate, and MIHEA already requires significantly more evaluations than DBN-GOMEA to find good solutions. However, the increase in execution time and evaluations can be worth it if the network structure of the best-found solutions improves. This trade-off can also be investigated in future works.

Balance of GOMEA and iAMaLGaM In MIHEA, the rate at which the discrete and continuous models are learned is balanced based on the number of evaluations each algorithm normally uses between consecutive model updates. Since GOMEA uses at most $2l_d - 1$ times more evaluations, the continuous model is updated $2l_d - 1$ times more often. As mentioned in [10], this balancing is crucial to avoid one of the two algorithms dominating the search, leaving some areas over-explored and others under-explored and potentially leading to premature convergence. Although the chosen balancing approach in MIHEA is reasonable in general, other options may yield better results on specific problems.

GAMBIT Moving beyond MIHEA, there exists a newer mixed-integer EA-based optimization algorithm. About a year after MIHEA was published, the same authors developed the Genetic Algorithm for Model-Based mixed-Integer opTimization (GAMBIT) [45, 46]. This algorithm extends MIHEA by adding more types of subsets to the FOS and by adding a population clustering mechanism. In GAMBIT, the FOS contains subsets of strictly discrete, strictly continuous or a mix of discrete and continuous variables. These subset types are processed, each using a different approach designed to deal with the different types of variable dependencies. The addition of mixed subsets and the clustering of the population make GAMBIT perform better on problems with cross-domain variable dependencies. This can prove to be useful for learning BNs from data, which has dependencies between the discretization of continuous nodes and the network structure. In future work, the performance of GAMBIT on structure learning of BNs with continuous data can be tested and compared to MIHEA.

References

[1] Matteo Fischetti et al. “Deep neural networks and mixed integer linear optimization”. In: *Constraints*, vol. 23, no. 3 (2018), pp. 296–309.

[2] Pierre Bonami et al. “An exact solution approach for portfolio optimization problems under stochastic and integer constraints”. In: *Operations research*, vol. 57, no. 3 (2009), pp. 650–670.

[3] Eray Demirel et al. “A mixed integer linear programming model to optimize reverse logistics activities of end-of-life vehicles in Turkey”. In: *Journal of Cleaner Production*, vol. 112 (2016), pp. 2101–2113.

[4] Krzysztof L Sadowski et al. “Exploring trade-offs between target coverage, healthy tissue sparing, and the placement of catheters in HDR brachytherapy for prostate cancer using a novel multi-objective model-based mixed-integer evolutionary algorithm”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2017, pp. 1224–1231.

[5] John H Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[6] Ingo Rechenberg. “Cybernetic solution path of an experimental problem”. In: *Roy. Aircr. Establ., Libr. transl.*, vol. 1122 (1965).

[7] E David. “Golderg Genetic Algorithms in Search, Optimization and Machine Learning”. In: *America: Addison Wesley* (1989), pp. 1–217.

[8] Stephanie Forrest. “Genetic algorithms: principles of natural selection applied to computation”. In: *Science*, vol. 261, no. 5123 (1993), pp. 872–878.

[9] Ran Cheng et al. “Model-based evolutionary algorithms: a short survey”. In: *Complex & Intelligent Systems*, vol. 4, no. 4 (2018), pp. 283–292.

[10] Krzysztof L Sadowski et al. “Combining model-based EAs for mixed-integer problems”. In: *International Conference on Parallel Problem Solving from Nature*. Springer. 2014, pp. 342–351.

[11] Rui Li et al. “Mixed integer evolution strategies for parameter optimization”. In: *Evolutionary computation*, vol. 21, no. 1 (2013), pp. 29–64.

[12] Karen Sachs et al. “Causal protein-signaling networks derived from multiparameter single-cell data”. In: *Science*, vol. 308, no. 5721 (2005), pp. 523–529.

[13] Seiya Imoto et al. “Combining microarrays and biological knowledge for estimating gene networks via Bayesian networks”. In: *Journal of bioinformatics and computational biology*, vol. 2, no. 01 (2004), pp. 77–98.

[14] M Berkan Sesen et al. “Bayesian networks for clinical decision support in lung cancer care”. In: *PloS one*, vol. 8, no. 12 (2013).

[15] Riza Demirer et al. “Bayesian networks: a decision tool to improve portfolio risk analysis”. In: *Journal of applied finance*, vol. 16, no. 2 (2006), p. 106.

[16] Claudia Vitolo et al. “Modeling air pollution, climate, and health data using Bayesian Networks: A case study of the English regions”. In: *Earth and Space Science*, vol. 5, no. 4 (2018), pp. 76–88.

[17] Tomas Beuzen et al. “A comparison of methods for discretizing continuous variables in Bayesian Networks”. In: *Environmental modelling & software*, vol. 108 (2018), pp. 61–66.

[18] Neville Kenneth Kitson et al. “A survey of Bayesian Network structure learning”. In: *Artificial Intelligence Review* (2023), pp. 1–94.

[19] Nir Friedman et al. “Discretizing Continuous Attributes While Learning Bayesian Networks”. In: *Proceedings of ICML-96, 13th international conference on machine learning*. Springer, 1996, pp. 157–165.

[20] Yi-Chun Chen et al. “Learning discrete Bayesian networks from continuous data”. In: *Journal of Artificial Intelligence Research*, vol. 59 (2017), pp. 103–132.

[21] Damy MF Ha et al. “Learning Discretized Bayesian Networks with GOMEA”. In: *arXiv preprint arXiv:2402.12175* (2024).

[22] Dirk Thierens et al. “Optimal mixing evolutionary algorithms”. In: *Proceedings of the 13th annual conference on Genetic and evolutionary computation*. 2011, pp. 617–624.

[23] Peter AN Bosman et al. “Enhancing the performance of maximum–likelihood Gaussian EDAs using anticipated mean shift”. In: *International Conference on Parallel Problem Solving from Nature*. Springer. 2008, pp. 133–143.

[24] Riccardo Manzini et al. “Optimization models for the dynamic facility location and allocation problem”. In: *International Journal of Production Research*, vol. 46, no. 8 (2008), pp. 2061–2086.

[25] Opher Baron et al. “Facility location: A robust optimization approach”. In: *Production and Operations Management*, vol. 20, no. 5 (2011), pp. 772–785.

[26] Zhi-Hui Zhan et al. “A survey on evolutionary computation for complex continuous optimization”. In: *Artificial Intelligence Review* (2022), pp. 1–52.

[27] Peter AN Bosman et al. “More concise and robust linkage learning by filtering and combining linkage hierarchies”. In: *Proceedings of the 15th annual conference on Genetic and evolutionary computation*. 2013, pp. 359–366.

[28] Ngoc Hoang Luong et al. “Multi-objective gene-pool optimal mixing evolutionary algorithms”. In: *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*. 2014, pp. 357–364.

[29] Anton Bouter et al. “Exploiting linkage information in real-valued optimization with the real-valued gene-pool optimal mixing evolutionary algorithm”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2017, pp. 705–712.

[30] Marco Virgolin et al. “Scalable genetic programming by gene-pool optimal mixing and input-space entropy-based building-block learning”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2017, pp. 1041–1048.

[31] Cristina Gonzalez et al. “Mathematical modelling of UMDAc algorithm with tournament selection. Behaviour on linear and quadratic functions”. In: *International Journal of Approximate Reasoning*, vol. 31, no. 3 (2002), pp. 313–340.

[32] Jörn Grahl et al. “The correlation-triggered adaptive variance scaling IDEA”. In: *Proceedings of the 8th annual conference on Genetic and evolutionary computation*. 2006, pp. 397–404.

[33] Jörn Grahl et al. *Behaviour of UMDAc with truncation selection on monotone functions*. Tech. rep. University of Mannheim, Department of Logistics, 2005.

[34] Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan kaufmann, 1988.

[35] Paul Arora et al. “Bayesian networks for risk prediction using real-world data: a tool for precision medicine”. In: *Value in Health*, vol. 22, no. 4 (2019), pp. 439–445.

[36] Lobna Bouchaala et al. “Improving algorithms for structure learning in Bayesian Networks using a new implicit score”. In: *Expert Systems with Applications*, vol. 37, no. 7 (2010), pp. 5470–5475.

[37] Joe Suzuki. “Learning Bayesian Network Structures When Discrete and Continuous Variables Are Present”. In: *Probabilistic Graphical Models: 7th European Workshop, PGM 2014, Utrecht, The Netherlands, September 17-19, 2014. Proceedings 7*. Springer. 2014, pp. 471–486.

[38] Kalia Orphanou et al. “Learning Bayesian network structures with GOMEA”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2018, pp. 1007–1014.

[39] Marc Teyssier et al. “Ordering-based search: a simple and effective algorithm for learning Bayesian networks”. In: *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*. 2005, pp. 584–590.

[40] Nir Friedman et al. “Learning bayesian network structure from massive datasets: the «sparse candidate» algorithm”. In: *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*. 1999, pp. 206–215.

[41] Ioannis Tsamardinos et al. “The max-min hill-climbing Bayesian network structure learning algorithm”. In: *Machine learning*, vol. 65 (2006), pp. 31–78.

[42] Hirotugu Akaike. “A new look at the statistical model identification”. In: *IEEE transactions on automatic control*, vol. 19, no. 6 (1974), pp. 716–723.

[43] Gideon Schwarz. “Estimating the dimension of a model”. In: *The annals of statistics* (1978), pp. 461–464.

[44] Jaime Shinsuke Ide et al. “Generating random Bayesian networks with constraints on induced width”. In: *ECAI*. Vol. 16. 2004, p. 323.

[45] Krzysztof L Sadowski et al. “A clustering-based model-building EA for optimization problems with binary and real-valued variables”. In: *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. 2015, pp. 911–918.

[46] Krzysztof L Sadowski et al. “GAMBIT: A parameterless model-based evolutionary algorithm for mixed-integer problems”. In: *Evolutionary computation*, vol. 26, no. 1 (2018), pp. 117–143.