The naive MIDEA: a baseline multi–objective EA

Peter A.N. Bosman and Dirk Thierens

Institute of Information and Computing Sciences
Utrecht University, The Netherlands
{Peter.Bosman, Dirk.Thierens}@cs.uu.nl

Abstract. Estimation of distribution algorithms have been shown to perform well on a wide variety of single–objective optimization problems. Here, we look at a simple - yet effective - extension of this paradigm for multi–objective optimization, called the naive MIDEA. The probabilistic model in this specific algorithm is a mixture distribution, and each component in the mixture is a univariate factorization. Mixture distributions allow for wide–spread exploration of the Pareto front thus aiding the important preservation of diversity in multi–objective optimization. Due to its simplicity, speed, and effectiveness the naive MIDEA can well serve as a baseline algorithm for multi–objective evolutionary algorithms.

1 Introduction

Estimation of distribution algorithms (EDAs) are a class of evolutionary algorithms that build at each generation a probabilistic model from the selected solutions. EDAs are mainly characterized by the kind of probabilistic model they learn. Somewhat surprisingly, it has become clear that even the use of a simple univariate factorization as probabilistic model often leads to good performance for discrete, single objective optimization problems. Considering their ease of implementation, univariate EDAs have become a baseline algorithm that can be used to generate reasonable good solutions quickly, thus setting a performance level that more elaborated algorithms need to surpass to justify their use. The goal of this paper is to show that a similar baseline algorithm can be constructed for multi–objective optimization problems. The resulting algorithm - called the naive MIDEA - is a simple, fast, and efficient multi–objective evolutionary algorithm (MOEA) based on the concept of Pareto dominance, and can thus be used as a baseline algorithm for more elaborate MOEAs.

The remainder of this paper is organized as follows. In Section 2 we specify the naive MIDEA algorithm as an instance of the more general MIDEA (mixture–based, multi–objective iterated density–estimation evolutionary algorithm) framework. In Section 3 we test the performance of the algorithm on four multi–objective, combinatorial optimization problems, compare the results with two state–of–the–art MOEAs, and discuss our findings. We present our conclusions in Section 4.

2 The naive MIDEA

2.1 Mixture probability distributions

A mixture probability distribution is a weighted sum of k probability distributions. Each probability distribution in the mixture distribution is called a mixture component. Let $\boldsymbol{\mathcal{Z}} = (Z_0, Z_1, \ldots, Z_{l-1})$ be a vector of random variables Z_i associated with the i–th problem variable. A mixture probability distribution is then defined as:

$$P^{mixture}(\boldsymbol{\mathcal{Z}}) = \sum_{i=0}^{k-1} \beta_i P^i(\boldsymbol{\mathcal{Z}}) \tag{1}$$

where $\beta_i > 0$, $i \in \{0, 1, \ldots, k-1\}$, and $\sum_{i=0}^{k-1} \beta_i = 1$. The β_i with which the mixture components are weighted in the sum are called mixing coefficients.

The general advantage of mixture probability distributions is that a larger class of dependency relations between the random variables can be expressed than when using non–mixture probability distributions since a mixture probability distribution makes a combination of multiple probability distributions. In many cases, simple probability distributions can be used as mixture components to get accurate descriptions of the data in different parts of the sample space. By using mixture probability distributions, a powerful, yet computationally tractable type of probability distribution can be used within EDAs, that provides for processing complicated interactions between a problem's variables.

For multi–objective optimization, mixture distributions have an additional advantage that renders them particularly useful. The specific advantage is geometrical in nature. If we, for instance, cluster the solutions as observed in the objective space and then estimate a simpler probability distribution in each cluster, the probability distributions in these clusters can portray specific information about the different regions along the Pareto optimal front that we are ultimately interested in. Drawing new solutions from the resulting mixture probability distribution gives solutions that are more likely to be well spread along the front as each mixture component delivers a subset of new solutions. The use of such a mixture distribution thus results in a parallel exploration along the current Pareto front. This parallel exploration may very well provide a better spread of new solutions along the Pareto front than when a single non–mixture distribution is used to capture information about the complete Pareto front.

To complete the construction of the mixture distribution we also need to determine the mixing coefficients β_i. This can be done in various ways. Here we set β_i proportional to the size of the i–th cluster with respect to the sum of the sizes of all clusters.

2.2 Selection operator

In multi–objective optimization we want the solutions to be as close to the Pareto optimal front as possible, and we want a good diverse representation

of the Pareto optimal front. In a practical application, we have no indication of how close we are to the Pareto optimal front. To ensure selection pressure toward the Pareto optimal front in the absence of such information, the best we can do is to find solutions that are dominated as little as possible by any other solution. A straightforward way to obtain selection pressure toward non–dominated solutions is therefore to count for each solution in the population the number of times it is dominated by another solution in the population, which is called the domination count of a solution [1, 5].

In the MIDEA selection we discern two cases. Call n the population size and $\tau \in [0...1]$ the selection threshold. If the number of non–dominated solutions - those with a domination count of 0 - is less than or equal to $\lfloor \tau n \rfloor$ we simply select the $\lfloor \tau n \rfloor$ solutions with lowest domination count (ties are broken at random). However, if the number of non–dominated solutions is larger than $\lfloor \tau n \rfloor$ we first collect all non–dominated solutions in a set $\boldsymbol{\mathcal{S}}^P$. Next, the final selection $\boldsymbol{\mathcal{S}}$ is obtained from $\boldsymbol{\mathcal{S}}^P$ using a nearest neighbor heuristic to enforce diversity. We start by picking a solution from $\boldsymbol{\mathcal{S}}^P$ with an optimal value for a randomly chosen objective and move it to the set $\boldsymbol{\mathcal{S}}$ which has now a single element. Note that the choice of objective is arbitrary as the goal is to find a diverse selection of solutions. For all solutions in $\boldsymbol{\mathcal{S}}^P$, the nearest neighbor distance is computed to the 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 $\boldsymbol{\mathcal{S}}$. This selection operator has a running time complexity of $\mathcal{O}(n^2)$ which is equal to the complexity for computing the domination counts.

2.3 The naive MIDEA

The naive MIDEA is an instance of the MIDEA framework for multi–objective optimization using EDAs [1, 9]. The MIDEA can be specified in pseudo–code as follows:

MIDEA
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 $\lfloor \tau n \rfloor$ solutions with the diversity preserving selection operator
2.3 Estimate a mixture probability distribution $P^{mixture}(\boldsymbol{\mathcal{Z}})$
2.4 Replace the non–selected solutions with new solutions drawn from $P^{mixture}(\boldsymbol{\mathcal{Z}})$
2.5 Evaluate the objectives of the new solutions

The naive MIDEA uses a simple univariate factorized probability distributions in each cluster: $P^{univariate}(\boldsymbol{\mathcal{Z}}) = \prod_{i=0}^{l-1} P(Z_i)$. For discrete random variables, this amounts to repeatedly counting frequencies and computing proportions for a single random variable. Since in each cluster we thus disregard all dependencies

between random variables, we call this specific MIDEA instance naive in analogy with the well–known naive Bayes classifier. However, the clusters are expected to already provide a large benefit for multi–objective optimization. Moreover, algorithms such as UMDA [8] and the compact GA [6] that also use the univariate marginal probability distribution (without clustering) have proved to be reasonably effective for single–objective optimization problems.

For computational efficiency reasons we apply a fast clustering algorithm. Possibly this adds to the naiveness of our naive MIDEA instance, but other clustering algorithms are easily implemented if required. The algorithm that we use here is the leader algorithm, which is one of the fastest partitioning algorithms. The use of it can thus be beneficial if the amount of overhead that is introduced by factorization mixture selection methods is desired to remain small. There is no need to specify in advance how many partitions there should be. The first solution to make a new partition is appointed to be its leader. The leader algorithm goes over the solutions exactly once. For each solution it encounters, it finds the first partition that has a leader being closer to the solution than a given threshold $\mathfrak{T}_d$. If no such partition can be found, a new partition is created containing only this single solution. To prevent the first partitions from becoming quite a lot larger than the later ones, we randomize the order in which the partitions are inspected. One of the drawbacks of the randomized leader algorithm is that it is not invariant given the sequence of the input solutions. Therefore, to be sure that the ordering of the solutions is not subject to large repeating sequences of solutions, we randomize the ordering of the solutions each time the leader algorithm is applied.
Summarizing, the pseudo-code of the naive MIDEA is as follows:

naive MIDEA (*instantiation of steps 2.3 and 2.4 of the general* MIDEA *framework*)
1 $(\boldsymbol{c}^0, \boldsymbol{c}^1, \ldots, \boldsymbol{c}^{k-1}) \leftarrow$ LeaderAlgorithm($\mathfrak{T}_d$) 2 **for** $i \leftarrow 0$ **to** $k-1$ **do** 2.1 $\beta_i \leftarrow \lvert\boldsymbol{c}^i\rvert / \lfloor \tau n \rfloor$ 2.2 **for** $j \leftarrow 0$ **to** $l-1$ **do** 2.2.1 Estimate the univariate marginal probability distribution $P^{i,j}(Z_j)$ for random variable Z_j from the solutions in the i–th cluster (i.e. $\boldsymbol{c}^i$) 3 **for** $i \leftarrow \lfloor \tau n \rfloor$ **to** $n-1$ **do** 3.1 Initialize a new solution $\boldsymbol{z}$ 3.2 Choose an index $q \in \{0, 1, \ldots, k-1\}$ with probability β_q 3.3 **for** $j \leftarrow 0$ **to** $l-1$ **do** 3.3.1 Draw a value for Z_j from the univariate marginal probability distribution $P^{q,j}(Z_j)$ associated with the q–th cluster 3.4 Add $\boldsymbol{z}$ to the set of new offspring.

3 Experiments

In this section we compare the naive MIDEA to two 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 [13] and the NSGA-II algorithm by Deb

et al. [3] showed superior performance compared to most other MOEAs [3, 11]. The multi–objective optimization problems are described in Section 3.1. The performance measures we use to score the results of the algorithms with are described in Section 3.2. In Section 3.3 we present our experimental setup. Finally, in Section 3.4 we discuss the obtained results.

3.1 Multi–objective optimization problems

The test–suite we have used consists of four multi–objective optimization problems with two different dimensionalities resulting in a total test size of 8 problems. Figure 1 specifies the four 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 to two and is denoted by m.

Name	Definition
MS (Maximum Satisfiability)	**Maximize** $(f_0(\boldsymbol{x}), f_1(\boldsymbol{x}), \ldots, f_{m-1}(\boldsymbol{x}))$ **Where** • $\forall\, i : f_i(\boldsymbol{x}) = \sum_{j=0}^{c_i - 1} \operatorname{sgn}\left(\left[\sum_{k=0}^{l-1} (C_i)_{jk} \otimes x_k\right]\right)$ • $\operatorname{sgn}(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ -1 & \text{if } x < 0 \end{cases}$ • $\otimes$ | 0 1 ; -1 | 1 0 — $\otimes$ | 0 1 ; 0 | 0 0 — $\otimes$ | 0 1 ; 1 | 0 1
KN (Knapsack)	**Maximize** $(f_0(\boldsymbol{x}), f_1(\boldsymbol{x}), \ldots, f_{m-1}(\boldsymbol{x}))$ **Where** • $\forall\, i : f_i(\boldsymbol{x}) = \sum_{j=0}^{l-1} P_{ij} x_j$ **Such that** • $\forall\, i : \sum_{j=0}^{l-1} W_{ij} x_j \leq c_i$
SC (Set Covering)	**Minimize** $(f_0(\boldsymbol{x}), f_1(\boldsymbol{x}), \ldots, f_{m-1}(\boldsymbol{x}))$ **Where** • $\forall\, i : f_i(\boldsymbol{x}) = \sum_{j=0}^{l-1} C_{ij} x_j$ **Such that** • $\forall\, i : \forall_{0 \leq j < r} : \sum_{k=0}^{l-1} (A_i)_{jk} x_k \geq 1$
MST (Minimal Spanning Tree)	**Minimize** $(f_0(\boldsymbol{x}), f_1(\boldsymbol{x}), \ldots, f_{m-1}(\boldsymbol{x}))$ **Where** • $\forall\, i : f_i(\boldsymbol{x}) = \sum_{j=0}^{l-1} W_{ij} x_j$ **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. 1. Binary multi–objective combinatorial optimization test problems.

Maximum satisfiability. 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.

Knapsack. The multi–objective knapsack problem was first used by Zitzler and Thiele [13] to test MOEAs. 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.

Set covering. 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 binary solution indicates at which locations the services are placed.

Minimal spanning tree. In the minimal spanning tree problem we are given an undirected graph (V, E) such that each edge has a certain weight. 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.

3.2 Performance indicators

To measure the performance of a MOEA we only consider the subset of all non–dominated solutions that is contained in the final population that results from running the MOEA. We call such a subset an approximation set and denote it by $\mathcal{S}$. The size of the approximation set depends on the settings used to run the MOEA with. To actually measure performance, performance indicators are used. A performance indicator is a function that, given an approximation set $\mathcal{S}$, returns a real value that indicates how good $\mathcal{S}$ is with respect to a certain feature that is measured by the performance indicator. More detailed information regarding the importance of using good performance indicators to evaluate MOEAs may be found in dedicated literature [2, 7, 12]. Here we will use three performance indicators:

1. The *Front Spread* (**FS**) indicator measures the size of the objective space covered by an approximation set [13]. A larger **FS** indicator value is preferable given equal values for the other indicators. The **FS** indicator for an

approximation set $\mathcal{S}$ is defined to be the maximum Euclidean distance inside the smallest m–dimensional bounding–box that contains $\mathcal{S}$:

$$\mathbf{FS}(\mathcal{S}) = \sqrt{\sum_{i=0}^{m-1} \max_{(\boldsymbol{z}^0, \boldsymbol{z}^1) \in \mathcal{S} \times \mathcal{S}} \{(f_i(\boldsymbol{z}^0) - f_i(\boldsymbol{z}^1))^2\}} \tag{2}$$

2. The *Front Occupation* (**FO**) indicator measures the size of the set of non–dominated solutions [10]. Since a larger set of trade–off points is more desirable, a larger **FO** indicator value is preferable given equal values for the other indicators.

$$\mathbf{FO}(\mathcal{S}) = |\mathcal{S}| \tag{3}$$

3. The *Front to Set Distance* indicator ($\boldsymbol{D}_{\boldsymbol{\mathcal{P}_F} \rightarrow \boldsymbol{\mathcal{S}}}$) computes for each solution in the discrete Pareto optimal set the distance to the closest solution in an approximation set $\mathcal{S}$ and takes the average as the indicator value:

$$\boldsymbol{D}_{\boldsymbol{\mathcal{P}_F} \rightarrow \mathcal{S}}(\mathcal{S}) = \frac{1}{|\boldsymbol{\mathcal{P}_S}|} \sum_{\boldsymbol{z}^1 \in \boldsymbol{\mathcal{P}_S}} \min_{\boldsymbol{z}^0 \in \mathcal{S}} \{d(\boldsymbol{z}^0, \boldsymbol{z}^1)\} \tag{4}$$

Since we are interested in performance as measured in the objective space, the distance between two multi–objective solutions $\boldsymbol{z}^0$ and $\boldsymbol{z}^1$ is the Euclidean distance between their objective values $f(\boldsymbol{z}^0)$ and $f(\boldsymbol{z}^1)$.
The $\boldsymbol{D}_{\boldsymbol{\mathcal{P}_F} \rightarrow \mathcal{S}}$ indicator represents both the goal of getting close to the Pareto optimal front as well as the goal of getting a diverse, wide–spread front of solutions. A smaller value for this performance indicator is preferable.
A performance indicator that is closely related to the $\boldsymbol{D}_{\boldsymbol{\mathcal{P}_F} \rightarrow \mathcal{S}}$ indicator, is the hypervolume indicator by Knowles and Corne [7]. In the hypervolume indicator, a point in the objective space is picked such that it is dominated by all points in the approximation sets that need to be evaluated. The indicator value is then equal to the hypervolume of the multi–dimensional region enclosed by the approximation set and the picked reference point. This value is an indicator of the region in the objective space that is dominated by the approximation set. The main difference between the hypervolume indicator and the $\boldsymbol{D}_{\boldsymbol{\mathcal{P}_F} \rightarrow \mathcal{S}}$ indicator is that for the hypervolume indicator a reference point has to be chosen. Different reference points lead to different indicator values. Moreover, different reference points can lead to indicator values that indicate a preference for different approximation sets. Since in the $\boldsymbol{D}_{\boldsymbol{\mathcal{P}_F} \rightarrow \mathcal{S}}$ indicator the true Pareto optimal front is used, the $\boldsymbol{D}_{\boldsymbol{\mathcal{P}_F} \rightarrow \mathcal{S}}$ indicator does not suffer from this drawback. Of course, a major drawback of the $\boldsymbol{D}_{\boldsymbol{\mathcal{P}_F} \rightarrow \mathcal{S}}$ indicator is that in a real application the true Pareto optimal front is not known beforehand. In that case, the Pareto front of all approximation sets could be used as a substitute for the actual Pareto optimal front.

$D_{\mathcal{P}_F \to \mathcal{S}}$								
EA	MS^{100}	KN^{100}	SC^{100}	MST^{105}	MS^{1000}	KN^{1000}	SC^{1000}	MST^{1035}
SPEA^{UX}	12.5	9.59	2.92	1.43	183	67.3	518	6.10
SPEA^{1X}	11.6	8.50	2.99	1.50	277	83.1	452	5.75
NSGA-II^{UX}	11.4	7.75	2.61	1.21	185	84.1	260	6.45
NSGA-II^{1X}	11.5	8.87	2.63	1.49	289	121.0	329	5.95
naive MIDEA	**7.95**	**4.13**	**1.52**	**1.19**	**37.2**	**30.4**	**117**	**3.39**

Fig. 2. Average of the $D_{\mathcal{P}_F \to \mathcal{S}}$ performance indicator on all combinatorial problems.

Front Spread **FS**								
EA	MS^{100}	KN^{100}	SC^{100}	MST^{105}	MS^{1000}	KN^{1000}	SC^{1000}	MST^{1035}
SPEA^{UX}	116	69.5	**64.6**	30.6	288	254	631	52.1
SPEA^{1X}	126	82.6	51.0	32.5	399	308	**636**	50.8
NSGA-II^{UX}	120	78.3	14.8	26.3	370	288	144	33.7
NSGA-II^{1X}	129	76.6	12.8	23.9	364	291	107	36.1
naive MIDEA	**172**	**115**	24.7	**34.3**	**538**	**453**	204	**57.0**

Fig. 3. Average of the **FS** performance indicator on all combinatorial problems.

3.3 Experimental setup

Optimization problem dimensionalities. We used test instances with dimensionality $l = 100$ and $l = 1000$. 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 at 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,

Front Occupation **FO**								
EA	MS^{100}	KN^{100}	SC^{100}	MST^{105}	MS^{1000}	KN^{1000}	SC^{1000}	MST^{1035}
SPEA^{UX}	46.8	46.5	**25.0**	42.8	49.4	49.5	26.2	48.8
SPEA^{1X}	46.1	**77.6**	24.3	**80.1**	49.9	49.7	**26.5**	**95.0**
NSGA-II^{UX}	33.5	35.5	7.80	32.3	35.4	33.1	7.50	64.7
NSGA-II^{1X}	41.1	37.5	6.80	24.5	42.0	36.4	7.20	64.8
naive MIDEA	**52.6**	57.8	10.4	23.7	**116**	**104**	6.27	60.0

Fig. 4. Average of the **FO** performance indicator on all combinatorial problems.

Population Size n								
EA	MS^{100}	KN^{100}	SC^{100}	MST^{105}	MS^{1000}	KN^{1000}	SC^{1000}	MST^{1035}
SPEA$^{\mathrm{UX}}$	25	25	25	25	25	25	25	25
SPEA$^{\mathrm{1X}}$	25	50	25	100	25	25	25	50
NSGA-II$^{\mathrm{UX}}$	350	325	300	200	200	250	200	250
NSGA-II$^{\mathrm{1X}}$	100	175	250	200	150	200	150	200
naive MIDEA	750	625	1300	4700	1100	1175	1625	1875

Fig. 5. Population sizes used for the combinatorial problems.

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 $[1; 10]$.

Statistically Significant Improvement Matrices	$D_{\mathcal{P}_F \to S}$						*Front Spread* **FS**						*Front Occ.* **FO**					
	SPEA$^{\mathrm{UX}}$	SPEA$^{\mathrm{1X}}$	NSGA-II$^{\mathrm{UX}}$	NSGA-II$^{\mathrm{1X}}$	naive MIDEA	**Sum**	SPEA$^{\mathrm{UX}}$	SPEA$^{\mathrm{1X}}$	NSGA-II$^{\mathrm{UX}}$	NSGA-II$^{\mathrm{1X}}$	naive MIDEA	**Sum**	SPEA$^{\mathrm{UX}}$	SPEA$^{\mathrm{1X}}$	NSGA-II$^{\mathrm{UX}}$	NSGA-II$^{\mathrm{1X}}$	naive MIDEA	**Sum**
SPEA$^{\mathrm{UX}}$	0	-1	-2	0	-8	-11	0	-4	1	0	-4	-7	0	-4	6	6	-2	6
SPEA$^{\mathrm{1X}}$	1	0	-4	0	-8	-11	4	0	8	7	-3	16	4	0	8	8	2	**22**
NSGA-II$^{\mathrm{UX}}$	2	4	0	4	-7	3	-1	-8	0	1	-8	-16	-6	-8	0	-1	-3	-18
NSGA-II$^{\mathrm{1X}}$	0	0	-4	0	-8	-12	0	-7	-1	0	-8	-16	-6	-8	1	0	-5	-18
naive MIDEA	8	8	7	8	0	**31**	4	3	8	8	0	**23**	2	-2	3	5	0	8

Fig. 6. Number of times an improvement was found to be statistically significant in the $D_{\mathcal{P}_F \to S}$, FS and FO performance indicators, 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.

Optimization problem constraints. The set covering, knapsack and minimal spanning tree problems have constraints. To deal with them, we can use a repair mechanism to transform infeasible solutions into feasible solutions. Another approach is based on the notion of constraint–domination introduced by Deb *et al.* [4]. This notion allows to deal with constrained multi–objective problems in a general fashion. A solution $\boldsymbol{z}^0$ is said to constraint–dominate solution $\boldsymbol{z}^1$ if any of the following is true:

1. Solution $\boldsymbol{z}^0$ is feasible and solution $\boldsymbol{z}^1$ is infeasible
2. Solutions $\boldsymbol{z}^0$ and $\boldsymbol{z}^1$ are both infeasible, but $\boldsymbol{z}^0$ has a smaller overall constraint violation

3. Solutions z^0 and z^1 are both feasible and $z^0 \succ z^1$

The overall constraint violation is the amount by which a constraint is violated, summed over all constraints. We have used this principle for the set covering problem. For the knapsack problem, an elegant repair mechanism was proposed earlier by Zitzler and Thiele [13]. 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 latter two problems.

Knapsack repair mechanism. If a solution violates a constraint, the repair mechanism iteratively removes items until all constraints 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.

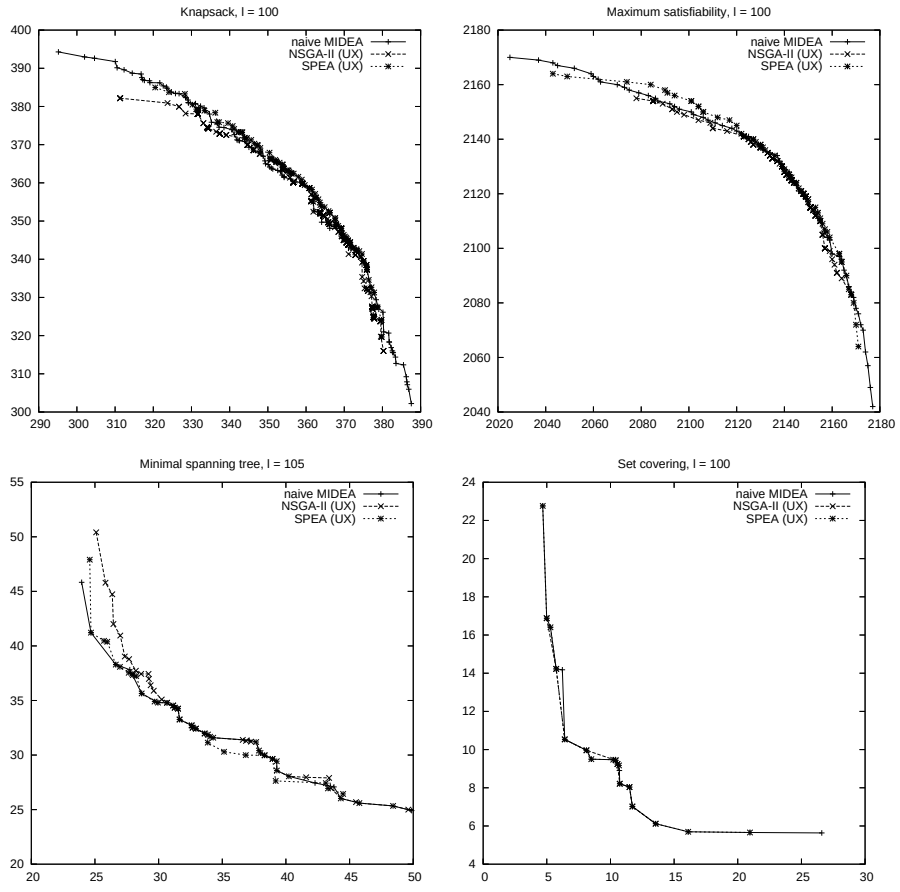

Fig. 7. Pareto fronts over 50 runs on all tested problems with dimensionality $l = 100$.

Minimal spanning tree repair mechanism. 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.

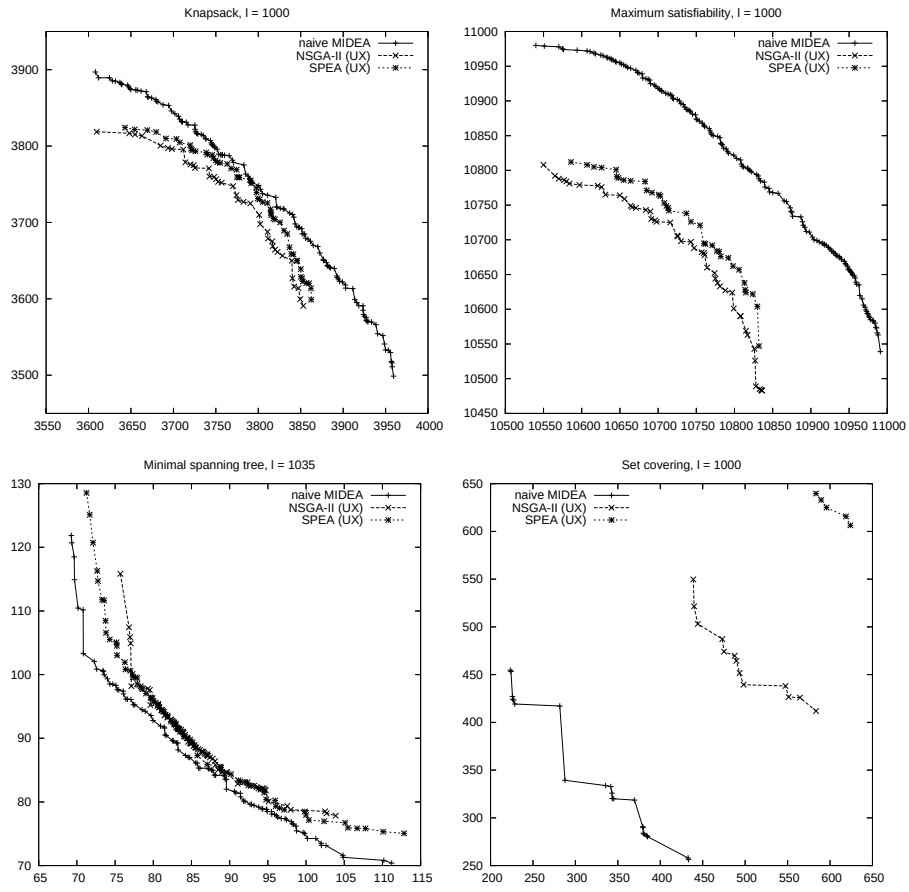

Fig. 8. Pareto fronts over 50 runs on all tested problems with dimensionality $l = 1000$.

General algorithmic setup. We ran every algorithm 50 times on each problem. In any single run we chose to allow a maximum of $20 \cdot 10^3$ evaluations for the problems of dimensionality $l = 100$ and a maximum of $100 \cdot 10^3$ evaluations for

the problems of dimensionality $l = 1000$. As a result of imposing the restriction of a maximum of evaluations, a value for the population size n exists for each MOEA such that the MOEA will perform best. For too large population sizes, the search will become a random search and for too small population sizes, there is not enough information to perform adequate model selection and induction. We therefore increased the population size in steps of 25 to find the best results. To actually select the best population size, we selected the result with the lowest value for the $\boldsymbol{D}_{\mathcal{P}_F \to \mathcal{S}}$ indicator.

Algorithms. We tested three MOEAs. In the following we will describe the details that are required in addition to the details given in earlier sections for constructing the actual MOEAs that we will use for testing.

1. For SPEA, we used uniform crossover and one–point crossover with a probability of 0.8. Bit–flipping mutation was used in combination with either of these recombination operators with a probability of 0.01. These settings were used previously by the SPEA authors [11]. We allowed the size of the external storage in SPEA to become as large as the population size.
2. For NSGA–II, we used the same crossover and mutation operators as above.
3. For the naive MIDEA, we used the leader clustering algorithm in the objective space 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 factorization selection 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. For the truncation percentile, we used the often used value $\tau = 0.3$.

3.4 Results

To compare the MOEAs, we investigated their average performance with respect to performance indicators introduced in Section 3.2. For the $\boldsymbol{D}_{\mathcal{P}_F \to \mathcal{S}}$ performance indicator, we used the Pareto front over all results obtained by all MOEAs.

For each of the performance indicators, we computed their average and standard deviation over the 50 runs to get an assessment of their performance. The averages are tabulated in Figures 2 through 4 (standard deviations can be found in the technical report). The best results are written in boldface. The population sizes that led to the best performance, are tabulated in Figure 5. 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 investigate these significances, 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 verified for each pair of algorithms whether the average obtained performance indicator 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 Figure 6. 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. A less detailed summary of the statistical significance tests is shown in Figure 9. In this figure histograms are used to indicate the sum of the results of the statistical significance tests for each algorithm compared with all other algorithms. The histogram represents the sums for the different tested dimensionalities and their average.

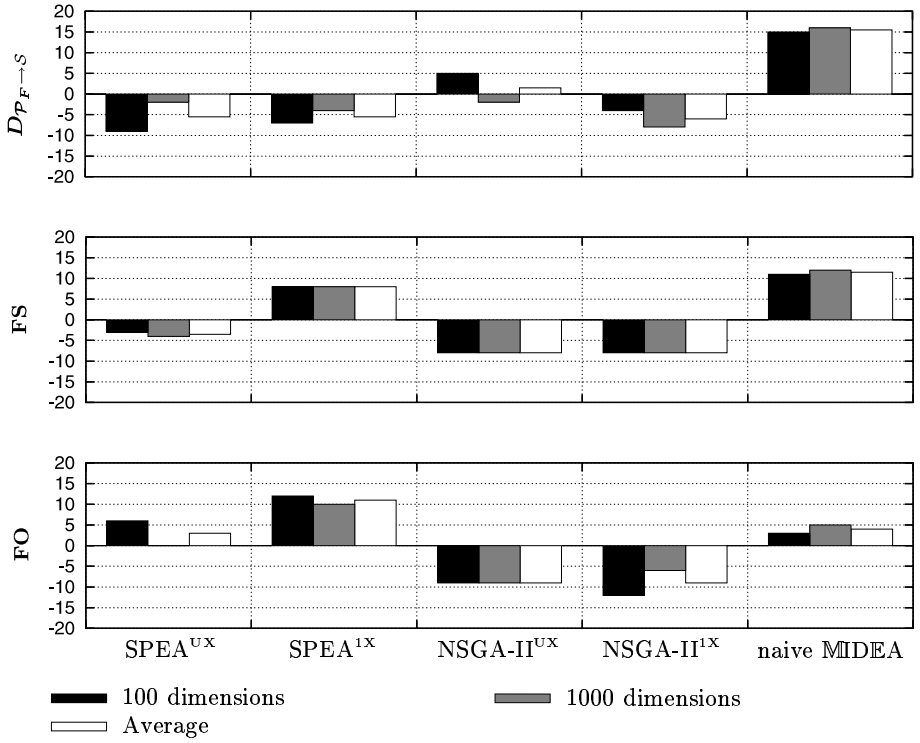

Fig. 9. A summary of the results of the statistical hypothesis tests performed for each pair of algorithms. For each algorithm, the sum of the outcome of the statistical hypothesis tests is shown for the combinatorial problems for each dimensionality separately. Furthermore, the average of these values is also shown, which serves as a global indicator of the performance of an algorithm relative to the other tested algorithms.

3.5 Discussion

The naive MIDEA performs obviously better when the dimensionality of the problem becomes larger. This is most likely due to the efficient diversity exploration and preservation in MIDEA. As the dimensionality of the problem goes up, the parameter search space becomes larger and the number of solutions in the objective space becomes larger as well. In Figures 7 and 8 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 the naive MIDEA obtains non–dominated solutions at the outer ends of the front for the knapsack problem with $l = 100$.

The naive MIDEA is arguably a very effective algorithm on the test suite used. Moreover, the naive MIDEA runs quickly, even for problems with many variables. The experimental results indicate that clustering the objective space to construct mixture probability distributions in MIDEAs leads to efficient MOEAs, even for simple univariate probabilistic models. It can be expected that the use of clustering is also an appealing technique for more traditional MOEAs that do not use probabilistic models. Actually, a MOEA that applies uniform crossover and restricted mating within the clusters should behave rather similar as the naive MIDEA. Whether one is 'more baseline' as the other seems to be a matter of personal taste.

4 Conclusions

In this paper we have presented the naive MIDEA for multi–objective optimization. The naive MIDEA clusters the selected solutions in the objective space, after which it estimates a univariate factorization in each cluster separately. New solutions are then drawn from the so–obtained mixture probability distribution. The naive MIDEA is a specific instance of the algorithmic framework MIDEA which is a general form of an EDA for multi–objective optimization in which a probabilistic model is learned. For the specific task of multi–objective optimization, the use of mixture distributions obtained by clustering the objective space has been observed to stimulate the desirable parallel exploration along the Pareto front. The naive MIDEA has only little computational overhead since clustering in the objective space can be done very fast as can the estimation of a univariate factorization. Furthermore, although no further exploitation of dependencies between a problem's variables is used in the naive MIDEA, the results obtained compare favorably to results obtained with algorithms in which clustering the objective space is not used. Concluding, the naive MIDEA has been found to be a fast, easy–to–use and effective algorithm for multi–objective optimization. Considering its simplicity, speed, and effectiveness the algorithm might play a role as baseline algorithm for MOEAs.

References

1. P. A. N. Bosman and D. Thierens. Multi–objective optimization with diversity preserving mixture–based iterated density estimation evolutionary algorithms. *International Journal of Approximate Reasoning*, 31:259–289, 2002.
2. P. A. N. Bosman and D. Thierens. The balance between proximity and diversity in multi–objective evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 7:174–188, 2003.
3. 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 et al., editor, *Parallel Problem Solving from Nature – PPSN VI*, pages 849–858. Springer, 2000.
4. 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, Berlin, 2001. Springer–Verlag.
5. C. M. Fonseca and P. J. Fleming. An overview of evolutionary algorithms in multiobjective optimization. *Evolutionary Computation*, 3(1):1–16, 1995.
6. G. Harik, F. Lobo, and D. E. Goldberg. The compact genetic algorithm. In *Proceedings of the 1998 IEEE International Conference on Evolutionary Computation*, pages 523–528. IEEE Press, 1998.
7. J. Knowles and D. Corne. On metrics for comparing non–dominated sets. In *Proceedings of the 2002 Congress on Evolutionary Computation CEC 2002*, pages 666–674, Piscataway, New Jersey, 2002. IEEE Press.
8. H. Mühlenbein and G. Paaß. From recombination of genes to the estimation of distributions I. binary parameters. In A. E. Eiben et al., editor, *Parallel Problem Solving from Nature – PPSN V*, pages 178–187. Springer, 1998.
9. D. Thierens and P. A. N. Bosman. Multi–objective mixture–based iterated density estimation evolutionary algorithms. In L. Spector et al., editor, *Proceedings of the GECCO–2001 Genetic and Evolutionary Computation Conference*, pages 663–670, San Francisco, California, 2001. Morgan Kaufmann.
10. D. A. Van Veldhuizen. *Multiobjective Evolutionary Algorithms: Classifications, Analyses, and New Innovations.* PhD thesis, Graduate School of Engineering of the Air Force Institute of Technology, WPAFB, Ohio, 1999.
11. E. Zitzler, K. Deb, and L. Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary Computation*, 8(2):173–195, 2000.
12. E. Zitzler, M. Laumanns, L. Thiele, C. M. Fonseca, and V. Grunert da Fonseca. Why quality assessment of multiobjective optimizers is difficult. In W. B. Langdon et al., editor, *Proceedings of the 2002 Genetic and Evolutionary Computation Conference*, pages 666–674, San Francisco, California, 2002. Morgan Kaufmann.
13. 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.