IDEAs Based On The Normal Kernels Probability Density Function

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

Dirk Thierens
Dirk.Thierens@cs.uu.nl

Department of Computer Science, Utrecht University
P.O. Box 80.089, 3508 TB Utrecht, The Netherlands

March 2000

Abstract

The IDEA framework is a general framework for iterated density estimation evolutionary algorithms. These algorithms use probabilistic models to guide the search in stochastic optimization. The estimation of densities for subsets of selected samples and the sampling from the resulting distributions, is the combination of the evolutionary recombination and mutation steps used in EAs. We investigate how the normal kernels probability density function can be used as the distribution of the problem variables in order to perform optimization using the IDEA framework. As a result, we present three probability density structure search algorithms, as well as a general estimation and sampling algorithm, all of which use the normal kernels distribution.

1 Introduction

Evolutionary approaches to optimization are stochastic and non–deterministic algorithms that attempt to use certain aspects of the structure of the search space to guide the search. Such an inductive search is mainly guided through statistics with at the basis a set of samples, often called a population. These samples are solutions to the optimization problem at hand. The solutions in the population are combined in order to perform induction and generate new solutions that will hopefully be closer to the optimum in a certain sense. As this process is iterated, convergence is intended to lead the algorithm to a final solution.

Genetic algorithms (GAs) [8, 12] and many variants thereof combine the material within a subset of the solutions. Often this is done through exchanging values for variables after which certain values are individually adapted. Another approach is to view a selection of the set of samples as being representative of some probability distribution. By estimating this probability distribution and sampling more solutions from it, a more global statistical type of inductive iterated search is performed. Such algorithms have been proposed using different types of probability distributions for discrete spaces [1, 2, 3, 10, 11, 13, 15, 17], as well as in a limited way for continuous spaces [4, 7, 18, 19]. An overview of such algorithms that perform optimization by building and using probabilistic models is given by Pelikan, Goldberg and Lobo [16].

The IDEA framework [4] allows the just mentioned *iterated density estimation evolutionary algorithms* to be defined easily in both the case of discrete as well as continuous random variables. Assuming that the continuous problem variables are normally distributed or can be approximated well by using a histogram distribution, algorithms within the IDEA framework have been proposed [4]. The main difference among the search algorithms in that framework, is the allowed order of interaction κ. For a large value of κ, many variables are allowed to be processed *together* instead of individually. Research has shown that processing groups of variables as a whole in order to exploit problem structure, often called using *linkage* information in evolutionary algorithms (EAs), is important for both GAs [9, 12] and IDEAs [5]. A search method has to find these subsets of variables. Such a method generally takes up more time as κ increases, which poses a trade–off between running time and possible optimization ability.

In this paper, we investigate the normal kernels distribution and define how this distribution can be used within the IDEA framework. This means that we define how we can estimate the probability density once some probability density structure (pds) is found by a search method and how we can sample from that probability density. Moreover, we define the same class of search methods as in previous work [4], but now for the use of the normal kernels distribution. Before moving to derive these algorithms, we point out that we use the same notation and conventions as in previous work [4] for writing about probability theory, variables and pseudo–code.

The remainder of this paper is organized as follows. In section 2 we shortly revisit the IDEA framework and go over the most important aspects. Also, we point out the requirements on any probability density function (pdf) in order to define certain probability structure search algorithms as well as the density estimation and sampling algorithms. Subsequently, we derive these algorithms in section 3. Notes on the running times and topics for further research are given in section 4. Our conclusions regarding this work are stated in section 5. In addition, the derivation of the required conditional pdf for the normal kernels distribution is given in appendix A.

2 The IDEA framework and pdf requirements

The IDEA framework is based on the rationale that in order to find a minimum of a cost function, we can attempt to estimate the distribution of the points that all have a cost lower than or equal to some threshold θ and subsequently sample from this distribution to get more points that have a cost lower than or equal to θ. By finding the distribution for $\theta^* = \min_Z\{C(Z)\}$ for minimization cost function $C(\cdot)$ and solution vectors Z that consist of l variables Z_0, Z_1, ..., Z_{l-1}, a single sample will give an optimal solution. This has been formalized in the *Iterated Density Estimation Evolutionary Algorithm* (IDEA) framework:

IDEA$(n, \tau, m, sel(), rep(), ter(), sea(), est(), sam())$	
Initialize an empty set of samples	$P \leftarrow \emptyset$
Add and evaluate n random samples	***for*** $i \leftarrow 0$ ***to*** $n-1$ ***do***
	$\quad P \leftarrow P \cup$ NewRandomVector(Z^i)
	$\quad c[i] \leftarrow C(Z^i)$
Initialize the iteration counter	$t \leftarrow 0$
Iterate until termination	***while*** $\neg ter()$ ***do***
Select $\lfloor \tau n \rfloor$ samples	$\quad \{Z^{(S)i} \mid i \in \mathcal{N}_\tau\} \leftarrow sel()$
Set θ_t to the worst selected cost	$\quad \theta_t \leftarrow c[Z^{(S)k}]$ such that
	$\quad\quad \forall_{i \in \mathcal{N}_\tau} \langle c[Z^{(S)i}] \leq c[Z^{(S)k}] \rangle$
Search for a pds $(\pi, \boldsymbol{\omega})$	$\quad (\pi, \boldsymbol{\omega}) \leftarrow sea()$
Estimate each pdf in $\hat{P}_{\pi,\boldsymbol{\omega}}(Z)$	$\quad \{\hat{P}(Z_{\omega_i} \mid \{Z_j \mid j \in \pi(\omega_i)\}) \mid i \in \mathcal{L}\} \leftarrow est()$
Create an empty set of new samples	$\quad O \leftarrow \emptyset$
Sample m new samples from $\hat{P}_{\pi,\boldsymbol{\omega}}(Z)$	$\quad$ ***for*** $i \leftarrow 0$ ***to*** $m-1$ ***do***
	$\quad\quad O \leftarrow O \cup sam()$
Replace a part of P with a part of O	$\quad rep()$
Evaluate the new samples in P	$\quad$ ***for each*** new $Z^i \in P$ ***do***
	$\quad\quad c[i] \leftarrow C(Z^i)$
Update the generation counter	$\quad t \leftarrow t+1$
Denote the required iterations by t_{end}	$t_{\text{end}} \leftarrow t$

In the IDEA framework, we have that $\mathcal{L} = \{0, 1, \ldots, l-1\}$, $\mathcal{N}_\tau = \{0, 1, \ldots, \lfloor \tau n \rfloor - 1\}$, $\tau \in [\frac{1}{n}, 1]$, $sel()$ is the selection operator, $rep()$ replaces a subset of P with a subset of O, $ter()$ is the termination condition, $sea()$ is a pds search algorithm, $est()$ estimates each pdf in $\hat{P}_{\pi,\boldsymbol{\omega}}(Z)$ and $sam()$ generates a single sample from $\hat{P}_{\pi,\boldsymbol{\omega}}(Z)$.

Note that in the IDEA, we have used the approximation notation $\hat{P}^{\theta_t}_{\pi,\boldsymbol{\omega}}(Z)$ instead of the true distribution $P^{\theta_t}_{\pi,\boldsymbol{\omega}}(Z)$. An approximation is required because the determined distribution is based upon samples and the underlying density model is an assumption on the true distribution. This means that even though we might achieve $\hat{P}^{\theta_t}_{\pi,\boldsymbol{\omega}}(Z) = P^{\theta_t}_{\pi,\boldsymbol{\omega}}(Z)$, in general this is not the case.

In previous work [4], it was pointed out that the functions $est()$, $sea()$ and $sam()$ are the most distinguished parts in the framework. A pds is found by the $sea()$ function. This pds defines the conditional probabilities that are subsequently used to estimate the density. Next, the ordering imposed by the pds can be used to sample from the conditional distributions thus found. To ensure that no variables are conditioned on variables for which sampling has not been done yet, the pds is constrained to be of a certain form. Given l random variables Z_0, Z_1, ..., Z_{l-1}, by scanning these variables from $Z_{\omega_{l-1}}$ down to Z_{ω_0}, such a conflict is not possible according to the definition of the pds $(\pi, \boldsymbol{\omega})$:

$$\hat{P}_{\pi,\boldsymbol{\omega}}(Z) = \prod_{i=0}^{l-1} \hat{P}(Z_{\omega_i} | Z_{\pi(\omega_i)_0}, Z_{\pi(\omega_i)_1}, \ldots, Z_{\pi(\omega_i)_{|\pi(\omega_i)|-1}}) \tag{1}$$

$$\text{such that } \forall_{i\in\{0,1,\ldots,l-1\}} \langle \omega_i \in \{0,1,\ldots,l-1\} \wedge \forall_{k\in\{0,1,\ldots,l-1\}-\{i\}} \langle \omega_i \neq \omega_k \rangle \rangle$$

$$\forall_{i\in\{0,1,\ldots,l-1\}} \langle \forall_{k\in\pi(\omega_i)} \langle k \in \{\omega_{i+1}, \omega_{i+2}, \ldots, \omega_{l-1}\} \rangle \rangle$$

The density estimation is done based on random variables Z_i, $i \in \{0,1,\ldots,l-1\}$. For each problem variable, one such random variable is introduced. These variables are assumed to be sampled from a certain distribution. This distribution is used in estimation and sampling in the IDEA framework. In previous work [4], it was shown how the normal distribution and the histogram distribution can be used to this end in the case of *continuous* random variables. Also, in the case of *discrete* random variables, it was shown how a histogram distribution can directly be used.

From equation 1, it follows that in order to use the IDEA framework for some pdf in general, we require to have its conditional distribution of one random variable Z_i being conditionally dependent on n other variables Z_{j_0}, Z_{j_1}, ..., $Z_{j_{n-1}}$. To this end, we note that we have from probability theory that if the pdf of some distribution can be written as $f(\cdot)$, the required conditional pdf equals:

$$f(y_i | y_{j_0}, y_{j_1}, \ldots, y_{j_{n-1}}) = \frac{f(y_i, y_{j_0}, y_{j_1}, \ldots, y_{j_{n-1}})}{f(y_{j_0}, y_{j_1}, \ldots, y_{j_{n-1}})} \tag{2}$$

In order to implement $sea()$ methods that search for the pds $(\pi, \boldsymbol{\omega})$, a search metric is required. In our previous work, the (differential) entropy measure based on the Kullback–Leibler divergence was used to define three search procedures. Both the multivariate differential entropy $h(Y_{j_0}, Y_{j_1}, \ldots, Y_{j_{n-1}})$ and the conditional differential entropy $h(Y_i | Y_{j_0}, Y_{j_1}, \ldots, Y_{j_{n-1}})$ measures are required. The fact that we require *differential* entropy is because we are using *continuous* random variables. As we shall use this metric again in this paper, we state the required definitions:

$$h(Y_{j_0}, Y_{j_1}, \ldots, Y_{j_{n-1}}) = \tag{3}$$

$$-\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\cdots\int_{-\infty}^{\infty} f_{j_0,j_1,\ldots,j_{n-1}}(y_0, y_1, \ldots, y_{n-1}) \ln(f_{j_0,j_1,\ldots,j_{n-1}}(y_0, y_1, \ldots, y_{n-1})) dy_0 dy_1 \ldots dy_{n-1}$$

$$h(Y_i | Y_{j_0}, Y_{j_1}, \ldots, Y_{j_{n-1}}) = \tag{4}$$

$$h(Y_i, Y_{j_0}, Y_{j_1}, \ldots, Y_{j_{n-1}}) - h(Y_{j_0}, Y_{j_1}, \ldots, Y_{j_{n-1}})$$

The search procedures that we present in this paper are the same as in our previous work [4]. The procedures search for a chain, a tree and an acyclic directed graph of dependencies. We define these algorithms again, but this time we use the normal kernels distribution for the variables.

3 Continuous IDEAs using the normal kernels distribution

The normal kernels pdf is based on the normal distribution, which is a normalized Gaussian:

$$f(y) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(y-\mu)^2}{2\sigma^2}} = g(y, \mu, \sigma) \tag{5}$$

In equation 5, we have that $\mu = E[y]$ is the mean of the sample points and $\sigma = \sqrt{E[(y-\mu)^2]}$ is the standard deviation of the sample points. The multivariate version of this density function in d dimensions with $\boldsymbol{y} = (y_0, y_1, \ldots, y_{d-1})$, is defined by:

$$f(\boldsymbol{y}) = \frac{(2\pi)^{-\frac{d}{2}}}{(\det \boldsymbol{\Sigma})^{\frac{1}{2}}} e^{-\frac{1}{2}(\boldsymbol{y}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{y}-\boldsymbol{\mu})} = g_m(\boldsymbol{y}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) \tag{6}$$

In equation 6, we have that $\boldsymbol{\mu} = (\mu_0, \mu_1, \ldots, \mu_{d-1}) = (E[y_0], E[y_1], \ldots, E[y_{d-1}])$ is the vector of means and $\boldsymbol{\Sigma} = E[(\boldsymbol{y}-\boldsymbol{\mu})^T(\boldsymbol{y}-\boldsymbol{\mu})]$ is the nonsingular covariance matrix, which is symmetric. Note that $\boldsymbol{\Sigma}(i,i) = \sigma_{ii} = \sigma_i^2 = E[(y_i - \mu_i)^2]$. It is important to note this for the remainder of the paper because it might be somewhat of a confusing notation as the diagonal of $\boldsymbol{\Sigma}$ thus contains the variances σ_i^2 instead of the σ_i themselves.

The non–parametric normal kernels density model in d dimensions, places a d–dimensional Gaussian kernel from equation 6 over every sample point $y^{(S)i}$, $i \in \{0, 1, \ldots, N-1\}$. The normalized sum over these N Gaussian functions constitutes the normal kernels pdf. Each kernel is identical in shape and therefore has the same $\boldsymbol{\Sigma}$ matrix. This matrix has zero entries off the diagonal. The entries on the diagonal are allowed to differ, which allows for different variances in different dimensions. This can be useful as it is convenient to scale the variance for a certain dimension with the range of the available samples. In order to avoid confusion with the definition of σ as being the standard deviation over the samples in the case of the univariate normal distribution, we write $\mathfrak{s}$ for the fixed standard deviation used in the normal kernels pdf. Analogously, we write $\mathfrak{S}$ instead of $\boldsymbol{\Sigma}$. Formalizing the normal kernels pdf, we can write the the density function underlying the univariate normal kernels distribution with fixed standard deviation $\mathfrak{s}$, defined on N sample points $y^{(S)i}$ as follows:

$$f(y) = \frac{1}{N} \sum_{i=0}^{N-1} \frac{1}{\mathfrak{s}\sqrt{2\pi}} e^{\frac{-(y-y^{(S)i})^2}{2\mathfrak{s}^2}} = \frac{1}{N} \sum_{i=0}^{N-1} g(y, y^{(S)i}, \mathfrak{s}) \tag{7}$$

The multivariate pdf for the normal kernels distribution in d dimensions with fixed standard deviations $\mathfrak{s}_i$, $\boldsymbol{y} = (y_0, y_1, \ldots, y_{d-1})$ and $\boldsymbol{y}^{(S)i} = (y_0^{(S)i}, y_1^{(S)i}, \ldots, y_{d-1}^{(S)i})$, is defined by:

$$f(y_0, y_1, \ldots, y_{d-1}) = \frac{1}{N} \sum_{i=0}^{N-1} \frac{(2\pi)^{-\frac{d}{2}}}{\prod_{j=0}^{d-1} \mathfrak{s}_j} e^{-\sum_{j=0}^{d-1} \frac{(y_j - y_j^{(S)i})^2}{2\mathfrak{s}_j^2}} = \frac{1}{N} \sum_{i=0}^{N-1} g_m(\boldsymbol{y}, \boldsymbol{y}^{(S)i}, \mathfrak{S}) \tag{8}$$

In the above equation, $\mathfrak{S}$ is the diagonal matrix containing the $\mathfrak{s}_i$ parameters we just mentioned:

$$\mathfrak{S} = \begin{bmatrix} \mathfrak{s}_0^2 & 0 & \ldots & 0 \\ 0 & \mathfrak{s}_1^2 & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \ldots & \mathfrak{s}_{d-1}^2 \end{bmatrix} \tag{9}$$

One interesting property of the normal kernels distribution is that by increasing the standard deviation values $\mathfrak{s}_i$, we allow for less detail in the final density estimation. Furthermore, an advantage over the use of the normal distribution pdf is that we have a better way of modelling clusters. Using the same datasets, shown in figure 1, as in the work that presented the algorithms using the normal distribution [4], the density function taken over the variables univariately is plotted in figures 2 and 3. Note that the localized aspect of the kernel method becomes clear immediately. Even though the samples were drawn from an uniform distribution, there are still

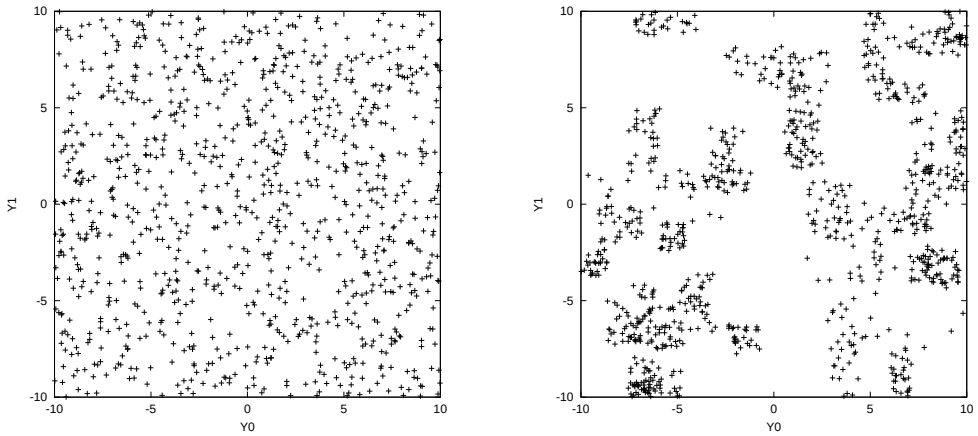

Figure 1: A uniform and a clustered data set over Y_0 and Y_1 with $D^Y = [-10, 10]$.

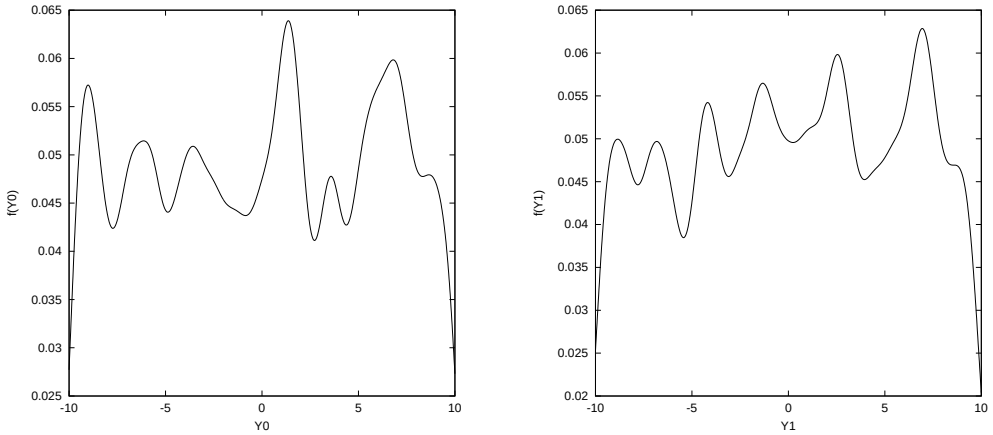

Figure 2: Fitting the univariate data from the uniform dataset with a normal kernels distribution with $\mathfrak{s} = 0.5$.

peaks in the distribution. It is clear that this method is well suited for representing local features of the distribution. Because of the local aspect, the kernels show a much better representation of the clusters as can be seen in the *joint* probability distribution using Gaussian kernels, which is depicted in figure 4. The joint distributions are not identical at all.

In order to define the search functions as well as general estimation and sampling algorithms in the case of the normal kernels distribution, it follows from section 2 that we require the *conditional* density function for one variable Y_0 conditioned on n other variables $Y_1, Y_2, \ldots, Y_n$. In appendix A it is shown that this required density function equals:

$$f(y_0|y_1, y_2, \ldots, y_n) = \sum_{i=0}^{N-1} \nu_i g(y_0, y_0^{(S)i}, \mathfrak{s}_0) \tag{10}$$

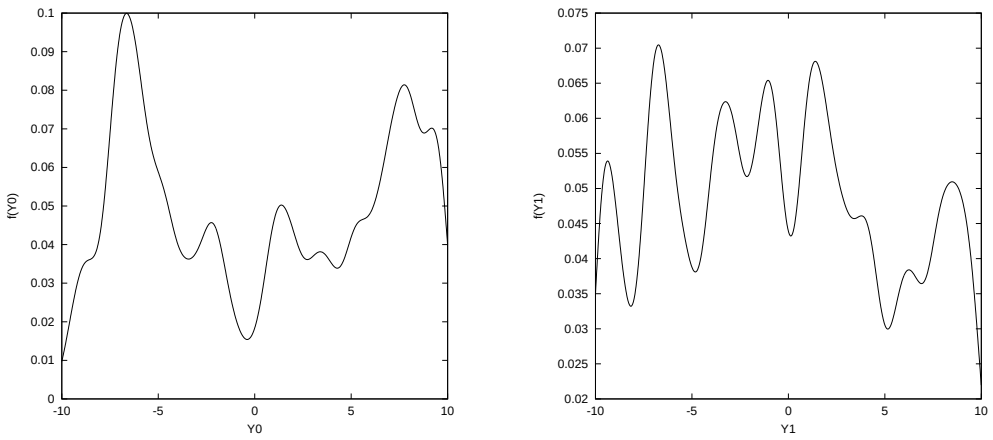

Figure 3: Fitting the univariate data from the clustered dataset with a normal kernels distribution model with $\mathfrak{s} = 0.5$.

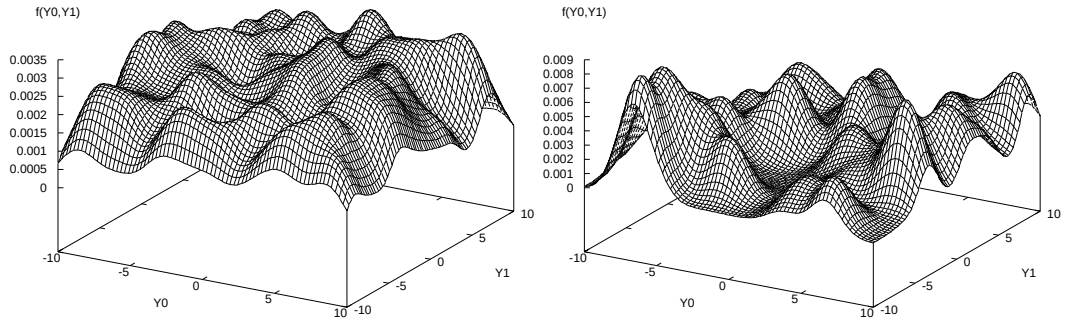

Figure 4: Fitting the joint data from the uniform (left) and clustered (right) datasets with a normal kernels distribution with $\mathfrak{s}_0 = \mathfrak{s}_1 = 1.0$.

$$\text{where } \nu_i = \frac{e^{-\sum_{j=1}^{n} \frac{(y_j - y_j^{(S)i})^2}{2\mathfrak{s}_j^2}}}{\sum_{k=0}^{N-1} e^{-\sum_{j=1}^{n} \frac{(y_j - y_j^{(S)k})^2}{2\mathfrak{s}_j^2}}}$$

In order to estimate the distribution using the normal kernels pdf, we only need to know the $\mathfrak{s}_i$ parameters. This can be seen from the fact that there are no additional variables to be computed in equation 10. The other variables that make up the density function are the domain values from the selected samples. The only thing that thus remains is the specification of $\mathfrak{s}_i$, $i \in \{0, 1, \ldots, l-1\}$. In our kernel approach, every kernel has the same variance in a certain dimension. Allowing different variances constitutes more of a mixture model. One approach is to specify a certain variance on beforehand and to scale this variance with the range of sample values in subsequent iterations. How to perform such a scaling or how to specify the initial value of the variance has an important influence on the behaviour of the evolutionary optimization algorithm. Therefore, we parameterize the determination of $\mathfrak{s}_i$ by using an auxiliary function $\varphi(\cdot)$. A simple example of such a function would be to compute the range for a variable Y_i and then to set the variance to a value based on that range, relative to the amount of kernels. This can be formally written as follows:

LINEARSCALING(α, i)
1 ***if*** range^i not yet determined this iteration ***then***
  1.1 $\min^i \leftarrow \min_{k \in \{0,1,\ldots,\lfloor \tau n \rfloor - 1\}} \{Y_i^{(S)k}\}$
  1.2 $\max^i \leftarrow \max_{k \in \{0,1,\ldots,\lfloor \tau n \rfloor - 1\}} \{Y_i^{(S)k}\}$
  1.3 $\text{range}^i \leftarrow \max^i - \min^i$
2 return$(\alpha \frac{\text{range}^i}{\lfloor \tau n \rfloor})$

In equations 8, 9 and 10, we referred to $\mathfrak{s}_i$ as the standard deviation of variable y_i. In the search algorithms however, we have to point out which variables exactly are subject to these equations. To this end, we write $\check{\mathfrak{s}}_i$ for the actual standard deviation to use for variable i. Furthermore, we introduce the following notation:

$$\begin{cases} \mathfrak{S}\langle j_0, j_1, \ldots, j_{n-1} \rangle = \begin{bmatrix} \check{\mathfrak{s}}_{j_0}^2 & 0 & \ldots & 0 \\ 0 & \check{\mathfrak{s}}_{j_1}^2 & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \ldots & \check{\mathfrak{s}}_{j_{n-1}}^2 \end{bmatrix} \\ \check{\mathfrak{s}}_i = \varphi(i) \end{cases} \tag{11}$$

Using equation 11, the multivariate normal distribution over problem variables $Y_{j_0}, Y_{j_1}, \ldots, Y_{j_{n-1}}$ can now be written as $g_m((y_{j_0}, y_{j_1}, \ldots, y_{j_{n-1}}), (Y_{j_0}^{(S)i}, Y_{j_1}^{(S)i}, \ldots, Y_{j_{n-1}}^{(S)i}), \mathfrak{S}\langle j_0, j_1, \ldots, j_{n-1} \rangle)$.

Before specifying the estimation algorithm, we note that we assume in the pseudo–code that $\check{\mathfrak{s}}_i$ $(\forall_{i \in \{0,1,\ldots,l-1\}})$ are global variables, for which we do not allocate additional memory in the algorithms. Using these definitions, we can now specify algorithm REKENOEST to estimate the density functions once the structure is known:

REKENOEST($\varphi(\cdot)$)
1 ***for*** $i \leftarrow 0$ ***to*** $l-1$ ***do***
  1.1 $\check{\mathfrak{s}}_i \leftarrow \varphi(i)$
2 ***for*** $i \leftarrow 0$ ***to*** $l-1$ ***do***
  2.1 $\hat{P}(Y_{\omega_i} | Y_{\pi(\omega_i)_0}, Y_{\pi(\omega_i)_1}, \ldots, Y_{\pi(\omega_i)_{|\pi(\omega_i)|-1}}) \leftarrow$
$$\sum_{q=0}^{\lfloor \tau n \rfloor - 1} \frac{e^{-\sum_{j=0}^{|\pi(\omega_i)|-1} \frac{(y_{\pi(\omega_i)_j} - Y_{\pi(\omega_i)_j}^{(S)q})^2}{2\check{\mathfrak{s}}_{\pi(\omega_i)_j}^2}}}{\sum_{k=0}^{\lfloor \tau n \rfloor - 1} e^{-\sum_{j=0}^{|\pi(\omega_i)|-1} \frac{(y_{\pi(\omega_i)_j} - Y_{\pi(\omega_i)_j}^{(S)k})^2}{2\check{\mathfrak{s}}_{\pi(\omega_i)_j}^2}}} \frac{1}{\check{\mathfrak{s}}_{\omega_i}\sqrt{2\pi}} e^{\frac{-(y_{\omega_i} - Y_{\omega_i}^{(S)q})^2}{2\check{\mathfrak{s}}_{\omega_i}^2}}$$

In algorithm ReKeNoEst, the estimation of the density function uses function variables y_{ω_i}. Function variable y_{ω_i} is the actual variable, meaning that we can sample from the density function once values for problem variables $Y_{\pi(\omega_i)_0}, Y_{\pi(\omega_i)_1}, \ldots, Y_{\pi(\omega_i)_{|\pi(\omega_i)|}}$ have been sampled. As noted in appendix A, sampling from the conditional distribution in equation 10 can be done by first computing the ν_q, $q \in \{0, 1, \ldots, \lfloor \tau n \rfloor - 1\}$ and by then sampling from the one–dimensional Gaussian $g(y_{\omega_i}, Y_{\omega_i}^{(S)q}, \mathfrak{s}_{\omega_i})$ with probability ν_q. We can now define algorithm ReKeNoSam for sampling from the conditional kernels distribution as follows:

ReKeNoSam($\varphi(\cdot)$)
1 $\tilde{\nu} \leftarrow$ ***new array*** of ***real*** with size $\lfloor \tau n \rfloor$
2 ***for*** $i \leftarrow l-1$ ***downto*** 0 ***do***
2.1 ***if*** $|\pi(\omega_i)| = 0$ ***then***
2.1.1 $k \leftarrow \lfloor$ Random01() $\cdot \lfloor \tau n \rfloor \rfloor$
2.1.2 $Y_{\omega_i} \leftarrow$ sampleNormalDistribution$(Y_{\omega_i}^{(S)k}, \check{\mathfrak{s}}_{\omega_i})$
2.2 ***else then***
2.2.1 ***for*** $k \leftarrow 0$ ***to*** $\lfloor \tau n \rfloor - 1$ ***do***
2.2.1.1 $\tilde{\nu}[k] \leftarrow e^{-\sum_{j=0}^{|\pi(\omega_i)|-1} \frac{(Y_{\pi(\omega_i)_j} - Y_{\pi(\omega_i)_j}^{(S)k})^2}{2\check{\mathfrak{s}}^2_{\pi(\omega_i)_j}}}$
2.2.1.2 ***if*** $k > 0$ ***then***
2.2.1.2.1 $\tilde{\nu}[k] \leftarrow \tilde{\nu}[k] + \tilde{\nu}[k-1]$
2.2.2 $\varsigma \leftarrow$ Random01() $\cdot \tilde{\nu}[\lfloor \tau n \rfloor - 1]$
2.2.3 ***for*** $k \leftarrow 0$ ***to*** $\lfloor \tau n \rfloor - 1$ ***do***
2.2.3.1 ***if*** $\varsigma \leq \tilde{\nu}[k]$ ***then***
2.2.3.1.1 $Y_{\omega_i} \leftarrow$ sampleNormalDistribution$(Y_{\omega_i}^{(S)k}, \check{\mathfrak{s}}_{\omega_i})$
2.2.3.1.2 ***breakfor***

Now, by using a fixed pds structure $(\pi, \boldsymbol{\omega})$, we can immediately use the normal kernels distribution in the IDEA framework. For instance, we can use algorithm UnivariateDistribution that was introduced in previous work [4] in order to use a univariate distribution, which leads to an approach in the line of the compact GA [11], the UMDA [14], PBIL [2] or PBIL$_C$ [18] and other approaches in continuous spaces [7, 19], all of which use a univariate distribution. Note that algorithm ReKeNoSam has been defined so as to do less work if the univariate distribution is used. In the case of that distribution, we namely have that $\forall_{i \in \{0,1,\ldots,\lfloor \tau n \rfloor - 1\}} \langle \nu_i = \frac{1}{N} \rangle$. This slight improvement in algorithm ReKeNoSam causes it to run in $\mathcal{O}(l)$ time instead of $\mathcal{O}(l \tau n)$ time in the special case of the univariate distribution. In order to derive the search algorithms for non–fixed structures using the Kullback–Leibler divergence, we noted in section 2 that we require to know the multivariate differential entropy for the pdf we wish to work with. In order to derive this entropy measure, we first note that we have the following for the multivariate normal kernels pdf:

$$\ln(f(y_0, y_1, \ldots, y_{n-1})) =$$
$$-\ln(N) - \ln\left((2\pi)^{\frac{n}{2}}\right) - \ln\left(\prod_{j=0}^{n-1} \mathfrak{s}_j\right) + \ln\left(\sum_{i=0}^{N-1} e^{-\sum_{j=0}^{n-1} \frac{(y_j - y_j^{(S)i})^2}{2\mathfrak{s}_j^2}}\right)$$
$$= -\ln\left(N(2\pi)^{\frac{n}{2}} \prod_{j=0}^{n-1} \mathfrak{s}_j\right) + \ln\left(\sum_{i=0}^{N-1} e^{-\sum_{j=0}^{n-1} \frac{(y_j - y_j^{(S)i})^2}{2\mathfrak{s}_j^2}}\right)$$

The term involving the logarithm over the sum cannot be evaluated further and will have to be approximated numerically, giving additional overhead to the method of using the Kullback–Leibler divergence in the IDEA framework. We find that the multivariate differential entropy over n variables $y_0, y_1, \ldots, y_{n-1}$ that have a normel kernels distribution following equation 8, can be written as follows:

$$h(y_0, y_1, \ldots, y_{n-1}) = \tag{12}$$
$$-\int f(\boldsymbol{y}) \ln(f(\boldsymbol{y})) d\boldsymbol{y}$$
$$= \ln\left(N(2\pi)^{\frac{n}{2}} \prod_{j=0}^{n-1} \mathfrak{s}_j\right) \int f(\boldsymbol{y}) d\boldsymbol{y} - \int f(\boldsymbol{y}) \ln\left(\sum_{i=0}^{N-1} e^{-\sum_{j=0}^{n-1} \frac{(y_j - y_j^{(S)i})^2}{2\mathfrak{s}_j^2}}\right) d\boldsymbol{y}$$
$$= \frac{1}{2}\ln\left(N^2(2\pi)^n \prod_{j=0}^{n-1} \mathfrak{s}_j^2\right) - \int f(\boldsymbol{y}) \ln\left(\sum_{i=0}^{N-1} e^{-\sum_{j=0}^{n-1} \frac{(y_j - y_j^{(S)i})^2}{2\mathfrak{s}_j^2}}\right) d\boldsymbol{y}$$

In the remainder of the paper, we write $f_I(\boldsymbol{y})$ for the last integral over n variables in equation 12. By using the information gathered so far, we are able to write out the algorithms for the chain search, the tree search and the graph search. We denote these algorithms as ReKeNoChainSea, ReKeNoTreeSea and ReKeNoGraphSea respectively. Assuming that function $\varphi(\cdot)$ takes constant time, the running time for algorithm ReKeNoGraphSeaExact is $\mathcal{O}(l^3 + l\tau n)$ which is subsumed by the running time of the greedy graph search algorithm for the case when $\kappa > 1$. For all of the search algorithms, we have that we do not need to recompute the $\mathfrak{s}_i$ parameters as well as the ranges when performing the distribution estimate, as all of the required information will already have been computed by the search algorithms.

ReKeNoChainSea($\varphi(\cdot)$)
1 $a \leftarrow$ ***new array*** of ***integer*** with size l
2 ***for*** $i \leftarrow 0$ ***to*** $l-1$ ***do***
  2.1 $a[i] \leftarrow i$
3 $e \leftarrow 0$
4 $\omega_{l-1} \leftarrow a[e]$
5 ***for*** $i \leftarrow 0$ ***to*** $l-1$ ***do***
  5.1 $\min^{a[i]} \leftarrow \min_{k \in \{0,1,\ldots,\lfloor \tau n \rfloor - 1\}} \{Y_{a[i]}^{(S)k}\}$
  5.2 $\max^{a[i]} \leftarrow \max_{k \in \{0,1,\ldots,\lfloor \tau n \rfloor - 1\}} \{Y_{a[i]}^{(S)k}\}$
  5.3 $\text{range}^{a[i]} \leftarrow \max^{a[i]} - \min^{a[i]}$
  5.4 $\mathfrak{s}_{a[i]} \leftarrow \varphi(a[i])$
  5.5 $h_{a[i]} \leftarrow \frac{1}{2}\ln(\lfloor \tau n \rfloor^2 2\pi \mathfrak{s}_{a[i]}^2) - f_I((y_{a[i]}))$
  5.6 ***if*** $h_{a[i]} < h_{\omega_{l-1}}$ ***then***
    5.6.1 $\omega_{l-1} \leftarrow a[i]$
    5.6.2 $e \leftarrow i$
6 $\pi(\omega_{l-1}) \leftarrow \emptyset$
7 $a[e] \leftarrow a[l-1]$
8 ***for*** $i \leftarrow l-2$ ***downto*** 0 ***do***
  8.1 $e \leftarrow 0$
  8.2 $\omega_i \leftarrow a[e]$
  8.3 ***for*** $q \leftarrow 0$ ***to*** i ***do***
    8.3.1 $h_{\omega_{i+1}a[q]} \leftarrow \frac{1}{2}\ln(\lfloor \tau n \rfloor^2 4\pi^2 \mathfrak{s}_{\omega_{i+1}}^2 \mathfrak{s}_{a[q]}^2) - f_I((y_{\omega_{i+1}}, y_{a[q]}))$
    8.3.2 $h_{a[q]\omega_{i+1}} \leftarrow h_{\omega_{i+1}a[q]}$
    8.3.3 ***if*** $h_{a[q]\omega_{i+1}} - h_{\omega_{i+1}} < h_{\omega_i\omega_{i+1}} - h_{\omega_{i+1}}$ ***then***
      8.3.3.1 $\omega_i \leftarrow a[q]$
      8.3.3.2 $e \leftarrow q$
  8.4 $\pi(\omega_i) \leftarrow \omega_{i+1}$
  8.5 $a[e] \leftarrow a[i]$
9 return$((\pi, \boldsymbol{\omega}))$

ReKeNoTreeSea($\varphi(\cdot)$)

1 $a \leftarrow$ ***new array*** of ***integer*** with size l
2 ***for*** $i \leftarrow 0$ ***to*** $l-1$ ***do***
  2.1 $a[i] \leftarrow i$
3 $b^t \leftarrow$ ***new array*** of ***integer*** with size $l-1$
4 $e \leftarrow$RandomNumber(l)
5 $\omega_{l-1} \leftarrow e$
6 $\pi(\omega_{l-1}) \leftarrow \emptyset$
7 $\min^{\omega_{l-1}} \leftarrow \min_{k \in \{0,1,\ldots,\lfloor \tau n \rfloor - 1\}} \{Y^{(S)k}_{\omega_{l-1}}\}$
8 $\max^{\omega_{l-1}} \leftarrow \max_{k \in \{0,1,\ldots,\lfloor \tau n \rfloor - 1\}} \{Y^{(S)k}_{\omega_{l-1}}\}$
9 $\mathrm{range}^{\omega_{l-1}} \leftarrow \max^{\omega_{l-1}} - \min^{\omega_{l-1}}$
10 $\mathfrak{s}_{\omega_{l-1}} \leftarrow \varphi(\omega_{l-1})$
11 $h_{\omega_{l-1}} \leftarrow \frac{1}{2}\ln(\lfloor \tau n \rfloor^2 2\pi \mathfrak{s}^2_{\omega_{l-1}}) - f_I((y_{\omega_{l-1}}))$
12 $a[e] \leftarrow a[l-1]$
13 ***for*** $i \leftarrow 0$ ***to*** $l-2$ ***do***
  13.1 $\min^{a[i]} \leftarrow \min_{k \in \{0,1,\ldots,\lfloor \tau n \rfloor - 1\}} \{Y^{(S)k}_{a[i]}\}$
  13.2 $\max^{a[i]} \leftarrow \max_{k \in \{0,1,\ldots,\lfloor \tau n \rfloor - 1\}} \{Y^{(S)k}_{a[i]}\}$
  13.3 $\mathrm{range}^{a[i]} \leftarrow \max^{a[i]} - \min^{a[i]}$
  13.4 $\mathfrak{s}_{a[i]} \leftarrow \varphi(a[i])$
  13.5 $h_{a[i]} \leftarrow \frac{1}{2}\ln(\lfloor \tau n \rfloor^2 2\pi \mathfrak{s}^2_{a[i]}) - f_I((y_{a[i]}))$
  13.6 $b^t[a[i]] \leftarrow \omega_{l-1}$
  13.7 $h_{b^t[a[i]]a[i]} \leftarrow \frac{1}{2}\ln(\lfloor \tau n \rfloor^2 4\pi^2 \mathfrak{s}^2_{b^t[a[i]]} \mathfrak{s}^2_{a[i]}) - f_I((y_{b^t[a[i]]}, y_{a[i]}))$
  13.8 $h_{a[i]b^t[a[i]]} \leftarrow h_{b^t[a[i]]a[i]}$
14 ***for*** $i \leftarrow l-2$ ***downto*** 0 ***do***
  14.1 $e \leftarrow arg\ \max_j \{h_{a[j]} + h_{b^t[a[j]]} - h_{a[j]b^t[a[j]]}\}$ $(j \in \{0,1,\ldots,i\})$
  14.2 $\omega_i \leftarrow a[e]$
  14.3 $\pi(\omega_i) \leftarrow b^t[\omega_i]$
  14.4 $a[e] \leftarrow a[i]$
  14.5 ***for*** $j \leftarrow 0$ ***to*** $i-1$ ***do***
    14.5.1 $h_{\omega_i a[j]} \leftarrow \frac{1}{2}\ln(\lfloor \tau n \rfloor^2 4\pi^2 \mathfrak{s}^2_{\omega_i} \mathfrak{s}^2_{a[j]}) - f_I((y_{\omega_i}, y_{a[j]}))$
    14.5.2 $h_{a[j]\omega_i} \leftarrow h_{\omega_i a[j]}$
    14.5.3 $I_{best} \leftarrow h_{a[j]} + h_{b^t[a[j]]} - h_{a[j]b^t[a[j]]}$
    14.5.4 $I_{add} \leftarrow h_{a[j]} + h_{\omega_i} - h_{a[j]\omega_i}$
    14.5.5 ***if*** $I_{best} < I_{add}$ ***then***
      14.5.5.1 $b^t[a[j]] \leftarrow \omega_i$
15 return($(\pi, \boldsymbol{\omega})$)

ReKeNoGraphSea(κ, $\varphi(\cdot)$)

1 ***if*** $\kappa = 0$ ***then***
  1.1 $(\pi, \boldsymbol{\omega}) \leftarrow$UnivariateDistribution()
2 ***else if*** $\kappa = 1$ ***then***
  2.1 $(\pi, \boldsymbol{\omega}) \leftarrow$ReKeNoGraphSeaExact($\varphi(\cdot)$)
3 ***else then***
  3.1 $(\pi, \boldsymbol{\omega}) \leftarrow$ReKeNoGraphSeaGreedy($\varphi(\cdot)$)
4 return($(\pi, \boldsymbol{\omega})$)

```
ReKeNoGraphSeaExact(φ(·))
 1  define type edarc as
                (stack of integer, stack of integer, stack of real)
 2  v^p ← new array of vector of integer with size l
 3  V ← new vector of integer
 4  A ← new vector of edarc
 5  for i ← 0 to l − 1 do
    5.1  min^i ← min_{k∈{0,1,...,⌊τn⌋−1}}{Y_i^{(S)k}}
    5.2  max^i ← max_{k∈{0,1,...,⌊τn⌋−1}}{Y_i^{(S)k}}
    5.3  range^i ← max^i − min^i
    5.4  𝔰_i ← φ(i)
    5.5  V_{|V|} ← i
 6  for i ← 0 to l − 1 do
    6.1  for j ← 0 to l − 1 do
         6.1.1  if i ≠ j then
                6.1.1.1  c ← ½ln(⌊τn⌋²4π²𝔰_j²𝔰_i²) − f_I((y_j, y_i))
                6.1.1.2  c ← c − ½ln(⌊τn⌋²2π𝔰_i²) + f_I((y_i))
                6.1.1.3  S^s ← new stack of integer
                6.1.1.4  S^t ← new stack of integer
                6.1.1.5  S^c ← new stack of real
                6.1.1.6  push(S^s, i)
                6.1.1.7  push(S^t, j)
                6.1.1.8  push(S^c, −c)
                6.1.1.9  A_{|A|} ← (S^s, S^t, S^c)
 7  γ ← min_{(S^s,S^t,S^c)∈A}{top(S^c)}
 8  for i ← 0 to |A| − 1 do
    8.1  (S^s, S^t, S^c) ← A_i
    8.2  c ← pop(S^c)
    8.3  c ← c + 1 − γ
    8.4  push(S^c, c)
 9  B ← GraphSeaOptimumBranching(V,A,l)
10  for i ← 0 to |B| − 1 do
    10.1  (S^s, S^t, S^c) ← B_i
    10.2  s ← pop(S^s)
    10.3  t ← pop(S^t)
    10.4  v^p[t]_{|v^p[t]|} ← s
11  return(GraphSeaTopologicalSort(v^p))
```

In previous work [4], we already noted that by adding an arc (v_0, v_1) to the pds graph in the greedy graph search algorithm, we only have to recompute the entropy for variable v_1. In the case of the normal distribution, the complete determinant of the matrix $\mathbf{\Sigma}$ for all of the involved variables did not have to be recomputed. This is also the case for the normal kernels distribution. However, in the case of using that pdf, the update rule is less involved. Using linear algebra, we find the following:

$$\det(\mathfrak{S}\langle j_0, j_1, \ldots, j_n\rangle) = \frac{\det(\mathfrak{S}\langle j_0, j_1, \ldots, j_{n-1}\rangle)}{\mathfrak{S}\langle j_0, j_1, \ldots, j_n\rangle^{-1}(n,n)} = \det(\mathfrak{S}\langle j_0, j_1, \ldots, j_{n-1}\rangle)\check{\mathfrak{s}}_{j_n}^2 \qquad (13)$$

ReKeNoGraphSeaGreedy(κ, $\varphi(\cdot)$)
1 $a \leftarrow$ ***new array*** of ***boolean*** in 2 dimensions with size $l \times l$
2 $v^p, v^s \leftarrow 2$ ***new arrays*** of ***vector*** of ***integer*** with size l
3 $h^n, d_f^n, d_p^n, c \leftarrow 4$ ***new arrays*** of ***real*** in 2 dimensions with size $l \times l$
4 $h^o, d_f^o, d_p^o \leftarrow 3$ ***new arrays*** of ***real*** with size l
5 ***for*** $i \leftarrow 0$ ***to*** $l-1$ ***do***
  5.1 $\min^i \leftarrow \min_{k \in \{0,1,\ldots,\lfloor \tau n \rfloor - 1\}} \{Y_i^{(S)k}\}$
  5.2 $\max^i \leftarrow \max_{k \in \{0,1,\ldots,\lfloor \tau n \rfloor - 1\}} \{Y_i^{(S)k}\}$
  5.3 $\text{range}^i \leftarrow \max^i - \min^i$
  5.4 $\mathfrak{s}_i \leftarrow \varphi(i)$
  5.5 $d_f^o[i] \leftarrow \check{\mathfrak{s}}_i^2$
  5.6 $d_p^o[i] \leftarrow 0$
  5.7 $h^o[i] \leftarrow \frac{1}{2}\ln(\lfloor \tau n \rfloor^2 2\pi d_f^o[i]) - f_I((y_i))$
6 ***for*** $i \leftarrow 0$ ***to*** $l-1$ ***do***
  6.1 ***for*** $j \leftarrow 0$ ***to*** $l-1$ ***do***
    6.1.1 ***if*** $i \neq j$ ***then***
      6.1.1.1 $a[i,j] \leftarrow$ ***true***
      6.1.1.2 $d_f^n[i,j] \leftarrow \check{\mathfrak{s}}_j^2 \check{\mathfrak{s}}_i^2$
      6.1.1.3 $d_p^n[i,j] \leftarrow \check{\mathfrak{s}}_i^2$
      6.1.1.4 $h^n[i,j] \leftarrow \frac{1}{2}\ln(\lfloor \tau n \rfloor^2 4\pi^2 d_f^n[i,j]) - f_I((y_j, y_i))$
      6.1.1.5 $h^n[i,j] \leftarrow h^n[i,j] - \frac{1}{2}\ln(\lfloor \tau n \rfloor^2 2\pi d_p^n[i,j]) + f_I((y_i))$
      6.1.1.6 $c[i,j] \leftarrow h^n[i,j] - h^o[j]$
  6.3 $a[i,i] \leftarrow$ ***false***
7 $\gamma \leftarrow l^2 - l$
8 ***while*** $\gamma > 0$ ***do***
  8.1 $(v_0, v_1) \leftarrow arg\ \min_{(i,j)}\{c[i,j] \mid a[i,j] \wedge (i,j) \in \{0,1,\ldots,l-1\}^2\}$
  8.2 ***if*** $c[v_0, v_1] > 0$ ***then***
    8.2.1 ***breakwhile***
  8.3 $\gamma \leftarrow \gamma -$ GraphSeaArcAdd(κ, v_0, v_1, a, v^p, v^s)
  8.4 $d_f^o[v_1] \leftarrow d_f^n[v_0, v_1]$
  8.5 $d_p^o[v_1] \leftarrow d_p^n[v_0, v_1]$
  8.6 $h^o[v_1] \leftarrow h^n[v_0, v_1]$
  8.7 ***for*** $i \leftarrow 0$ ***to*** $l-1$ ***do***
    8.7.1 ***if*** $a[i, v_1]$ ***then***
      8.7.1.1 $d_f^n[i, v_1] \leftarrow d_f^o[v_1]\mathfrak{s}_i^2$
      8.7.1.2 $h^n[i, v_1] \leftarrow \frac{1}{2}\ln(\lfloor \tau n \rfloor^2 (2\pi)^{|v^p[v_1]|+2} d_f^n[i, v_1]) -$
        $f_I((y_{v_1}, y_{v^p[v_1]_0}, y_{v^p[v_1]_1}, \ldots, y_{v^p[v_1]_{|v^p[v_1]|-1}}, y_i))$
      8.7.1.3 $d_p^n[i, v_1] \leftarrow d_p^o[v_1]\mathfrak{s}_i^2$
      8.7.1.4 $h^n[i, v_1] \leftarrow h^n[i, v_1] - \frac{1}{2}\ln(\lfloor \tau n \rfloor^2 (2\pi)^{|v^p[v_1]|+1} d_p^n[i, v_1]) +$
        $f_I((y_{v^p[v_1]_0}, y_{v^p[v_1]_1}, \ldots, y_{v^p[v_1]_{|v^p[v_1]|-1}}, y_i))$
      8.7.1.5 $c[i,j] \leftarrow h^n[i, v_1] - h^o[v_1]$
9 return(GraphSeaTopologicalSort(v^p))

4 Discussion

Optimization algorithms that build and use probabilistic models, such as the IDEAs, take up more time in finding a probability density function and subsequently sampling from it, as the allowed complexity of the pds goes up. As noted in previous work [4], it is therefore important to be aware of the running times of the parts of an algorithm. We note that because of equation 12, we have to numerically approximate an integral every time we compute the joint differential entropy. We shall assume at this point that this takes constant time and shall therefore disregard it in the presentation of the asymptotic running times of the algorithms in this paper:

Algorithm	Complexity
ReKeNoChainSea()	$\mathcal{O}(l^2 + l\tau n)$
ReKeNoTreeSea()	$\mathcal{O}(l^2 + l\tau n)$
ReKeNoGraphSea()	$\mathcal{O}(l^3\kappa + l^2 + l\tau n)$
ReKeNoEst()	$\mathcal{O}(l\tau n)$
ReKeNoSam()	$\mathcal{O}(l\kappa\tau n)$

The running time for algorithm ReKeNoEst was determined as the worst case with respect to the fact that the ranges still have to be computed. This means that it hasn't been done by a search algorithm previously. Also, we assume that $\varphi(\cdot) = \text{LinearScaling}(\alpha, \cdot)$. Otherwise, the running time of algorithm ReKeNoEst would be $\mathcal{O}(1)$.

The competence of the algorithms presented in this paper should be tested on test functions. Preliminary results have already shown however that the normal kernels distribution approach leads to algorithms that give good results on certain problems but that they are very very sensitive to parameter settings such as α in algorithm LinearScaling. However, the running times for those algorithms are much larger if $\kappa > 1$, than when using only the normal distribution, especially if the population size increases. If we base the estimation on $\lfloor \tau n \rfloor$ samples and generate $n - \lfloor \tau n \rfloor$ new samples every iteration, the running time for sampling in the case of using the normal distribution, taking τ as a constant, amounts to $\mathcal{O}(nl\kappa)$. In the case of the normal kernels distribution however, the running time amounts to $\mathcal{O}(n^2 l\kappa)$.

This is one of the reasons why a normal mixture distribution might provide some very useful properties. Sampling from a normal mixture distribution with M kernels will most likely take $\mathcal{O}(Mnl\kappa)$ time. If M is taken as a constant, we find the same running time as in the case of using the normal distribution. Furthermore, the mixture model is not as global and cluster–insensitive as is the normal distribution and neither perhaps over–sensitive to clusters as is the normal kernels distribution. Therefore, further exploring the possibilities of using mixture models seems very worthwhile.

We have used the Kullback–Leibler divergence as a metric to guide the search for a pds. Note that using this metric as well as the normal kernels distribution is merely an *instance* of the IDEA framework and that the derived algorithms should be seen independently from it. Furthermore, using the Kullback–Leibler divergence is not very effective if there are no strong constraints on the pds to be learned because the Kullback–Leibler divergence is merely a distance metric to the full joint pds. This means that any search algorithm guided by this metric will simply search for a model as close as possible to that distribution. A better means of exploiting problem structure would be to explore the dependencies in the data and base conditionalities on statistical tests. Combining this with the use of normal mixture models could very well lead to competent evolutionary algorithms for continuous optimization problems.

5 Conclusions

We have presented mathematical tools that allow us to define iterated density estimation evolutionary algorithms using the normal kernels distribution. The search metric we used for a set of search algorithms is the differential entropy. The normal kernels distribution has some advantages over the normal distribution. One of these is the fact that it allows us to estimate certain aspects of a probability density in a better way. One of the main drawbacks is that this comes at the expense of the running time of the resulting algorithms.

The presented algorithms yet have to be brought into practice. However, as the variety in probability density functions applied to optimization by iterated density estimation evolutionary algorithms increases, more promising methods for problems with certain characteristics are created.

References

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

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

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

[4] P.A.N. Bosman and D. Thierens. An algorithmic framework for density estimation based evolutionary algorithms. Utrecht University Technical Report UU–CS–1999–46. ftp://ftp.cs.uu.nl/pub/RUU/CS/techreps/CS-1999/1999-46.ps.gz, 1999.

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

[6] T.M. Cover and J.A. Thomas. *Elements of Information Theory*. John Wiley & Sons Inc., 1991.

[7] M. Gallagher, M. Fream, and T. Downs. Real–valued evolutionary optimization using a flexible probability density estimator. In W. Banzhaf, J. Daida, A.E. Eiben, M.H. Garzon, V. Honavar, M. Jakiela, and R.E. Smith, editors, *Proceedings of the GECCO–1999 Genetic and Evolutionary Computation Conference*, pages 840–846. Morgan Kaufmann Publishers, 1999.

[8] D.E. Goldberg. *Genetic Algorithms In Search, Optimization, And Machine Learning*. Addison–Wesley, Reading, 1989.

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

[10] G. Harik. Linkage learning via probabilistic modeling in the ecga. IlliGAL Technical Report 99010. ftp://ftp-illigal.ge.uiuc.edu/pub/papers/IlliGALs/99010.ps.Z, 1999.

[11] 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.

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

[13] H. Mühlenbein, T. Mahnig, and O. Rodriguez. Schemata, distributions and graphical models in evolutionary optimization. *Journal of Heuristics*, 5:215–247, 1999.

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

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

[16] M. Pelikan, D.E. Goldberg, and F. Lobo. A survey of optimization by building and using probabilistic models. IlliGAL Technical Report 99018. ftp://ftp-illigal.ge.uiuc.edu/pub/papers/IlliGALs/99018.ps.Z, 1999.

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

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

[19] I. Servet, L. Trave-Massuyes, and D. Stern. Telephone network traffic overloading diagnosis and evolutionary computation technique. In J.K. Hao, E. Lutton, E. Ronald, M. Schoenauer, and D. Snyers, editors, *Proceedings of Artificial Evolution '97*, pages 137–144. Springer Verlag, LNCS 1363, 1997.

A Multivariate conditional normal kernels distribution

Using equations 2 and 8, we can write the required conditional pdf as follows:

$$f(y_0|y_1,y_2,\ldots,y_n) =$$

$$\frac{\frac{1}{N}\sum_{i=0}^{N-1}\frac{(2\pi)^{-\frac{n+1}{2}}}{\prod_{j=0}^{n}\mathfrak{s}_j}e^{-\sum_{j=0}^{n}\frac{(y_j-y_j^{(S)i})^2}{2\mathfrak{s}_j^2}}}{\frac{1}{N}\sum_{i=0}^{N-1}\frac{(2\pi)^{-\frac{n}{2}}}{\prod_{j=1}^{n}\mathfrak{s}_j}e^{-\sum_{j=1}^{n}\frac{(y_j-y_j^{(S)i})^2}{2\mathfrak{s}_j^2}}}$$

$$= \frac{\frac{1}{N}\sum_{i=0}^{N-1}\frac{(2\pi)^{-\frac{n+1}{2}}\prod_{j=1}^{n}\mathfrak{s}_j}{(2\pi)^{-\frac{n}{2}}\prod_{j=0}^{n}\mathfrak{s}_j}e^{-\sum_{j=0}^{n}\frac{(y_j-y_j^{(S)i})^2}{2\mathfrak{s}_j^2}}}{\frac{1}{N}\sum_{i=0}^{N-1}e^{-\sum_{j=1}^{n}\frac{(y_j-y_j^{(S)i})^2}{2\mathfrak{s}_j^2}}}$$

$$=_{\text{def}} \left\{ \begin{array}{ll} \lambda_i &= \sum_{j=1}^{n}\frac{(y_j-y_j^{(S)i})^2}{2\mathfrak{s}_j^2} \\ \Lambda &= \sum_{k=0}^{N-1}e^{-\lambda_k} \end{array} \right\}$$

$$\frac{1}{N}\frac{N}{\Lambda}\sum_{i=0}^{N-1}\frac{1}{\mathfrak{s}_0\sqrt{2\pi}}e^{-\sum_{j=0}^{n}\frac{(y_j-y_j^{(S)i})^2}{2\mathfrak{s}_j^2}}$$

$$= \frac{1}{N}e^{\ln(N)-\ln(\Lambda)}\sum_{i=0}^{N-1}\frac{1}{\mathfrak{s}_0\sqrt{2\pi}}e^{\frac{-(y_0-y_0^{(S)i})^2}{2\mathfrak{s}_0^2}-\sum_{j=1}^{n}\frac{(y_j-y_j^{(S)i})^2}{2\mathfrak{s}_j^2}}$$

$$= \frac{1}{N}\sum_{i=0}^{N-1}\frac{1}{\mathfrak{s}_0\sqrt{2\pi}}e^{\frac{-(y_0-y_0^{(S)i})^2}{2\mathfrak{s}_0^2}+\ln(N)-\ln(\Lambda)-\lambda_i}$$

$$= \frac{1}{N}\sum_{i=0}^{N-1}\frac{1}{\mathfrak{s}_0\sqrt{2\pi}}e^{\frac{-(y_0-y_0^{(S)i})^2-2\mathfrak{s}_0^2(-\ln(N)+\ln(\Lambda)+\lambda_i)}{2\mathfrak{s}_0^2}}$$

$$= \frac{1}{N}\sum_{i=0}^{N-1}\frac{1}{\mathfrak{s}_0\sqrt{2\pi}}e^{\frac{b_i}{2\mathfrak{s}_0^2}}$$

If the expression that will result is to be of a closed form, so that we may write $f(y_0|y_1,y_2,\ldots,y_n) = \sum_{i=0}^{N-1} g(y,\tilde{\mu}_0^i,\mathfrak{s}_0)$, rewriting b_i should lead us to $\tilde{\mu}_0^i$:

$$\begin{aligned} b_i &= -y_0^2 + 2y_0y_0^{(S)i} - (y_0^{(S)i})^2 - 2\mathfrak{s}_0^2(-\ln(N)+\ln(\Lambda)+\lambda_i) \\ &= -(y_0^2 - 2y_0y_0^{(S)i} + (y_0^{(S)i})^2 + 2\mathfrak{s}_0^2(-\ln(N)+\ln(\Lambda)+\lambda_i)) \\ &=_{\text{def}} \left\{ \begin{array}{l} \alpha_i = -2y_0^{(S)i} \\ \beta_i = (y_0^{(S)i})^2 + 2\mathfrak{s}_0^2(-\ln(N)+\ln(\Lambda)+\lambda_i) \end{array} \right\} - (y_0^2 + \alpha_i y_0 + \beta_i) \end{aligned}$$

So that when we realize that $(y_0-\tilde{\mu}_0^i)^2 = y_0 - 2\tilde{\mu}_0^i y_0 + (\tilde{\mu}_0^i)^2$, we find that if we have $(\frac{\alpha_i}{-2})^2 = \beta_i$, we may write:

$$\tilde{\mu}_0^i = \sqrt{\beta_i} = \frac{\alpha_i}{-2} = y_0^{(S)i}$$

However, we do not have in general that $(\frac{\alpha_i}{-2})^2 = \beta^i$, as this would lead to the univariate distribution. Given the uniform dataset in figure 1, figure 5 shows an example of a case in which we do not have $(\frac{\alpha_i}{-2})^2 = \beta^i$.

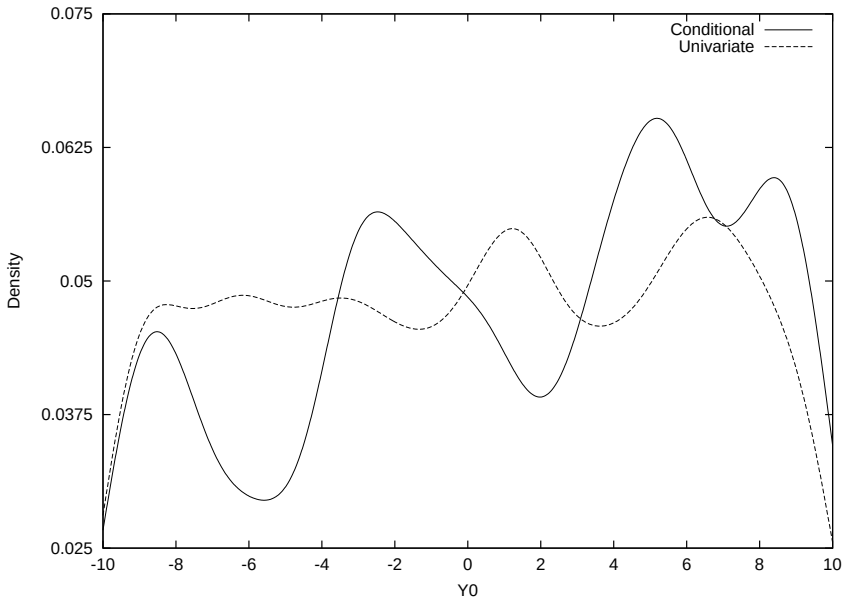

Figure 5: The univariate distribution density estimate $f(y_0)$ and the conditional distribution density estimate $f(y_0|y_1)$ with $y_1 = -10$ for the uniform dataset, using the normal kernels distribution with $\mathfrak{s}_0 = \mathfrak{s}_1 = 1.0$

Because the required conditional density function is not of a closed form, it perhaps seems that it is no longer straightforward to sample from it. However, this is not entirely the case. This becomes clear if we introduce the following definition:

$$\nu_i \;=\; e^{-\ln(\Lambda)-\lambda_i} = \frac{1}{\Lambda} e^{-\lambda_i} \tag{14}$$

We may now continue the derivation of the required conditional pdf as follows:

$$\begin{aligned} & f(y_0|y_1, y_2, \ldots, y_n) \\ = \; & \frac{1}{N} \sum_{i=0}^{N-1} \frac{1}{\mathfrak{s}_0\sqrt{2\pi}} e^{\frac{-(y_0 - y_0^{(S)i})^2}{2\mathfrak{s}_0^2}} e^{\ln(N)-\ln(\Lambda)-\lambda_i} \\ = \; & \sum_{i=0}^{N-1} \nu_i \frac{1}{\mathfrak{s}_0\sqrt{2\pi}} e^{\frac{-(y_0 - y_0^{(S)i})^2}{2\mathfrak{s}_0^2}} \\ = \; & \sum_{i=0}^{N-1} \nu_i g(y_0, y_0^{(S)i}, \mathfrak{s}_0) \end{aligned}$$

As the last expression in the above derivation is a weighted sum of univariate Gaussian density functions, we can quite easily sample from the conditional density function. Given values for $y_1, y_2, \ldots, y_n$, we can compute $\nu_i, i \in \{0, 1, \ldots, N-1\}$. We then select the univariate Gaussian pdf $g(y_0, y_0^{(S)i}, \mathfrak{s}_0)$ with probability $\nu_i / \sum_{k=0}^{N-1} \nu_k$. From the definition of ν_i, it follows that $\sum_{k=0}^{N-1} \nu_k = 1$, so actually we may select the univariate Gaussian pdf $g(y_0, y_0^{(S)i}, \mathfrak{s}_0)$ with probability ν_i.