PolyEDA: Combining Estimation of Distribution Algorithms and Linear Inequality Constraints

Jörn Grahl and Franz Rothlauf

Working Paper 2/2004
January 2004

Working Papers in Information Systems

University of Mannheim
Department of Information Systems 1
D-68131 Mannheim/Germany
Phone +49 621 1811691, Fax +49 621 1811692
E-Mail: wifo1@uni-mannheim.de
Internet: `http://www.bwl.uni-mannheim.de/wifo1`

PolyEDA: Combining Estimation of Distribution Algorithms and Linear Inequality Constraints

Jörn Grahl
Dept. of Information Systems 1
University of Mannheim
D-68131 Mannheim/Germany
jgrahl@rumms.uni-mannheim.de

Franz Rothlauf
Dept. of Information Systems 1
University of Mannheim
D-68131 Mannheim/Germany
rothlauf@uni-mannheim.de

January 26, 2004

Abstract

Estimation of distribution algorithms (EDAs) are population-based heuristic search methods that use probabilistic models and which have been successfully applied to continuous optimization problems. When applied to constrained optimization problems, most EDAs (as well as genetic algorithms) handle constraints by penalizing invalid solutions. This paper presents PolyEDA, a new EDA approach that is able to directly consider linear inequality constraints by using Gibbs sampling. Gibbs sampling allows us to sample new individuals inside the boundaries of the polyhedral search space described using a set of linear inequality constraints by iteratively constructing a density approximation that lies entirely inside the polyhedron. Due to its ability to consider linear inequality constraints, PolyEDA can be used for highly constrained optimization problems, where even the generation of valid solutions is a non-trivial task. Results for different variants of a constrained Rosenbrock problem show a higher performance of PolyEDA in comparison to a standard EDA using rejection sampling.

1 Introduction

Estimation of distribution algorithms (EDA) are population-based optimization methods that use probabilistic models to guide their search [1]. In contrast to genetic algorithms (GA), EDAs do not use classical search operators, but crossover and mutation are replaced by the following two steps:

1. A probabilistic model is built of selected solutions.
2. New random solutions are sampled from the probabilistic model.

EDAs have shown promising results [2, 3] when used for combinatorial, discrete, and continuous problems. When using EDAs for continuous real-world planning and optimization problems, there are often a number of additional

problem-specific constraints which significantly affect the performance of optimization algorithms [4]. For example, many variables in real-world problems have lower and upper bounds and the feasible regions of the search space are constrained using linear inequality constraints [5]. Linear inequality constraints are a powerful approach for describing problem-specific knowledge, are common and well understood in the field of mathematical programming, and form the basis of many traditional optimization techniques. During the last few years many methods have been proposed for handling constraints when using GAs or EDAs. The most common are: (1) methods that repair invalid solutions, (2) methods that use penalties, (3) methods which distinguish between feasible and infeasible solutions, and (4) methods that are based on decoders (compare [6]). The general idea when dealing with constraints is to penalize invalid solutions.

The purpose of this paper is to develop a new EDA approach, PolyEDA, that is able to consider linear inequality constraints without penalizing infeasible solutions. PolyEDA is designed in such a way that no infeasible solutions are created during the optimization process. Therefore, the different parts of an EDA like the construction of a probabilistic model, or the sampling of new solutions, must be modified such that the inequality constraints are satisfied. Consequently, PolyEDA uses factorizations of truncated multi-normal distributions that consider the inequality constraints for the building of probabilistic models. Furthermore, the sampling of new solutions according to the given constraints is done using Gibbs sampling [7, 8]. Gibbs sampling allows us to sample inside the boundaries of the polyhedral search space described using a set of linear inequality constraints by iteratively constructing a density approximation which lies entirely inside the polyhedron. Therefore, the Gibbs sampler uses well known univariate conditional distributions instead of calculating the highly complicated multivariate constrained densities directly. In contrast to standard EDAs, PolyEDA is able to optimize highly constrained optimization problems, and example results on constrained Rosenbrock problems show a higher performance of PolyEDA in comparison to a standard EDA using rejection sampling.

The paper is structured as follows. The following section introduces some basic results from polyhedral theory that are relevant for linear inequality constraints. Section 2.2 outlines the Gibbs sampling approach for multivariate random number generation considering linear constraints. In section 3, the functionality of PolyEDA is outlined by discussing all of its elements, namely techniques for sampling the first generation (Sect. 3.2), the principles of model-selection (Sect. 3.3), the estimation of parameters (Sect. 3.4), and the generation of new solutions (Sect. 3.5). In section 4, PolyEDA is applied to the constrained Rosenbrock problem and its performance is compared to a standard EDA. The paper ends with some concluding remarks and a short outlook into future work.

The notation and symbols we use throughout this paper are based on the notation used in the IDEA-Framework (see [9]).

2 Polyhedrons and Gibbs Sampling

2.1 Polyhedral Theory

In most linear programming approaches, the feasible search space is described by using a set of linear inequality constraints. Using certain assumptions, the set of points in the search space that is feasible under a finite set of inequalities is a polyhedron. Many classical optimization methods like the simplex algorithm [10], cutting planes, or branch and cut techniques [11], are based on such a polyhedral description of the search space. The mathematical grounding of these approaches is the polyhedral theory. We introduce some basic definitions [11] from polyhedral theory that are relevant for PolyEDA and which are used in the later sections.

Definition 1 *A polyhedron $P \subseteq R^n$ is a set of points that satisfy a finite number of linear inequalities. The linear inequalities are described using $P = \{y \in R^n : Ax \leqslant c\}$, where (A, c) is an $m \times (n+1)$ matrix.*

A polyhedron is a convex set. Thus, the set of points that are feasible under a set of linear inequalities is a convex polyhedron. In the following we want to use linear inequalities with the structure

$$\boldsymbol{a} \leqslant \boldsymbol{D}\boldsymbol{y} \leqslant \boldsymbol{b}, \tag{1}$$

where the $(n \times n)$ matrix $\boldsymbol{D}$ has rank n. The vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ are $(n \times 1)$ each. Equation 1 allows the formulation of maximal n linearly independent inequalities. Because each system of linear inequalities can be transformed into a system with the structure $Ax \leqslant c$ the set of points that are feasible under (1) is a convex polyhedron.

2.2 Gibbs Sampling

Gibbs sampling is a statistical method that allows us to generate random variables from highly complicated distributions without calculating their density functions. Complex calculations like the recurrent evaluation of normal integrals can be replaced by a series of computational easier calculations. Gibbs sampling was introduced by [7]. Later, the approach was modified and improved by [12]. A good introduction into Gibbs sampling can be found in [13]. One of the first applications of Gibbs sampling is shown in [8], where it is used for the digital restoration of images. In this paper, the Gibbs sampler is used to generate random variables from multivariate normal distributions that are subject to linear constraints [14].

2.2.1 The Gibbs Algorithm.

The following paragraphs describe how Gibbs sampling can be used for creating multivariate random numbers.

We assume that we want to draw an $n-$dimensional random vector $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)'$ from a multivariate density $f(\boldsymbol{x})$. In addition, we assume (e.g.

due to the complexity of the density function) that there is no method available for performing this task directly. Gibbs sampling can be used if all of the following conditional distributions are known:

$$x_i | \{x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n\} \sim P_i(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n), \tag{2}$$

where $i = 1 \ldots n$ and P_i denotes the conditional distribution of x_i given all other variables x_j $(j \neq i)$. A second condition for using Gibbs sampling is that a method is available that allows the efficient generation of random numbers from the conditional distributions P_i.

Based on these assumptions we can describe the functionality of Gibbs sampling: Let $x^{0'}$ be a $n-$dimensional (starting) point in the multivariate distribution $f(\boldsymbol{x})$. Then, Gibbs sampling iteratively creates the random variables:

$$x_i^1 | \left\{x_1^1, \ldots, x_{i-1}^1, x_{i+1}^0, \ldots, x_n^0\right\} \sim P_i(x_1^1, \ldots, x_{i-1}^1, x_{i+1}^0, \ldots, x_n^0) \quad \forall (i = 1 \ldots n) \tag{3}$$

Having done this for the first time, exactly n random numbers have been generated. The creation of these n random number is the first iteration of the Gibbs sampling algorithm. The following iterations are carried out exactly the same. Therefore, after generating the i'th random number of the j'th iteration, we have:

$$x_i^j | \left\{x_1^j, \ldots, x_{i-1}^j, x_{i+1}^{j-1}, \ldots, x_n^{j-1}\right\} \sim P_i(x_i^j, \ldots, x_{i-1}^{j-1}, x_{i+1}^{j-1} \ldots, x_n^{j-1}) \tag{4}$$

After completing the jth iteration, the random vector has the following structure:

$$\boldsymbol{x}^{j'} = (x_1^j, \ldots, x_n^j)' \tag{5}$$

The central element of Gibbs sampling is that with growing j the distribution of $\boldsymbol{x}^{j'}$ converges against the correct multivariate distribution of $\boldsymbol{x}$ [15].

It should be noted, that it is not necessary to calculate this multivariate density directly. Instead, random numbers are drawn from conditional distributions. As this is often less complex, Gibbs sampling has become a popular method in many areas of statistics. The second aspect is that this vector does not constitute a complete sample, but merely an $n-$dimensional point of the multivariate density $f(\boldsymbol{x})$. In order to generate a sample of size k, the above steps can be repeated k times. Alternative methods for generating samples of a specific size can be found in [13].

2.2.2 Sampling from Truncated Multinormal Distributions.

The method outlined in the previous paragraphs can be used to generate random vectors according to complex multivariate distributions. In the following paragraphs, we explain a Gibbs sampler for generating i.i.d. (identically, independently distributed) random vectors from multivariate normal distributions that are subject to linear constraints. The algorithm has been developed by [14]. For technical and mathematical details on truncated multinormal distributions compare [16, p. 204].

Supposing we want to generate $n-$dimensional random vectors $\boldsymbol{x}$ that follow a multivariate normal distribution, but at the same time consider linear inequality constraints:

$$\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}), \quad \text{s.t.} \quad \boldsymbol{a} \leqslant \boldsymbol{D}\boldsymbol{x} \leqslant \boldsymbol{b}, \tag{6}$$

where $\boldsymbol{x} = (x_0, x_1, \ldots, x_{n-1})$ and $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ is the $n-$variate multinormal distribution with mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$. Furthermore, $-\infty$ and $+\infty$ may be elements of $\boldsymbol{a}$ and $\boldsymbol{b}$ and the matrix $\boldsymbol{D}$ is $(n \times n)$ and of rank n. This allows the formulation of maximal n linearly independent inequality relationships. Generating random numbers from (6) is equal to the generation of random numbers from

$$\boldsymbol{z} \sim \mathcal{N}(0, \boldsymbol{T}), \qquad \boldsymbol{\alpha} \leqslant z \leqslant \boldsymbol{\beta} \tag{7}$$

with

$$\boldsymbol{T} = \boldsymbol{D}\Sigma\boldsymbol{D}', \quad \boldsymbol{\alpha} = \boldsymbol{a} - \boldsymbol{D}\boldsymbol{\mu}, \quad \text{and} \quad \boldsymbol{\beta} = \boldsymbol{b} - \boldsymbol{D}\boldsymbol{\mu}. \tag{8}$$

The vector $\boldsymbol{x}$ can be calculated from $\boldsymbol{z}$ as

$$\boldsymbol{x} = \boldsymbol{\mu} + \boldsymbol{D}^{-1}\boldsymbol{z}$$

Many different methods have been developed for the sampling of random values from the distribution described by 7. An overview of such methods can be found in [17]. A problem all methods have to solve is that, in general, iterated calculations of the normal integral are necessary when sampling random numbers according to equation 7. Also, these methods are most often incapable of generating i.i.d. samples. By using a Gibbs sampler, i.i.d. samples from 7 can be generated without performing iterated calculations of the normal integral [14]. A necessary condition for using a Gibbs-sampler is previous knowledge regarding the conditional distribution of one random variable x_i on all other random variables x_j, $(i \neq j)$. These conditional distributions are truncated univariate normal. As a result, when using a Gibbs sampler the recurrent evaluation of normal integrals can be replaced by repeated sampling from a truncated univariate normal distribution. This is a less complex problem, and highly efficient techniques for this purpose exist [14].

3 PolyEDA: Combining EDAs with Linear Constraints

This section describes PolyEDA, an EDA that is able to consider a set of linear inequality constraints during optimization. It uses a continuous problem representation and a solution is represented by the vector $\boldsymbol{y} = (y_1, y_2, \ldots, y_n)'$. PolyEDA is the result of combining EDA with a system of linear inequalities of the type

$$\boldsymbol{a} \leqslant \boldsymbol{D}\boldsymbol{y} \leqslant \boldsymbol{b}. \tag{9}$$

The set of $n-$dimensional points that satisfy (9) is a polyhedron (compare section 2.1). PolyEDA is able to sample new solutions according to the boundaries of this search space and to consider problem-specific knowledge. Modeling

problem-specific knowledge as a set of linear inequality constraints is a common technique in evolutionary computation [6] and mathematical programming [18].

To consider linear constraints in PolyEDA, the elements of the EDA (model selection, parameter estimation, and sampling) must be modified. The following sections outline the necessary adaptations.

3.1 Probabilistic Model

In PolyEDA we use a probabilistic model that consists of factorizations of truncated multinormal distributions. This probabilistic model is based on multivariate normal factorizations as used in the IDEA-Framework [9] and additionally incorporates a set of linear inequality constraints. The truncated distributions reflect the constrained nature of the search space. All $n-$dimensional points that are feasible under the linear inequality constraints have positive, non-negative probabilities. All infeasible points have a probability of 0. Then, the model can be formulated as

$$\begin{aligned} P_{(\boldsymbol{v},\boldsymbol{\theta})}(\boldsymbol{Y})(\boldsymbol{y}) &= \prod_{i=0}^{|\boldsymbol{v}|-1} P^{\mathcal{N}}_{(\boldsymbol{\mu}_{v_i},\boldsymbol{\Sigma}_{v_i})}(Y_{\boldsymbol{v}_i})(\boldsymbol{y}) \\ s.t. \qquad & \boldsymbol{a} \leq \boldsymbol{D}\boldsymbol{y} \leq \boldsymbol{b}, \end{aligned} \tag{10}$$

where a solution is represented by a vector $\boldsymbol{y} = (y_0, y_1, \ldots, y_{(n-1)})'$ of random variables. $\boldsymbol{y}$ can be separated into subsets of random variables and all subsets follow multivariate truncated normal distributions. Random variables of different subsets are independent of each other; random variables in the same subset depend on each other. For indicating the different subsets we use node vectors $\boldsymbol{v}_i$. The entries of each node vector are the indices of the variables of one subset (all variables that depend on each other) and $\boldsymbol{v}_i \wedge \boldsymbol{v}_j = ()$ for all $i \neq j$. This means, that each random variable occurs only in one node vector. The partition vector $\boldsymbol{v}$ consists of all $\boldsymbol{v}_i$ and describes the structure of the whole factorization. The set of parameters $\boldsymbol{\theta}$ consists of the mean vectors $\boldsymbol{\mu}_{\boldsymbol{v}_i}$ and covariance matrices $\boldsymbol{\Sigma}_{\boldsymbol{v}_i}$ of each node vector.

The matrix $\boldsymbol{D}$ is $(n \times n)$ and of rank n. Individual elements of $\boldsymbol{a}$ and $\boldsymbol{b}$ must be real values, using $\pm\infty$ is not allowed. This allows the formulation of n inequality constraints.

3.2 Sampling the Initial Population

In EDA, the first population is sampled uniformly over all possible solutions. Therefore, when using the model from equation 10, we have to uniformly sample i.i.d. solutions inside the polyhedron, which is described by the inequality constraints $\boldsymbol{a} \leq \boldsymbol{D}\boldsymbol{y} \leq \boldsymbol{b}$. To the best of our knowledge, currently no efficient method is available for doing this. Thus, for sampling the initial population, we use rejection sampling, which means that we sample uniformly inside a cube that entirely covers the polyhedron. We calculate lower and upper bounds for each of the y_i and all randomly sampled solutions that lie outside the polyhedron

are rejected and sampled again; all solutions that lie inside the polyhedron are accepted and make up the initial population.

As the performance of rejection-sampling depends mainly on its acceptance rate, we seek to maximize the number of accepted solutions by choosing close bounds for the y_i. The smallest rectangular region that entirely covers the polyhedral search space can by calculated by the Fourier-Motzkin-elimination (see [19]). This technique eliminates variables from the inequality system $\boldsymbol{a} \leq \boldsymbol{D}\boldsymbol{y} \leq \boldsymbol{b}$. In order to generate the lower and upper bounds a_i and b_i for the random variable y_i, all variables $y_j, j \neq i$ are eliminated from the system, leaving a single inequality $a_i \leqslant y_i \leqslant b_i$. Doing this n times we get n inequalities which are used as lower and upper bounds for the sampling of the initial population.

3.3 Model Selection

In the model selection step, the structure $\boldsymbol{v}$ of the factorization that fits best to the current population is searched. If linear inequalities are considered, the factorization is subdivided into a variable part and a fixed part. The variable part depends on statistical properties of a population and describes the interactions between the variables due to the fitness function. The fixed part of the factorization depends on the linear inequalities and describes the interactions between different elements of $\boldsymbol{v}$.

To generate the variable part of the factorization, we use the greedy factorization selection method outlined in [9]. This heuristic method uses local search steps, beginning with a univariate factorization. The decision between two candidate factorizations is based on a negative log-likelihood metric that penalizes complexity of the factorization. The variable part of the factorization has to be generated in every iteration of the EDA. It reflects the dependencies between the random variables in the current area of the search space.

The fixed part of the factorization can be generated from the inequality system $\boldsymbol{a} \leq \boldsymbol{D}\boldsymbol{y} \leq \boldsymbol{b}$. To do this, the lines of the matrix $\boldsymbol{D}$ have to be examined. Let the sets S_j $(j = 1 \ldots n)$ denote the positions in which the matrix $\boldsymbol{D}$ has entries $\neq 0$ in the jth line. Then, all variables $y_{(S_j)}$ depend on each other following a truncated multivariate distribution.

The first n node vectors are determined by the n sets of variables $y_{(S_j)}$. Then, it is checked whether the same random variables occur in more than one node vector $\boldsymbol{v}_i$. If this is the case, these vectors are merged and duplicate entries are deleted. This step is repeated, until every random variable appears in only one node vector. It should be noted, that the fixed part of the factorization needs only to be generated once. Since the matrix $\boldsymbol{D}$ does not change during the optimization, the fixed part can be generated before the optimization and the fixed factorization remains unchanged in later generations. It reflects the dependencies that are necessary in order to consider the linear constraints.

After generating the variable and the fixed part of the factorization, these two parts are combined to create the complete factorization. All dependencies between the variables that are a result of the fixed parts of the factorization are considered first. Then, the interactions that come from the variable part are considered by checking whether they do not already occur in the fixed part.

Finally, it is examined whether some random variables occur in more than one node vector. If this is the case, these vectors are merged and duplicated entries are deleted. This step is repeated until every random variable appears in only one node vector.

3.4 Estimation of Parameters

The set of parameters that has to be estimated consists of the mean vector $\boldsymbol{\mu}$ and the covariance matrix $\boldsymbol{\Sigma}$. Well-known maximum likelihood estimators exist for the estimation of $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ from a sample $\boldsymbol{S} = (\mathcal{S}_0, \mathcal{S}_1, \ldots, \mathcal{S}_{|\mathcal{S}|-1})$ [16].

$$\hat{\boldsymbol{\mu}}_{\boldsymbol{v}_i} = \frac{1}{|\boldsymbol{\mathcal{S}}|} \sum_{j=0}^{|\boldsymbol{\mathcal{S}}|-1} (\boldsymbol{\mathcal{S}}_j)_{\boldsymbol{v}_i}, \tag{11}$$

$$\hat{\boldsymbol{\Sigma}}_{\boldsymbol{v}_i} = \frac{1}{|\boldsymbol{\mathcal{S}}_j|} \sum_{j=0}^{|\boldsymbol{\mathcal{S}}|-1} ((\boldsymbol{\mathcal{S}}_j)_{\boldsymbol{v}_i} - \hat{\boldsymbol{\mu}}_{\boldsymbol{v}_i})((\boldsymbol{\mathcal{S}}_j)_{\boldsymbol{v}_i} - \hat{\boldsymbol{\mu}}_{\boldsymbol{v}_i})' \tag{12}$$

Unfortunately, these estimators are designed for estimation from (multi)normal samples and do not consider truncated normal samples. Nonetheless, we use these estimators in PolyEDA being fully aware of the fact that modified estimators need to be developed that consider the truncation of the distributions. We believe that developing estimators that consider truncated distributions would result in a significant enhancement of PolyEDA.

3.5 Sampling New Solutions

The sampling of new solutions according to equation 10 is not trivial as i.i.d. random vectors must be generated from multinormal distributions that consider the linear inequality constraints $\boldsymbol{a} \leq \boldsymbol{Dy} \leq \boldsymbol{b}$. Therefore, PolyEDA uses the Gibbs sampling algorithm outlined in section 2.2.2 for sampling new solutions. This ensures that only feasible solutions are sampled and new populations lie entirely inside the given boundaries of the search space. As a result, PolyEDA does not need to use penalties to consider linear inequality constraints.

4 Experiments

In the following paragraphs PolyEDA is exemplarily applied to some versions of the Rosenbrock problem. We want to illustrate how PolyEDA considers linear inequality constraints during optimization in the probabilistic model and we show that the direct sampling of feasible solutions results in higher performance in comparison to standard EDAs using rejection sampling. We are aware of the fact that to fully evaluate the performance of PolyEDA more exhaustive tests on a large number of different test problems are necessary. However, as the purpose of this paper is on introducing and explaining the functionality of PolyEDA we postpone exhaustive tests until future publications.

4.1 Problem Definition

The Rosenbrock's function is a highly non-linear function that is commonly used for the test of numerical optimization methods. Rosenbrock's function is defined as

$$\text{minimize:} \quad f(\boldsymbol{y}) = \sum_{i=0}^{l-2} \left[100 \left(y_{i+1} - y_i^2 \right)^2 + (y_i - 1)^2 \right], \tag{13}$$

where l is the dimensionality of the problem. The optimal solution $\boldsymbol{y}^\star$ has fitness $f(\boldsymbol{y}^\star) = 0$ and is located at $y_i^\star = 0$ $(i = 1 \dots l)$. In our test problem Rosenbrock's function is defined for $y_i \in [-5.12; 5.12]$. We used three test problems of dimension ten, 20, and 40. In each of these test problems, l linear inequalities of the following type have to be considered:

$$1.0 \leqslant y_i \leqslant 2.0 \quad \forall i = 1 \dots l \tag{14}$$

These inequalities make up a rectangular feasible region. The optimal solution of Rosenbrock's function is at the edge of the search space.

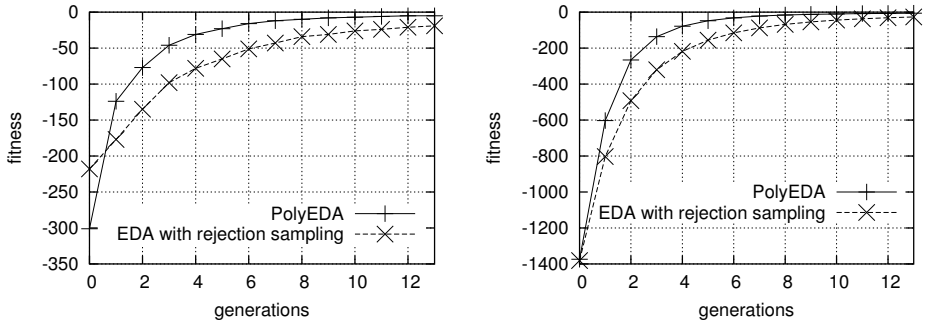
4.2 Experimental Results

In our experiments, we compared a standard EDA to PolyEDA outlined in section 4.1. For both EDAs we used a population size of $N = 300$ of which the 100 best solutions are selected. A statistical model (compare section 3.1) is build from these 100 best solutions and in the next generation 300 offspring are generated according to this model using a sampling algorithm. In PolyEDA we used Gibbs sampling as described in section 2.2.2 for the creation of new solutions. The number of iterations j that has been used by the Gibbs sampler to approximate the truncated distributions has been set to $j = 100$.

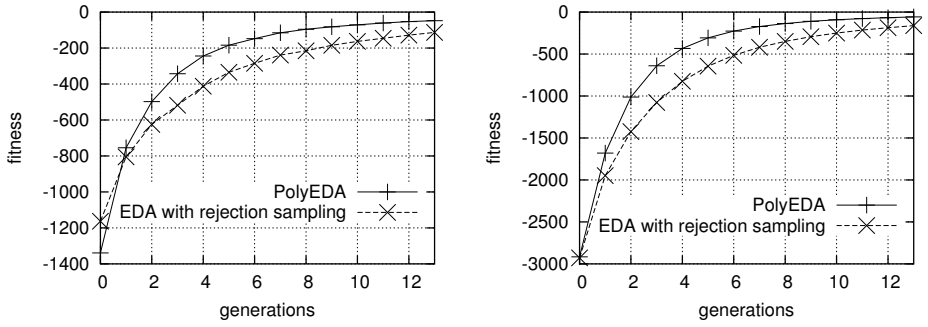
Both, the standard EDA and PolyEDA use factorizations of multivariate normal distributions as outlined in [9]. The only difference lies in considering the linear inequalities. PolyEDA considers the linear inequality constraints when sampling new solutions by using Gibbs-sampling. In the standard EDA new solutions are sampled using the unconstrained multi-normal distributions and neglecting the linear constraints. To consider the additional constraints newly generated solutions that are infeasible are rejected and not considered for the creation of the statistical model (rejection-sampling). This means, that invalid solutions (solutions that were infeasible under the linear inequalities outlined in section 4.1), were rejected and sampled again (until the population is filled).

PolyEDA and the standard EDA with rejection-sampling have been compared on all three instances of the constrained Rosenbrock function. We performed 15 runs for every problem instance. Fig. 1 shows the mean of the average fitness of the best solution in a population (left) and the average fitness of the population (right) over the number of generations for different sizes of the Rosenbrock function.

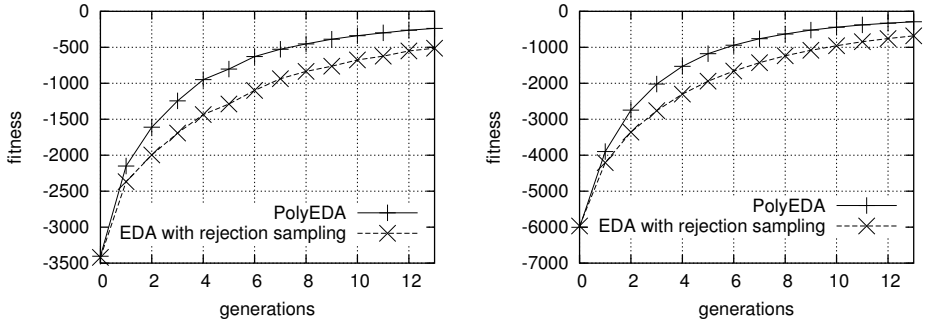
The plots show that PolyEDA outperforms a standard EDA using rejection sampling for the constraint Rosenbrock problem. Although both approaches



(a) $l = 10$



(b) $l = 20$



(c) $l = 40$

Figure 1: The plots show the mean of the average fitness of the best solution (left) and the average fitness of a population (right) over the number of generations. Results are presented for the constraint Rosenbrock function of size $l = 10$ (Fig. 1(a)), $l = 20$ (Fig. 1(b)), and $l = 40$ (Fig. 1(c)). The constraints are chosen such that the optimal solution is at the edge of the feasible solution space. The results show that PolyEDA outperforms a standard EDA using rejection sampling independently of the size of the problem.

Table 1: Average percentage of infeasible solutions

dimension l	10	20	40
rejection sampling	8.58 %	23.1 %	60.8 %
PolyEDA	0 %		

use the same population size ($N = 300$), PolyEDA results in better results in terms of average population fitness as well as average best fitness. Obviously, using a probabilistic model and a sampling technique that considers the given linear constraints is advantageous and lead to more efficient EDAs.

Standard EDAs using rejection sampling have the problem that many of the sampled individuals are infeasible. Table 1 shows the average ratio of infeasible solutions to all generated solutions. The results are averaged over all fifteen runs and all generations. When applying standard EDA with rejection sampling to the 40-dimensional constraint Rosenbrock function, more than 60 percent of all generated solutions are infeasible which results in a great overhead. This situation can become even worse for highly constraint optimization problems. This problem of standard EDAs illustrates the advantage of PolyEDA which did not generate a single infeasible solution.

5 Conclusions and Further Work

This work presented the functionality of PolyEDA, an EDA that is able to consider linear inequality constraints during the optimization without penalizing infeasible solutions. In section 2.1 the paper reviewed some foundations of polyhedral theory and Gibbs sampling which are necessary for PolyEDA. Section 3 explained in detail PolyEDA and focused on the different aspects like sampling of the first generation, model selection, the estimation of parameters, and the generation of new solutions from the probabilistic model. Finally, section 4 compared exemplarily the performance of PolyEDA to a standard EDA using rejection sampling for some variants of the constrained Rosenbrock function.

PolyEDA is a new type of EDA that allows us to directly consider linear inequality constraints. It was designed in such a way that it avoids the creation of infeasible solutions. The used probabilistic model is based on factorizations of multivariate truncated normal distributions. In this model, all solutions that are feasible under the linear inequality constraints have positive probabilities, solutions that are infeasible have a probability of zero. For the sampling of feasible solutions from this model, a Gibbs sampler is used. Gibbs sampling allows the generation of random values from the multivariate truncated distribution without calculating their density. By using a Gibbs sampler, the complex calculation of the density can be avoided and replaced by the generation of random values from univariate truncated normal distributions.

This paper illustrated exemplarily the performance of PolyEDA for some small examples of the constraint Rosenbrock problem. In further work, PolyEDA should be applied to a larger variety of linearly constraint optimization problems and also compared to different other techniques that can be used for solving con-

strained optimization problems. Furthermore, PolyEDA should be continuously improved. The work presented here focused on the sampling of solutions from multivariate truncated distributions by using Gibbs sampling and neglected a proper estimation of truncated distributions. This aspect should be addressed in future work.

References

[1] Mühlenbein, H., Paaß, G.: From recombination of genes to the estimation of distributions I. Binary parameters. In: Lecture Notes in Computer Science 1411: Parallel Problem Solving from Nature - PPSN IV. (1996) 178–187

[2] Larrañaga, P., Etxeberria, R., Lozano, J.A., Peña, J.M.: Optimization in continuous domains by learning and simulation of Gaussian networks. In Wu, A.S., ed.: Proceedings of the 2000 Genetic and Evolutionary Computation Conference Workshop Program. (2000) 201–204

[3] Pelikan, M., Goldberg, D.E., Lobo, F.: A survey of optimization by building and using probabilistic models. Technical Report IlliGAL Report 99018, University of Illinois at Urbana-Champaign (1999)

[4] Michalewicz, Z.: Heuristic methods for evolutionary computation techniques. Journal of Heuristics **1** (1995) 177–206

[5] Michalewicz, Z., Deb, K., Schmidt, M., Stidsen, T.J.: Towards understanding constraint-handling methods in evolutionary algorithms. In Angeline, P.J., Michalewicz, Z., Schoenauer, M., Yao, X., Zalzala, A., eds.: Proceedings of the Congress on Evolutionary Computation. Volume 1., Mayflower Hotel, Washington D.C., USA, IEEE Press (1999) 581–588

[6] Michalewicz, Z., Schoenauer, M.: Evolutionary computation for constrained parameter optimization problems. Evolutionary Computation **4** (1996) 1–32

[7] Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H., Teller, E.: Equation of state calculations by fast computing machines. Journal of Chemical Physics **21** (1953) 1087–1091

[8] Geman, S., Geman, D.: Stochastic relaxation, gibbs distributions, and the bayesian restauration of images. IEEE Transactions on Pattern Analysis and Machine Intelligence **6** (1984) 721–741

[9] Bosman, P.A.N.: Design an Application of Iterated Density-Estimation Evolutionary Algorithms. PhD thesis, University of Utrecht, Institute of Information and Computer Science (2003)

[10] Vanderbei, R.J.: Linear Programming, Foundations and Extensions. Volume 2 of International Series In Operations Research And Management Science. Kluwer Academic Publishers, Boston (2001)

[11] Nemhauser, G.L., Wolsey, L.A.: Integer and Combinatorial Optimization. Wiley-Interscience Series in Discrete Mathematics and Optimization (1999)

[12] Hastings, W.K.: Monte carlo sampling methods using markov chains and their applications. Biometrika **57** (1970) 97–109

[13] Casella, G., George, E.I.: Explaining the gibbs sampler. The American Statistician **46** (1992) 167–174

[14] Geweke, J.: Efficient simulation from the multivariate normal and student-t distributions subject to linear constraints and the evaluation of constraint probabilities. Technical report, University of Minnesote, Dept. of Economics (1991)

[15] Gelfand, A.E., Smith, A.F.M.: Sampling-based approaches to calculating marginal densities. Journal of the American Statistical Association **87** (1990) 398–409

[16] Kotz, S., Balakrishnan, N., Johnson, N.L.: Continuous Multivariate Distributions. Volume 2 of Wiley Series in Probability and Statistics. John Wiley and Sons (2000)

[17] Hajivassiliou, V.A., McFadden, D. L.and Ruud, P.A.: Simulation of multivariate normal orthan probabilities: Methods and programs. Technical report, M.I.T. (1990)

[18] Williams, H.: Model Building in Mathematical Programming. Volume 4. Auflage. Wiley (1999)

[19] Duffin, R.: On fourier's analysis of linear inequality systems. Mathematical Programming Study **1** (1974) 71–95