Multi–objective optimization with diversity preserving mixture–based iterated density estimation evolutionary algorithms

Peter A.N. Bosman, Dirk Thierens

Institute of Information and Computing Sciences, Utrecht University
P.O. Box 80.089, 3508 TB Utrecht, The Netherlands

Abstract

Stochastic optimization by learning and using probabilistic models has received an increasing amount of attention over the last few years. Algorithms within this field estimate the probability distribution of a selection of the available solutions and subsequently draw more samples from the estimated probability distribution. The resulting algorithms have displayed a good performance on a wide variety of single–objective optimization problems, both for binary as well as for real–valued variables. Mixture distributions offer a powerful tool for modeling complicated dependencies between the problem variables. Moreover, they allow for elegant and parallel exploration of a multi–objective front. This parallel exploration aids the important preservation of diversity in multi–objective optimization. In this paper, we propose a new algorithm for evolutionary multi–objective optimization by learning and using probabilistic mixture distributions. We name this algorithm MIDEA (Multi–objective Mixture–based Iterated Density Estimation Evolutionary Algorithm). To further improve and maintain the diversity that is obtained by the mixture distribution, we use a specialized diversity preserving selection operator. We verify the effectiveness of our approach in two different problem domains and compare it with two other well–known efficient multi–objective EAs.

Key words: Multi–objective evolutionary algorithms, Probabilistic model learning, Numerical optimization, Combinatorial optimization

Email addresses: Peter.Bosman@cs.uu.nl (Peter A.N. Bosman), Dirk.Thierens@cs.uu.nl (Dirk Thierens).

1 Introduction

Evolutionary algorithms (EAs) are commonly applied to black box optimization in which we have no prior knowledge of the problem structure. The only available mechanism is the grading of solutions. Assuming that the underlying problem within the black box is structured, finding and using this structure is a more preferable way to traverse the search space than is a random search. In EAs, such an inductive search is guided by a collection of available samples, often called a population, which are solutions to the optimization problem. The solutions are combined in order to perform induction and generate new solutions that will hopefully be closer to the optimum solution. As this process is iterated, convergence is intended to lead the algorithm to a final solution.

Genetic algorithms (GAs) [25, 30] and many variants thereof, combine the solution information in a selection of the available samples. Often this is done by exchanging values for variables in a recombination phase, after which certain values are adapted in a mutation phase. Alternatively, the selection of samples can be seen as being representative of some probability distribution. Estimating this probability distribution and drawing more samples from it, is a more global statistical inductive type of iterated search. Algorithms that use such techniques have obtained an increasing amount of attention over the last few years, obtaining promising results on a large variety of problems [32].

The estimated probability distribution can contain *dependencies* between the problem variables. These dependencies represent some structural aspects of the optimization problem. By using this information, more efficient optimization can be achieved. This is related to the notion of *linkage* in GAs. In binary representation, linkage refers to the structural cohesion of the bits in the coding string with respect to the search space. Such linkage arises for instance when a group of bit positions together code a non–linear contribution to the overall solution quality (fitness) of a binary string. Processing these bits together in a single *building block* is known to significantly aid GA optimization. More precisely, ignoring the need for linkage processing leads to exponential growth in population size requirements for solving deceptive problems that contain these non–linear fitness contributing building blocks [47]. Such linkage information can be captured and processed efficiently with probabilistic models [28, 35, 39, 43]. Although the notion of linkage and problem structure is different for real–valued continuous problem variables, good results have been obtained using real–valued continuous probabilistic models as well [4, 12, 48].

In this paper, we apply the use of probabilistic models in EAs to multi–objective optimization. In previous work [8], we have given a general algorithmic framework for using probabilistic model learning in evolutionary search. This framework is named IDEA (*Iterated Density Estimation Evolutionary*

Algorithm). In this paper, we show how we can make efficient instances of this framework for multi–objective optimization. We call such instances MIDEA (*Multi–objective Mixture–based* IDEA) variants. We will show how MIDEA instances can be implemented for both binary as well as real problem variables.

The remainder of this paper is organized as follows. In Section 2, we first present some definitions for multi–objective optimization. In Section 3, we give a definition of probabilistic models and present a way in which they can be learned from a set of samples. Furthermore, we show how this can be used iteratively for optimization, resulting in a definition of the IDEA framework. In Section 4 we show how we can instantiate the IDEA framework to get efficient algorithms for multi–objective optimization. In Section 5 we validate the performance of MIDEAs on eight test problems and compare the results with two other state–of–the–art multi–objective evolutionary algorithms (MOEAs). We discuss our findings in Section 6 and present our conclusions in Section 7.

2 Multi–objective optimization

Multi–objective optimization differs from single–objective optimization in that we have a multiple of *objectives* that we wish to optimize simultaneously without an expression of weight or preference for any of the objectives. Often, these multiple objectives are conflicting. Such problems naturally arise in many real world situations. An example of conflicting objectives that often arises in industry, is when we want to minimize the costs of some production process while at the same time we also want to minimize the pollution caused by the same production process. Such conflicting objectives give rise to a key characteristic of multi–objective optimization problems, which is the existence of sets of solutions that cannot be ordered in terms of preference when only considering the objective function values simultaneously. To formalize this notion, four relevant concepts exist. Assuming that we have m objectives $f_i(\boldsymbol{x})$, $i \in \boldsymbol{\mathcal{M}} = \{0, 1, \ldots, m-1\}$, that, without loss of generality, we seek to minimize, these four concepts can be defined as follows:

(1) **Pareto dominance**
A solution $\boldsymbol{x}$ is said to (Pareto) *dominate* a solution $\boldsymbol{y}$ (denoted $\boldsymbol{x} \succ \boldsymbol{y}$) **iff** $(\forall i \in \boldsymbol{\mathcal{M}} : f_i(\boldsymbol{x}) \leq f_i(\boldsymbol{y})) \wedge (\exists i \in \boldsymbol{\mathcal{M}} : f_i(\boldsymbol{x}) < f_i(\boldsymbol{y}))$

(2) **Pareto optimal**
A solution $\boldsymbol{x}$ is said to be *Pareto optimal* **iff** $\neg\exists \boldsymbol{y} : \boldsymbol{y} \succ \boldsymbol{x}$

(3) **Pareto optimal set**
The set $\boldsymbol{\mathcal{P}_S}$ of all Pareto optimal solutions: $\boldsymbol{\mathcal{P}_S} = \{\boldsymbol{x} | \neg\exists \boldsymbol{y} : \boldsymbol{y} \succ \boldsymbol{x}\}$

(4) **Pareto optimal front**
The set $\boldsymbol{\mathcal{P}_F}$ of all objective function values corresponding to the solutions in $\boldsymbol{\mathcal{P}_S}$: $\boldsymbol{\mathcal{P}_F} = \{(f_0(\boldsymbol{x}), f_1(\boldsymbol{x}), \ldots, f_{m-1}(\boldsymbol{x})) | \boldsymbol{x} \in \boldsymbol{\mathcal{P}_S}\}$

The Pareto optimal set $\boldsymbol{\mathcal{P}_S}$ is a definition of all trade–off optimal solutions in the parameter space. The Pareto optimal front $\boldsymbol{\mathcal{P}_F}$ is the same set of solutions, only regarded in the objective space. The size of either set can be infinite, in which case it is impossible to find the optimal set or front with a finite number of solutions. Regardless of the size of $\boldsymbol{\mathcal{P}_S}$ or $\boldsymbol{\mathcal{P}_F}$, it is commonly accepted that we are interested in finding a good representation of these sets with a finite number of solutions. The definition of a good representation, is difficult however. The reason for this is that it is desirable to obtain a diverse set of solutions as well as it is desirable to obtain a front or set that is close to the optimal one. Furthermore, it depends on the mapping between the parameter space and the objective space whether a good spread of the solutions in the parameter space is also a good spread of the solutions in the objective space. However, it is common practice [16] to search for a good diversity of the solutions along the Pareto *front*. The reason for this is that a decision–maker will ultimately have to pick a single solution. Therefore, it is often best to present a wide variety of trade–off solutions for the specified goals.

The notion of searching a space by maintaining a population of solutions is characteristic of EAs, which makes them natural candidates for multi–objective optimization aiming to cover a good approximation of the Pareto optimal front. A strongly increasing amount of research has indeed been done in the field of evolutionary multi–objective optimization in recent years [16] with very promising results.

3 Probabilistic models and IDEAs

Before we describe the algorithmic framework IDEA, we first discuss some related work on probabilistic model learning in EAs in Section 3.1. We present the IDEA framework in Section 3.2 and discuss the techniques that we use for learning probabilistic models in Sections 3.3 and 3.4.

3.1 Related work

For binary variables, first attempts at estimating a probability distribution and subsequently sampling new solutions from it, have lead to approaches in which a univariate factorization is used [1, 36]. In the probability distribution that is based on this factorization, all of the variables are processed independently of each other. Therefore, these approaches are not able to efficiently optimize problems that contain sets of interacting variables, just as is the case for the GA with uniform crossover [47]. To take interacting variables into account, Holland [30] already recognized that it would be beneficial if the GA

would exploit this so–called *linkage* information. Through other approaches, such as the mGA [27], the fmGA [26], the GEMGA [3], the LLGA [33] and the BBF–GA [49], the simple GA was extended to be able to process interactions between the problem variables. In the field of using probabilistic models, a first attempt to process the interactions resulted in algorithms that allow for pairwise interactions in the probability distribution [2, 5, 41]. However, regardless of how well a probability distribution based on pairwise interactions is estimated, it is not sufficient to efficiently optimize higher–order building block problems [6, 41]. To this end, we require the processing of higher–order multivariate interactions in the probability distribution. This can for instance be obtained by using a factorized probability distribution in which the factors can consist of multiple variables. Algorithms that allow for such probability distributions have subsequently been presented and tested successfully on various difficult problems with interacting variables [21, 28, 35, 40].

After multivariate interactions, mixture distributions have been used by several researchers. The mixture distributions can be obtained by clustering [38] or by using the expectation maximization algorithm [42, 43]. By using mixture distributions, a powerful, yet computationally tractable probability distribution is incorporated in EAs that is able to process complicated non–linear interactions between the problem variables. Furthermore, multi–modal optimization in which the goal is to obtain multiple optima in the fitness landscape, is also possible using mixture distributions. Another development is the use of decision graphs and niching [39]. Decision graphs provide an efficient means of expressing and storing multivariate interactions, whereas the combination with niching allows for solving very difficult hierarchical deceptive problems.

Next to approaches for binary spaces, real–valued continuous probabilistic models have been used as well. In most cases, the normal probability density function (pdf) is used. Similar to the binary approaches, the univariate factorization was used at first [24, 45]. The univariate factorization was also applied in combination with the normal mixture pdf [24] as well as a histogram pdf [7, 48]. It has been shown using the normal pdf, that capturing multivariate interactions between the problem variables in the probability distribution can be beneficial for real–valued variables as well [4, 8]. Mixture distributions have also been used in real–valued continuous probabilistic models, resulting in more efficient optimization of problems with non–linear interactions [11].

All of the former algorithms have been regarded for single–objective optimization. The only attempt so far to apply the EAs that learn and use probabilistic models to multi–objective optimization was a pilot study in which the IDEA framework with mixture distributions was used [46]. It was shown that good results can already be obtained by using mixture distributions as they are a natural means to preserve and promote diversity. However, there was no additional special mechanism for maintaining this diversity. This leads to larger

requirements on the population size to ensure proper convergence and a good termination criterion to ensure that diversity is not reduced or lost. In this paper, we improve this algorithm and show how learning and using probabilistic models in EAs can lead to efficient multi–objective optimization.

3.2 Definition of probabilistic models and the IDEA *framework*

The IDEA is a framework that uses probabilistic models in optimization. Before we describe how probabilistic models are used, we first introduce some notation and definitions. We take the elementary building block of probabilistic models to be the pdf which we can use to express multivariate joint probability densities with. We can now define a probabilistic model $\mathcal{M}$ to consist of some structure ς that describes a composition of pdfs, and a set of parameters $\boldsymbol{\theta}$ for the pdfs implied by ς, $\mathcal{M} = (\varsigma, \boldsymbol{\theta})$. The form of the multivariate joint pdf to fit over every factor implied by ς is chosen beforehand, such as the normal pdf. The way in which the parameters $\boldsymbol{\theta}$ are fit, is also predefined. We denote the parameter set that is obtained in this manner by $\boldsymbol{\theta} \xleftarrow{fit} \varsigma$. With these assumptions, the structure ς is sufficient to identify the probability distribution. Therefore, we denote the resulting probability distribution by P_ς.

We assume that we have an l–dimensional, single–objective optimization problem $f(y_0, y_1, \ldots, y_{l-1})$ which, without loss of generality, we seek to minimize. A population of n samples of dimensionality l is maintained. If we have no prior information on the problem that we wish to minimize, we might as well generate n random samples. In each iteration t, we select $\lfloor \tau n \rfloor$ samples ($\tau \in [\frac{1}{n}; 1]$) and let Υ_t be the *worst* selected sample fitness. Ideally, this selection of points is representative of the complete fitness landscape that lies below Υ_t or at least the attractors that we are interested in. For each problem variable y_i, we introduce a random variable Y_i. We then estimate the distribution of the selected samples and thereby find $\hat{P}_\varsigma^{\Upsilon_t}(\boldsymbol{\mathcal{Y}}) = \hat{P}_\varsigma^{\Upsilon_t}(Y_0, Y_1, \ldots, Y_{l-1})$ as an approximation to the true uniform distribution $P^{\Upsilon_t}(\boldsymbol{\mathcal{Y}})$ over all points $\boldsymbol{y}$ with $f(\boldsymbol{y}) \leq \Upsilon_t$. Next, new samples are drawn from $\hat{P}_\varsigma^{\Upsilon_t}(\boldsymbol{\mathcal{Y}})$ to further explore the part of the search space that we are interested in. Finally, some of the new samples are used to replace some of the current samples in the population. By iterating selection, density estimation and sample generation, we obtain the *Iterated Density Estimation Evolutionary Algorithm* (IDEA) [8].

If we select by taking the best $\lfloor \tau n \rfloor$ samples, draw $n - \lfloor \tau n \rfloor$ new samples from $\hat{P}_\varsigma^{\Upsilon_t}(\boldsymbol{\mathcal{Y}})$, and finally replace the worst $n - \lfloor \tau n \rfloor$ samples in the population with these new samples, we have that $\Upsilon_{t+1} = \Upsilon_t - \varepsilon$ with $\varepsilon \geq 0$. This assures that $\Upsilon_0 \geq \Upsilon_1 \geq \ldots \geq \Upsilon_{t_{\text{end}}}$. We call an IDEA so constructed *monotonic*. Since the best solutions survive every generational step, this approach is also called *elitist* in EA terminology.

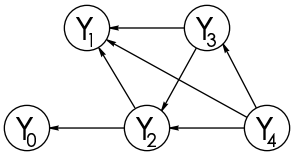

Fig. 1. A Bayesian factorization graph that represents the probability distribution $P_{\boldsymbol{\pi}}(\boldsymbol{\mathcal{Y}}) = P(Y_0|Y_2)P(Y_1|Y_2,Y_3,Y_4)P(Y_2|Y_3,Y_4)P(Y_3|Y_4)P(Y_4)$.

3.3 Probabilistic model selection

At some stage in each generation in the IDEA, we have to search for a structure ς. This search is commonly known as *model selection*. One example of a class of structures that we can use, is the class of *factorizations*. A factorization *factors* the probability distribution over $\boldsymbol{\mathcal{Y}}$ into a product of pdfs. In this paper, we focus on *Bayesian factorizations* and *Bayesian factorization mixtures*. The use of such structures has resulted in successful and promising applications of IDEAs [8, 11, 12]. In the following subsections, we describe the properties of these structures along with algorithms to derive them from sample data.

3.3.1 Bayesian factorizations

A Bayesian factorized probability distribution, or Bayesian factorization for short, is a product of conditional pdfs $P(Y_i|Y_{\boldsymbol{\pi}_i}) = P(Y_{\{i\}\cup\boldsymbol{\pi}_i})/P(Y_{\boldsymbol{\pi}_i})$. The structure of a Bayesian factorization is given by the parent vector of length l, $\boldsymbol{\pi} = (\boldsymbol{\pi}_0, \boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_{l-1})$, that indicates for each variable index i, a set of variable indices $\boldsymbol{\pi}_i \subseteq \{0,1,\ldots,l-1\} - \{i\}$ that variable Y_i is conditioned on in the Bayesian factorization. The Bayesian factorized probability distribution over all random variables is given by $P_{\boldsymbol{\pi}}(\boldsymbol{\mathcal{Y}}) = \prod_{i=0}^{l-1} P(Y_i|Y_{\boldsymbol{\pi}_i})$. The information that we need to identify the structure of the probabilistic model with, is in this case thus given by $\varsigma = \boldsymbol{\pi}$. By identifying a vertex with each variable Y_i and an arc $Y_i \rightarrow Y_j$ if and only if Y_j is conditioned on Y_i ($i \in \boldsymbol{\pi}_j$), we get the Bayesian factorization *graph*. A Bayesian factorization is valid if and only if its Bayesian factorization graph is *acyclic*. Such representations of probability distributions are also called *graphical models* in literature [19]. An example of a Bayesian factorization and its associated factorization graph is given in Figure 1.

Bayesian factorizations are capable of modelling various types of dependencies between the random variables [37], that express whether the probability at drawing a new value for a certain variable is dependent on the values already obtained for other variables. This allows for capturing and respecting of important relations between variables when generating a new collection of solutions after estimating a probability distribution over a selected collection of solutions in IDEAs. As a result, the structure of the optimization problem

can be exploited better than when the variables are regarded independently of each other, which leads to more efficient optimization with respect to the required number of times a solutions needs to be evaluated [21, 35, 40].

The actual types of dependency that we can express and process using Bayesian factorizations, depends on the pdf that is used to estimate the factors $P(Y_i|Y_{\boldsymbol{\pi}_i})$. In the case of a normal pdf, Bayesian factorizations allow to express linear dependencies. In the case of binary variables, the standard frequency counting pdf is capable of modelling both linear as well as non–linear interactions between the problem variables. For instance, assume that we have a function of six binary variables $f(x_0, x_1, \ldots, x_5) = g(h(x_0, x_1), h(x_2, x_3), h(x_4, x_5))$. If $g(y_0, y_1, y_2) = y_0 + y_1 + y_2$, we get an additively decomposable function. If we use Bayesian factorizations, such problems can be effectively solved in terms of the minimum requirement on the population size and the number of function evaluations, regardless of the behavior of h [21, 35, 40]. However, if g and h are both non–linear, we cannot effectively process the non–linear *hierarchical* interactions by only using Bayesian factorizations. One way to overcome this problem is to use *mixtures* in combination with local structures such as decision graphs, that are capable of efficiently representing dependencies between the actual values for sets of variables [23, 39].

A general greedy way to learn probabilistic models from data can be obtained by assuming that we have some current probabilistic model $\mathcal{M}_0$. Furthermore, we have a *metric* that evaluates the goodness of a model. In most metrics, a better model has a *lower* score. To get a greedy algorithm, a set of operations is given that can be performed on $\mathcal{M}_0$ to get a set of new candidate models. From this set, the model with the lowest score is selected. If its score is lower than that of $\mathcal{M}_0$, it replaces $\mathcal{M}_0$. This procedure is repeated until no improvement can be made any further. A metric that has often proved to be successful in iterated density estimation approaches, is commonly known as the *Bayesian Information Criterion* (BIC) [11, 44]. This metric should be minimized. It scores a factorization by its negative log–likelihood, but adds a penalty term that increases with the complexity of the model and the size of the sample set. The complexity of the model is expressed by the number of parameters $|\boldsymbol{\theta}|$ that need to be estimated. Let $\boldsymbol{\mathcal{S}} = \{\boldsymbol{y^0}, \boldsymbol{y^1}, \ldots, \boldsymbol{y^{|\mathcal{S}|-1}}\}$ be the selected set of l–dimensional samples. The BIC metric for a certain probabilistic model $\mathcal{M}$ and a sample set $\boldsymbol{\mathcal{S}}$, can be formalized as follows:

$$BIC(\mathcal{M}, \boldsymbol{\mathcal{S}}) = \underbrace{-\sum_{i=0}^{|\boldsymbol{\mathcal{S}}|-1} \ln\left(\hat{P}_{\mathcal{M}}(\boldsymbol{\mathcal{Y}})(\boldsymbol{y^i})\right)}_{\text{Error}(\hat{P}_{\mathcal{M}}(\boldsymbol{\mathcal{Y}})|\boldsymbol{\mathcal{S}})} + \underbrace{\frac{1}{2}\ln(|\boldsymbol{\mathcal{S}}|)|\boldsymbol{\theta}|}_{\text{Complexity}(\hat{P}_{\mathcal{M}}(\boldsymbol{\mathcal{Y}})|\boldsymbol{\mathcal{S}})} \quad (1)$$

The greedy approach that we take, starts from a graph with no arcs, which represents the univariate factorization. The only graph operation that we allow is the addition of an arc such that no cycles are introduced.

In the BIC metric in equation 1, we are required to evaluate the negative log–likelihood. According to the definition, we can do this by evaluating the model at each sample point and sum the logarithms of these model evaluations. However, this requires more time as the size of the sample set $\boldsymbol{\mathcal{S}}$ goes up. It can be shown that when we estimate the model parameters $\boldsymbol{\theta} \xleftarrow{fit} \varsigma$ with a maximum likelihood, the *entropy* $h(P_{\boldsymbol{\mathcal{M}}}(\boldsymbol{\mathcal{Y}})) = \int P_{\boldsymbol{\mathcal{M}}}(\boldsymbol{\mathcal{Y}}) \ln(P_{\boldsymbol{\mathcal{M}}}(\boldsymbol{\mathcal{Y}}))$ equals the *average* negative log–likelihood [10, 31], $|\boldsymbol{\mathcal{S}}|h(P(\boldsymbol{\mathcal{Y}})) = -\sum_{i=0}^{|\boldsymbol{\mathcal{S}}|-1} \ln\left(\hat{P}_{\mathcal{M}}(\boldsymbol{\mathcal{Y}})(\boldsymbol{y^i})\right)$. Evaluating the entropy can sometimes be done more efficiently than evaluating the negative log–likelihood. Moreover, when we only allow the addition of an arc $v_0 \rightarrow v_1$ to the factorization graph, it can be shown that the negative log–likelihood difference can be expressed as the difference in entropy for the factor $\hat{P}(Y_{v_1}|Y_{\boldsymbol{\pi}_{v_1}})$ before and after the arc addition [9, 14].

The running time complexity of the greedy algorithm is $\mathcal{O}(l^3\kappa + |\boldsymbol{\theta}|)$, where κ is the maximum number of parents for any variable. Although leaving κ unrestricted does allow for capturing higher–order interactions that are important for efficient optimization of higher–order building block problems, it is also a computationally intensive approach. At the interaction level of $\kappa \leq 1$, only second order interactions between the problem variables can be processed efficiently with respect to the minimum requirement on the population size and the number of function evaluations. However, specialized algorithms for finding an optimal Bayesian factorization exist that run in $\mathcal{O}(l^2)$ time. As we do not know in advance what degree of interaction a certain problem has, we can therefore argue to first try modelling only lower order dependencies before we apply more advanced algorithms to model higher order dependencies.

One algorithm that finds a model at the interaction level of $\kappa \leq 1$, is the optimal dependency tree approach by Chow and Liu [15]. This algorithm computes a Bayesian factorization such that its directed graph is a tree. This algorithm was first used by Baluja and Davies in a probabilistic model learning EA [2]. Moreover, learning a *mixture* of trees has been shown to be a very powerful probability distribution estimation technique [34, 43].

3.3.2 Factorization mixtures

The structure of the sample set may be highly non–linear, forcing us to use probabilistic models of a high complexity to retain some of this non–linearity. However, especially using relatively simple pdfs such as the normal pdf, the non–linear interactions cannot be captured even with higher–order factorizations. The use of clusters allows us to break up non–linear interactions so that we can use simple models to get an adequate representation of the sample set. An example of this is depicted in Figure 2. Furthermore, computationally efficient clustering algorithms exist that provide useful results [9, 29].

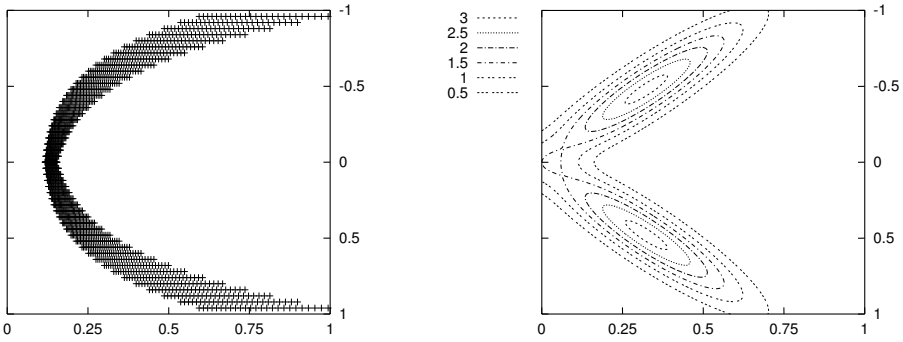

Fig. 2. A non–linear dependency in the sample set (left) and the contour lines of the estimated probability distribution using two normal pdfs after clustering (right).

We can estimate a factorized probability distribution in each cluster. By doing so, we obtain a *factorization mixture*. We let k be the number of clusters and let $\boldsymbol{\mathcal{K}} = \{0, 1, \ldots, k-1\}$. For a mixture of Bayesian factorizations, we write $\boldsymbol{\pi}^{\boldsymbol{\mathcal{K}}} = (\boldsymbol{\pi}^{\mathbf{0}}, \boldsymbol{\pi}^{\mathbf{1}}, \ldots, \boldsymbol{\pi}^{\boldsymbol{k-1}})$. The resulting probability distribution is a weighted sum of the individual factorized probability distributions over each cluster:

$$\hat{P}_{\boldsymbol{\pi}^{\boldsymbol{\mathcal{K}}}}(\boldsymbol{\mathcal{Y}}) = \sum_{i=0}^{k-1} \beta_i \hat{P}_{\boldsymbol{\pi}^{\boldsymbol{i}}}(\boldsymbol{\mathcal{Y}}) \qquad \forall i \in \boldsymbol{\mathcal{K}} : \beta_i \geq 0, \quad \sum_{i=0}^{k-1} \beta_i = 1 \tag{2}$$

The mixing coefficients β_i can be set in different ways. For function optimization, β_i is usually set to the proportion of the size of cluster i with respect to $|\boldsymbol{\mathcal{S}}|$. By proportionally assigning a larger value to β_i for clusters with a better average fitness, niching can be introduced elegantly into probabilistic model learning EAs [38]. In the case of multi–objective optimization, we propose to assign each cluster an equally large mixing coefficient, $\beta_i = 1/k$. The reason for this is that we want to distribute the points as good as possible along the Pareto front. Since it gives each cluster an equal probability at producing new points, it maximizes the parallel exploration along the Pareto front.

Clustering the sample set and subsequently estimating a factorization in each cluster, is a fast way to constructing mixture probability distributions through which dependencies in different parts of the sample set can be expressed. The drawback of this approach is that from a probabilistic viewpoint, the resulting probability distribution estimation is almost certain not to be a maximum likelihood estimate. However, the use of clustering does allow the modelling of non–linear dependencies between the variables. Furthermore, the EM al-

gorithm is highly time–consuming, even if the structure $\boldsymbol{\pi}^{\boldsymbol{\kappa}}$ is fixed for each mixture component. In a clustering–based approach, we can still use the greedy incremental penalization metric algorithm to estimate a factorization in each cluster and thereby model linear relations in each cluster.

To stimulate diversity preservation and exploration along the Pareto *front*, we use clustering by computing distances in the *objective* space. In order to perform clustering, we use the randomized leader algorithm. This algorithm has been observed to be a fast and flexible adaptive clustering algorithm that gives useful results in IDEAs [11]. The first sample to make a new cluster is appointed to be its leader. The algorithm goes over the sample set only once. For each sample, it finds the cluster with the nearest leader. If this leader is nearer than a given threshold $\mathfrak{T}_d$, the sample is added to that cluster. Otherwise, a new cluster is created with this single sample as its leader. To prevent the first clusters to become much larger than the ones that were constructed later, the order in which the clusters are scanned, is randomized.

3.4 Probabilistic model fitting

Once a probabilistic model structure ς has been selected, the parameters for the pdfs have to be estimated. Note that the estimation of the parameters is required after model selection *only* if the probabilistic model structure was given by some oracle. This occurs for instance when we specify in advance that we only want to use the univariate factorization. Whenever we are learning a structure however, we will be required to compute the pdf parameters since we must evaluate the penalized likelihood of the model. In our experiments, we have used maximum likelihood estimations to obtain $\boldsymbol{\theta}$. In the case of real variables we used the normal pdf [9]. For binary variables, a maximum likelihood estimation is obtained by computing average frequencies.

4 $\mathbb{M}$IDEA: Multi–objective mixture–based IDEA

To obtain IDEA instances that are suitable for multi–objective optimization, we propose to instantiate two steps in the framework. Firstly, we partially instantiate the search for probabilistic models by enforcing mixture distributions by means of clustering as described in Section 3.3.2. Secondly, to stimulate the preservation of diversity along the Pareto front, we instantiate the selection mechanism by using a diversity preserving truncation selection operator.

The selection operator that we propose first computes the domination count of all individuals. The domination count of a solution equals the number of

times the solution is dominated by other solutions in the population [22]. Subsequently, a pre–selection $\boldsymbol{\mathcal{S}}^P$ is made of $\lfloor \delta\tau n \rfloor$ individuals ($\delta \in [1; \frac{1}{\tau}]$) by using truncation selection on the domination count (select the best $\lfloor \delta\tau n \rfloor$ solutions). However, if the solution with the largest domination count to end up in $\boldsymbol{\mathcal{S}}^P$ by truncation selection has a domination count of 0, *all* individuals with a domination count of 0 are selected instead, resulting in $|\boldsymbol{\mathcal{S}}^P| \geq \lfloor \delta\tau n \rfloor$. This ensures that once the search starts to converge onto a certain Pareto front, we enforce diversity over all of the available solutions on the front. The final selection $\boldsymbol{\mathcal{S}}$ is obtained from $\boldsymbol{\mathcal{S}}^P$ by using a nearest neighbor heuristic to promote diversity. First, an individual with a maximum value for objective 0 is deleted from $\boldsymbol{\mathcal{S}}^P$ and added to $\boldsymbol{\mathcal{S}}$. Note that the choice for maximality in objective 0 is arbitrary as the key is to find a diverse selection of solutions. To stimulate this, we can select a solution that is maximal or minimal along any objective. For all solutions in $\boldsymbol{\mathcal{S}}^P$, the nearest neighbor distance is computed to the single solution in $\boldsymbol{\mathcal{S}}$. The distance that we use is the Euclidean distance scaled to the sample range in each objective. The solution in $\boldsymbol{\mathcal{S}}^P$ with the *largest* distance is then deleted from $\boldsymbol{\mathcal{S}}^P$ and added to $\boldsymbol{\mathcal{S}}$. The distances in $\boldsymbol{\mathcal{S}}^P$ are updated by investigating whether the distance to the newly added point in $\boldsymbol{\mathcal{S}}$ is smaller than the currently stored distance. These last two steps are repeated until $\lfloor \tau n \rfloor$ solutions are in the final selection. This selection operator has a running time complexity of $\mathcal{O}(n^2)$. This is no worse than the minimum of $\mathcal{O}(n^2)$ for computing the domination counts which is required in all MOEAs. The monotonic $\mathbb{M}$IDEA variant that we use in our experiments can be written in pseudo–code as follows:

$\mathbb{M}$IDEA (*monotonic instance*)
1 Initialize a population of n random solutions and evaluate their objectives 2 Iterate until termination 2.1 Compute the domination counts 2.2 Select solutions with the diversity preserving selection operator 2.3 Search for a (mixture) structure $\varsigma = \pi_{\boldsymbol{\mathcal{K}}}$ 2.4 Replace the non–selected solutions with new solutions drawn from $\hat{P}_\varsigma(\boldsymbol{\mathcal{Y}})$ 2.5 Evaluate the objectives of the new solutions

5 Experiments

We compare $\mathbb{M}$IDEA to two well–known state–of–the–art MOEAs that aim at obtaining a diverse set of solutions along the Pareto front. The SPEA algorithm by Zitzler and Thiele [51] and the NSGA-II algorithm by Deb et al. [17] showed superior performance compared to most other MOEAs [17, 50].

To test the algorithms, we use a test suite consisting of eight problems. We varied the dimensionality of these problems in order to get a total of sixteen problem instances to test the MOEAs on. In the following subsections, we describe our test problems and the obtained results.

5.1 Multi–objective optimization problems

Our test suite consists of problems over both real–valued variables as well as binary variables. In the case of MIDEA, this implies that we can use both the normal pdf as well as the discrete average frequency count. In our problem definitions, we denote a vector of l binary or real–valued problem variables by $\boldsymbol{x}$ and $\boldsymbol{y}$ respectively for short. The problems with real–valued variables are all defined for two objectives as shown in Figure 3.

We introduce function BT_1 into the field of multi–objective optimization. It differs from the other three functions in that it has explicit multivariate (linear) interactions between the problem variables. All of the other problems can therefore be adequately optimized by using a univariate factorization. The Pareto optimal front is given by $f_1(\boldsymbol{y}) = 1 - y_0$.

Functions ZDT_4 and ZDT_6 were introduced by Zitzler et al. [50]. It is very hard to obtain the optimal front $f_1(\boldsymbol{y}) = 1 - \sqrt{y_0}$ in ZDT_4 since there are many local fronts. The density of solutions in ZDT_6 increases as we move away from the Pareto optimal front. Furthermore, this function has a non–uniform density of solutions *along* the Pareto optimal front as there are more solutions as $f_0(\boldsymbol{y})$ goes up to 1. The Pareto optimal front is given by $f_1(\boldsymbol{y}) = 1 - f_0(\boldsymbol{y})^2$ with $f_0(\boldsymbol{y}) \in [1 - e^{-1/3}; 1]$. Finally, function CTP_7 was introduced by Deb et al. [18]. Its Pareto optimal front differs slightly from that of ZDT_4, but otherwise shares the multimodal front problem. In addition, this problem has constraints in the objective space, which makes finding a diverse representation of the Pareto front more difficult.

In Figure 4, we have specified four binary multi–objective optimization problems. Next to being binary, these problems are also multi–objective variants of well–known combinatorial optimization problems. The number of objectives for these problems is not restricted by two and is denoted by m.

In the maximum satisfiability problem, we are given a propositional formula in conjunctive normal form. The goal is to satisfy as many clauses as possible. The solution string is a truth assignment to the involved literals. These formulas can be represented by a matrix in which row i specifies what literals appear either positive (1) or negative (-1) in clause i. In the multi–objective variant of this problem, we have m of such matrices and only a single solution to satisfy as many clauses as possible in each objective at the same time.

Name	Definition	Range
BT_1	**Minimize** $(f_0(\boldsymbol{y}), f_1(\boldsymbol{y}))$ **Where** • $f_0(\boldsymbol{y}) = y_0$ • $f_1(\boldsymbol{y}) = 1 - f_0(\boldsymbol{y}) + 10^7 - \frac{100}{\left(10^{-5} + \sum_{i=1}^{l-1} \left\lvert \sum_{j=1}^{i} y_i \right\rvert\right)}$	• $y_0 \in [0; 1]$ • $y_i \in [-3; 3]$ $(1 \leq i < l)$
ZDT_4	**Minimize** $(f_0(\boldsymbol{y}), f_1(\boldsymbol{y}))$ **Where** • $f_0(\boldsymbol{y}) = y_0$ • $f_1(\boldsymbol{y}) = \gamma\left(1 - \sqrt{\frac{f_0(\boldsymbol{y})}{\gamma}}\right)$ • $\gamma = 1 + 10(l\text{–}1) + \sum_{i=1}^{l-1}\left(y_i^2 - 10\cos(4\pi y_i)\right)$	• $y_0 \in [0; 1]$ • $y_i \in [-5; 5]$ $(1 \leq i < l)$
ZDT_6	**Minimize** $(f_0(\boldsymbol{y}), f_1(\boldsymbol{y}))$ **Where** • $f_0(\boldsymbol{y}) = 1 - e^{-4y_0}\sin^6(6\pi y_0)$ • $f_1(\boldsymbol{y}) = \gamma\left(1 - \left(\frac{f_0(\boldsymbol{y})}{\gamma}\right)^2\right)$ • $\gamma = 1 + 9\left(\sum_{i=1}^{l-1} \frac{y_i}{9}\right)^{0.25}$	• $y_i \in [0; 1]$ $(0 \leq i < l)$
CTP_7	**Minimize** $(f_0(\boldsymbol{y}), f_1(\boldsymbol{y}))$ **Where** • $f_0(\boldsymbol{y}) = y_0$ • $f_1(\boldsymbol{y}) = \gamma\left(1 - \frac{f_0(\boldsymbol{y})}{\gamma}\right)$ • $\gamma = 1 + 10(l\text{–}1) + \sum_{i=1}^{l-1}\left(y_i^2 - 10\cos(4\pi y_i)\right)$ **Such That** • $\cos(-\frac{5\pi}{100})f_1(\boldsymbol{y}) - \sin(-\frac{5\pi}{100})f_0(\boldsymbol{y}) \geq 40\lvert\sin(5\pi\left[\sin(-\frac{5\pi}{100})f_1(\boldsymbol{y}) + \cos(-\frac{5\pi}{100})f_0(\boldsymbol{y})\right])\rvert^6$	• $y_0 \in [0; 1]$ • $y_i \in [-5; 5]$ $(1 \leq i < l)$

Fig. 3. Multi–objective real–valued continuous optimization test problems.

The multi–objective knapsack problem was first used to test MOEAs on by Zitzler and Thiele [51]. We are given m knapsacks with a specified capacity and n items. Each item can have a different weight and profit in every knapsack. Selecting item i in a solution implies placing it in every knapsack. A solution may not cause exceeding the capacity of any knapsack.

In the set covering problem, we are given l locations at which we can place some service at a specified cost. Furthermore, associated with each location is a set of regions $\subseteq \{0, 1, \ldots r-1\}$ that can be serviced from that location. The goal is to select locations such that *all* regions are serviced against minimal costs. In the multi–objective variant of set covering, m services are placed at a location. Each service however covers its own set of regions when placed at a certain location and has its own cost associated with a certain location. A

Name	Definition	
MS (Maximum Satisfiability)	**Maximize**	$(f_0(\boldsymbol{x}), f_1(\boldsymbol{x}), \ldots, f_{m-1}(\boldsymbol{x}))$
	Where	• $\forall i \in \boldsymbol{\mathcal{M}} : f_i(\boldsymbol{x}) = \sum_{j=0}^{c_i-1} \operatorname{sgn}\left(\left[\sum_{k=0}^{l-1} (C_i)_{jk} \otimes x_k\right]\right)$
		• $\operatorname{sgn}(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ -1 & \text{if } x < 0 \end{cases}$
		• $\begin{array}{c\|cc} \otimes & 0 & 1 \\ \hline -1 & 1 & 0 \end{array}$ $\begin{array}{c\|cc} \otimes & 0 & 1 \\ \hline 0 & 0 & 0 \end{array}$ $\begin{array}{c\|cc} \otimes & 0 & 1 \\ \hline 1 & 0 & 1 \end{array}$
KN (Knapsack)	**Maximize**	$(f_0(\boldsymbol{x}), f_1(\boldsymbol{x}), \ldots, f_{m-1}(\boldsymbol{x}))$
	Where	• $\forall i \in \boldsymbol{\mathcal{M}} : f_i(\boldsymbol{x}) = \sum_{j=0}^{l-1} P_{ij} x_j$
	Such That	• $\forall i \in \boldsymbol{\mathcal{M}} : \sum_{j=0}^{l-1} W_{ij} x_j \leq c_i$
SC (Set Covering)	**Minimize**	$(f_0(\boldsymbol{x}), f_1(\boldsymbol{x}), \ldots, f_{m-1}(\boldsymbol{x}))$
	Where	• $\forall i \in \boldsymbol{\mathcal{M}} : f_i(\boldsymbol{x}) = \sum_{j=0}^{l-1} C_{ij} x_j$
	Such That	• $\forall i \in \boldsymbol{\mathcal{M}} : \forall 0 \leq j < r : \sum_{k=0}^{l-1} (A_i)_{jk} x_k \geq 1$
MST (Minimal Spanning Tree)	**Minimize**	$(f_0(\boldsymbol{x}), f_1(\boldsymbol{x}), \ldots, f_{m-1}(\boldsymbol{x}))$
	Where	• $\forall i \in \boldsymbol{\mathcal{M}} : f_i(\boldsymbol{x}) = \sum_{j=0}^{l-1} W_{ij} x_j$
	Such That	• $\forall S \subseteq V : \sum_{x_j \in (S \times (V-S))} x_j \geq 1$
		• $\forall S \subseteq V : \sum_{x_j \in (S \times S)} x_j \leq \lvert S \rvert - 1$

Fig. 4. Multi–objective binary combinatorial optimization test problems.

binary solution indicates the selected locations at which to place all services. All regions must be covered by all services.

The last binary optimization problem that we use, is the minimal spanning tree problem. We are given an undirected graph (V, E) such that each edge has a certain weight associated with it. We are interested in selecting edges $E_T \subseteq E$ such that (V, E_T) is a spanning tree. The objective is to find a spanning tree such that the weight of all its edges is minimal. In the multi–objective variant of this problem, each edge can have a different weight in each objective.

It is important to note that we have used random instances for the combinatorial optimization problems. In the case of only a single objective, random instances may on average be easy for some combinatorial problems. However, in the case of multiple objectives, finding the Pareto front is usually much more difficult, even if the efficient algorithms are available for the single–objective case [20]. Therefore, the instances used in our test suite are not expected to be over–easy. Furthermore, the problems also serve to indicate differences between the different multi–objective algorithmic approaches other than the

fact that dependencies between problem variables can be exploited. This relative performance of the algorithms may be well observed using our proposed test–suite. On the other hand, the degree of interaction between the problem variables in randomly generated problem instances may not be too large, which may cause optimization algorithms that regard the problem variables independently of each other to be the most efficient.

5.2 Experiment setup and results

We tested two variants of every algorithm. For SPEA, we used uniform crossover (UX) and one–point crossover (1X) with a probability of 0.8. Bit flipping mutation was used in either case with a probability of 0.01. These settings were suggested elsewhere by the SPEA authors [50]. We allowed the archive size of SPEA to become as large as the population size. For the real problems, we encoded every variable with 30 bits. For NSGA–II, we used the same crossover and mutation operators and the same encoding for the real variables. For $\mathbb{M}$IDEA, we used the leader clustering algorithm in the objective space by selecting a distance threshold such that four clusters were constructed on average. If the number of clusters becomes too large, the requirements for the population size increases in order to facilitate proper model learning in each cluster. We do not suggest that the number of clusters we use is optimal, but it will serve to indicate the effectiveness of parallel exploration along the Pareto front as well as diversity preservation. In each cluster, we either used the univariate factorization or we learned a Bayesian factorization in the case of real–valued variables. However, in the case of 100 dimensional real–valued variable problems, we used the greedy learning strategy discussed in Section 3.3 and allowed only at most a single parent for any variable. In the case of binary variables, we used the optimal dependency algorithm by Chow and Liu to learn a tree factorization in each cluster. To further investigate the influence of the different components in the $\mathbb{M}$IDEA algorithm, we also performed tests in which only a single cluster is used. Furthermore, we also replaced the use of estimating probability distributions by the use of one–point crossover and uniform crossover with mutation as used in the SPEA and NSGA–II algorithms. In the case of clustering in combination with the use of crossover operators, restricted mating was employed in order to ensure clustered exploration along the front. In restricted mating crossover, an offspring is produced using two parent solutions that are picked from the same cluster. For the truncation percentile, we used the rule of thumb by Mühlenbein and Mahnig [35] and set τ to 0.3. Furthermore, we set the diversity preservation parameter to $\delta = 1.5$.

For the real–valued problems, we tested all algorithms with both $l = 10$ and $l = 100$ problem variables. In addition, we chose to allow $20 \cdot 10^3$ and $100 \cdot 10^3$ function evaluations respectively in any single run. Since we allowed for a max-

imum of function evaluations, there is a value for n at which the algorithm will perform best. For too large population sizes, the search will move towards a random search and for too small population sizes, there is not enough information to perform adequate model learning and induction. We therefore increased the population size (n) in steps of 25 to find the best results. To actually select the best population size, we selected the result with the lowest AFD score, which is a metric that is explained later in this Section. For the binary problems, we used test instances with $l = 100$ and $l = 1000$. We again allowed $20 \cdot 10^3$ and $100 \cdot 10^3$ function evaluations respectively. For the maximum satisfiability problem, we generated the test instances by generating 2500 clauses for $l = 100$ and 12500 clauses for $l = 1000$ with a random number of literals between 1 and 5. For the knapsack problem, we generated instances by generating random weights in $[1; 10]$ and random profits in $[1; 10]$. The capacity of a knapsack was set to half of the total weight of all the items, weighted according to that knapsack objective. For set covering, the costs were generated at random in $[1; 10]$. We used 250 regions and 2500 regions to be serviced for $l = 100$ and $l = 1000$ respectively. We varied the problem difficulty through the region–location adjacency relation. This relation was generated by making each location adjacent to 70 and 50 randomly selected regions for $l = 100$ and $l = 1000$ respectively. Finally, for the minimum spanning tree problem, we used full graphs with 105 edges (15 vertices) and 1035 edges (46 vertices). The dimensionality of these problems is therefore not precisely 100 and 1000. The weights of the edges were generated randomly in $[1; 10]$.

One last issue remains before we can test the algorithms. Some of the problems have constraints. To deal with them, we can use a repair mechanism to apply to infeasible solutions. Another approach is introduced by the notion of constraint–domination introduced by Deb et al. [18]. This notion allows for a general scheme to deal with constrained multi–objective problems. A solution $\boldsymbol{x}$ is said to *constraint–dominate* solution $\boldsymbol{y}$ if any of the following is true:

(1) Solution $\boldsymbol{x}$ is *feasible* and solution $\boldsymbol{y}$ is *infeasible*
(2) Solutions $\boldsymbol{x}$ and $\boldsymbol{y}$ are both *infeasible*, but $\boldsymbol{x}$ has a smaller overall constraint violation
(3) Solutions $\boldsymbol{x}$ and $\boldsymbol{y}$ are both *feasible* and $\boldsymbol{x} \succ \boldsymbol{y}$

In the above definition, the overall constraint violation is amount by which a constraint is violated, summed over all constraints. We have used this principle for problems CTP_7 and set covering. For knapsack and minimal spanning tree, we have used a repair mechanism. For the knapsack problem, an elegant repair mechanism was proposed in earlier MOEA research [51]. For the minimal spanning tree problem, the number of constraints grows exponentially with the problem size l. We therefore propose to use repair mechanisms for these two problems. For the knapsack problem, if a solution violates a constraint, the repair algorithm iteratively removes items until all constrains are satisfied.

The order in which the items are investigated, is determined by the maximum profit/weight ratio. The items with the lowest profit/weight ratio are removed first. For the minimal spanning tree problem, first the edges are removed from the currently constructed graph and they are sorted according to their weight. Next, they are added to the graph such that no cycles are introduced. This is done by only allowing edges to be introduced *between* the connected components in the graph. If after this phase, the number of connected components has not been reduced to 1, all edges between the connected components are regarded in increasing weight and again the connected components are merged until a single component is left.

We ran every algorithm 50 times on each problem. To compare the algorithm, we look at their average performance with respect to three different metrics. The most important one is the *average front distance* **AFD**. This metric is the average minimal Euclidean distance over all points in a given default front ($\mathcal{F}_D$) front to another front $\mathcal{F}$. In the case of the real–valued problems, we know the optimal front. The default front in this case consists of a uniformly sampled set of 5000 solutions along the Pareto optimal front. Since we do not know the Pareto optimal front for the binary optimization problems, we use the Pareto front over all results obtained by all algorithms as the default front. Because the distance is computed over all points in the default front instead of over all points in the front found in a certain run, this measure gives us a sense of how well each part of the optimal or best known front is covered on average. It also gives us a sense of distance to the optimal front.

$$\mathbf{AFD}(\mathcal{F}_D, \mathcal{F}) = \frac{1}{|\mathcal{F}_D|} \sum_{\boldsymbol{x} \in \mathcal{F}_D} \min \left\{ \sum_{i=0}^{m-1} (\boldsymbol{x}_i - \boldsymbol{y}_i)^2 \;\middle|\; \boldsymbol{y} \in \mathcal{F} \right\} \tag{3}$$

If two algorithms obtain a somewhat comparable AFD score, it is interesting to look at other properties of the obtained results. A second metric we use is the *front spread* **FS**. This metric gives a notion of the size of the objective space covered by the Pareto front. It is the maximum Euclidean distance inside the m–dimensional hypercube that is obtained by taking the maximum distance among the points in the front in each dimension:

$$\mathbf{FS}(\mathcal{F}) = \sqrt{\sum_{i=0}^{m-1} \max\{(\boldsymbol{x}_i - \boldsymbol{y}_i)^2 \mid (\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{F} \times \mathcal{F}\}} \tag{4}$$

The third and final metric is the *front occupation* **FO**, which is simply the number of points on the front:

$$\mathbf{FO}(\mathcal{F}) = |\mathcal{F}| \tag{5}$$

For each of the metrics, we computed their average and standard deviation over the 50 runs to get an assessment of their performance. The averages are tabulated in Figures 6 through 11. The best results are written in boldface. For each algorithm, the type of recombination is indicated as a superscript. The $\mathbb{M}$IDEA algorithms are indicated by a single $\mathbb{M}$ symbol. For all tested $\mathbb{M}$IDEA algorithms, the subscript indicates whether only a single cluster was used. Without a subscript, the leader algorithm was used in the objective space. The population sizes that led to the best performance, are tabulated in Figures 12 and 13. For an extensive documentation of the results, including the standard deviations, we refer the interested reader to a technical report [13]. Although the average behavior is the most interesting, the standard deviations are vital to determine whether the differences in the average behavior of the different algorithms are significant. To this end, we have performed Aspin–Welch–Satterthwaite (AWS) statistical hypothesis T–tests at a significance level of $\alpha = 0.05$. The AWS T–test is a statistical hypothesis test for the equality of means in which the equality of variances is not assumed. For each problem, we statistically verified for each pair of algorithms whether the average obtained metric values differ significantly. We assigned a value of 1 if an algorithm scored significantly better and a value of -1 if an algorithm scored significantly worse. We summed the so obtained matrices over all problems to get the statistically significant improvement matrices that are shown in figures 14 through 16. We also computed the sum for each algorithm of its significant improvement values over all other algorithms to indicate the summed relative statistically significant performance of the algorithms.

One of the things that stands out, is that the $\mathbb{M}$IDEA variants always perform best in the case in which the dimensionality of the problem is larger ($l = 100$ for the real–valued problems, $l = 1000$ for the binary problems). This is most likely due to the more powerful diversity preservation in $\mathbb{M}$IDEA. As the dimensionality of the problem goes up, the parameter search space becomes larger. In the case of the binary combinatorial problems, the number of solutions in the objective space becomes larger as well. Because of the enforced parallel exploration in $\mathbb{M}$IDEA and the good diversity preservation, better results are obtained on average as the dimensionality of the problem increases. In Figure 5 the Pareto fronts over 50 runs for all algorithms are plotted on one problem from each problem class and dimensionality. The better diversity preservation and proper distribution of the points along the front can be seen clearly for the problems of larger dimensionality. For the lower dimensionality problems, better diversity preservation can also be observed, which is most exemplified by the fact that $\mathbb{M}$IDEA obtains non–dominated solutions at the outer ends of the front for the knapsack problem with $l = 100$.

The fact that the use of mixtures by clustering the objective space allows for enhanced diversity exploration and preservation, can also be observed by the difference between the spread obtained by $\mathbb{M}$IDEA with crossover operators

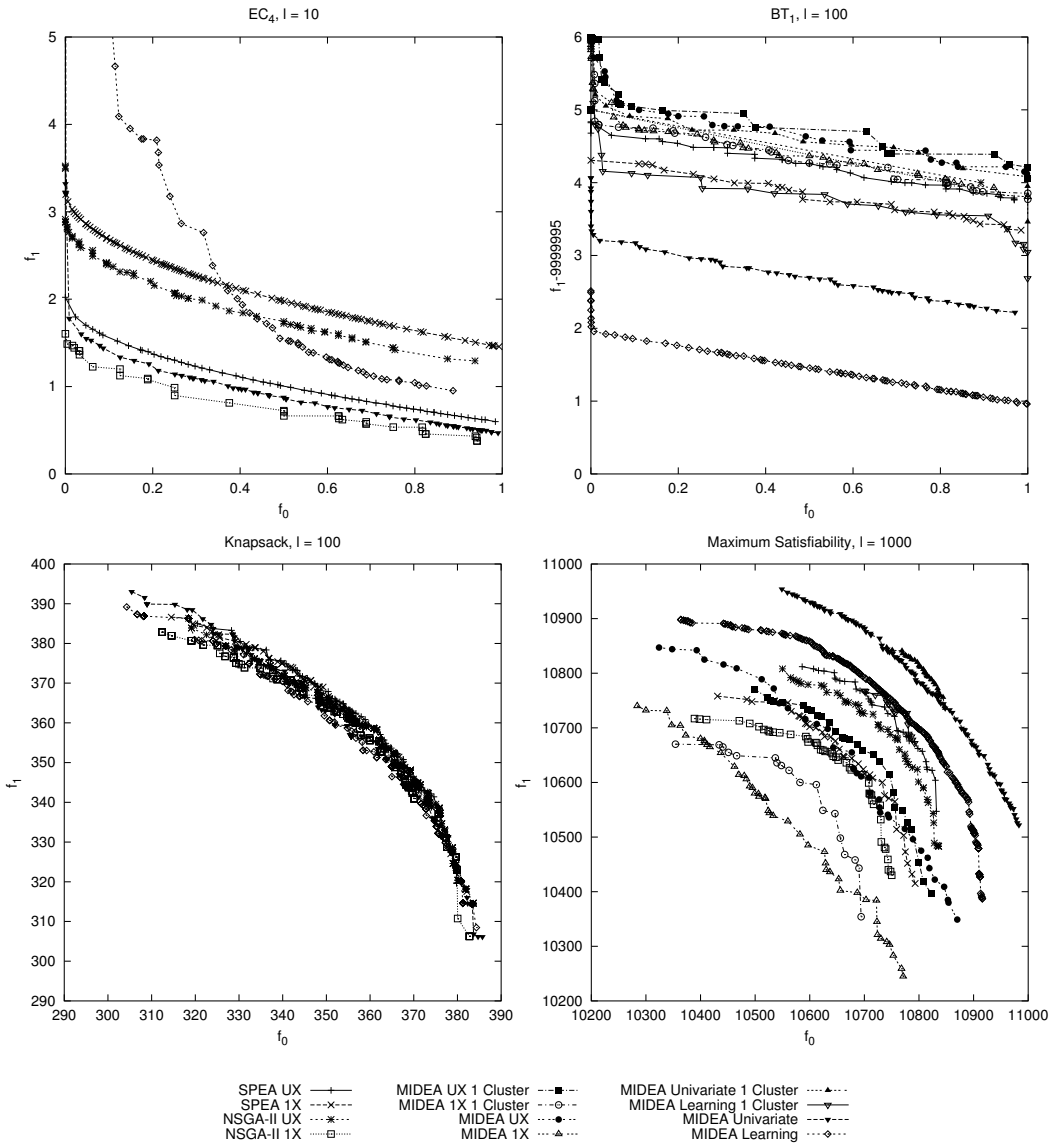

Fig. 5. Pareto fronts over 50 runs on a few of the tested problems. For the EC_4 problem in 10 dimensions and the knapsack problem in 100 dimensions, only the SPEA, NSGA-II and MIDEA with clustering and probabilistic model learning variants are shown. For the BT_1 problem in 100 dimensions and the maximum satisfiability problem in 1000 dimensions, the results for all algorithms are shown.

using only a single cluster versus the case in which on average four clusters are used. A wider spread of solutions is found when clustering is enabled.

On the BT_1 problem, modelling interactions in MIDEA leads to better results than those obtained by the other MOEAs. Thus, exploiting interactions can be beneficial in multi–objective optimization. For the BT_1 problem with $l = 10$, if we allow for $5 \cdot 10^5$ evaluations, the MIDEA variant that learns Bayesian factorizations is even capable of finding near optimal solutions whereas the other MOEAs were observed not to be able to produce comparable results.

Average Front Distance **AFD**								
EA	BT_1^{10}	ZDT_4^{10}	ZDT_6^{10}	CTP_7^{10}	BT_1^{100}	ZDT_4^{100}	ZDT_6^{100}	CTP_7^{100}
SPEA$^{\mathrm{UX}}$	$100{\cdot}10^5$	4.62	0.193	7.97	$100{\cdot}10^5$	470	7.64	499
SPEA$^{\mathrm{1X}}$	$100{\cdot}10^5$	3.90	0.172	7.31	$100{\cdot}10^5$	447	7.06	476
NSGA-II$^{\mathrm{UX}}$	$100{\cdot}10^5$	4.39	0.303	7.25	$100{\cdot}10^5$	360	5.99	348
NSGA-II$^{\mathrm{1X}}$	$100{\cdot}10^5$	**1.40**	0.328	**3.32**	$100{\cdot}10^5$	297	6.59	303
$\mathbb{M}_{1\text{ Cluster}}^{\mathrm{UX}}$	$100{\cdot}10^5$	4.43	0.358	6.63	$100{\cdot}10^5$	374	6.72	378
$\mathbb{M}_{1\text{ Cluster}}^{\mathrm{1X}}$	$100{\cdot}10^5$	1.89	0.291	4.13	$100{\cdot}10^5$	336	6.81	345
$\mathbb{M}^{\mathrm{UX}}$	$100{\cdot}10^5$	3.98	0.354	7.27	$100{\cdot}10^5$	311	5.96	326
$\mathbb{M}^{\mathrm{1X}}$	$100{\cdot}10^5$	2.03	0.311	3.95	$100{\cdot}10^5$	328	6.74	335
$\mathbb{M}_{1\text{ Cluster}}^{\mathrm{Univariate}}$	$100{\cdot}10^5$	14.0	1.08	16.5	$100{\cdot}10^5$	774	3.06	875
$\mathbb{M}_{1\text{ Cluster}}^{\mathrm{Learning}}$	$100{\cdot}10^5$	11.2	**0.00239**	15.3	$100{\cdot}10^5$	597	**0.434**	600
$\mathbb{M}^{\mathrm{Univariate}}$	$100{\cdot}10^5$	5.00	0.306	8.64	$100{\cdot}10^5$	157	4.60	**161**
$\mathbb{M}^{\mathrm{Learning}}$	$\mathbf{998{\cdot}10^4}$	11.5	0.287	12.6	$\mathbf{100{\cdot}10^5}$	**144**	1.30	165

Fig. 6. Average of the **AFD** metric on all real–valued problems.

Average Front Distance **AFD**								
EA	MS^{100}	KN^{100}	SC^{100}	MST^{105}	MS^{1000}	KN^{1000}	SC^{1000}	MST^{1035}
SPEA$^{\mathrm{UX}}$	6.10	5.14	2.93	1.31	147	67.3	442	3.39
SPEA$^{\mathrm{1X}}$	7.15	5.14	2.99	1.39	237	83.1	376	3.09
NSGA-II$^{\mathrm{UX}}$	**4.52**	**4.22**	**1.79**	1.09	147	56.1	185	3.55
NSGA-II$^{\mathrm{1X}}$	8.03	5.40	2.64	1.36	250	90.5	254	3.18
$\mathbb{M}_{1\text{ Cluster}}^{\mathrm{UX}}$	8.18	5.62	2.13	1.54	194	76.5	241	4.37
$\mathbb{M}_{1\text{ Cluster}}^{\mathrm{1X}}$	10.5	6.65	2.49	1.52	326	136	364	3.96
$\mathbb{M}^{\mathrm{UX}}$	9.01	5.01	2.12	**0.906**	213	59.3	254	4.63
$\mathbb{M}^{\mathrm{1X}}$	12.7	6.17	2.69	1.07	359	146	410	4.32
$\mathbb{M}_{1\text{ Cluster}}^{\mathrm{Univariate}}$	8.35	9.95	1.92	1.95	124	105	**85.9**	5.58
$\mathbb{M}_{1\text{ Cluster}}^{\mathrm{Learning}}$	16.5	6.52	2.39	1.82	147	73.1	136	2.90
$\mathbb{M}^{\mathrm{Univariate}}$	9.07	5.34	2.01	1.15	**26.5**	**22.3**	129	2.41
$\mathbb{M}^{\mathrm{Learning}}$	16.5	7.23	5.38	1.39	113	64.0	472	**2.13**

Fig. 7. Average of the **AFD** metric on all combinatorial problems.

Front Spread **FS**								
EA	BT_1^{10}	ZDT_4^{10}	ZDT_6^{10}	CTP_7^{10}	BT_1^{100}	ZDT_4^{100}	ZDT_6^{100}	CTP_7^{100}
SPEA$^{\mathrm{UX}}$	225	51.4	5.22	44.9	2.06	692	1.85	733
SPEA$^{\mathrm{1X}}$	369	55.8	5.26	46.3	2.31	736	3.02	773
NSGA-II$^{\mathrm{UX}}$	179	3.60	1.09	1.76	0.413	35.2	0.756	29.3
NSGA-II$^{\mathrm{1X}}$	23.4	8.93	1.03	1.31	1.02	33.4	0.665	13.9
$\mathbb{M}_{1\text{ Cluster}}^{\mathrm{UX}}$	655	8.55	2.90	39.1	2.18	395	3.43	365
$\mathbb{M}_{1\text{ Cluster}}^{\mathrm{1X}}$	78.6	2.46	1.92	1.41	2.27	94.0	1.40	88.6
$\mathbb{M}^{\mathrm{UX}}$	685	40.8	4.11	41.8	2.15	740	4.75	737
$\mathbb{M}^{\mathrm{1X}}$	262	3.38	3.94	58.9	2.29	359	2.30	371
$\mathbb{M}_{1\text{ Cluster}}^{\mathrm{Univariate}}$	293	70.8	1.15	84.7	1.82	393	0.180	347
$\mathbb{M}_{1\text{ Cluster}}^{\mathrm{Learning}}$	$129 \cdot 10^1$	84.9	3.00	87.4	2.12	635	2.20	342
$\mathbb{M}^{\mathrm{Univariate}}$	$209 \cdot 10^1$	90.4	**5.29**	114	2.45	636	**8.10**	619
$\mathbb{M}^{\mathrm{Learning}}$	$\mathbf{164 \cdot 10^2}$	**197**	3.68	**188**	**3.28**	$\mathbf{175 \cdot 10^1}$	3.97	$\mathbf{183 \cdot 10^1}$

Fig. 8. Average of the **FS** metric on all real–valued problems.

Front Spread **FS**								
EA	MS^{100}	KN^{100}	SC^{100}	MST^{105}	MS^{1000}	KN^{1000}	SC^{1000}	MST^{1035}
SPEA$^{\mathrm{UX}}$	109	69.5	**64.6**	30.6	288	254	631	52.1
SPEA$^{\mathrm{1X}}$	123	82.6	51.0	32.5	399	308	**636**	50.8
NSGA-II$^{\mathrm{UX}}$	110	71.8	16.1	26.3	370	288	144	33.7
NSGA-II$^{\mathrm{1X}}$	129	76.6	12.8	23.9	364	291	107	36.6
$\mathbb{M}_{1\text{ Cluster}}^{\mathrm{UX}}$	114	82.0	19.6	19.8	389	347	164	37.1
$\mathbb{M}_{1\text{ Cluster}}^{\mathrm{1X}}$	122	81.4	17.3	21.5	421	301	154	44.3
$\mathbb{M}^{\mathrm{UX}}$	173	114	21.3	**37.3**	**706**	**585**	319	**77.3**
$\mathbb{M}^{\mathrm{1X}}$	**169**	98.9	20.5	32.2	681	521	249	76.4
$\mathbb{M}_{1\text{ Cluster}}^{\mathrm{Univariate}}$	68.3	39.4	16.2	17.4	111	106	11.9	22.7
$\mathbb{M}_{1\text{ Cluster}}^{\mathrm{Learning}}$	123	83.8	18.4	17.8	153	199	142	36.9
$\mathbb{M}^{\mathrm{Univariate}}$	167	**108**	23.8	26.3	559	485	168	55.1
$\mathbb{M}^{\mathrm{Learning}}$	162	104	20.6	24.1	579	544	143	46.2

Fig. 9. Average of the **FS** metric on all combinatorial problems.

Front Occupation **FO**								
EA	BT_1^{10}	ZDT_4^{10}	ZDT_6^{10}	CTP_7^{10}	BT_1^{100}	ZDT_4^{100}	ZDT_6^{100}	CTP_7^{100}
SPEA$^{\mathrm{UX}}$	**60.9**	99.0	50.0	43.5	49.8	27.6	18.7	26.7
SPEA$^{\mathrm{1X}}$	38.7	**187**	49.6	43.2	48.8	27.4	29.3	26.8
NSGA-II$^{\mathrm{UX}}$	5.42	59.7	47.5	**59.3**	100	5.80	6.00	4.00
NSGA-II$^{\mathrm{1X}}$	29.5	32.7	31.2	9.98	75.0	5.00	6.60	3.00
$\mathbb{M}_{1\text{ Cluster}}^{\mathrm{UX}}$	9.92	41.7	8.06	9.00	14.4	12.8	14.4	12.6
$\mathbb{M}_{1\text{ Cluster}}^{\mathrm{1X}}$	13.4	30.3	6.52	11.9	16.5	7.10	6.64	5.94
$\mathbb{M}^{\mathrm{UX}}$	13.9	10.0	8.48	8.62	19.1	20.0	19.6	21.7
$\mathbb{M}^{\mathrm{1X}}$	9.94	31.4	7.32	15.6	17.4	12.2	9.76	12.2
$\mathbb{M}_{1\text{ Cluster}}^{\mathrm{Univariate}}$	5.74	6.88	4.90	4.14	36.7	6.9	2.55	3.20
$\mathbb{M}_{1\text{ Cluster}}^{\mathrm{Learning}}$	6.06	8.36	**258**	4.96	13.1	5.25	**369**	3.75
$\mathbb{M}^{\mathrm{Univariate}}$	12.5	68.7	56.3	34.0	64.5	**106**	27.7	**78.9**
$\mathbb{M}^{\mathrm{Learning}}$	30.1	26.4	197	32.1	**111**	50.8	163	43.0

Fig. 10. Average of the **FO** metric on all real–valued problems.

Front Occupation **FO**								
EA	MS^{100}	KN^{100}	SC^{100}	MST^{105}	MS^{1000}	KN^{1000}	SC^{1000}	MST^{1035}
SPEA$^{\mathrm{UX}}$	96.4	49.8	27.2	49.6	49.4	49.5	26.2	50.0
SPEA$^{\mathrm{1X}}$	96.4	87.6	27.7	124	49.9	49.7	26.5	98.6
NSGA-II$^{\mathrm{UX}}$	200	**175**	150	200	35.9	33.1	7.50	250
NSGA-II$^{\mathrm{1X}}$	99.9	**175**	**212**	200	42.9	37.0	7.20	249
$\mathbb{M}_{1\text{ Cluster}}^{\mathrm{UX}}$	32.2	18.4	7.80	63.8	17.6	16.6	4.40	14.4
$\mathbb{M}_{1\text{ Cluster}}^{\mathrm{1X}}$	41.9	28.7	10.3	62.2	18.9	15.3	5.70	21.8
$\mathbb{M}^{\mathrm{UX}}$	73.3	30.2	11.8	61.1	28.5	27.6	6.7	24.8
$\mathbb{M}^{\mathrm{1X}}$	71.3	62.9	17.2	352	34.4	18.5	5.90	27.2
$\mathbb{M}_{1\text{ Cluster}}^{\mathrm{Univariate}}$	111	14.3	10.2	105	39.3	16.4	28.4	15.7
$\mathbb{M}_{1\text{ Cluster}}^{\mathrm{Learning}}$	227	34.7	13.5	42.8	19.9	40.7	6.43	286
$\mathbb{M}^{\mathrm{Univariate}}$	132	38.1	19.3	**299**	108	57.2	**255**	41.6
$\mathbb{M}^{\mathrm{Learning}}$	**390**	164	34.9	93.5	**548**	**67.0**	30.3	$\mathbf{105 \cdot 10^1}$

Fig. 11. Average of the **FO** metric on all combinatorial problems.

Population Size n								
EA	BT_1^{10}	ZDT_4^{10}	ZDT_6^{10}	CTP_7^{10}	BT_1^{100}	ZDT_4^{100}	ZDT_6^{100}	CTP_7^{100}
SPEA$^{\mathrm{UX}}$	50	50	25	25	25	25	25	25
SPEA$^{\mathrm{1X}}$	25	100	25	25	25	25	25	25
NSGA-II$^{\mathrm{UX}}$	200	200	100	100	100	200	200	150
NSGA-II$^{\mathrm{1X}}$	200	375	75	300	75	200	150	300
$\mathbb{M}_{1\text{ Cluster}}^{\mathrm{UX}}$	75	100	25	25	100	125	200	125
$\mathbb{M}_{1\text{ Cluster}}^{\mathrm{1X}}$	100	450	25	300	125	325	100	175
$\mathbb{M}^{\mathrm{UX}}$	225	25	25	25	125	200	200	300
$\mathbb{M}^{\mathrm{1X}}$	150	475	25	725	125	200	100	150
$\mathbb{M}_{1\text{ Cluster}}^{\mathrm{Univariate}}$	150	50	75	50	100	75	375	50
$\mathbb{M}_{1\text{ Cluster}}^{\mathrm{Learning}}$	150	75	425	75	175	100	700	100
$\mathbb{M}^{\mathrm{Univariate}}$	275	125	200	125	250	200	800	200
$\mathbb{M}^{\mathrm{Learning}}$	450	200	250	150	225	300	400	250

Fig. 12. Population sizes used for the real–valued problems.

Population Size n								
EA	MS^{100}	KN^{100}	SC^{100}	MST^{105}	MS^{1000}	KN^{1000}	SC^{1000}	MST^{1035}
SPEA$^{\mathrm{UX}}$	50	25	25	25	25	25	25	25
SPEA$^{\mathrm{1X}}$	50	50	25	100	25	25	25	50
NSGA-II$^{\mathrm{UX}}$	200	175	150	200	200	250	200	250
NSGA-II$^{\mathrm{1X}}$	100	175	250	200	150	200	150	250
$\mathbb{M}_{1\text{ Cluster}}^{\mathrm{UX}}$	100	150	225	1250	175	200	100	350
$\mathbb{M}_{1\text{ Cluster}}^{\mathrm{1X}}$	100	175	125	1200	150	150	150	375
$\mathbb{M}^{\mathrm{UX}}$	175	175	100	950	175	250	125	450
$\mathbb{M}^{\mathrm{1X}}$	150	175	150	675	200	125	100	425
$\mathbb{M}_{1\text{ Cluster}}^{\mathrm{Univariate}}$	325	250	350	2700	650	500	150	475
$\mathbb{M}_{1\text{ Cluster}}^{\mathrm{Learning}}$	425	400	500	1800	1000	700	500	1500
$\mathbb{M}^{\mathrm{Univariate}}$	250	200	400	900	475	600	475	775
$\mathbb{M}^{\mathrm{Learning}}$	500	450	850	1950	750	1250	1900	1250

Fig. 13. Population sizes used for the combinatorial problems.

Average Front Distance **AFD**													
Statistically Significant Improvement Matrix	SPEA^{UX}	SPEA^{1X}	$\text{NSGA-II}^{\text{UX}}$	$\text{NSGA-II}^{\text{1X}}$	$\mathbb{M}_{1\text{ Cluster}}^{\text{UX}}$	$\mathbb{M}_{1\text{ Cluster}}^{\text{1X}}$	$\mathbb{M}^{\text{UX}}$	$\mathbb{M}^{\text{1X}}$	$\mathbb{M}_{1\text{ Cluster}}^{\text{Univariate}}$	$\mathbb{M}_{1\text{ Cluster}}^{\text{Learning}}$	$\mathbb{M}^{\text{Univariate}}$	$\mathbb{M}^{\text{Learning}}$	**Sum**
SPEA^{UX}	0	-6	-8	-2	1	-1	-4	-2	4	1	-9	-1	-27
SPEA^{1X}	6	0	-8	-3	-3	1	-4	2	6	-1	-9	-1	-14
$\text{NSGA-II}^{\text{UX}}$	8	8	0	3	11	5	4	3	7	3	-7	1	46
$\text{NSGA-II}^{\text{1X}}$	2	3	-3	0	3	10	-2	9	4	-2	-9	-2	13
$\mathbb{M}_{1\text{ Cluster}}^{\text{UX}}$	-1	3	-11	-3	0	0	-5	-1	5	1	-8	-4	-24
$\mathbb{M}_{1\text{ Cluster}}^{\text{1X}}$	1	-1	-5	-10	0	0	-5	0	2	-2	-11	-3	-34
$\mathbb{M}^{\text{UX}}$	4	4	-4	2	5	5	0	4	4	3	-8	0	19
$\mathbb{M}^{\text{1X}}$	2	-2	-3	-9	1	0	-4	0	2	-5	-10	-2	-30
$\mathbb{M}_{1\text{ Cluster}}^{\text{Univariate}}$	-4	-6	-7	-4	-5	-2	-4	-2	0	-7	-10	-9	-60
$\mathbb{M}_{1\text{ Cluster}}^{\text{Learning}}$	-1	1	-3	2	-1	2	-3	5	7	0	-9	-3	-3
$\mathbb{M}^{\text{Univariate}}$	9	9	7	9	8	11	8	10	10	9	0	5	95
$\mathbb{M}^{\text{Learning}}$	1	1	-1	2	4	3	0	2	9	3	-5	0	19

Fig. 14. Number of times an improvement was found to be statistically significant in the AFD metric, summed over all tested problems. The numbers in a single row indicate the summed number of significantly better or worse results compared to the algorithms in the different columns.

The number of evaluations that were allowed in this benchmark, is relatively small. Although a good assessment is obtained in how good the tested approaches are at rapidly obtaining a good approximation of the global Pareto optimal front, the use of learning techniques to exploit problem structure usually requires a larger number of evaluations in order for their effectiveness to be displayed. This is even more so if mixtures of factorizations are used, since the number of solutions in each mixture component is smaller than the complete number of selected solutions. This behavior can be seen from the results since the AFD metric is more often improved by using learning techniques if a single cluster is used than when multiple clusters are used.

Regarding the full set of benchmarks, $\mathbb{M}$IDEA is capable of obtaining results that are at least comparable with the most efficient state–of–the–art multi–objective evolutionary algorithms. Furthermore, from our results it seems that NSGA–II is the most competitive. However, there is an added value to the use of $\mathbb{M}$IDEA in that it is able to obtain and maintain a larger and more diverse Pareto front by parallel front exploration and diversity preserving selection. The experiments underline these results as the front spread (Figures 8 and 9), front occupation (Figures 10 and 11) and the global Pareto fronts in Figure 5

Front Spread **FS**													
Statistically Significant Improvement Matrix	SPEA^{UX}	SPEA^{1X}	NSGA-II^{UX}	NSGA-II^{1X}	$\mathbb{M}_{1\text{ Cluster}}^{UX}$	$\mathbb{M}_{1\text{ Cluster}}^{1X}$	$\mathbb{M}^{UX}$	$\mathbb{M}^{1X}$	$\mathbb{M}_{1\text{ Cluster}}^{Univariate}$	$\mathbb{M}_{1\text{ Cluster}}^{Learning}$	$\mathbb{M}^{Univariate}$	$\mathbb{M}^{Learning}$	**Sum**
SPEA^{UX}	0	-7	9	8	3	5	-4	0	11	3	-6	-6	16
SPEA^{1X}	7	0	16	14	8	11	-2	2	11	7	-5	-5	64
NSGA-II^{UX}	-9	-16	0	5	-12	-6	-15	-14	3	-9	-14	-13	-100
NSGA-II^{1X}	-8	-14	-5	0	-10	-7	-16	-14	1	-7	-16	-15	-111
$\mathbb{M}_{1\text{ Cluster}}^{UX}$	-3	-8	12	10	0	4	-12	-8	9	0	-15	-13	-24
$\mathbb{M}_{1\text{ Cluster}}^{1X}$	-5	-11	6	7	-4	0	-14	-14	5	-3	-15	-14	-62
$\mathbb{M}^{UX}$	4	2	15	16	12	14	0	7	11	9	1	1	92
$\mathbb{M}^{1X}$	0	-2	14	14	8	14	-7	0	9	5	-5	-4	46
$\mathbb{M}_{1\text{ Cluster}}^{Univariate}$	-11	-11	-3	-1	-9	-5	-11	-9	0	-12	-15	-16	-103
$\mathbb{M}_{1\text{ Cluster}}^{Learning}$	-3	-7	9	7	0	3	-9	-5	12	0	-12	-13	-18
$\mathbb{M}^{Univariate}$	6	5	14	16	15	15	-1	5	15	12	0	-2	100
$\mathbb{M}^{Learning}$	6	5	13	15	13	14	-1	4	16	13	2	0	100

Fig. 15. Number of times an improvement was found to be statistically significant in the FS metric, summed over all tested problems. The numbers in a single row indicate the summed number of significantly better or worse results compared to the algorithms in the different columns.

indicate a better performance. This increased performance is furthermore also statistically significant, as can be seen in figures 15 and 16. The use of clustering to obtain mixture probability distributions clearly leads to a significant increase of performance in the preservation and exploration of diversity. However, using only a single cluster in combination with the diversity preserving selection method in $\mathbb{M}$IDEA, on average yields inferior results compared to NSGA–II. This is an indication of the fact that the use of mixture probability distributions in combination with the selection method of NSGA–II is likely to lead to an even more efficient and effective diversity preserving multi–objective evolutionary algorithm.

6 Discussion

We have tested different MOEAs on multiple problems and compared their results. These MOEAs differ in two aspects. On the one hand, they differ in the way that the solutions are represented and recombined into new offspring. On the other hand, they differ in the way that selection is performed and

Front Occupation **FO**													
Statistically Significant Improvement Matrix	$SPEA^{UX}$	$SPEA^{1X}$	$NSGA\text{-}II^{UX}$	$NSGA\text{-}II^{1X}$	$\mathbb{M}_{1\text{ Cluster}}^{UX}$	$\mathbb{M}_{1\text{ Cluster}}^{1X}$	$\mathbb{M}^{UX}$	$\mathbb{M}^{1X}$	$\mathbb{M}_{1\text{ Cluster}}^{Univariate}$	$\mathbb{M}_{1\text{ Cluster}}^{Learning}$	$\mathbb{M}^{Univariate}$	$\mathbb{M}^{Learning}$	**Sum**
$SPEA^{UX}$	0	-3	2	4	13	14	13	12	12	8	-4	-8	63
$SPEA^{1X}$	3	0	1	3	16	16	16	14	14	8	-2	-6	83
$NSGA\text{-}II^{UX}$	-2	-1	0	1	8	8	7	5	10	3	-2	-7	30
$NSGA\text{-}II^{1X}$	-4	-3	-1	0	7	9	7	5	11	2	-6	-9	18
$\mathbb{M}_{1\text{ Cluster}}^{UX}$	-13	-16	-8	-7	0	-3	-11	-7	1	-2	-16	-13	-95
$\mathbb{M}_{1\text{ Cluster}}^{1X}$	-14	-16	-8	-9	3	0	-8	-11	3	-2	-14	-14	-90
$\mathbb{M}^{UX}$	-13	-16	-7	-7	11	8	0	1	5	1	-14	-15	-46
$\mathbb{M}^{1X}$	-12	-14	-5	-5	7	11	-1	0	8	4	-12	-12	-31
$\mathbb{M}_{1\text{ Cluster}}^{Univariate}$	-12	-14	-10	-11	-1	-3	-5	-8	0	-6	-15	-12	-97
$\mathbb{M}_{1\text{ Cluster}}^{Learning}$	-8	-8	-3	-2	2	2	-1	-4	6	0	-6	-11	-33
$\mathbb{M}^{Univariate}$	4	2	2	6	16	14	14	12	15	6	0	-4	87
$\mathbb{M}^{Learning}$	8	6	7	9	13	14	15	12	12	11	4	0	111

Fig. 16. Number of times an improvement was found to be statistically significant in the FO metric, summed over all tested problems. The numbers in a single row indicate the summed number of significantly better or worse results compared to the algorithms in the different columns.

diversity is maintained. The use of probabilistic model learning in EAs is relatively new. Therefore, it would also be interesting to see what results are obtained when probabilistic model learning techniques are used with features such as layered selection in NSGA-II and the external non–dominated archive in SPEA. The results in this paper indicate that even more effective MOEAs may be constructed by doing so. At this point, we note that the MIDEA framework is an efficient new tool for multi–objective optimization, since its performance is at least comparable to current state–of–the art MOEAs.

The results in this paper show that exploiting interactions can be beneficial in multi–objective optimization. We have provided the BT_1 function of which we know that modelling multivariate interactions improves evolutionary optimization. However, for more practical applications such as the binary combinatorial problems, it is unclear at the outset whether or not the modelling of interactions will indeed contribute to more efficient optimization. It would therefore be interesting to see how the results vary when we look at the number of function evaluations required to obtain a certain quality of solutions. In such a case, the population size could grow larger to allow for more solutions in a single cluster. In this way, more reliable probabilistic models can be built,

leading perhaps to better results than when the univariate factorization is used. Furthermore, it would also be interesting to find and define more problems on which the use of the univariately factorized probability distributions or simple crossover schemes such as uniform crossover produce worse results than when multivariate dependencies are modelled and processed.

In the MIDEA as currently proposed, the clustering algorithm is not capable of determining how many clusters are actually required to obtain a certain degree of diversity or what the requirements on the population size are, to be able to perform proper model learning in each cluster. Although general greedy incremental learning algorithms could be used to adaptively select the number of mixtures, this would imply a much larger running time since factorizations need to be recomputed if the number of clusters changes. From our experiments however, using only a few clusters already indicates a significant increase in the front spread for a wide variety of test problems.

Using a mixture of univariate factorizations has been shown in this paper to give good results on a variety of problems. Furthermore, using the univariate factorization is fast. Since the clustering algorithm also requires little time, we essentially have a very fast and good performing multi–objective optimization algorithm. Before trying to exploit linkage information by processing higher order dependencies, we can therefore first test whether disregarding this information gives satisfactory results.

7 Conclusions

In this paper, we have proposed and used the algorithmic framework MIDEA for multi–objective optimization by learning and using probabilistic models. To this end, a probability distribution is learned from a selection of solutions and subsequently more solutions are drawn from this probability distribution. For the specific task of multi–objective optimization, we proposed the use of mixture distributions obtained by clustering to stimulate parallel exploration along the Pareto front. Furthermore, we proposed a specialized diversity preserving selection operator to preserve diversity and a good distribution of the points in the space of the objectives to be optimized.

By testing elitist MIDEAs on sixteen test problems that differ in problem domain and dimensionality, we showed that MIDEAs are capable of obtaining results that are at least comparable to results that may be obtained with current state–of–the–art MOEAs. Furthermore, MIDEAs display excellent and unmatched behavior in obtaining diversity and a good distribution of trade–off solutions, especially for higher–dimensional problems.

References

[1] S. Baluja and R. Caruana. Removing the genetics from the standard genetic algorithm. In A. Prieditis and S. Russell, editors, *Proceedings of the 1995 International Conference on Machine Learning*, pages 38–46. Morgan Kauffman, 1995.

[2] S. Baluja and S. Davies. Using optimal dependency–trees for combinatorial optimization: Learning the structure of the search space. In D. H. Fisher, editor, *Proceedings of the 1997 International Conference on Machine Learning*, pages 30–38. Morgan Kauffman, 1997.

[3] S. Bandyopadhyay, H. Kargupta, and G. Wang. Revisiting the GEMGA: Scalable evolutionary optimization through linkage learning. In *Proceedings of the 1998 IEEE International Conference on Evolutionary Computation*, pages 603–608. IEEE Press, 1998.

[4] E. Bengoetxea, T. Miquélez, P. Larrañaga, and J. A. Lozano. Experimental results in function optimization with EDAs in continuous domains. In P. Larrañaga and J. A. Lozano, editors, *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation.* Kluwer Academic Publishers, 2001.

[5] J. S. De Bonet, C. Isbell, and P. Viola. MIMIC: Finding optima by estimating probability densities. *Advances in Neural Information Processing*, 9, 1996.

[6] P. A. N. Bosman and D. Thierens. Linkage information processing in distribution estimation algorithms. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the GECCO–1999 Genetic and Evolutionary Computation Conference*, pages 60–67. Morgan Kaufmann, 1999.

[7] P. A. N. Bosman and D. Thierens. Continuous iterated density estimation evolutionary algorithms within the IDEA framework. In M. Pelikan, H. Mühlenbein, and A. O. Rodriquez, editors, *Proceedings of the Optimization by Building and Using Probabilistic Models OBUPM Workshop at the GECCO–2000 Genetic and Evolutionary Computation Conference*, pages 197–200. Morgan Kaufmann, 2000.

[8] P. A. N. Bosman and D. Thierens. Expanding from discrete to continuous estimation of distribution algorithms: The IDEA. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature – PPSN VI*, pages 767–776. Springer, 2000.

[9] P. A. N. Bosman and D. Thierens. Mixed IDEAs. Utrecht University Technical Report UU–CS–2000–45, 2000.

[10] P. A. N. Bosman and D. Thierens. Negative log–likelihood and statistical hypothesis testing as the basis of model selection in IDEAs. In A. Feelders, editor, *Proceedings of the Tenth Belgium–Netherlands Conference on Machine Learning*, pages 109–116. Tilburg University, 2000.

[11] P. A. N. Bosman and D. Thierens. Advancing continuous IDEAs with mixture distributions and factorization selection metrics. In M. Pelikan and K. Sastry, editors, *Proceedings of the Optimization by Building and Using Probabilistic Models OBUPM Workshop at the GECCO–2001 Genetic and Evolutionary Computation Conference*, pages 208–212. Morgan Kaufmann Publishers, 2001.

[12] P. A. N. Bosman and D. Thierens. Exploiting gradient information in continuous iterated density estimation evolutionary algorithms. In B. Kröse, M. de Rijke, G. Schreiber, and M. van Someren, editors, *Proceedings of the 13th Belgium–Netherlands Artificial Intelligence Conference BNAIC'01*, pages 69–76, 2001.

[13] P. A. N. Bosman and D. Thierens. A thorough documentation of obtained results on real–valued continuous and combinatorial multi–objective optimization problems using diversity preserving mixture–based iterated density estimation evolutionary algorithms. Utrecht University Technical Report, 2002.

[14] W. Buntine. Operations for learning with graphical models. *Journal of Artificial Intelligence Research*, 2:159–225, 1994.

[15] C. K. Chow and C. N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14:462–467, 1968.

[16] C. A. Coello Coello. A comprehensive survey of evolutionary-based multiobjective optimization techniques. *Knowledge and Information Systems. An International Journal*, 1(3):269–308, 1999.

[17] K. Deb, S. Agrawal, A. Pratab, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature – PPSN VI*, pages 849–858. Springer, 2000.

[18] K. Deb, A. Pratap, and T. Meyarivan. Constrained test problems for multi–objective evolutionary optimization. In E. Zitzler, K. Deb, L. Thiele, C. A. Coello Coello, and D. Corne, editors, *First International Conference on Evolutionary Multi-Criterion Optimization*, pages 284–298. Springer–Verlag, 2001.

[19] D. Edwards. *Introduction to Graphical Modelling*. Springer Verlag, 1995.

[20] M. Ehrgott and X. Gandibleux. An annotated bibliography of multi–objective combinatorial optimization. Technical Report 62/2000, Fachbereich Mathematik, Universität Kaiserslautern, Kaiserslautern, 2000.

[21] R. Etxeberria and P. Larrañaga. Global optimization using Bayesian networks. In A. A. O. Rodriguez, M. R. S. Ortiz, and R. S. Hermida, editors, *Proceedings of the Second Symposium on Artificial Intelligence CIMAF–1999*, pages 332–339. Institute of Cybernetics, Mathematics and Physics, 1999.

[22] C. M. Fonseca and P. J. Fleming. An overview of evolutionary algorithms in multiobjective optimization. *Evolutionary Computation*, 3(1):1–16, 1995.

[23] N. Friedman and M. Goldszmidt. Learning Bayesian networks with local structure. In E. Horvits and F. Jensen, editors, *Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence UAI–1996*, pages 252–262. Morgan Kaufmann, 1996.

[24] M. Gallagher, M. Fream, and T. Downs. Real–valued evolutionary optimization using a flexible probability density estimator. In W. Banzhaf *et al.*, editor, *Proceedings of the GECCO–1999 Genetic and Evolutionary Computation Conference*, pages 840–846. Morgan Kauffman, 1999.

[25] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison–Wesley, Reading, 1989.

[26] D. E. Goldberg, K. Deb, H. Kargupta, and G. Harik. Rapid, accurate optimization of difficult problems using fast messy genetic algorithms. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 56–64. Morgan Kauffman, 1993.

[27] D. E. Goldberg, B. Korb, and K. Deb. Messy genetic algorithms: Motivation, analysis and first results. *Complex Systems*, 10:385–408, 1989.

[28] G. Harik. Linkage learning via probabilistic modeling in the ECGA. IlliGAL Technical Report 99010, 1999.

[29] J. A. Hartigan. *Clustering Algorithms*. John Wiley & Sons, Inc., 1975.

[30] J. H. Holland. *Adaptation in Natural and Artificial Systems*. Ann Arbor: University of Michigan Press, 1975.

[31] S. Kullback. *Information Theory And Statistics*. New York: Dover, 1968.

[32] P. Larrañaga and J. A. Lozano. *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation*. Kluwer Academic Publishers, 2001.

[33] F. G. Lobo, K. Deb, D. E. Goldberg, G. R. Harik, and L. Wang. Compressed introns in a linkage learning genetic algorithm. In W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, and R. Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 551–558. Morgan Kauffman, 1998.

[34] M. Meilă and M. I. Jordan. Estimating dependency structure as a hidden variable. In M. I. Jordan M. J. Kearns and S. A. Solla, editors, *Advances in Neural Information Processing Systems*, volume 10, pages 584–590. MIT Press, 1998.

[35] H. Mühlenbein and T. Mahnig. FDA – a scalable evolutionary algorithm for the optimization of additively decomposed functions. *Evolutionary Computation*, 7:353–376, 1999.

[36] H. Mühlenbein and G. Paaß. From recombination of genes to the estimation of distributions I. binary parameters. In A. E. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature – PPSN V*, pages 178–187. Springer, 1998.

[37] J. Pearl. *Causality: Models, Reasoning and Inference*. Cambridge University Press, 2000.

[38] M. Pelikan and D. E. Goldberg. Genetic algorithms, clustering, and the breaking of symmetry. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature – PPSN VI*, pages 385–394. Springer, 2000.

[39] M. Pelikan and D. E. Goldberg. Escaping hierarchical traps with competent genetic algorithms. In L. Spector, E. D. Goodman, A. Wu, W. B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon, and E. Burke, editors, *Proceedings of the GECCO–2001 Genetic and Evolutionary Computation Conference*, pages 511–518. Morgan Kaufmann, 2001.

[40] M. Pelikan, D. E. Goldberg, and E. Cantú-Paz. BOA: The Bayesian optimization algorithm. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the GECCO–1999 Genetic and Evolutionary Computation Conference*, pages 525–532. Morgan Kauffman, 1999.

[41] M. Pelikan and H. Mühlenbein. The bivariate marginal distribution algorithm. In R. Roy, T. Furuhashi, K. Chawdry, and K. Pravir, editors, *Advances in Soft Computing - Engineering Design and Manufacturing*, pages 521–535, London, 1999. Springer-Verlag.

[42] J. M. Peña, J. A. Lozano, and P. Larrañaga. Benefits of data clustering in multimodal function optimization via EDAs. In P. Larrañaga and J. A. Lozano, editors, *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation.* Kluwer Academic Publishers, 2001.

[43] R. Santana, A. Ochoa, and M. R. Soto. The mixture of trees factorized distribution algorithm. In L. Spector, E. D. Goodman, A. Wu, W. B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon, and E. Burke, editors, *Proceedings of the GECCO–2001 Genetic and Evolutionary Computation Conference*, pages 543–550. Morgan Kaufmann, 2001.

[44] G. Schwarz. Estimating the dimension of a model. *Annals of Statistics*, 6:461–464, 1978.

[45] M. Sebag and A. Ducoulombier. Extending population–based incremental learning to continuous search spaces. In A. E. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature – PPSN V*, pages 418–427. Springer-Verlag, 1998.

[46] D. Thierens and P. A. N. Bosman. Multi–objective mixture–based iterated density estimation evolutionary algorithms. In L. Spector, E. D. Goodman, A. Wu, W. B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon, and E. Burke, editors, *Proceedings of the GECCO–2001 Genetic and Evolutionary Computation Conference*, pages 663–670, San Francisco, California, 2001. Morgan Kaufmann.

[47] D. Thierens and D. E. Goldberg. Mixing in genetic algorithms. In S. Forrest, editor, *Proceedings of the fifth conference on Genetic Algorithms*, pages 38–45. Morgan Kaufmann, 1993.

[48] S. Tsutsui, M. Pelikan, and D. E. Goldberg. Evolutionary algorithm using marginal histogram in continuous domain. In M. Pelikan and K. Sastry, editors, *Proceedings of the Optimization by Building and Using Probabilistic Models OBUPM Workshop at the GECCO–2001 Genetic and Evolutionary Computation Conference*, pages 230–233. Morgan Kaufmann Publishers, 2001.

[49] C. H. M. van Kemenade. Building block filtering and mixing. In *Proceedings of the 1998 IEEE International Conference on Evolutionary Computation*, pages 505–510. IEEE Press, 1998.

[50] E. Zitzler, K. Deb, and L. Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary Computation*, 8(2):173–195, 2000.

[51] E. Zitzler and L. Thiele. Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271, 1999.