Exploiting Gradient Information in Continuous Iterated Density Estimation Evolutionary Algorithms

Peter A.N. Bosman
Peter.Bosman@cs.uu.nl

Dirk Thierens
Dirk.Thierens@cs.uu.nl

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

Abstract

For continuous optimization problems, evolutionary algorithms (EAs) that build and use probabilistic models have obtained promising results. However, the local gradient information of the fitness function is not used in these EAs. In the case of optimization of continuous differentiable functions, it may be less efficient to disregard this information. In this paper, we therefore hybridize pure continuous iterated density estimation evolutionary algorithms (IDEAs) by using the conjugate gradient algorithm on a selection of the solutions. We test the resulting algorithm on a few well known difficult continuous differentiable function optimization problems. The results indicate that exploiting gradient information in probabilistic model building EAs leads to more efficient continuous optimization.

1 Introduction

Finding and using the structure of the fitness landscape can aid EAs in optimization. One approach to doing so, is by learning a probabilistic model from the selected solutions and by using it in sampling new solutions. The solutions are regarded as being representative of some probability distribution. Estimating this probability distribution and sampling more solutions from it, is a global statistical type of inductive iterated search. Such algorithms have been successfully applied to various optimization problems in the case of discrete (binary) variables [7, 9, 10, 12] as well as continuous (real) variables [1, 3, 6]. In the estimation of a probability distribution of a set of samples, no assumption is made on the source of the samples. This implies that for continuous, differentiable functions that we seek to optimize with IDEAs, *gradient information* is disregarded.

It has been shown [3] that continuous IDEAs can efficiently process interactions between the problem variables, even for problems of very large dimensionality. Furthermore, IDEAs have also given promising results on a variety of difficult continuous differentiable function optimization problems [5]. In these IDEAs, density estimation is based upon either the normal pdf or on the normal mixture pdf. Although both pdfs have lead to good results, IDEAs based on them

have outperformed each other on different types of optimization problems. The normal mixture pdf enables IDEAs to better cope with non–linear interactions between the problem variables. However, to prevent overfitting, many more samples are often required. One problem with repeatedly drawing new samples from a probability distribution, is that gradient information is ignored in continuous differentiable function optimization. As a result, once the search has located an interesting basin of attraction, efficiently finding the optimum is prohibited. To speed up convergence, it therefore appears to be a good idea to hybridize the IDEA to exploit gradient information. Our goal in this paper is to verify whether this is the case for continuous differentiable function optimization.

The remainder of this paper is organized as follows. In section 2, we define the notion of probabilistic models and point out how these can be used in EAs. In sections 3 and 4, we go into probabilistic model selection. Next, we take a look at what pdfs we can use in section 5. We discuss hybridization of IDEAs by adding gradient search in section 6 and present our experiments in section 7. Further research is discussed in section 8 and our final conclusions are drawn in section 9.

2 Probabilistic models and IDEAs

The IDEA is a framework that uses probabilistic models in evolutionary optimization. We take the elementary building block of probabilistic models to be the *probability density function* (pdf). We define a probabilistic model $\mathcal{M}$ to consist of some structure ς that describes a composition of pdfs, and a vector of parameters $\boldsymbol{\theta}$ for the pdfs implied by ς, $\mathcal{M} = (\varsigma, \boldsymbol{\theta})$. The pdf to fit over every factor implied by ς is chosen on beforehand, such as the normal pdf. The way in which the parameters $\boldsymbol{\theta}$ are fit, is also predefined on beforehand. With these assumptions, we denote the resulting probability distribution by P_ς.

We assume that we have an l–dimensional continuous optimization problem $C(y_0, y_1, \ldots, y_{l-1})$ which without loss of generality we seek to minimize. A population of n samples is maintained. We select $\lfloor \tau n \rfloor$ samples ($\tau \in [\frac{1}{n}, 1]$) in each iteration t and let θ_t be the worst selected sample cost. We then estimate the distribution of the selected samples and thereby find $\hat{P}_\varsigma^{\theta_t}(\boldsymbol{\mathcal{Y}}) = \hat{P}_\varsigma^{\theta_t}(Y_0, Y_1, \ldots, Y_{l-1})$ as an approximation to the true uniform distribution $P^{\theta_t}(\boldsymbol{\mathcal{Y}})$ over all points $\boldsymbol{y}$ with $C(\boldsymbol{y}) \leq \theta_t$. New samples can then be drawn from $\hat{P}_\varsigma^{\theta_t}(\boldsymbol{\mathcal{Y}})$ and be used to replace some of the current samples. If we select by taking the best $\lfloor \tau n \rfloor$ samples, draw $n - \lfloor \tau n \rfloor$ new samples from $\hat{P}_\varsigma^{\theta_t}(\boldsymbol{\mathcal{Y}})$, and finally replace the worst $n - \lfloor \tau n \rfloor$ samples in the population with these new samples, we have that $\theta_{t+1} = \theta_t - \varepsilon$ with $\varepsilon \geq 0$. This assures that $\theta_0 \geq \theta_1 \geq \ldots \geq \theta_{t_{\text{end}}}$. We call an IDEA so constructed *monotonic*. For a more detailed algorithmic description of the IDEA framework, we refer the reader to previous work [3].

3 Factorization selection

To estimate the probability distribution of the selected samples in the IDEA, we first search for a model structure. In this section, we only focus on the specific

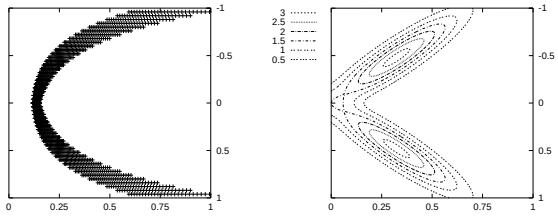

Figure 1: A non–linear dependency in the sample set (left) and the contour lines of the density estimation using two normal pdfs after clustering (right).

structure of a *conditional factorization*. The use of this structure has resulted in successful and promising applications of IDEAs [3, 5]. A conditional factorization is a product of multivariate conditional pdfs $\prod_{i=0}^{l-1} P(Y_i|Y_{\pi(i)})$. By identifying a vertex with each variable Y_i and an arc (Y_i, Y_j) if and only if Y_j is conditionally dependent on Y_i ($Y_j \in \pi(Y_i)$), we get the conditional factorization *graph*. A conditional factorization is valid if and only if its factorization graph is *acyclic*.

We have to learn such a conditional factorization from the vector of selected samples. To this end, a variety of approaches can be taken [11]. We use an incremental algorithm that starts from the empty graph with no arcs. Each iteration, the arc to add is selected to be the arc that increases some metric the most. If no addition of any arc further increases the metric, the final factorization graph has been found. The metric that we use in this paper, is commonly known as the *Bayesian Information Criterion* (BIC). This metric should be minimized. It scores a factorization by its negative log–likelihood, but adds a penalty term that increases with the complexity of the factorization and the size of the sample vector. Let $\boldsymbol{\mathcal{S}} = (\boldsymbol{y^0}, \boldsymbol{y^1}, \ldots, \boldsymbol{y^{|\mathcal{S}|-1}})$ be the selected vector of l–dimensional samples. The BIC metric is parameterized by a regularization parameter λ that determines the amount of penalization:

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

4 Factorization mixture selection

The structure of the sample vector may be highly non–linear. This non–linearity can force us to use probabilistic models of a high complexity to retain some of this non–linearity. However, especially using relatively simple pdfs such as the normal pdf, the non–linear interactions cannot be captured even with higher order models. The use of clusters allows us to efficiently break up non–linear interactions so that we can use simple models to get an adequate representation of the sample vector. An example of this is depicted in figure 1. Furthermore, computationally efficient clustering algorithms exist that provide useful results.

A factorized probability distribution is estimated in each cluster. By doing so, we obtain a *mixture distribution*. We let k be the amount of clusters and let $\boldsymbol{\mathcal{K}} = (0, 1, \ldots, k-1)$. We write $\mathfrak{f}$ for a factorization. For a mixture of factorizations, we write $\mathfrak{f}_{\boldsymbol{\mathcal{K}}} = (\mathfrak{f}_0, \mathfrak{f}_1, \ldots, \mathfrak{f}_{k-1})$. The resulting probability distribution is a weighted sum of the individual probability distributions over each cluster:

$$\hat{P}_{\mathfrak{f}_{\boldsymbol{\mathcal{K}}}}(\boldsymbol{\mathcal{Y}}) = \sum_{i=0}^{k-1} \beta_i \hat{P}^i_{\mathfrak{f}_i}(\boldsymbol{\mathcal{Y}}) \tag{2}$$

For function optimization, the β_i are usually set proportional to the size of cluster i. In order to perform clustering, we use the randomized leader algorithm. This algorithm has been observed to be a fast and flexible adaptive clustering algorithm that gives useful results in IDEAs [5]. The first sample to make a new cluster is appointed to be its leader. The leader algorithm goes over the sample vector exactly once. For each sample, it finds the cluster with the closest leader. If this leader is closer than a given threshold $\mathfrak{T}_d$, the sample is added to that cluster. Otherwise, a new cluster is created with this single sample as its leader.

5 Probability Density Functions

The normal pdf has received the most attention in continuous probabilistic model building EAs and has led to good results [3, 5]. The sample average in dimension j is $\overline{Y_j} = \frac{1}{|\boldsymbol{\mathcal{S}}|} \sum_{i=0}^{|\boldsymbol{\mathcal{S}}|-1} (\boldsymbol{y^i})_j$. Given a vector of indices $\boldsymbol{a}$, the sample covariance matrix over variables $Y_{\boldsymbol{a}}$ is $\boldsymbol{\Sigma_a} = \frac{1}{|\boldsymbol{\mathcal{S}}|} \sum_{i=0}^{|\boldsymbol{\mathcal{S}}|-1} (y^i_{\boldsymbol{a}} - \overline{Y}_{\boldsymbol{a}})(y^i_{\boldsymbol{a}} - \overline{Y}_{\boldsymbol{a}})^T$. To compute the BIC measure, we require to compute the negative log–likelihood of each conditional pdf in the factorization. However, the *entropy* $h(\cdot)$ is equal to the average negative log–likelihood of the sample set if the pdfs were fitted to be of a maximum likelihood [4]. The entropy of the normal pdf can be evaluated significantly faster than the negative log–likelihood. The required conditional pdf and the entropy are [3]:

$$f_{\mathcal{N}}(y_{\boldsymbol{a}_0} | y_{\boldsymbol{a}-\boldsymbol{a}_0}) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(y_{\boldsymbol{a}_0}-\mu)^2}{2\sigma^2}} \tag{3}$$

$$\text{where} \begin{cases} \sigma = \frac{1}{\sqrt{\boldsymbol{\Sigma}^{-1}_{\boldsymbol{a}}(0,0)}} \\ \mu = \frac{\overline{Y_{\boldsymbol{a}_0}} \boldsymbol{\Sigma}^{-1}_{\boldsymbol{a}}(0,0) - \sum_{i=1}^{|\boldsymbol{a}|-1} (y_{\boldsymbol{a}_i} - \overline{Y_{\boldsymbol{a}_i}}) \boldsymbol{\Sigma}^{-1}_{\boldsymbol{a}}(i,0)}{\boldsymbol{\Sigma}^{-1}_{\boldsymbol{a}}(0,0)} \end{cases}$$

$$h(Y_{\boldsymbol{a}_0} | Y_{\boldsymbol{a}-\boldsymbol{a}_0}) = h(Y_{\boldsymbol{a}}) - h(Y_{\boldsymbol{a}-\boldsymbol{a}_0}), \tag{4}$$

$$h(Y_{\boldsymbol{a}}) = \frac{1}{2}(|\boldsymbol{a}| + \ln((2\pi)^{|\boldsymbol{a}|} \det(\boldsymbol{\Sigma_a})))$$

By clustering we obtain an *approximation* to the maximum likelihood normal mixture distribution. An attempt to fit a normal mixture pdf with a maximum likelihood, leads to the *expectation maximization* (EM) algorithm [2]. Even though the EM algorithm is theoretically rigorous, it can easily get stuck in local minima, resulting in a sub optimal fit. This is especially true in higher dimensions. Finding

a factorization with the normal mixture pdf is far too time consuming to be of practical use because of the repeated application of the costly EM algorithm [5]. When a univariate factorization is used, each normal pdf in the mixture has a diagonal covariance matrix. The resulting IDEA requires less function evaluations to optimize epistatic functions with many local optima, but is less efficient than a clustering approach for non–linear problems [5]. Furthermore, the time requirements for the clustering approach are significantly smaller.

6 Hybridization using local gradient information

In order to use the local gradient information of a function, a straightforward manner is to perform *gradient descent* (in the case of minimization). This is an iterative approach that alters a point by moving it a short distance in the direction of the greatest rate of decrease in the optimization function. By using *line minimization*, the distance that is moved in the direction of the steepest descent, takes the search to a point at which the gradient in that direction is 0. Subsequently, a new direction is taken until the search converges. However, following the direction of steepest descent in each step is in general not optimal. The reason for this is that each subsequent search direction is orthogonal to the previous one. This can cause the search to oscillate around the optimal direction towards the optimum. The *conjugate gradient* algorithm [8] overcomes this problem. In this algorithm, each subsequent search direction is *conjugate* with the previous one. This means that the new direction is chosen so that the component of the gradient in the direction of the previous step remains zero along the new direction, resulting in more efficient local optimization.

In order to apply the conjugate gradient algorithm, we require to be able to approximate the gradient. At a single point, this requires l evaluations. For higher dimensional functions, using gradient search therefore becomes much more expensive. To construct the hybrid IDEA, we apply the conjugate gradient algorithm to $\lfloor \tau_G n \rfloor$ randomly selected solutions at the end of each generation. We do not suggest that this hybridization approach is optimal, but it will point out whether a gradient search algorithm can help us in the optimization of continuous differentiable functions. We refer to the resulting hybrid IDEA as the GLIDE EA (*Gradient Leveraged Iterated Density Estimation Evolutionary Algorithm*) or GLIDE for short:

GLIDE
1 Create n random solutions of dimensionality l
2 Repeat until termination
2.1 Create new population using IDEA
2.2 Perform conjugate gradient search on $\tau_G n$ randomly selected solutions

7 Experiments

The continuous optimization problems we used for testing are the following:

C_0	$\frac{1}{4000}\sum_{i=0}^{l-1}(y_i-100)^2-\prod_{i=0}^{l-1}\cos\left(\frac{y_i-100}{\sqrt{i+1}}\right)+1$	$[-600,600]^l$	*Minimize*
C_1	$-\sum_{i=0}^{l-1}\sin(y_i)\sin^{20}\left(\frac{(i+1)y_i^2}{\pi}\right)$	$[0,\pi]^l$	*Minimize*
C_2	$\sum_{i=0}^{l-2}100(y_{i+1}-y_i^2)^2+(1-y_i)^2$	$[-5.12,5.12]^l$	*Minimize*

Function C_0 is Griewank's function, C_1 is Michalewicz's function and C_2 is Rosenbrock's function. Both C_0 and C_1 are highly epistatic functions with many local optima. However, given a fixed precision, C_0 becomes easier as l increases. Function C_2 is highly non–linear. It has a curved valley along which the quality of the solutions is much better than in its neighborhood. Furthermore, this valley has a unique minimum of 0 itself. In all our testing, we used *monotonic* IDEAs. We used the rule of thumb by Mühlenbein and Mahnig [9] for FDA and set τ to 0.3. We ran tests so as to find the best results within a maximum of 10^7 evaluations and a maximum population size n of 10^5. If all of the solutions differed by less than $5 \cdot 10^{-7}$, termination was enforced. All results were averaged over 10 runs. We tested GLIDE using the normal distribution by searching for conditional factorizations using the BIC metric with $\lambda = \frac{1}{2}$. We also tested GLIDE using the normal mixture distribution by applying clustering to get approximately 10 clusters. The mixture coefficients β_i were set to the proportional cluster sizes. We allowed the conjugate gradient algorithm to run for 10 iterations each time it was called. We set $\tau_G \in \{0, 0.1, 0.5\}$. Note that for $\tau_G = 0$, we get a pure IDEA. We computed the gradient information by using $\Delta y_i = 10^{-13}$. Furthermore, we have used the Polak–Ribiere variant of the conjugate gradient algorithm [13].

In figure 2, we present the minimum required population size $n_{\min}$, the average best solution, the average amount of evaluations and the *Relative Run Time* RRT. Let $\mathrm{FT}(x)$ be the time to perform x random evaluations and let TRT be the *Total Run Time* on the same processing system. Then, $\mathrm{RRT}(x) = \mathrm{TRT}/\mathrm{FT}(x)$. We determined RRT as $\mathrm{RRT}(10^6)$. The RRT index is a processing system independent fair comparison measure. We sort the results by this index secondarily instead of the amount of evaluations, as it *truly* reflects the required amount of time. We have tested GLIDE variants that use the normal distribution (indicated by $\mathfrak{f}$) as well as the normal mixture distribution obtained by clustering (indicated by $\mathfrak{f}_{\mathcal{K}}$). For comparison, we have also tested a *Random Restart Conjugate Gradient* ($\mathbb{R}$CG) algorithm in which only the conjugate gradient algorithm is repeatedly started a random starting point. The maximum amount of iterations in the conjugate gradient algorithm in the case of $\mathbb{R}$CG was taken to be 10^7.

For lower dimensional problems, GLIDE outperforms the pure IDEAs. However, as l increases, IDEAs without gradient information obtain better results on C_0 and C_1. The reason for this is twofold. On the one hand, gradient search requires many evaluations as l increases just to evaluate the gradient. As this is done many times, the maximum amount of evaluations limit is reached sooner. On the other hand, at the beginning of the EA, we have not much of a clue about the locations of the interesting basins of attraction in the fitness landscape. Therefore, it is not yet of much use to start using a gradient search algorithm. As the search progresses towards one or more attractors of the function, it becomes more interesting to use gradient search. So it is to be expected that the results obtained by GLIDE can be improved by postponing gradient search until it becomes more

Alg.	$n_{\min}$	$\overline{C}$	$\overline{\text{evals}}$	RRT
$C_0, l = 5$				
(f, 0.5)	50	0.000000	50928	0.0547
(f, 0.1)	50	0.000000	52093	0.0605
(f$_{\mathcal{K}}$, 0.1)	525	0.000000	331374	0.4674
(f$_{\mathcal{K}}$, 0.5)	425	0.000000	484342	0.5313
(f, 0.0)	175	0.000000	1112144	2.5253
$\mathbb{R}$CG	—	0.013304	5307754	5.5496
(f$_{\mathcal{K}}$, 0.0)	17000	0.015748	10^7	51.1732
$C_0, l = 25$				
(f, 0.0)	200	0.000000	12410	0.0334
(f, 0.5)	25	0.000000	27233	0.0438
(f, 0.1)	25	0.000000	14751	0.0675
(f$_{\mathcal{K}}$, 0.5)	25	0.000000	24637	0.0907
(f$_{\mathcal{K}}$, 0.1)	75	0.000000	29454	0.2482
$\mathbb{R}$CG	—	0.000000	4072119	3.6863
(f$_{\mathcal{K}}$, 0.0)	3000	0.000000	197343	10.6522
$C_0, l = 100$				
(f, 0.0)	400	0.000000	49884	0.5102
(f, 0.5)	50	0.000000	408864	0.6351
(f, 0.1)	50	0.000000	186171	0.7425
(f$_{\mathcal{K}}$, 0.0)	1500	0.000000	168609	16.0661
(f$_{\mathcal{K}}$, 0.5)	300	0.000000	2703191	16.4785
(f$_{\mathcal{K}}$, 0.1)	500	0.000000	2071105	24.2137
$\mathbb{R}$CG	—	1390.943857	3414982	3.0439

Alg.	$n_{\min}$	$\overline{C}$	$\overline{\text{evals}}$	RRT
$C_1, l = 5$				
(f, 0.5)	50	-4.687658	89085	0.1986
(f, 0.1)	100	-4.687658	134549	0.3170
(f$_{\mathcal{K}}$, 0.1)	400	-4.687658	187996	0.4574
(f$_{\mathcal{K}}$, 0.5)	300	-4.687658	340929	0.7681
(f$_{\mathcal{K}}$, 0.0)	19500	-4.687658	1095199	7.3141
(f, 0.0)	24000	-4.676699	10^7	58.2003
$\mathbb{R}$CG	—	-4.672658	3685044	7.9689
$C_1, l = 25$				
(f, 0.0)	150	-24.135704	2396333	21.4501
(f, 0.5)	25	-21.112268	2932662	8.6118
(f, 0.1)	25	-20.901198	2734527	18.5832
(f$_{\mathcal{K}}$, 0.5)	700	-19.839100	4652799	11.7873
(f$_{\mathcal{K}}$, 0.1)	800	-18.579309	1608802	6.7266
$\mathbb{R}$CG	—	-16.063736	5203679	10.6908
(f$_{\mathcal{K}}$, 0.0)	19000	-15.070724	10^7	97.2095
$C_1, l = 100$				
(f, 0.0)	≥ 100	$\leq$ -76.314818	≥ 4669486	≥ 1351.6456
(f$_{\mathcal{K}}$, 0.5)	175	-67.716262	10^7	58.6796
(f, 0.5)	50	-59.859316	10^7	29.5821
(f$_{\mathcal{K}}$, 0.1)	1500	-59.849551	10^7	50.0092
(f, 0.1)	50	-59.579408	10^7	65.2234
$\mathbb{R}$CG	—	-44.887450	5502399	11.2490
(f$_{\mathcal{K}}$, 0.0)	≥ 100	$\leq$ -23.433810	≥ 136349	≥ 462.4847

(A performance lowerbound is indicated by $\geq$ and $\leq$)
(For these results, the time limit was exceeded)

Alg.	$n_{\min}$	$\overline{C}$	$\overline{\text{evals}}$	RRT
$C_2, l = 5$				
(f, 0.1)	25	0.000000	6150	0.0218
(f$_{\mathcal{K}}$, 0.1)	50	0.000000	10432	0.0296
(f$_{\mathcal{K}}$, 0.5)	25	0.000000	15599	0.0322
(f, 0.5)	25	0.000000	14157	0.0330
(f$_{\mathcal{K}}$, 0.0)	5500	0.000000	150683	1.8107
$\mathbb{R}$CG	—	0.000000	5584752	11.7180
(f, 0.0)	50000	1.253981	10^7	159.9769
$C_2, l = 25$				
(f, 0.5)	25	0.000000	154307	0.4823
(f, 0.1)	25	0.000000	49924	0.6339
(f$_{\mathcal{K}}$, 0.1)	25	0.000000	52953	1.2719
(f$_{\mathcal{K}}$, 0.5)	25	0.000000	175624	1.3690
$\mathbb{R}$CG	—	0.000000	5206170	8.6328
(f$_{\mathcal{K}}$, 0.0)	70000	20.365538	10^7	338.8301
(f, 0.0)	70000	22.272674	10^7	241.4619

Alg.	$n_{\min}$	$\overline{C}$	$\overline{\text{evals}}$	RRT
$C_2, l = 100$				
$\mathbb{R}$CG	—	0.000000	3713514	6.7658
(f, 0.1)	50	0.000000	994431	12.0070
(f, 0.5)	25	0.000000	1405699	15.6587
(f$_{\mathcal{K}}$, 0.1)	25	0.000000	409042	73.3926
(f$_{\mathcal{K}}$, 0.5)	25	0.000000	1442753	76.8516
(f, 0.0)	19000	96.509540	10^7	352.1634
(f$_{\mathcal{K}}$, 0.0)	10^5	96.664236	10^7	656.6042

Figure 2: Results for all algorithms on C_0, C_1 and C_2.

beneficial to use it. These two issues are also demonstrated by the results obtained by $\mathbb{R}$CG, since they are worse on C_0 and C_1 as l goes up. For C_2, all algorithms that use gradient search outperform the pure IDEAs. The reason for this is that there are no local optima. The non–linear valley is efficiently located by the IDEA and the gradient search procedure finds the unique minimum. The combination of continuous IDEAs and gradient search is indeed effective. Furthermore, the use of the single normal distribution in GLIDE seems preferable over the normal mixture distribution obtained by clustering.

8 Discussion

The use of gradient search in IDEAs has resulted in an improvement over the results obtained so far for problems of low dimensionality l. From the results, it has however become unclear what value for τ_G is better. It is to be expected that for different optimization problems, different values for τ_G give better results. Furthermore, we have already argued that better results can probably obtained if gradient search is postponed until it is really beneficial to use it. It would therefore be interesting to see whether the results can be improved by setting τ_G adaptively. One way to do so is to look at the rate of success of the IDEA and the gradient search procedure and increase τ_G if the latter performs better than the former.

9 Conclusions

We have proposed a hybrid iterated density estimation evolutionary algorithm that uses the conjugate gradient algorithm to obtain more efficient optimization of continuous differentiable functions. The results in this paper indicate that such a hybridization can improve the results of both algorithms separately. These results may yet be further improved by altering the hybridization scheme to only use gradient search at times when it is really beneficial to use it.

References

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

[2] J. Bilmes. A gentle tutorial of the EM algorithm and its application to parameter estimation for gaussian mixture and hidden markov models. ICSI TR-97-021, 1997

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

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

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

[6] M. Gallagher, M. Fream, and T. Downs. Real–valued evolutionary optimization using a flexible probability density estimator. In W. Banzhaf *et al.*, editor, *Proc. of the GECCO–1999 Genetic and Evol. Comp. Conf.*, pages 840–846. M.K. Pub., 1999

[7] G. Harik and D.E. Goldberg. Linkage learning through probabilistic expression. *Comp. methods in applied mechanics and engineering*, 186:295–310, 2000

[8] M.R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 6:409–436, 1952

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

[10] A. Ochoa, H. Mühlenbein, and M. Soto. A factorized distribution algorithm using single connected bayesian networks. In M. Schoenauer *et al.*, editor, *Parallel Problem Solving from Nature – PPSN VI*, pages 787–796. Springer, 2000

[11] M. Pelikan, D.E. Goldberg, and F. Lobo. A survey of optimization by building and using probabilistic models. IlliGAL Technical Report 99018, 1999

[12] M. Pelikan, D.E. Goldberg, and K. Sastry. Bayesian optimization algorithm, decision graphs and occam's razor. In E. Burke *et al.*, editor, *Proceedings of the GECCO–2001 Genetic and Evolutionary Computation Conference*, pages 519–526. M.K. Pub., 2001

[13] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical Recipies In C: The Art Of Scientific Computing.* Cambridge University Press, 1992