Distribution Tree-Building Real-valued Evolutionary Algorithm

Petr Pošík

Faculty of Electrical Engineering, Department of Cybernetics
Czech Technical University in Prague
Technická 2, 166 27 Prague 6, Czech Republic
posik@labe.felk.cvut.cz

Abstract. This article describes a new model of probability density function and its use in estimation of distribution algorithms. The new model, the distribution tree, has interesting properties and can form a solid basis for further improvements which will make it even more competitive. Several comparative experiments on continuous real-valued optimization problems were carried out and the results are promising. It outperformed the genetic algorithm using the traditional crossover operator several times, in the majority of the remaining experiments it was comparable to the genetic algorithm performance.

1 Introduction

This article addresses the problem of the real-valued optimization with box constraints. The stochastic genetic and evolutionary algorithms (GAs, EAs) are often considered to be a very flexible tool for searching the space of potential solutions. Due to the dependencies among the problem variables the ordinary GEAs are often not able to solve the task at hand reliably. One possible solution of this *linkage problem* is presented by the so-called estimation of distribution algorithms (EDAs).

In EDAs, new individuals are not created by means of traditional crossover operators, rather each generation a probabilistic model describing the distribution of good individuals in the search space is created and new individuals are sampled from it. We can view the model-building-and-sampling as a generalized multi-parent crossover operator. The characteristics of the resulting EDA are mainly determined by the probabilistic model used during the evolution. If the model is flexible enough, the algorithm can use the dependencies among the problem variables to guide the search more efficiently than just the plain selection (which guides the search in ordinary EAs). The structure of an EDA is depicted in Fig. 1. The EDA framework emerged in the field of GAs (see e.g. [4], [6] for surveys). In the real-valued optimization we can find 'continuous counterparts' of the algorithms developed originally for the bit strings, some approaches which assume the independence of individual variables (the univariate marginal distribution algorithm, UMDA, see e.g. [7], [8]), and on the other side of the

```
EDAStructure
1   Initialize and evaluate the population
2   Repeat
3      Select individuals which should serve as a basis for model creation
4      Create a probabilistic model of distribution of selected individuals
5      Sample new individuals from created model
6      Insert new individuals into population
7      Mutate some individuals in population (not a typical part of EDA)
8      Evaluate new and modified individuals
9   Until a termination condition is met
```

Fig. 1. Structure of the estimation of distribution algorithm.

spectra there are approaches which use very general kinds of models like the finite mixtures of multivariate normal distributions (see e.g. [3], [1]) or bayesian networks (see [5]) that are able to cover very complex kinds of interactions .

The following sections include the first examination of EDA using the distribution tree model. In section 2, the way of distribution tree construction is described. Section 3 includes the description of carried out experiments along with their results. Finally, in section 4 you will find a short summary and conclusions.

2 Distribution Trees

The ideal probabilistic model for EDAs should have the following features:

- *Easy and fast to build.* The model is built every generation; a fast way to create it must exist.
- *Generative.* We need to create new population members in accordance with the distribution encoded in the model. If we do not know how to sample from the model, we are not able to create new individuals.
- *Flexible enough.* The model must be able to cover the interactions between variables (at least to some extent). It should also be able to decompose or factorize the problem if possible.

The distribution tree (DiT[1]) is an attempt to construct such kind of model. It is based mainly on the Classification and Regression Trees (CART) framework introduced by Breiman [2], but it differs from CART in several important aspects.

On the contrary to CART, the primary objective of DiT is not to present a model to classify or predict new, previously unseen data points, but rather to generate new individuals so that the distribution of them in the search space is very similar to that of the original data points.

[1] In this article for the distribution tree model the abbreviation 'DiT' is used in order to prevent messing it with the 'DT' abbrev. commonly used for the decision trees.

When presented with the training set of data (the population members selected for mating), the distribution tree is built by recursively searching for the 'best' axis-parallel split. The leaf nodes of DiT present a complete partitioning of the search space. The shape of each partition can be described as a multi-dimensional hyper-rectangle whose edges are aligned with the coordinate axes. Thus, the final model forms a mixture of multivariate uniform distributions.

When compared to other structure-building algorithms, e.g. the Ocenasek's MBOA [5], the DiT is much simpler model. The tree here is a kind of temporary structure; after the model is created we can forget the tree, only the leaf nodes are important because they form a mixture of uniform distributions. The structure in MBOA plays much more important role as it captures the dependency structure of individual variables. The DiT algorithm is able to cover some kind of interactions but only as a side-effect.

2.1 Growing the Distribution Tree

Let us denote the set of data points at hand (the population) as $\mathbb{P} = \{\boldsymbol{x}^i\}_{i=1}^N$, N is the population size, $\boldsymbol{x}^i = (x_1^i, x_2^i, \ldots, x_D^i)$ is the i-th individual, D is the dimensionality of the search space. Further, there are three parameters of the algorithm: $minToSplit$, describing the minimal node size[2] allowed to be split, $minInNode$, the minimal number of data points that must remain in the left and right part of the node after splitting, and $maxPValue$, the maximal p-value for which we consider the split statistically significant.

The tree building algorithm is very simple (see Fig. 2). The algorithm parameters are assigned to the steps where they apply (written in parentheses).

Function SplitNode	
1	If there are not enough data points in the node ($minToSplit$) then exit
2	Find best split among all possible splits ($minInNode$)
3	If there is a 'good-enough' split ($maxPValue$),
3.1	Divide the data points into left and right node
3.2	Apply the SplitNode function recursivelly on left and right node
3	Return the just realized node split

Fig. 2. Procedure for splitting the node.

The heart of this procedure is constituted by the way we search for the best split. We can place the split between each pair of the successive data points in each dimension. If we have N D-dimensional data points in the node, we have $D \times (N-1)$ possible splits. Each of these candidate splits is evaluated via

[2] To prevent misunderstanding, let me point out that 'size of node' or 'node size' refers to the number of data points which belong to the respective node. When I describe the 'physical size' of node in DiT I use the term 'node volume'.

hypothesis testing. We assume that in each leaf node the data points should have approximately the uniform distribution. This assumption is tested via the χ^2-test. Eventually, the split 'which gives us the least similarity between the uniform and the observed distribution' is selected, if the test says 'there is a sufficiently high probability of big difference between the uniform and the observed distribution'. Otherwise we have no reason to split this node even if it has a large size.

```
Function FindBestSplit
1       For all dimensions
1.1         Compute the density of individuals in this node
1.2         Build a list of candidate split points
1.3         For all candidate split points
1.3.1           Determine expected frequencies in left and right subnode
1.3.2           Determine observed frequencies in left and right subnode
1.3.3           Compute the test statistic
1.3.4           Carry out the χ²-test, determine the p-value
1.3.5           If the best split so far was found, remember it
2       Return the best split
```

Fig. 3. Procedure for searching the best split.

For each particular candidate split we can count the points in the left and the right subnode which are the observed sizes, $N_{\mathrm{L}}^{\mathrm{obs}}$ and $N_{\mathrm{R}}^{\mathrm{obs}}$. For both subnodes we can also directly compute the expected sizes, $N_{\mathrm{L}}^{\mathrm{exp}}$ and $N_{\mathrm{R}}^{\mathrm{exp}}$, knowing the size of node as a whole and the relative volumes of the left and right subnodes. The χ^2-test allows us to compare the observed sizes with the expected ones. First, we compute the test statistic

$$Chi^2 = \frac{(N_{\mathrm{L}}^{\mathrm{exp}} - N_{\mathrm{L}}^{\mathrm{obs}})^2}{N_{\mathrm{L}}^{\mathrm{exp}}} + \frac{(N_{\mathrm{R}}^{\mathrm{exp}} - N_{\mathrm{R}}^{\mathrm{obs}})^2}{N_{\mathrm{R}}^{\mathrm{exp}}} \ . \tag{1}$$

The Chi^2 is a random variable with χ^2 distribution with 1 degree of freedom. It describes to what extent the observed and expected frequencies differ from each other. Thus, we can compute the probability of observing this or greater Chi^2 assuming the uniform distribution in the node as 1 minus the value of the cumulative distribution function of χ^2 distribution with 1 degree of freedom at point Chi^2.

$$p = 1 - CDF\chi^2(1, Chi^2) \tag{2}$$

This way we can select the split that gives us the highest discrimination from the uniform distribution with a sufficiently high probability, i.e. among all candidate splits we select the one with the lowest p-value, and we realize the split if the p-value is lower than $maxPValue$.

2.2 Sampling from the Distribution Tree

The process of creating new individuals, i.e. sampling from the distribution tree, is very straightforward. The leaf nodes of the tree fully cover the whole search space. New individuals are sampled this way: each leaf has a certain number of 'parent' individuals which belong to that node after the DiT creation; the same number of 'offsprings' is created in each leaf just by several calls to the uniform random number generator. Since an empty leaf node is not possible, this ensures that the search will not stop in any area of the search space.

3 Experiments

The carried out experiments compared the DiT evolutionary algorithm to several other evolutionary techniques. For all experiments, I used the evolutionary model of Tsutsui et al. [8]. Let the population size be N. Each iteration consists of the following phases:

1. Based on the current population, create N new individuals (i.e. build a probabilistic model and sample from it in case of EDAs, or use the crossover to create N offsprings in case of GAs) and evaluate them.
2. Join the old and the new population to get a data pool of size $2N$.
3. Use the truncation selection to select the better half of data points (returning the population size back to N).

These phases were repeated until the number of evaluations exceeded 50,000.

3.1 Test Suite

For the performance evaluation of the EA using DiT several reference optimization problems were used. The test functions were selected to show some of the strengths and weaknesses of the DiT-EA and are listed in Table 1.

Table 1. Test functions and related parameters. The f function can be described as a polyline going through points $(0,0)$, $(1,5)$, $(2,0)$, $(7,4)$, $(12,0)$.

Function	Expression	Domain
2D Two Peaks	$F = 10 - \sum_{i=1}^{2} f(x_i)$	$\langle 0, 12\rangle^2$
20D Two Peaks	$F = 100 - \sum_{i=1}^{20} f(x_i)$	$\langle 0, 12\rangle^{20}$
2D Griewangk	$F = 1 + \sum_{i=1}^{2} \frac{x_i^2}{4000} - \prod_{i=1}^{2} \cos \frac{x_i}{\sqrt{i}}$	$\langle -5, 5\rangle^2$
10D Griewangk	$F = 1 + \sum_{i=1}^{10} \frac{x_i^2}{4000} - \prod_{i=1}^{10} \cos \frac{x_i}{\sqrt{i}}$	$\langle -5, 5\rangle^{10}$
2D Rosenbrock	$F = 100 \times (x_1^2 - x_2)^2 + (1 - x_1)^2$	$\langle -2.048, 2.048\rangle^2$
10D Rosenbrock	$F = \sum_{i=1}^{5} (100 \times (x_{2i-1}^2 - x_{2i})^2 + (1 - x_{2i-1})^2)$	$\langle -2.048, 2.048\rangle^{10}$

The Two Peaks function was used in the work of Tsutsui et al. [8]. The number of its local optima grows exponentially with the dimensionality, on the

other hand it is completely separable function. The global optimum lies in the point $(1, 1, \ldots, 1)$ and the optimal value is 0. Algorithm which is not able to decompose this problem to a set of 1D problems has a very small chance to find the global optimum.

The Griewangk function is an example of a non-separable function. It has one global optimum in the origin of the coordinate system and several (the exact number depends on the dimensionality) local optima surrounding (and hiding) the global one. Many evolutionary algorithms fall in trouble with this function.

The Rosenbrock (a.k.a. Banana) function has a high degree of dependency between the variables. It is very hard to optimize using ordinary evolutionary algorithms. The basic form of this function is two-dimensional. The optimal value of this function is 0 for point $(1, 1)$.

3.2 Involved Evolutionary Techniques

The distribution tree evolutionary algorithm (DiT-EA) is compared to the genetic algorithm (with a low and a high resolution), to the histogram UMDA, to the random search and to the line search heuristic.

Sampling individuals one by one with the uniform distribution on the whole search space – that is the simple *random search*. In order to give some indication of how complex the test problems are, I tried to optimize them also with the so-called *line search* (LS) method [9] which is a heuristic very efficient for separable problems. The discretization step for the LS was set to 0.01. This means that for Two Peaks function the LS algorithm evaluates 1201 data points for each dimension. If the LS gets stuck (no further improvement possible), it is restarted from another randomly generated point.

Further I compared the DiT-EA to 2 instantiations of GAs. It is very hard to compare a GA to an EDA in the continuous domain. While the EDA searches the space of real numbers (or rational numbers when evolving *in silico*), the GA evolves binary strings and searches a discretized space, and thus solves a different, much easier task. The resolution can deteriorate the GA performance e.g. due to insufficient precision. Thus, in experiments I used two GAs – one with a low resolution (allowing the GA to find such a solution the distance of which from the global optimum is not larger than 0.001), and second with a high resolution (allowing the GA to find such a solution the distance of which from the global optimum is not larger than 2.22^{-16}). The actual settings for individual test functions can be found in Table 2.

The univariate marginal distribution algorithm (UMDA) with the equi-height histogram model was described e.g. in [7, 8]. The number of bins for each histogram is set in such a way that if all the bins had equal width, none of them would be larger than 0.1 (see Table 2).

The distribution tree, its construction and sampling is described in section 2. The only parameters to be set are the minimal number of individuals that must remain in each node ($minInNode$ set to 3), the minimal node size allowing the split in the node ($minToSplit$ set to 5), and the maximal p-value for which we consider a split statistically significant ($maxPValue$ set to 0.001).

Table 2. Summary of settings of several algorithms involved in the study

Algorithm	Function					
	Two Peaks		Griewangk		Rosenbrock	
GA-low	Crossover:	Uniform, 100%				
	Resolution:	14 bits	Resolution:	14 bits	Resolution:	12 bits
GA-high	Crossover:	Uniform, 100%				
	Resolution:	56 bits	Resolution:	56 bits	Resolution:	55 bits
Hist-UMDA	Histogram type:	Equi-height				
	Number of Bins:	120	Number of Bins:	100	Number of Bins:	41

In all experiments, no mutation was used. The study is aimed at generalized crossover operators and the mutation would introduce a noise which would make it impossible to distinguish the effects of mutation and crossover. Moreover, typical mutation operators for binary and real chromosomes are incomparable.

3.3 Results and Discussion

The overall results for all problems are shown in Table 3. For population based algorithms experiments with population sizes of 20, 50, 100, 200, 400, 600, and 800 were carried out. Each experiment was repeated 20 times, and the average best fitness and its standard deviation based on all 20 runs was measured. The population size which resulted in the best average score after 50,000 evaluations was selected to be included in the table.

Table 3. Results of experiments on all test functions. The first row in each cell shows the best fitness averaged over 20 runs and its standard deviation in parentheses. The second row shows the population size for which the results are reported.

Func	Algorithm					
	Random	Line Search	GA low res.	GA high res.	Hist UMDA	DiT EA
2D Two Peaks	0,1788 (0,0752)	0 (0)	0,0018 (0)	2,5e-11 (3,9e-11)	0 (0)	4,8e-13 (2,3e-13)
			400	600	200	600
2D Griewangk	2,1e-4 (2,5e-4)	0 (0)	2,3e-5 (8,7e-5)	1,3e-7 (5,1e-7)	0,0024 (0,0026)	0 (0)
			600	600	800	400
2D Rosenbrock	0,0012 (0,0021)	0,0051 (0,0053)	5,3e-5 (1,4e-4)	9e-7 (3,1e-6)	0,0044 (0,0042)	5e-5 (9,6e-5)
			800	800	800	600
20D Two Peaks	35,3914 (1,8198)	0 (0)	0,8168 (0,7827)	0,9627 (0,6612)	0,0027 (0,0059)	27,1593 (2,7677)
			400	400	400	100
10D Griewangk	0,4874 (0,0874)	0,2975 (0,3412)	0,0063 (0,0031)	0,0083 (0,0057)	0,0033 (0,002)	2,5e-4 (4,9e-4)
			400	400	800	600
10D Rosenbrock	27,0552 (10,418)	1,168 (0,4596)	1,9947 (1,3471)	2,2752 (0,9745)	0,6373 (0,1714)	1,8554 (0,6581)
			400	400	400	600

As expected, the Hist-UMDA was very efficient when solving separable problems, i.e. the 2D and 20D Two Peaks function, and the 10D Rosenbrock function

as well. Although the variables in the basic 2D Rosenbrock function are not independent, due to the way the 10D version was constructed, each pair of variables in the 10D Rosenbrock function is independent of the rest.

The DiT-EA was the best competitor when solving both variants of the Griewangk function (see Fig. 4 for an illustration). In both cases it was able to find a solution of significantly better quality. In the majority of the rest of the involved test problems, its efficiency was better or comparable to that of GAs. The 20D Two Peaks function is the only exception (see Fig. 5). The DiT-EA was able to find solutions only slightly better than those obtained by random search. I hypothesize that this poor behavior is caused by the great number of local optima of this function (to be precise, the 20D Two Peaks function has 2^{20} local optima). To capture the problem structure, the DiT-EA would need a huge population. Hundreds of individuals allow the algorithm to make only limited number of leaf nodes with 'almost the same' density.

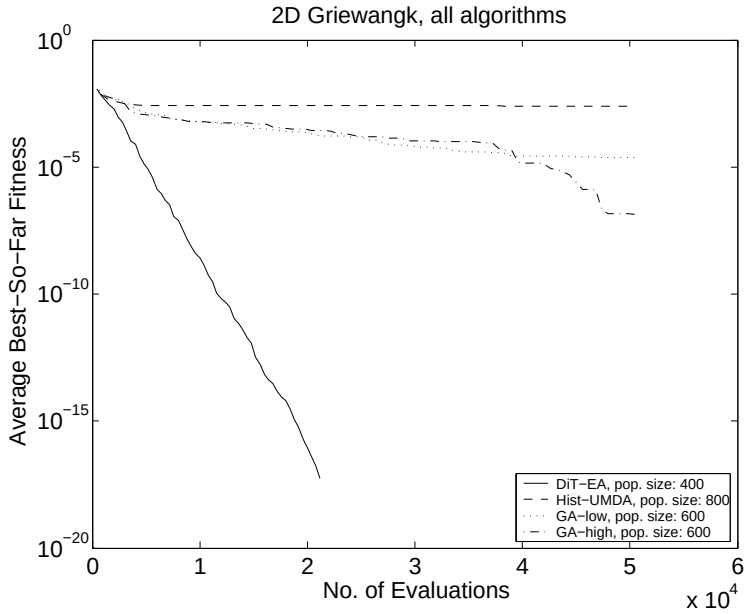

Fig. 4. Evolution of the involved algorithms on the test problem for which the DiT-EA performed best.

For the 2D functions, the GA-high found better solutions than the GA-low. GA with low resolution suffered from the insufficient bit string length. For the multidimensional functions, however, the difference between the two GAs vanished and they performed almost identically along the whole evolution when compared by a human eye.

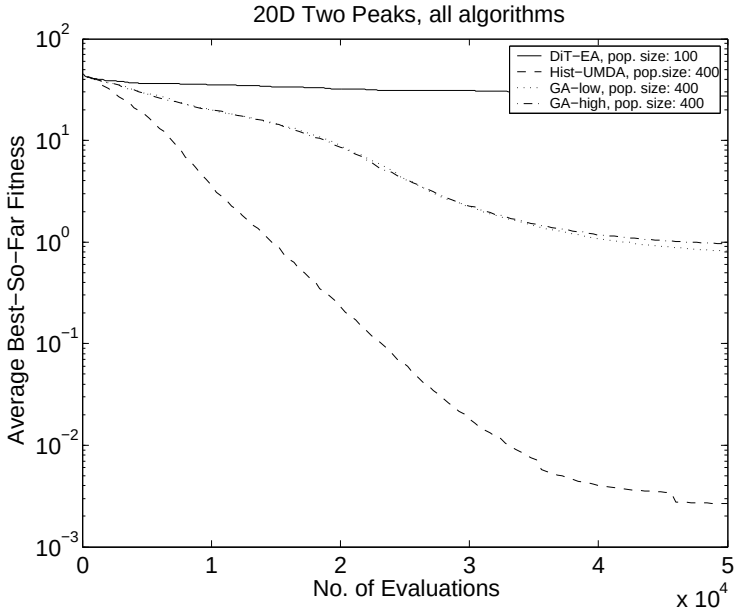

Fig. 5. Evolution of the involved algorithms on the test problem for which the DiT-EA performed worst.

4 Summary, Conclusions and Future Work

The article described an estimation of distribution algorithm using the distribution tree model. The model is developed here and to the best of the author's knowledge it is original and has not been used anywhere else yet. The DiT-EA was compared to other search algorithms on several test functions and it showed up to be a strong competitor in the field of the real-valued optimization with box constraints. The experiments showed that the members of EDA class of algorithms (histogram UMDA and DiT-EA) in almost all cases outperformed the GAs — they were able to find more precise solutions with less time needed.

It can be stated that the DiT-EA is very efficient when solving problems with several well-distinguished local optima. The population members after selection then form clusters in the search space and the DiT-EA seems to be able to identify them and to use this information in evolution (the DiT model tries to identify rectangular areas of the search space with significantly different densities). Furthermore, the DiT presents a model built on the basis of rigid statistical testing which is not rather common in the field of EAs.

The DiT has its own bottlenecks, of course. First, it is able to create only axis-parallel splits of the search space which can result in a limited ability to search the promising areas of the space if they look like valleys which are not parallel to the coordinate axes. This drawback can be reduced by employing a

kind of coordinate transformations in the node. The Independent Component Analysis seems to be very suitable method for making the DiT model rotationally invariant.

Second, the presented version of the algorithm is not able to decompose the task at hand as was shown on the 20D Two Peaks function. Using some technique of independency detection in each node would allow the algorithm to hierarchically decompose the problem into a set of problems of lower dimensionality. Such a model would be comparable to models capturing the dependency structure of variables, e.g. the MBOA. In that case the model deserves much thorougher analysis, e.g. in the scalability area.

Another possibility for future work is the fact that the model building algorithm allows very simple generalization for discrete variables. Only the procedure for creating the set of candidate splits would have to be changed.

Acknowledgments

This research was supervised by Doc. Ing. Jiří Lažanský, CSc. and was supported by the Grant agency of the Czech Republic with the grant No. GACR 102/02/0132 entitled "Use of Genetic Principles in Evolutionary Algorithms".

References

[1] Peter A.N. Bosman and Dirk Thierens. Continuous iterated density estimation evolutionary algorithms within the IDEA framework. In *Workshop Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, pages 197–200, 2000.

[2] Leo Breiman, Jerome Friedman, Charles J. Stone, and R. A. Olshen. *Classification and Regression Trees*. Kluwer Academic Publishers, January 1984.

[3] Marcus R. Gallagher, Marcus Frean, and Tom Downs. Real-valued evolutionary optimization using a flexible probability density estimator. In *Genetic and Evolutionary Computation Conference (GECCO-1999)*.

[4] Pedro Larrañaga and Jose A. Lozano, editors. *Estimation of Distribution Algorithms*. GENA. Kluwer Academic Publishers, 2002.

[5] Jiří Očenášek and Jiří Schwarz. Estimation of Distribution Algorithm for Mixed Continuous-Discrete Optimization Problems. In *2nd Euro-International Symposium on Computational Intelligence*, IOS Press, Kosice, Slovakia, 2002, pages 227–232. ISBN 3-540-444139-5, ISSN 0302-9743.

[6] Martin Pelikan, David E. Goldberg, and Fernando Lobo. A survey of optimization by building and using probabilistic models. Technical Report IlliGAL Report No. 98018, University of Illinois, Urbana-Champaign, September 1999.

[7] Petr Pošík. Comparing various marginal probability models in evolutionary algorithms. In Pavel Ošmera, editor, *MENDEL 2003*, volume 1, pages 59–64, Brno, 2003. Brno University. ISBN 80-214-2411-7.

[8] Shigeyoshi Tsutsui, Martin Pelikan, and David E. Goldberg. Evolutionary algorithm using marginal histogram models in continuous domain. Technical Report IlliGAL Report No. 2001019, University of Illinois, Urbana-Champaign, March 2001.

[9] D. Whitley, K. Mathias, S. Rana, and J. Dzubera. Evaluating evolutionary algorithms. *Artificial Intelligence*, 85:245–276, 1996.