Journal of Combinatorial Optimization (2022) 44:1077–1098
https://doi.org/10.1007/s10878-022-00879-6

Estimation of distribution algorithms using Gaussian Bayesian networks to solve industrial optimization problems constrained by environment variables

Vicente P. Soloviev[1] · Pedro Larrañaga[1] · Concha Bielza[1]

Accepted: 20 June 2022 / Published online: 1 July 2022

Abstract
Many real-world optimization problems involve two different subsets of variables: decision variables, and those variables which are not present in the cost function but constrain the solutions, and thus, must be considered during optimization. Thus, dependencies between and within both subsets of variables must be considered. In this paper, an estimation of distribution algorithm (EDA) is implemented to solve this type of complex optimization problems. A Gaussian Bayesian network is used to build an abstraction model of the search space in each iteration to identify patterns among the variables. As the algorithm is initialized from data, we introduce a new hyper-parameter to control the influence of the initial data in the decisions made during the EDA execution. The results show that our algorithm improves the cost function more than the expert knowledge does.

Keywords Estimation of distribution algorithms · Gaussian Bayesian network · Evolutionary algorithms · Optimization · Industry · Environment variables

1 Introduction

Optimizing a cost function consists of finding the set of variables that results in the minimum output from the function. When only a few variables are involved in the problem, off-the-shelf solvers, such as Gurobi (Gurobi Optimization, LLC 2021) or CPlex (Bliek1ú et al. 2014) can be applied. However, there are factors that increase the

✉ Vicente P. Soloviev
vicente.perez.soloviev@fi.upm.es

Pedro Larrañaga
pedro.larranaga@fi.upm.es

Concha Bielza
mcbielza@fi.upm.es

1 Computational Intelligence Group, Universidad Politécnica de Madrid, Madrid, Spain

complexity of the optimization tasks, and hence, the complexity of the approaches that handle the problem (Chong and Zak 2004). Some factors are the drastic increase of the number of variables involved in the optimization problem (large scale optimization) such as in portfolio optimization problems (Markowitz 1952), the imposition of several constraints in the problem (constrained optimization) such as in the n-queens problem (Rivin et al. 1994), the optimization of more than one objective for the same problem (multi-objective optimization) such as the multi-objective knapsack problem (Bazgan et al. 2009), the setting of evidences in the model so that the cost function depends on some fixed values for some of the variables (evidence optimization) which is common in the industrial areas such as textile optimization depending on characteristics of the textile (Sahani and Linden 2002), or the optimization of a cost function which varies along runtime (dynamic optimization) such as feature subset selection in data streams (Huang et al. 2015), among others.

Another factor that increases the complexity of the optimization problem is the type of variables (continuous, discrete) the problem deals with: continuous optimization or discrete optimization. Usually, solving a discrete optimization problem implies a higher complexity approach due to the combinatorial explosion that occurs in all but the smallest problems. Some approaches imply transforming the discrete variables into continuous ones to avoid this combinatorial step. Continuous optimization tends to be easier to solve as the smoothness of the objective and the constraint functions at a specific solution can provide information about the neighborhood of that solution. Continuous optimization approaches often assume an underlying probability distribution in the involved variables to treat them more efficiently and reduce the complexity. Dealing with discrete and continuous variables in the same problem presents an issue when treating the data in a multivariate way. Therefore, usually on these occasions the discrete variables are transformed to continuous, or vice versa, although there are approaches involving both types of variables.

In such cases, other approaches such as heuristics (Talbi 2009) should be considered to solve the problem. Heuristic approaches can be executed quickly and offer a feasible and good solution for the optimization problem.

When optimizing a cost function, different solutions with similar costs can be provided by the algorithm. Thus, a non-deterministic algorithm must be implemented to analyse the different ways of reaching an optimal combination. The need for stochasticity can be met by population-based metaheuristics, such as swarm intelligence (Chakraborty and Kar 2017) and evolutionary algorithms (EAs). EAs are increasingly demanded for solving optimization problems as they can achieve very good results across a wide range of problem domains (Dasgupta and Michalewicz 2014). There are many open research lines regarding EAs: genetic algorithms (GAs) (Holland 1975), evolutionary strategies (Rechenberg 1973), evolutionary programming (Fogel et al. 1966) or genetic programming (Cramer 1985).

All research lines regarding EAs follow similar steps adopted from natural evolution: iteratively, the algorithm evaluates the quality of solutions given a cost function, and reproduces new solutions based on the best ones identified. New solutions are reproduced following different techniques, which differentiate each of the approaches. The main disadvantage found in traditional EAs is that they have a bad behaviour when some characteristics are present in the problem: they do not explicitly consider dependencies among the variables involved, and hence, solutions reproduced by the algorithm are not able to exploit the information found in the data. This limitation is overcome by using a probabilistic graphical model (PGM) (Koller and Friedman 2009) to reproduce new solutions. This type of EAs are the estimation of distribution algorithms (EDAs) (Mühlenbein and Paass 1996).

The typical optimization problems that are usually referenced in the literature involve optimizing a cost function with a specific number of variables that are introduced to the algorithm in order to find the best solution. However, there are many real-world optimization problems, in which, variables that are not present in the cost function, influence the behaviour of the algorithm during the optimization process. Therefore, the total set of variables ($\boldsymbol{X}$) is decomposed into two subsets: those that are present in the cost function and can change their value during the optimization process ($\boldsymbol{Y} = \{y_1, \ldots, y_p\}$) which will be henceforth called as decision variables, and others, which are the inputs of the algorithm, and remain constant during the process ($\boldsymbol{Z} = \{z_1, \ldots, z_c\}$).

$$\boldsymbol{X} = \{\boldsymbol{Y}, \boldsymbol{Z}\} = \{y_1, \ldots, y_p, z_1, \ldots, z_c\}$$

The relations within and between both subsets of variables must be considered in order to find coherent solutions to the problem.

In this paper, we aim to solve one of these problems. The motivation for the problem comes from a real situation in an industry: a chemical process is carried out for the dissolution of a solid substance. Depending on the properties of the substance, the solvents to be used may change. Moreover, the final product after dissolving is stored in a tank and must comply with some further restrictions. Thus, the combination of solvents depends not only on the substance properties, but also on the final product restrictions. The industry aims to optimize the process and relies on a system to decide which is the optimum combination of solvents ($\boldsymbol{Y}$) for the specific environment variables ($\boldsymbol{Z}$): some substance properties and final product restrictions. In this problem, optimization is evaluated from an economic point of view: each solvent is associated with a price, and it is desired to minimize their total costs. The optimization process uses the historic records of the laboratory technicians that dissolved the substance so far. Figure 1 shows a sketch of the process.

Depending on the properties of the substance and the restrictions imposed on the final tank, the solvent combinations vary, since the dependencies between the variables are different. If the dependencies between and within both subsets of variables are not considered, only the p variables involved in the cost function will be observed during the optimization process. Thus, the cheapest combination of solvents found by the algorithm will be to use 100% of the cheapest solvent and not using the others, as they would increase the cost. Optimizing only the solvent variables could be solved using a simpler optimizer, such as a gradient descent. The aim of this paper is to find the cheapest solvent combination, but keeping the patterns identified in the initial data between and within both subsets of variables. These patterns constrain the cost, and, for this reason, the implemented algorithm must identify the patterns in the initial data, and generate valid solutions according to them.

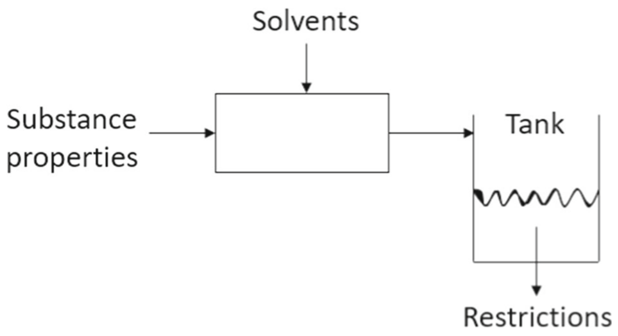

Fig. 1 Sketch of the chemical process to optimize. The combination of solvents is influenced by the substance properties and the quality that the final product must attain (Color figure online)

The cost function is then given by,

$$C(\boldsymbol{Y}) = \sum_{i=1}^{p} y_i c_i \tag{1}$$

where y_i and c_i are the percentage of each solvent in the mixture ($\sum_{i=1}^{p} y_i = 1$) and its cost, respectively, and p is the total number of solvents. The optimizer should search for the optimum values of y_i such that the dependencies with the environment variables hold.

The problem consists of $p + c$ continuous variables: p solvents which each one specifies the percentage of the total mix of solvents, and c substance properties and restrictions. The substance properties specifically describe the substance, such as thermodynamic properties or chemical elements presence in the substance, and some final product restrictions are for example the density, volatility or viscosity of the final liquid or a quality index calculated by the industry. Thus, the algorithm receives as input a total of c constant values $b_1, \ldots, b_c$, and must output the optimum combination of the p solvents. In this particular problem, the available historic records of how laboratory technicians have combined the solvents has a total of 1056 instances.

The application of most population-based algorithms and conventional solvers does not consider explicitly the relationship between the variables or does not allow the existence of a subset of constant inputs. In this paper, a novel technique of EDAs is prepared to overcome this limitation of traditional population-based algorithms. It includes a PGM to identify conditional independence relationships between and within both $\boldsymbol{Y}$ and $\boldsymbol{Z}$ subsets of variables. The PGM allows to set a subset of constant variables as observed evidence of the model and then sample from the rest of the variables. Moreover, a new hyper-parameter is added to the implementation to control the influence of the initial data in the algorithm's decision making.

A PGM specifies the factorization of the joint distribution over a set of variables. In this paper, a PGM learnt in each generation of the algorithm builds an abstract representation of the relationships among the variables (solvents percentages, substance properties, and restrictions) of the promising solutions. In this representation, the patterns among variables can be identified in order to find better solutions. The use

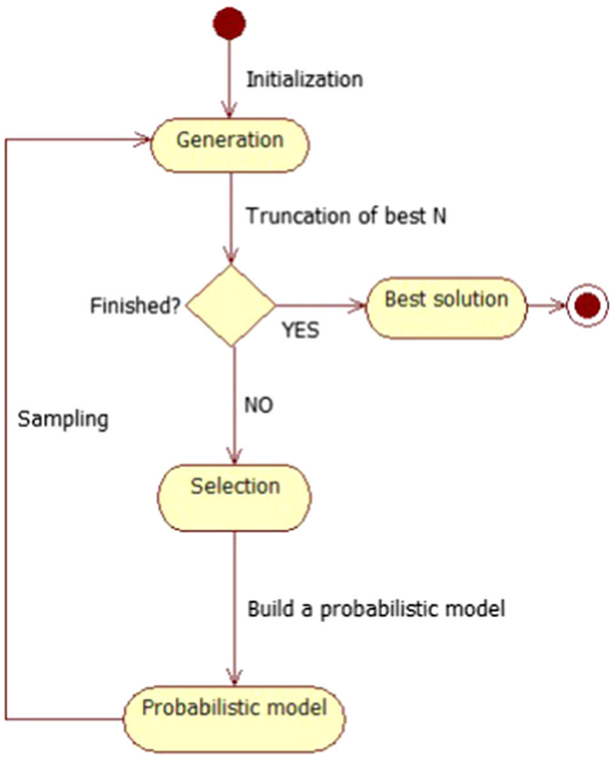

Fig. 2 General flowchart of estimation of distribution algorithms (Color figure online)

of PGMs provides the algorithm with the ability to deal with problems that involve variables with strong dependencies among them.

So far, EDAs have been widely used to solve different optimization problems such as route planning for vehicles (Pérez-Rodríguez and Hernández-Aguirre 2019), harvesting agricultural fields (Utamima et al. 2019), energy-efficient robots manufacturing (Sun et al. 2020), permutation-based combinatorial problems (Irurozki et al. 2018), and public transport driver scheduling (Shen et al. 2017). EDAs are also used coupled with reinforcement learning (RL) approaches as they demand a large amount of data, and EDAS can satisfy this needness by working as a generative model of valid data (Li et al. 2014).

The main scheme of EDAs is shown in Fig. 2. The algorithm, in each iteration, selects the best individuals from the previous generation, or in the case of the first iteration, from the initialization. This selection is made according to the cost function that is to be optimized in the optimization problem. From this ranking, some individuals are selected, and reproduction is performed. For the reproduction step, a probabilistic model is built from the promising individuals selected, and new individuals are sampled from it, to form the next generation. The algorithm can be initialized randomly or with a subset of some initial solutions, allowing for different stopping criteria, and can return not only the best individual found, but also, a set of optimum solutions.

Depending on the type of variables that the optimization problem deals with, a discrete or a continuous EDA is used. When dealing with continuous variables, it is common to assume an underlying probability distribution. The usual choice is a Gaussian distribution. As discrete EDAs were studied before continuous EDAs, most of the first continuous EDAs were adaptations of discrete EDAs to continuous environments.

Another categorization can be done analysing the dependencies among the variables. If no dependencies among variables are considered, then variables are independent, and a univariate EDA is used. Some examples of univariate EDAs are the univariate marginal distribution algorithm (UMDA) (Mühlenbein and Paass 1996), which in each iteration of the algorithm learns the parameters of the marginal distributions of the variables; and the population-based incremental learning (PBIL) (Sebag and Ducoulombier 1998), which updates the existing model using the statistics learnt from the new selection of the actual generation. An approximation of both UMDA and PBIL algorithms to the continuous domain is discussed in Larrañaga and Lozano 2001. This type of EDAs cannot be used in complex optimization problems which involve considering dependencies among variables, as are univariate algorithms.

If variables are dependent by pairs, such as in a tree or a chain structure, then a bivariate EDA is used. An example is mutual information maximizing input clustering for continuous domains ($\mathrm{MIMIC}_{\mathrm{c}}^{\mathrm{G}}$) (Larrañaga et al. 2000). The algorithm uses a probabilistic model with chain structures. It uses the entropy between pairs of variables to build the model structure of each iteration of the algorithm.

If no restrictions are stipulated on the dependencies between the variables, then a multivariate EDA is used. An example of these algorithms is the estimation of Gaussian networks algorithm (EGNA) (Larrañaga and Lozano 2001), which adapts the discrete EDA estimation of Bayesian network algorithm (EBNA), to continuous environments using GBNs (see Sect. 2). One of the difficulties with this type of EDAs is the structure learning in each iteration. EGNA proposes two approaches: start from a complete directed acyclic graph (DAG) and deciding which edges can be modified or deleted, or perform a greedy search to choose the ideal DAG. Another multivariate continuous EDA which uses GBNs is real-coded Bayesian optimization algorithm (rBOA) (Ahn et al. 2004). The iterated density estimation evolutionary algorithm (IDEA) (Bosman and Thierens 2000) and estimation of multivariate normal algorithm (EMNA) (Larrañaga and Lozano 2001) learn a multivariate Gaussian density from which new individuals are sampled. In our optimization problem, dependencies must be considered as a multivariate model and variables involved are continuous and fit a Gaussian distribution, therefore, a multivariate continuous EDA using GBN is implemented. The EDA implemented in this paper shares characteristics with the EGNA approach.

The paper is organized as follows. In Sect. 2, a brief background about concepts used in the implementation of the algorithm is presented. Section 3 explains the implementation of the algorithm. Section 4 presents some results of the evolution of the optimization process and the behaviour of the algorithm for different problem instances and different parameter tuning. Section 5 rounds the paper off with the conclusions.

2 Background

2.1 Bayesian networks

A Bayesian network (BN) (Koller and Friedman 2009) is a representation of a multivariate probability distribution, by means of a DAG, whose nodes are the random variables of the domain and the arcs among them represent the probabilistic influence of some nodes on others. Each arc represents a conditional probability distribution (CPD) between two variables. Each variable has a local probability distribution $P(X_i)$, and the CPDs are the probabilities of one given their parents in the graph $P(X_i|\mathbf{Pa}(X_i))$.

The joint probability of a multivariate model would be,

$$P(X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} P(X_i|\mathbf{Pa}(X_i)), \tag{2}$$

where n is the total number of variables, and $\mathbf{Pa}(X_i)$ are the parent nodes of variable X_i. Note that, $X_1, X_2, \ldots, X_n$ is the set of variables involved in the BN.

For sets $\mathbf{X}$, $\mathbf{Y}$ and $\mathbf{Z}$ of variables in a distribution P, $\mathbf{X}$ is independent of $\mathbf{Y}$ given $\mathbf{Z}$ if for all values of $\mathbf{X}$, $\mathbf{Y}$ and $\mathbf{Z}$, P satisfies $(\mathbf{X} \perp\!\!\!\perp \mathbf{Y}|\mathbf{Z})$. In a BN, a variable is independent of all other variables $\mathbf{U}$ given its Markov blanket ($\mathbf{MB}$), which includes its parents, its children and the parents of its children. Thus, $\mathbf{X} \perp\!\!\!\perp \mathbf{U}|\mathbf{MB}(\mathbf{X})$.

There are several algorithms for learning a structure from data. Three approaches can be considered: (i) constraint-based algorithms (Verma and Pearl 1990) which recover probabilistic relations with conditional independence tests; (ii) score-based algorithms (Madigan et al. 1995) which perform a heuristic search in the space of graph structures to maximize the score function that measures how well the structures fits the data; and (iii) hybrid algorithms which are a mixture of both constraint-based and score-based algorithms.

When learning a BN it is possible to set mandatory arcs in the structure, although the learning algorithm may not find these arcs. The list of arcs that must be included in the DAG is the *whitelist*. In the same way, a list of forbidden arcs can be specified: the *blacklist*.

2.2 Gaussian Bayesian Networks

A Gaussian Bayesian network (GBN) is a BN in which all of the variables are continuous and all the conditional probability distributions are linear Gaussian. The joint probability of all variables is a multivariate Gaussian distribution. In using a GBN, it is assumed that all distributions (marginal, joint, and conditional) fit a Gaussian. If this assumption is not satisfied, models based on the GBN will provide non-reliable solutions.

To perform an inference from a BN, a probability distribution of interest is calculated. In the case of a GBN, inferences are performed by determining the Gaussian probability distribution of a set of variables. To infer a conditional probability, the probability distribution of a set of variables ($\mathbf{X_1}$) is calculated given a fixed value of one or more variables ($\mathbf{X_2}$). This is $P(\mathbf{X_1}|\mathbf{X_2})$. To work with Gaussian probability distributions, some formulas are needed (Murphy 2012). The conditional probabilities are calculated as follows: if $\mathbf{X} = (\mathbf{X_1}, \mathbf{X_2})$ is jointly Gaussian with parameters

$$\boldsymbol{\mu} = \begin{pmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{pmatrix}, \boldsymbol{\Sigma} = \begin{pmatrix} \boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{pmatrix}, \boldsymbol{\Lambda} = \boldsymbol{\Sigma}^{-1} = \begin{pmatrix} \boldsymbol{\Lambda}_{11} & \boldsymbol{\Lambda}_{12} \\ \boldsymbol{\Lambda}_{21} & \boldsymbol{\Lambda}_{22} \end{pmatrix}$$

then marginal probabilities are given by,

$$\begin{aligned} P(\mathbf{x}_1) &= \mathcal{N}(\mathbf{x}_1|\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_{11}) \\ P(\mathbf{x}_2) &= \mathcal{N}(\mathbf{x}_2|\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_{22}) \end{aligned}$$

and the posterior conditional is given by,

$$\begin{aligned} P(\mathbf{x}_1|\mathbf{x}_2) &= \mathcal{N}\left(\mathbf{x}_1|\boldsymbol{\mu}_{1|2}, \boldsymbol{\Sigma}_{1|2}\right) \\ \boldsymbol{\mu}_{1|2} &= \boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}(\mathbf{x}_2 - \boldsymbol{\mu}_2) \\ &= \boldsymbol{\mu}_1 - \boldsymbol{\Lambda}_{11}^{-1}\boldsymbol{\Lambda}_{12}(\mathbf{x}_2 - \boldsymbol{\mu}_2) \\ &= \boldsymbol{\Sigma}_{1|2}(\boldsymbol{\Lambda}_{11}\boldsymbol{\mu}_1 - \boldsymbol{\Lambda}_{12}(\mathbf{x}_2 - \boldsymbol{\mu}_2)) \\ \boldsymbol{\Sigma}_{1|2} &= \boldsymbol{\Sigma}_{11} - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21} = \boldsymbol{\Lambda}_{11}^{-1} \end{aligned}$$

3 Proposed solution

As mentioned above, an EDA using a GBN is used to solve the optimization problem. The pseudocode in Algorithm 1 shows how the proposed EDA is implemented.

Algorithm 1 EDA

```
1: $G_0 \leftarrow N$ individuals from the historic records
2: $G_{aux} \leftarrow$ Select similar situation individuals from the historic records
3: for $l = 1, 2, \ldots$ until stopping criterion is met do
4:     $G_{l-1} \leftarrow$ Select $E$ individuals from $G_{l-1}$
5:     $G_{l-1} \leftarrow$ Append $G_{aux}$ to $G_{l-1}$
6:     $GBN_l \leftarrow$ Learn GBN from $G_{l-1}$
7:     $Pe(x) \leftarrow$ Compute the joint density function from $GBN_l$
8:     $G_l \leftarrow$ Sample $N$ individuals from $Pe(x)$
9: end for
```

In the pseudocode, the main functions of the algorithm are shown. Initially, G_0 is an input to the code. Then, in each iteration, the algorithm selects the top individuals of the generation considering the cost function. The algorithm learns a GBN, a multivariate Gaussian distribution is then derived from the GBN, and new individuals are sampled from joint distribution of it. Each set of new individuals constitutes a generation that is the input for the next iteration of the algorithm. The stopping criteria may be based on a maximum number of iterations, a state in which no better solutions are found by the algorithm, or a state in which the algorithm is not able to generate valid individuals.

In this implementation the stopping criterion is a number of iterations after which the algorithm cannot improve the best global cost. The main functions used in the algorithm are described below.

The GBN learnt in each generation is an abstract representation of the region of the search space explored, which guides the algorithm to promising areas. The individuals sampled from the GBN keep the patterns identified in the data used to learn the multivariate model. The use of GBNs allows generating solutions that were not contemplated in the learning of the GBN.

The use of some algorithms is restricted to the amount of data availability. For example, using neural networks demands a large amount of data to be trained. However, building a BN does not require such a large amount of data. The EDA can be run correctly with the 1056 instances we dispose of for the initialization step. Another advantage of the use of GBNs is that the DAG can be visualized in each generation in order to show patterns in data, or to easily analyse the algorithm behaviour.

3.1 Initialization

The initialization of the algorithm has a large influence on how the algorithm moves through the search space during execution. The aim of this paper is to implement an algorithm which finds the optimum solution of the problem keeping the patterns identified in the data. A random initialization would generate some random patterns that would be kept through iterations and this is not desired. This could be addressed implementing a *whitelist* in the GBN in order to fix the arcs that are desired in the generated samples, but this would over-bias the algorithm, as exploration would be limited.

In our approach, the initialization is based on the historic records of how technicians of the laboratory have been manipulating the solvents. In the chemical laboratory, different sensors record values of the variables used in the problem. First, some sensors measure the substance to be dissolved providing the properties variables. Second, the tank where the final product is to be stored has some further restrictions that must be accomplished, and are also sensorized. Also, the amount of each solvent introduced in the solvent mixture is recorded. These values are the historic records. We assume that the staff has been manipulating the solvents to obtain the best results, considering the restrictions and the final cost. As these historic records are considered nearly optimum, the algorithm is expected to converge soon.

3.2 Truncation

When a generation is obtained in each iteration, the individuals whose values in the subset Y of variables minimizes the cost function, are interesting and must therefore be selected. Once they are selected, all the individuals chosen have the same relevance regardless of the cost of the individual. Moreover, those selected individuals may have a very low probability of occurrence in the historic records. This can lead the algorithm to learn confusing patterns, and future generations will be sampled incorrectly. For this reason, two modifications are made in the baseline of the algorithm.

First, the cost is added as a new node of the GBN. By adding this new variable, a certain coherence is ensured among all the samplings. If the cost is considered as a new variable, the majority of the individuals in a generation will be sampled around the mean cost according to the shape of the normal distributions, and thus, anomalous individuals are less likely to be obtained when sampling. This does not avoid anomalous individuals to be sampled, because a small part of the samples will be in the tails of the normal distributions, but maximum dispersion among individuals is reduced. This new variable is calculated with the cost function given in Eq. (1). In this way, the GBNs learnt in each iteration are more coherent. Other multi-objective EDAs has also added the cost as nodes of the BN, such as Karshenas et al. 2014.

Second, the likelihood of each individual in the historic records is considered. If only the individuals that minimize the cost function were selected, it would be possible that very anomalous individuals would be selected, as individuals from different promising spaces, even located at the tails of the distribution, may be found. To avoid this, both, the cost of the individual and the likelihood of the individual in the historic records must be considered in the truncation ranking. An individual that costs very cheap is very attractive, but if its likelihood is low in the historic records, it might not be worth choosing. Therefore, those individuals whose cost is not as cheap as the so attractive ones of the generation, but their likelihood is higher, are preferable. For this reason, the likelihood in the historic records must be considered.

The ideal is to find a balance between the likelihood in the historic and the randomness of the samples of the individuals in the multivariate model. Consequently, a new hyper-parameter α is added to the implementation, which controls the influence of the likelihood in minimizing the cost function. We define the parameter as a value in the range [0, 1]. If the likelihood is not taken into account ($\alpha = 0$), the solutions can be dispersed, and different executions of the EDA can provide very diverse solutions that may be very cheap. However, if the likelihood is totally considered ($\alpha = 1$), the optimizer can nearly not optimize the solutions in the historic records, as the EDA will try to find a similar solution to those that can be found in the historic records, and not a solution based on the patterns found in the data. An analysis of this parameter is provided below. Thus, the function used in the truncation is,

$$C_\alpha(\boldsymbol{x}) = \frac{\sum_{i=1}^{p} y_i c_i}{\sum_{i=1}^{p} c_i} - \alpha L(\boldsymbol{x}|M), \tag{3}$$

where $L(\boldsymbol{x}|M)$ is the likelihood of an individual $\boldsymbol{x}$ in the Gaussian multivariate model M estimated from the initial generation (the historic records), and y_i and c_i are the solvent amounts percentages and costs, respectively. The first term of the equation has been normalized so that a comparison with the second term is possible, as likelihood is represented as a probability. For $\alpha \to 1$ the historic relevance tends to have the same relevance as the cost of the individual. For $\alpha = 1$ the cost and the historic influence have the same relevance in Eq. (3).

It is expected that the likelihood of the individuals of each generation will increase. Because of this, Eq. (3) is not the cost function that should be minimized. It is only a function used to evaluate the individuals in the truncation step, and establish a ranking, to guide the selection of the top individuals.

With these two modifications it may happen that some very cheap individuals are rejected from a generation due to their anomalous costs in comparison with those of the rest of the population. In this case, these rejected individuals will appear in future generations, where their likelihoods and costs are more consistent with those of the other individuals in the population. This favours the building of coherent GBNs in each iteration, as more coherence among individuals is obtained.

3.3 Problem formulation

The mathematical formulation of the problem is presented in this section. The problem involves $p + c$ variables, $\boldsymbol{X} = \{\boldsymbol{Y}, \boldsymbol{Z}\} = \{y_1, \ldots, y_p, z_1, \ldots, z_c\}$, where $\boldsymbol{Y}$ are the decision variables and $\boldsymbol{Z}$ are the environment variables fixed as constant.

The optimization problem is,

$$\begin{array}{ll} \text{minimize} & \min\limits_{\boldsymbol{x}} \left(\dfrac{\sum_{i=1}^{p} y_i c_i}{\sum_{i=1}^{p} c_i} - \alpha L(\boldsymbol{x}|M) \right) \\ \text{subject to} & f_i(\boldsymbol{x}) = b_i, \ \text{for } i = 1, \ldots, c, \end{array}$$

where $c_i \in \mathbb{R}^+$ are constant terms pre-defined in the problem by $\boldsymbol{C} = \{c_1, \ldots, c_p\}$, the parameter $\alpha \in [0, 1]$, and $\sum_{i=1}^{p} y_i = 1$.

Here, the vector $\boldsymbol{x} = (y_1, \ldots, y_p, z_1, \ldots, z_c)$ is the optimization variable of the problem, the function $C_\alpha(\boldsymbol{x}) : \boldsymbol{R}^{p+c} \rightarrow \boldsymbol{R}$ is the objective function defined in Eq. 3, the functions $f_i : \boldsymbol{R}^c \rightarrow \boldsymbol{R}, i = 1, \ldots, c$ are the constraint functions and the constants $b_1, \ldots, b_c$ are the constraint values that the constraint functions should meet. A vector $\boldsymbol{x}^*$ is optimal if it has the lowest objective value among all vectors that satisfy the constraints. Thus, for any $\boldsymbol{x}$ with $f_1(\boldsymbol{x}) = b_1, \ldots, f_c(\boldsymbol{x}) = b_c$ we have $C_\alpha(\boldsymbol{x}) \geq C_\alpha(\boldsymbol{x}^*)$.

Despite the fact that the cost function is defined in Eq. (1) in terms of the decision variables y_i, the environment variables $\boldsymbol{Z}$ in the problem statement must be considered using Eq. (3) during the optimization process. However, as this approach has an industrial perspective, the results shown in Sect. 4 represent the economic cost calculated with Eq. (1) for each solution $\boldsymbol{x}$ provided by the algorithm.

3.4 Estimation of the density function

When the top individuals are selected using Eq. (3), the structure of the GBN can be learnt. As the only variables that the EDA can optimize are the subset $\boldsymbol{Y}$ of variables, the environment variables are inputs of the problem and hence, evidences of the multivariate Gaussian distribution. Thus, in each generation the subset of environment variables will assign to each individual the same value. When the learning algorithm tries to find dependencies among variables, all the fixed environment variables are found to be independent, and if some dependence is found it would be spurious. This

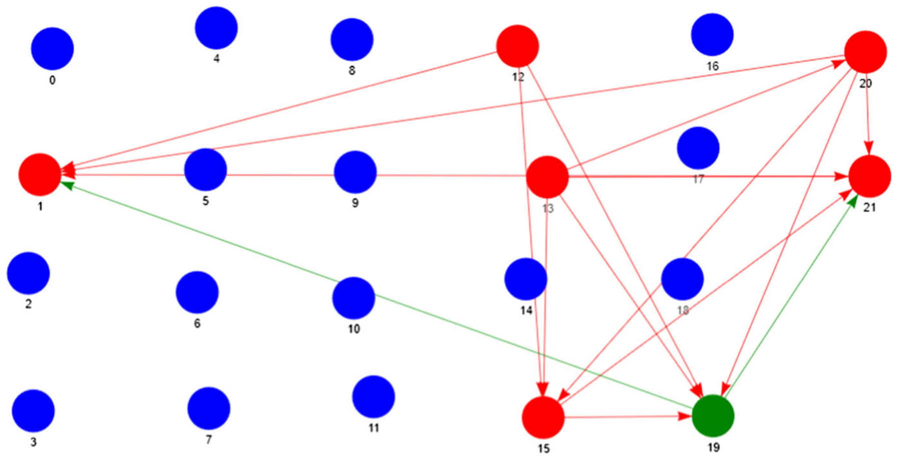

Fig. 3 DAG learnt in an EDA execution where the input variables are fixed as evidence of the GBN. Only the y_i variables (red nodes) are connected in the DAG, and the rest are independent. The blue nodes represent the environment variables and the green node is the cost (Color figure online)

means that the variables included in the subset $\boldsymbol{Y}$ of variables would only influence and be influenced by each other, without considering the subset $\boldsymbol{Z}$ of variables. This would be a situation in which the algorithm finds the optimum combination of solvent amounts, without considering the rest of the variables: $\boldsymbol{Y} \perp\!\!\!\perp \boldsymbol{Z}$. Figure 3 shows this type of situation where the red nodes represent the subset $\boldsymbol{Y}$ of variables, the green node represents the cost function, and the blue nodes are the environment variables $\boldsymbol{Z}$.

The patterns identified in the historic records must be translated to the GBN learnt from the top individuals of each generation. Therefore, some individuals from the historic records are added to the selection made in the truncation phase of the algorithm. These added individuals G_{aux} (see Algorithm 1) are selected from the historic records in such a way that individuals similar to the problem situation are selected, that is, similar environment variables. If a random set of individuals is added from the historic records, a lot of dispersion may be introduced, or solutions from different zones of the search space may be selected, and hence, confusing patterns could be learnt by the algorithm. For this reason, clustering is implemented in the historic records considering the subset of environment variables. In this way, G_{aux} is incorporated into the selection of the E (see Algorithm 1) individuals made in the truncation. G_{aux} will provide to the E selection of individuals, the patterns of the search space explored by the algorithm in order to identify the dependencies among the variables.

Once G_{aux} is added to the selection made in the truncation phase, the GBN can be learnt. The score-based hill climbing algorithm is used. The subset of environment variables are the inputs of the problem, and thus, evidences of the GBN. If z are the input values that the subset $\boldsymbol{Z}$ of variables take, the multivariate Gaussian distribution built is $P(Y|Z = z)$ with the evidences as fixed values in the multivariate Gaussian distribution. It is possible to add expert knowledge to the GBN. Forbidden relations among variables can be specified in a *black list* in order to not consider them, and mandatory relations among variables can be specified in a *white list*.

When the optimal combination of y_i is found, an optimal structure of the GBN is learnt for specific substance properties and further restrictions. Thus, this GBN can be used to perform inferences and analyse different combinations of y_i. The structure can be also used by experts to infer patterns among variables of the system. By introducing the cost as a node of the GBN, it can be used by the experts to calculate some posterior probabilities of the cost $\boldsymbol{C}$ given some variables values $x_1, \ldots, x_t$, for example, $P(C|x_1, \ldots, x_t)$ for classification tasks.

3.5 Sampling

Once the GBN is built, the next generation can be sampled. It is expected that the dispersion among the individuals in the initial iterations is higher than in later iterations. The behaviour of the algorithm is designed to make the space search move towards the optimal space and to reduce the dispersion. The more iterations of the algorithm, the lower dispersion between individuals, until convergence of the algorithm is reached and the dispersion among individuals is minimum. Thus, the individuals in the last iterations should be similar and centred in the optimal solution found.

Despite the fact that the multivariate Gaussian distribution is estimated from real data, it is possible that non-real data are sampled. These samplings must be discarded. Otherwise, the algorithm will tend towards a minus infinite cost.

Depending on the input values the number of samplings removed may be different. In some situations, the optimizer may find solutions in which some of the y_i variables are reduced to nearly zero, so due to the Gaussian distribution, some of the samplings will lead to solutions with values less than zero. However, other input variables will make the optimizer stabilize the amounts at a realistic percentage, and the number of removed samplings will be smaller. The frequency of appearance of these invalid individuals is small, so it does not imply a significant computational cost.

4 Results

As the system was implemented to optimize a process in an industry, the model was run considering expert knowledge. The optimizer results were validated with expert technicians of the chemical laboratory.

In the problem we aim to solve in this paper, the substance properties and the final liquid restrictions in the tank are given as inputs (environment variables), and the optimal combination of the six solvent amounts are returned as output. In this section, some real problem results are provided. To validate the optimizer, real historic records situations have been selected. The values of the subset of environment variables are selected and the process and final results of the optimizer are analyzed by expert knowledge in the field. Expert knowledge is of great importance in this validation process as it must corroborate that the existing relationships that are identified between the data are correct.

Figure 4 shows the best cost evolution with an increasing number of iterations. In each iteration, note that the algorithm tries to converge towards a better solution

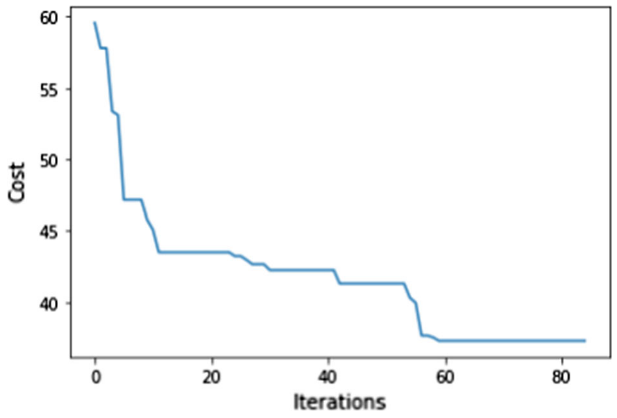

Fig. 4 Best cost (Eq. (1)) found in each iteration of an EDA execution for a specific substance with restrictions (Color figure online)

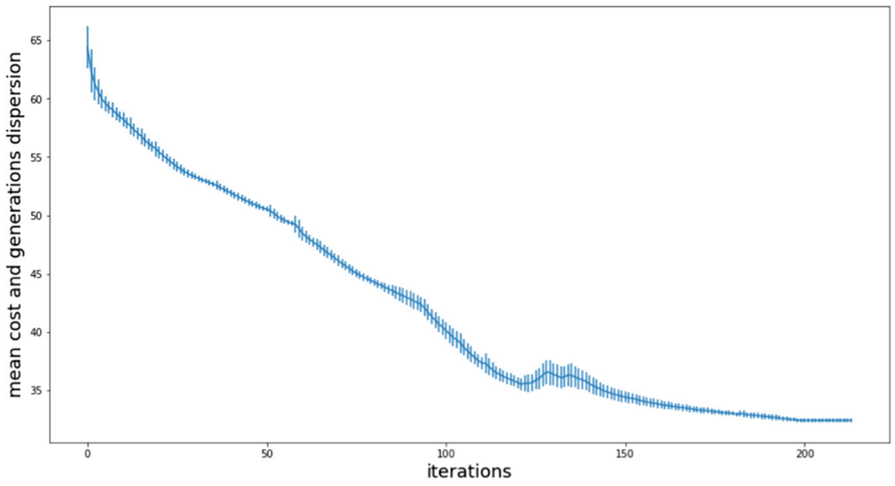

Fig. 5 Mean cost (Eq. (1)) and dispersion among individuals in each generation for a specific substance with restrictions (Color figure online)

than it has so far. When the algorithm cannot improve the best solution found after an specific number of iterations, the algorithm converges and returns the best solution found. When the curve flattens, the EDA has found the optimal area of the initial search space (iteration 60). One of the advantages of this algorithm is that not only a single solution can be returned. The algorithm can return a set of solutions with similar costs, in order to select one preferable by laboratory technicians.

The expected behaviour of the algorithm evolution is that the dispersion among individuals is reduced as the number of iterations increases. Figure 5 shows the evolution of the mean cost and the dispersion among individuals along an EDA execution. The dispersion does not have to decrease in a linear way, but a decreasing trend is appreciated in its evolution. The way to measure this dispersion is the distance from the mean cost of the generation.

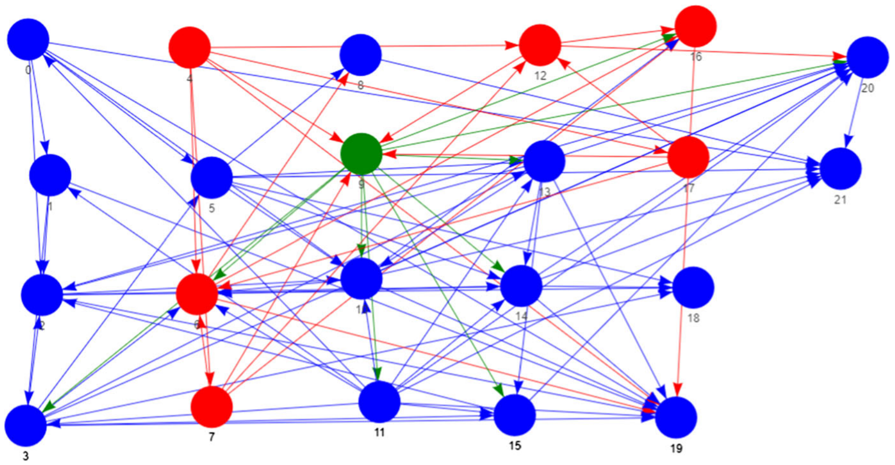

Fig. 6 DAG of the GBN resulting from an EDA execution. The optimal GBN is obtained for specific substance properties and restrictions. No nodes are independent. The blue nodes represent the substance properties and restrictions, the green node is the cost, and the red nodes are the solvent amounts (Color figure online)

The ability to return a set of optimal solutions to the problem is not the only notable advantage of this algorithm. Another one is that once the algorithm finds the optimal area of the search space, it is possible to save the GBN of this area as the optimal structure for the concrete problem. This GBN can be used with different purposes. The GBN found for an EDA execution is shown in Fig. 6, which is a representation of the optimal space that the EDA found in the initial search space. Note the difference between both learned structures in Figs. 3 and 6. Figure 3 shows a situation in which environment variables ($\boldsymbol{Z}$) are independent nodes and only the solvent variables ($\boldsymbol{Y}$) are influenced by each other as they are continuously being optimized during the process: $\boldsymbol{Y} \perp\!\!\!\perp \boldsymbol{Z}$. However, this is solved by adding G_{aux} in each generation of the algorithm (see Algorithm 1). This way, we ensure that dependencies between both $\boldsymbol{Y}$ and $\boldsymbol{Z}$ subsets of variables are considered and the algorithm reaches a realistic situation as shown in Fig. 6, where environment variables influence the decisions in the solvent variables.

Notions of expert knowledge in the field will not be explained, but different aspects of the DAG shown in Fig. 6 must be analyzed. In a BN a variable depends on those in its MB. All y_i variables are dependent on each other. This relation is obvious as are percentages of a total amount of solvent mix. If some of the solvents amount are reduced, others must compensate this reduction ($\sum_{i=1}^{p} y_i = 1$). Thus, all the solvents are dependent on each other. The cost node depends on all the solvents, and other environment variables. The modification of some of the solvent amount is directly related to the cost variation. Other relations can be appreciated, such as some environment variables which are directly related to the cost variable. This GBN can be used to perform different inferences, to try different combinations of solvents amounts, and infer the most probable solvent combination always keeping in mind that the GBN represents an optimal area in the initial search space.

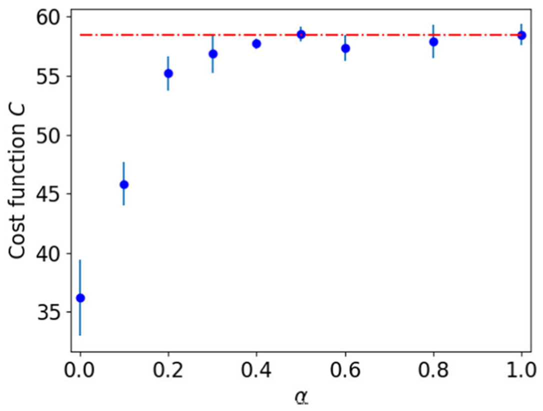

Fig. 7 Mean cost as a function of the α parameter (Eq. (3)). For each value of α, 20 simulations were run. The dashed line corresponds to the cost of solvents traditionally used in the laboratory processes for this specific substance and set of restrictions (Color figure online)

Figure 7 shows the value of the cost function C as a function of the α parameter (Eq. (3)). For the same problem (same inputs), the EDA was run 20 times for each value of α. C_α (Eq. (3)) was used to perform the truncation, and find the minimum cost function (Eq. (1)). Note that the mean C value increases with α while its dispersion among different solutions decreases. As $\alpha \to 1$, C tends toward the value determined by expert knowledge ($C \approx 58.5$). However, although the dispersion increases for $\alpha = 0$, for all the simulations the optimal costs found is below that determined by theoretical expertise. The figure also shows that the cost converges to a constant value (dashed line) for $\alpha > 0.5$, which represents the cost based on expert knowledge.

For $\alpha > 0.5$, the optimizer obtained similar costs and small dispersion among solutions. In Eq. (3) the cost of the solvent combination tends to have the same relevance as the likelihood in the historic records. Thus, the algorithm will not provide solutions that do not fit well with the historic records. The solution provided by the algorithm will be the same or nearly the same as that of the historic records, as the laboratory records are assumed to be nearly optimum. For highest values of α the optimizer performs as a predictive model, as the solutions provided are the ones that can be found in the historic records.

For low values of the α parameter, the solutions provided have a larger dispersion. The algorithm is therefore more stochastic than it is for high values of the α parameter. For each iteration, only the cost of the solvent combination is considered, and thus, the individuals that have a lower density in the historic Gaussian multivariate data will not be removed and can lead to new solutions not contemplated. As the likelihood is nearly not considered, the solutions provided by the algorithm are more stochastic, and thus, in different executions, different GBNs can be learnt. This explains the dispersion among the different executions shown in Fig. 7 for low values of α.

Facing an industrial application, it must be considered the α hyper-parameter tunning. Despite the fact that the theoretical basis on which the implementation of the algorithm is based is correct, using those solutions found by the algorithm for α close to zero may carry a risk; the algorithm may have learnt wrong patterns during runtime so not reliable solutions may have been found, or more constraints should be added to translate the *savoir-faire* of the technicians.

Thus, it is necessary to find a balance between the stochasticity of the algorithm and the solutions already found in the historic records. This balance must be found using

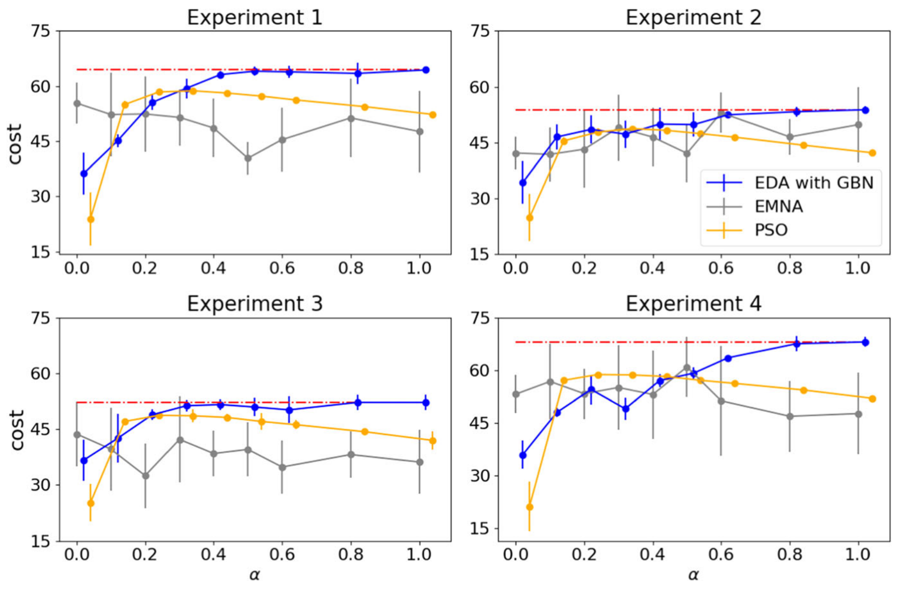

Fig. 8 Mean cost and standard deviation of EDA with a Gaussian Bayesian network, and the EMNA and PSO algorithms as a function of the α parameter (Eq. (3)) for the different experiments. For each value of α, 20 simulations were run. The dashed line corresponds to the cost of solvents traditionally used in the laboratory processes for this specific substance and set of restrictions (Color figure online)

expert knowledge, or even carrying out virtual simulations of the chemical process to properly tune this novel hyper-parameter. The new hyper-parameter can be discussed from the exploitation-exploration point of view. For high values of the parameter, the exploration of the search is minimum while the exploitation of the already existent solutions in the historic data is maximum. Though, low values of the parameter leads to explore the search space.

In order to perform a deeper analysis, the proposed approach has been executed for different instances of the problem. We have designed four experiments in which the substance to be dissolved is the same, but different environment variables are present.

- Experiment 1. The viscosity index of the dissolution result has a value of 10% below the mean found in the historical data.
- Experiment 2. The viscosity index of the dissolution result has a value of 10% above the mean found in the historical data.
- Experiment 3. The volatility index of the dissolution result and the quality index defined by the industry are decreased a 10% below the mean found in the historical data.
- Experiment 4. The volatility index of the dissolution result and the quality index defined by the industry are increased a 10% above the mean found in the historical data.

The experiments have been executed for the Estimation of Multivariate Normal Algorithm (EMNA) (Larrañaga and Lozano 2001), the Particle Swarm Optimization (PSO) (Kennedy and Eberhart 1995) and our approach. The EMNA approach has a

Table 1 Mean and standard deviation for each of the four designed experiments and different values of α executed by our approach (Fig. 8)

α	Experiment 1	Experiment 2	Experiment 3	Experiment 4
0.0	36.22 ± 5.66	34.27 ± 5.76	36.67 ± 5.55	37.46 ± 3.99
0.1	45.17 ± 1.77	46.52 ± 3.45	42.55 ± 6.50	48.03 ± 1.23
0.2	55.54 ± 2.01	48.55 ± 3.72	48.84 ± 1.46	52.26 ± 4.26
0.3	59.40 ± 2.70	47.29 ± 3.68	51.29 ± 1.57	50.33 ± 3.22
0.4	63.14 ± 0.85	49.97 ± 4.42	51.61 ± 1.41	56.40 ± 1.95
0.5	64.05 ± 1.24	49.84 ± 3.26	50.95 ± 2.50	59.49 ± 1.58
0.6	63.87 ± 1.71	52.47 ± 0.77	50.20 ± 3.54	63.50 ± 0.71
0.8	63.45 ± 2.97	53.30 ± 1.34	52.20 ± 2.12	67.50 ± 2.12
1.0	64.40 ± 0.88	53.84 ± 0.91	51.18 ± 2.12	68.00 ± 1.41

Table 2 Mean and standard deviation for each of the four designed experiments and different values of α executed by the EMNA approach (Fig. 8)

α	Experiment 1	Experiment 2	Experiment 3	Experiment 4
0.0	55.35 ± 5.590	42.22 ± 4.360	43.68 ± 8.727	53.15 ± 5.432
0.1	52.25 ± 11.46	41.86 ± 7.251	39.61 ± 11.08	56.74 ± 10.74
0.2	52.41 ± 10.25	43.23 ± 10.24	32.52 ± 8.702	53.26 ± 7.166
0.3	51.48 ± 7.700	49.01 ± 8.911	42.24 ± 11.57	55.03 ± 12.01
0.4	48.58 ± 7.990	46.48 ± 7.833	38.46 ± 6.135	53.04 ± 12.53
0.5	40.37 ± 4.380	42.13 ± 7.845	39.56 ± 7.312	60.90 ± 8.631
0.6	45.43 ± 8.640	53.03 ± 5.448	34.84 ± 7.151	51.26 ± 15.58
0.8	51.33 ± 10.69	46.53 ± 4.821	38.17 ± 6.170	46.87 ± 10.06
1.0	47.64 ± 11.07	49.80 ± 10.22	36.18 ± 8.651	47.64 ± 11.62

similar pseudo-code as our approach but instead of learning a Gaussian Bayesian network in each iteration of the evolutionary algorithm, it learns a multivariate Gaussian distribution from where it is sampled (lines 6-7 of Alg 1). PSO in each iteration updates its parameters in order to minimize the cost function but not considering dependencies among variables. This way, we compare the difference between using Gaussian Bayesian networks and not using them during the optimization process.

The results are shown in Fig. 8 and in Tables 1, 2 and 3. If we analyze each of the experiments individually, it can be seen that the behavior of our approach is as expected. For low values of α it finds very cheap solutions, and the cost increases with α, until the cost settles to the value determined by expert knowledge for $\alpha \rightarrow 1$, shaping a curve with a logarithmic profile. However, the behaviour of the EMNA and PSO is not as predictable as our approach for different values of α. For $\alpha \rightarrow 0$ our approach seems to find better solutions than EMNA. However, for $\alpha \rightarrow 1$, EMNA and PSO find better solutions than our approach, but always very far from the costs provided by the experts and with high values of standard deviation. EMNA and PSO are not learning the patterns found in data and thus, they are converging to solutions

Table 3 Mean and standard deviation for each of the four designed experiments and different values of α executed by the PSO approach (Fig. 8)

α	Experiment 1	Experiment 2	Experiment 3	Experiment 4
0.0	36.22 ± 5.66	34.27 ± 5.76	36.67 ± 5.55	35.89 ± 3.99
0.1	45.17 ± 1.77	46.52 ± 3.45	42.55 ± 6.50	47.98 ± 1.23
0.2	55.54 ± 2.01	48.55 ± 3.72	48.84 ± 1.46	54.47 ± 4.26
0.3	59.40 ± 2.70	47.29 ± 3.68	51.29 ± 1.57	49.00 ± 3.22
0.4	63.14 ± 0.85	49.97 ± 4.42	51.61 ± 1.41	57.03 ± 1.95
0.5	64.05 ± 1.24	49.84 ± 3.26	50.95 ± 2.50	59.12 ± 1.58
0.6	63.87 ± 1.71	52.47 ± 0.77	50.18 ± 3.54	63.52 ± 0.71
0.8	63.45 ± 2.97	53.31 ± 1.34	52.24 ± 2.11	67.45 ± 2.15
1.0	64.40 ± 0.88	53.84 ± 0.91	51.98 ± 2.08	68.02 ± 1.41

Table 4 Mean and standard deviation of the number of iterations and runtime until convergence for our approach, EMNA and PSO after 20 simulations

Algorithm	# iterations	CPU time
EDA with GBN	1 ± 1	1 ± 1
EMNA	1.12 ± 4.90	0.33 ± 2.98
PSO	1.43 ± 0.37	0.39 ± 0.20

This experiment has been carried out with population size $N = 100$ and $\alpha = 0.6$.The number of fitness evaluations would be NQ, where Q is the number of iterations until convergence. All the results have been normalized taking as reference the results of our approach. The experiment was conducted on a Windows 10 machine with an Intel i7-5820K processor and 16 Gb of RAM

that are cheaper but are not of interest for the company when fully considering the likelihood of solutions in the historical data (Eq. (3)). Facing an industrial solution, when our approach considers the likelihood completely, the laboratory technicians will obtain solutions which they are accustomed to, and as α is decreased, solutions that are somewhat more risky and therefore cheaper will be obtained, but they still take into account the patterns found in the data. It is interesting how the results obtained by the PSO approach when the likelihood is not considered ($\alpha = 0$) outperforms the ones obtained by EMNA and by our approach, and the small standard deviations observed in Table 3 for $\alpha > 0$. From this, we conjecture that PSO is falling in local optima solutions which seems to have been avoided by EMNA and our approach.

The set of experiments shows how in general experiments 2 and 3 are cheaper than the others, which makes sense because in both experiments we lead to solutions in which the final product is closer to a solid than to a liquid, and therefore it is not necessary to dissolve the substance as much as in experiments 1 and 4.

Table 4 shows a quantitative comparison of the EMNA, PSO and our approach analyzing the number of fitness evaluations and computation time until convergence. All the approaches have been evaluated under the same conditions. We can observe how our approach takes the larger runtime compared to the two other algorithms. This is due to the GBN learning in each iteration, which is a characteristic that makes our algorithm to take more than double of the runtime. Our approach seems to need a smaller number of fitness evaluations until convergence compared to its competitors. However, the main advantage of our approach is that we are able to obtain solutions that can approximate those given by experts, parameterized by α, while EMNA and PSO are not able.

Our approach seems to be a good option in those industrial situations where it is crucial to provide similar solutions to the ones offered by the experts ($\alpha \to 1$) but also when cheaper solutions are sought ($\alpha \to 0$) under relaxed computation time requisites.

5 Conclusions

In summary, the aim of this paper was to propose a solution for those optimization problems in which there exist two types of subsets of variables: a subset of variables that define the cost function and can be optimized, and a subset of fixed variables that are the input of the problem. The dependencies between and within both subsets must be considered in order to keep the patterns observed in data. To solve this, an EDA is used which incorporates a GBN that is rebuilt in each iteration to find the optimum model for the optimum area of the search space.

As the algorithm is initialized based on the historic records, a novel hyper-parameter α was introduced to control the influence of the historic conditions on the individuals that constitute each generation. Our results show that cheaper solutions can be obtained when the algorithm is not constrained by the historic conditions ($\alpha = 0$). In this case, the dispersion of the solutions among different EDA executions is higher, but the worst solution for $\alpha = 0$ is cheaper than the solutions in which historic records are considered ($\alpha > 0$). This way, this hyper-parameter must be tuned. Our approach is compared with the EMNA and PSO algorithms in which no GBNs are used to identify the relationships among variables in each iteration. The results show that our approach can approximate the solution given by experts when $\alpha \to 1$ unlike EMNA or PSO.

Future work should consider using kernel estimation rather than Gaussian distributions, as a higher precision can be obtained, as in Wang and Li 2018. In this paper, the hyper-parameter tunes the influence of the historic in the decisions made during runtime by comparing the individuals likelihood in the historic records. A different heuristic could be applied, such as comparing the individuals likelihood in the previous generation, in such a way that, the decisions made during runtime would be influenced by the previous generation, and not by the initial data. Future work also includes adapting this approach to dynamic environments in which environment variables or the cost function vary during runtime.

Acknowledgements We gratefully acknowledge David Quesada and Gabriel Valverde from the Computational Intelligence Group for fruitful discussions.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature. This work has been partially supported by the Spanish Ministry of Science and Innovation through the PID2019-109247GB-I00 and RTC2019-006871-7 projects, by the BBVA Foundation (2019 Call) through the "Score-based nonstationary temporal Bayesian networks. Applications in climate and neuroscience" project. Vicente P. Soloviev has been supported by the predoctoral grant FPI PRE2020-094828 from the Spanish Ministry of Science and Innovation.

Code availability The code is available in **EDAspy** Python package. More implementations of the algorithm can be found in the package such as a binary EDA for feature selection or a univariate continuous EDA for hyper-parameter optimizations. The package can be found at https://github.com/VicentePerezSoloviev/EDAspy and downloaded from Pypi.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

Ahn CW, Ramakrishna R, Goldberg D (2004) Real-coded Bayesian optimization algorithm: Bringing the strength of BOA into the continuous world. In: Genetic and Evolutionary Computation, pp 840–851

Bazgan C, Hugot H, Vanderpooten D (2009) Solving efficiently the 0–1 multi-objective knapsack problem. Comput. & Oper. Res. 36(1):260–279

Bliek1ú C, Bonami P, Lodi A (2014) Solving mixed-integer quadratic programming problems with IBM-CPLEX: A progress report. In: Proceedings of the 26th RAMP symposium, pp 16–17

Bosman PAN, Thierens D (2000) Expanding from discrete to continuous estimation of distribution algorithms: The idea. In: Proceedings of the 6th International Conference on Parallel Problem Solving from Nature, Springer, pp 767–776

Chakraborty A, Kar AK (2017) Swarm Intelligence: A Review of Algorithms. Springer, pp 475–494

Chong EK, Zak SH (2004) An Introduction to Optimization. John Wiley & Sons

Cramer NL (1985) A representation for the adaptive generation of simple sequential programs. In: Proceedings of an International Conference on Genetic Algorithms and the Applications, pp 183–187

Dasgupta D, Michalewicz Z (2014) Evolutionary Algorithms in Engineering Applications. Springer

Fogel LJ, Owens AJ, Walsh MJ (1966) Artificial Intelligence through Simulated Evolution. Wiley

Gurobi Optimization, LLC (2021) Gurobi Optimizer Reference Manual. https://www.gurobi.com

Holland JH (1975) Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence. University of Michigan Press

Huang H, Yoo S, Kasiviswanathan SP (2015) Unsupervised feature selection on data streams. In: Proceedings of the 24th ACM International on Conference on Information and Knowledge Management, pp 1031–1040

Irurozki E, Ceberio J, Santamaria J, Santana R, Mendiburu A (2018) Algorithm 989: perm_mateda: A MATLAB Toolbox of Estimation of Distribution Algorithms for Permutation-based Combinatorial Optimization Problems. ACM Trans. Math. Softw. (TOMS) 44(4):1–13

Karshenas H, Santana R, Bielza C, Larrañaga P (2014) Multiobjective estimation of distribution algorithm based on joint modeling of objectives and variables. IEEE Trans. Evol. Comput. 18(4):519–542

Kennedy J, Eberhart R (1995) Particle swarm optimization. In: Proceedings of ICNN'95-International Conference on Neural Networks, IEEE, vol 4, pp 1942–1948

Koller D, Friedman N (2009) Probabilistic Graphical Models: Principles and Techniques. The MIT Press

Larrañaga P, Lozano JA (2001) Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation. Kluwer Academic Publishers

Larrañaga P, Etxeberria R, Lozano J, Peña J (2000) Optimization in continuous domains by learning and simulation of Gaussian networks. Proceedings of the 2000 Genetic and Evolutionary Computation Conference Workshop Program pp 201–204

Li X, He W, Hirasawa K (2014) Creating stock trading rules using graph-based estimation of distribution algorithm. In: 2014 IEEE Congress on Evolutionary Computation (CEC), pp 731–738

Madigan D, York J, Allard D (1995) Bayesian graphical models for discrete data. Int. Stat. Rev./Rev. Int. de Statistique, 63:215–232

Markowitz HM (1952) Portfolio selection. J. Financ. 7:77–91

Mühlenbein H, Paass G (1996) From recombination of genes to the estimation of distributions I. Binary parameters. In: Proceedings of the 4th International Conference on Parallel Problem Solving from Nature, Springer, p 178–187

Murphy KP (2012) Machine Learning: A Probabilistic Perspective. The MIT Press, Cambridge, MA

Pérez-Rodríguez R, Hernández-Aguirre A (2019) A hybrid estimation of distribution algorithm for the vehicle routing problem with time windows. Comput. & Ind. Eng. 130:75–96

Rechenberg I (1973) Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution. JSTOR

Rivin I, Vardi I, Zimmermann P (1994) The n-queens problem. Am. Math. Mon. 101(7):629–639

Sahani M, Linden J (2002) Evidence optimization techniques for estimating stimulus-response functions. Advances in Neural Information Processing Systems 15:317–324

Sebag M, Ducoulombier A (1998) Extending population-based incremental learning to continuous search spaces. In: Proceedings of the 5th International Conference on Parallel Problem Solving from Nature, Springer, p 418–427

Shen Y, Li J, Peng K (2017) An estimation of distribution algorithm for public transport driver scheduling. Int. J. Oper. Res. 28(2):245–262

Sun BQ, Wang L, Peng ZP (2020) Bound-guided hybrid estimation of distribution algorithm for energy-efficient robotic assembly line balancing. Comput. & Ind. Eng. 146:106604

Talbi EG (2009) Metaheuristics: From Design to Implementation, vol 74. John Wiley & Sons

Utamima A, Reiners T, Ansaripoor AH (2019) Evolutionary estimation of distribution algorithm for agricultural routing planning in field logistics. Procedia Comput. Sci. 161:560–567

Verma T, Pearl J (1990) Equivalence and synthesis of causal models. In: Proceedings of the 6th Annual Conference on Uncertainty in Artificial Intelligence, Elsevier, UAI '90, p 255–270

Wang X, Li X (2018) Estimation of distribution algorithm for initial orbit determination of too-short-arc based on kernel density estimation. Chin. Astron. Astrophys. 42:141–154