IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received January 3, 2022, accepted January 25, 2022, date of publication February 7, 2022, date of current version February 15, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3149280

Efficient Density Estimation for High-Dimensional Data

AREF MAJDARA, (Member, IEEE), AND SAEID NOOSHABADI, (Senior Member, IEEE)
Department of Electrical and Computer Engineering, Michigan Technological University, Houghton, MI 49931, USA
Corresponding author: Aref Majdara (amajdara@mtu.edu)

ABSTRACT Multivariate density estimation methods typically work well in low dimensions and their extension to data analytics in high dimensions domain has proven challenging. For density estimation in high-dimensional big data domains, the non-parametric Bayesian sequential partitioning (BSP) algorithm provides an efficient way of partitioning the sample space, based on Bayesian inference. In this paper, we present a detailed analysis of BSP and provide a computationally efficient copula-transformed data structure and algorithm for use in density estimation for data analytics in high dimensions. Using the copula-transformed data structure, we implement the density estimation for marginals in both BSP and kernel density estimation (KDE) methods. The data structures and algorithm are suitably designed for most efficient rendering into parallel processing paradigms of open multi-processing (OPENMP®) and message passing interface (MPI).

INDEX TERMS Multivariate density estimation, non-parametric, high-dimensional, Bayesian sequential partitioning, copula transform.

I. INTRODUCTION

A variety of modern real-world big data applications including sensing technologies, security, financial trading, epidemiology, networks and scientific experiments, heavily rely on an adequate analysis of transient data streams [1]. To extract meaningful information from these big data streams, techniques in data-driven machine learning and analytics provide promising solutions [2]. A fundamental building block of many data mining and analysis approaches is density estimation, providing a very natural way of investigating the properties of datasets [3]. Density estimates can give valuable indication of such features as skewness and multimodality in the data when modeling the probabilistic or stochastic structure of a dataset [4]. It also provides a well-defined estimation of a continuous data distribution, a fact which makes its adaptation to data streams desirable. They can be used as the basis of a range of statistical analyses and machine learning techniques, including non-parametric discriminant analysis [5], classification, feature analysis [6], cluster analysis, and bump hunting [7]. Statistical analysis of big data typically requires analytics in high-dimensional domain, where the commonly used techniques, such as kernel density estimation (KDE) [8], fail to perform [1], [41]. Density estimation is defined as the process of constructing an estimate of the probability density function (PDF), from a set of observed data. For a D-dimensional random vector $\boldsymbol{X} = (X_1, \cdots, X_D)$, its PDF $f_{\boldsymbol{X}}(x)$, where the vector $x = (x_1, x_2, \cdots, x_D)$, is used to calculate the probability of $\boldsymbol{X}$ lying in a certain D-dimensional domain $D = (\mathcal{D}_1, \mathcal{D}_2, \cdots, \mathcal{D}_D)$, is defined as,

$$P(\boldsymbol{X} \in D) = \int_D f_{\boldsymbol{X}}(x)\mathbf{d}x \tag{1}$$

where $\mathrm{d}x = \mathrm{d}x_1\mathrm{d}x_2 \cdots \mathrm{d}x_D$.

Typical parametric methods [4], [10], [11] of density estimation assume that the data is coming from a known family of distributions, *e.g.* normal with mean μ and variance σ^2, and try to estimate the parameters. However, in high-dimensional big data domain, parametric methods become largely inefficient, as due to sparsity of data in some areas of the sample space (*a.k.a. curse of dimensionality* [10], [12]), the number of parameters rapidly increases with the sample size and dimension.

Non-parametric methods are in general more suited for high-dimensional big datasets, where the data have no characteristic structure. In these methods, the number of parameters is not fixed, which makes them more flexible than parametric methods. Kernel density estimation (KDE) [4], [10], [11], [13]–[15] is one of the most commonly used

The associate editor coordinating the review of this manuscript and approving it for publication was Chee Keong Kwoh.

non-parametric estimation methods. A kernel density estimator for a D-dimensional dataset with N samples is defined by [4],

$$\hat{f}(\boldsymbol{x}) = \frac{1}{Nh^D} \sum_{i=1}^{N} K\left(\frac{\boldsymbol{x} - \boldsymbol{X_i}}{h}\right) \tag{2}$$

where K is the KDE function, $\boldsymbol{X_i}$ represents the i^{th} sample and h is called the *bandwidth*. More generally, a different bandwidth can be used for each of the dimensions, with h replaced with $\mathbf{H}$, a $D \times D$, symmetric, and positive-definite matrix KDE expressed as,

$$\hat{f}(x, \mathbf{H}) = \frac{1}{N} \sum_{i=1}^{N} |\mathbf{H}|^{-1/2} K(\mathbf{H}^{-1/2}(\boldsymbol{x} - \boldsymbol{X_i})) \tag{3}$$

A proper choice of bandwidth becomes critical in high-dimensional problems [16]–[23] and in fact, KDE method fails to work when the dimension becomes higher than 4 or 5 [9].

The other non-parametric method is the *histogram* with fixed multi-dimensional bin volume. It works based on the simple idea of dividing the sample space into equal multi-dimensional *bins* of volume h^D and then counting the number of data points in each bin as a measure of its density. However, with equally spaced bins, it is not possible to adapt to spatially varying smoothness [24]. Even more significantly, in multivariate space, the density function may vary unevenly across the dimensions.

To deal with this, various histogram methods with adaptive choice of the bin volume have been proposed [24], [25]. For a variable bin volume histogram with fixed bin size projected along each dimension, the bin volume can be expressed as $h = h_1 h_2 \cdots h_D$. For a more general case of variable bin sizes along each dimension, a D-dimensional bin volume can be expressed as $h_{\mathbf{j}=(j_1,\cdots,j_D)} = \prod_{d=1}^{D} h_{j_d}$, with $1 \leq j_d \leq j_{d_{max}}$. In general $h_{j_d} \neq h_{k_d}$, or more generally $h_{\mathbf{j}} \neq h_{\mathbf{k}}$. For a sample $\boldsymbol{X} = \mathbf{x}$, located in the D-dimensional bin volume of $h_{\mathbf{j}}$, the density is estimated as [4],

$$f(x) = \frac{1}{N} \times \frac{n_{\mathbf{j}}}{h_{\mathbf{j}}} \tag{4}$$

where N is the total number of samples, and $n_{\mathbf{j}}$ is the data count in bin volume $h_{\mathbf{j}}$.

Even with variable bin sizes, the direct application of histogram with regular grid structure becomes impractical as the number of bins required grows exponentially with the dimension. As an example, for a 10-dimensional test case in MATLAB®, the maximum number of bins that can be allocated before the system runs out of memory is 8^{10}. This corresponds to only eight bin segments along each dimension!

To significantly reduce the number of bins, requires partitioning the sample space into irregular partitions of various sizes. The method of adaptive histogram, in which the irregular bin volumes are chosen in a data-dependent way, has been proposed before [24]–[28]. Data-dependent *Bayesian sequential partitioning* (BSP) method using sequential importance sampling (SIS) [29], [30] has been proposed [9] as an efficient way of partitioning the sample space, based on Bayesian inference [1]. Non-parametric density estimation methods, like BSP, have an appeal in physical sciences, due to the fact that they allow embedding of physical prior belief in the analysis. Further, they provide a straightforward path to obtain predictive distribution, and more generally, spectral inference, by means of posterior draws [31].

In this paper, we present a copula-transformed BSP algorithm in details, analyze its computational complexity over a range of input parameters, and propose an efficient set of data structures and algorithm for the density estimation in high dimensions that are suitably designed for most efficient rendering into parallel processing paradigms of open multi-processing (OPENMP®) and message passing interface (MPI)). To evaluate the efficiency of our design, we have employed a set of synthetic datasets, to measure the algorithmic accuracy, using the Kullback-Leibler (KL) divergence [24] metric, and evaluate the computational complexity of the BSP algorithm.

This paper is organized as follows. Section II presents density estimation using BSP algorithm. Section III presents our use of copula transform technique to reduce the complexity of the BSP algorithm. Section IV presents the proposed data structures for the efficient implementation of BSP and discusses different implementation-related issues. The simulation results for some example cases are presented in Section V. Section VI discusses the analysis of the copula-transformed BSP computational complexity and parallelization of the proposed algorithm using the various features of the algorithm. We will look at some related work in Section VII. Finally, Section VIII contains the concluding remarks.

II. HIGH-DIMENSIONAL DENSITY ESTIMATION USING BAYESIAN SEQUENTIAL PARTITIONING (BSP)

A. BINARY PARTITIONING SCHEME

As an alternative to regular histograms, the sample space can be partitioned using a *binary partitioning* (BP) scheme, in which only binary cuts are allowed, i.e. each subregion can only be cut into two smaller subdivisions. The most convenient choice for location of the binary cut would be cutting in the middle of the subregion, i.e., into two equal halves [1]. Fig. 1 illustrates the mid-point BP scheme, in 2-dimensional space. The method works on the idea of choosing the best partitioning scenario after a certain number of cuts, based on some chosen criterion. However, it is not practically feasible to exhaustively generate all the possible scenarios, because the number of paths grows very rapidly, even in low dimensions. For a D-dimensional sample space, at each level j, there are $(j-1) \times D$ possible ways to cut each of the existing partition subregions. It can be shown that the total number of possible ways to partition the sample space into j BP subregions is $(j-1)! \times D^{(j-1)}$. For the 2-dimensional space shown in Fig. 1, generation of 50 BP subregions,

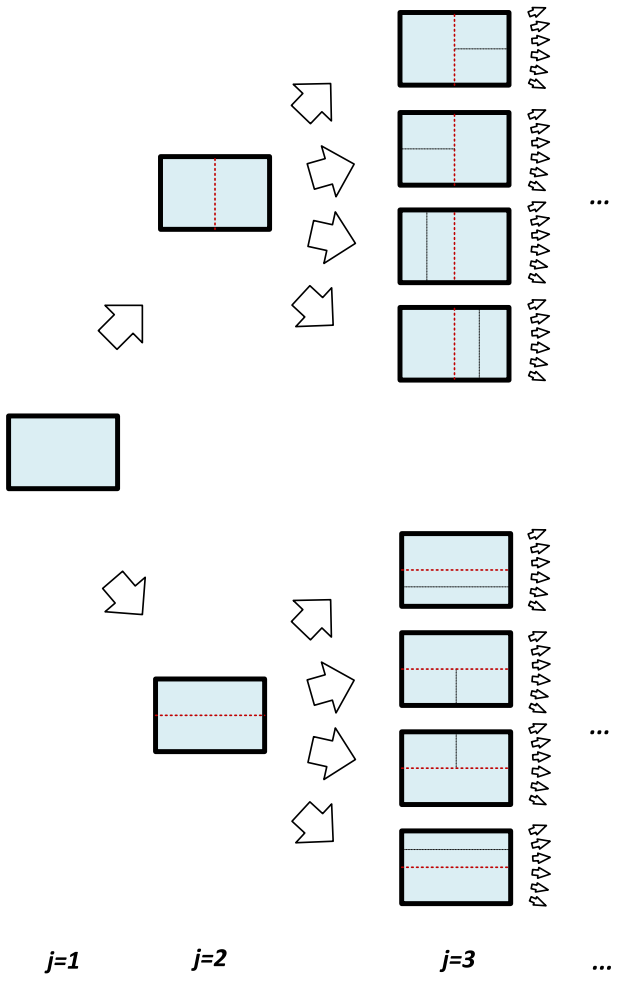


FIGURE 1. Example of mid-point binary partitioning scheme in 2D sample space.

requires evaluation of 3.4×10^{77} possibilities! Thus, instead of exhaustively creating and examining all possible sample partitions, we need to randomly generate a certain number of sample partitions to maintain a good diversity [1]. In [9], a posterior probability is proposed for this purpose, as part of the BSP method that will be described next.

A rather less common choice for the location of the binary cuts is cutting the subregion at the median point where resulting subregions will have the same number of samples [32], rather than equal volumes. In comparison to the mid-point BP scheme, this scheme requires an additional step of searching for the median of the data, to determine the location of the cut. While the BSP method described in the next subsection is based on mid-point BP scheme, we will also discuss the median-based BP scheme in Section III-C.

B. BSP ALGORITHM

Consider a D-dimensional dataset with N sample points, expressed as an $N \times D$ matrix. In BSP, the smallest D-dimensional sample space containing the dataset is progressively divided into subregions where the density in each of the divided subregions is estimated by simply counting the number of data points that it contains. The algorithm follows a BP scheme, *i.e.* each cut at a given level j splits one of the subregions into two equal halves. At $j = 1$, we start with the entire sample space. At $j = 2$, we examine all possible cuts, *i.e.* one cut along each dimensions, with a total of D cuts. Using the subregion densities, one of the dimensions is suitably chosen for the splitting the sample space using the BP. To improve the quality of density estimation, M independent paths are tried at each level (M sample partitions). At each level j, path ${g_j}^m = \{cut_2^m, cut_3^m, \cdots, cut_{j-1}^m\}$, $(m = 1, \cdots, M)$; the sample space contains $(j-1)$ subregions $(p = 1, \cdots, j-1)$, with subregion p having a volume of v_p, and containing n_p data points. There are $(j-1) \times D$ possibilities for the j^{th} cut. Enumerating the possibilities as $pd = 11, \cdots, 1D, \cdots, (j-1)D$, a conditional probability $s_{j_{pd}}$ is calculated for each of these cuts as [9]:

$$s_{j_{pd}}(cut_{j_{pd}}|g_{j-1}) = C_{j-1} 2^{n_p} \frac{\Gamma(n_{pd}^{(1)})\Gamma(n_{pd}^{(2)})}{\Gamma(n_p)} \tag{5}$$

where $\Gamma()$ denotes the Gamma function [33], $n_{pd}^{(1)}$ and $n_{pd}^{(2)}$ are data points in each of the resulting halves due to the cut in subregion p along dimension d, and C_{j-1} is a normalizing constant. The sum of all $(j-1) \times D$ conditional probabilities are normalized to unity to construct a probability mass function (PMF), which is used to make the random cut at level j in the chosen subregion p and dimension d, to generate the new subregion $p = j$. Subsequent to the cut, the data structures holding the information (the number of data points, volumes and coordinates) for the two new subregions, are updated accordingly. This process is repeated until either the best possible partition is obtained, or the number of cuts reaches the maximum value set by the user. Once the optimum partition is obtained, probability density is estimated for each subregion $1 \le p \le j$ (a D-dimensional bin), as $n_p/(Nv_p)$.

The best partition is the one that gives the lowest error between the actual and estimated densities. However, since the actual density of the data is unknown for real datasets, the algorithm needs to be able to determine the best partition without relying on the knowledge of the actual density.

It has been shown in [9] that the log of the posterior distribution of a sample partition, $\pi(m)$ is a linear function of the KL divergence (KLD) between the actual and estimated densities, with a negative slope. Thus, in order to minimize the KLD, we use the sample partition m with highest $\log(\pi(m))$. To do so, for each sample partition $m \in 1, \cdots, M$, with j levels, a partition *score* is defined as [9]:

$$\begin{aligned} score(m) &= \log(\pi(m)) \\ &= -\beta j + \log\left(\frac{\mathrm{B}(n_1+\alpha, \cdots, n_j+\alpha)}{\mathrm{B}(\alpha, \cdots, \alpha)}\right) \\ &\quad - \sum_{p=1}^{j} n_p \log(|v_p|) \end{aligned} \tag{6}$$

where $\alpha \in [0, 0.5]$ and $\beta \in [0.5, 1]$ are two constants. v_p and n_p are the volume and the number of data points in the subregion p, respectively. $\mathrm{B}(u_1, \cdots, u_K)$ denotes the multivariate version of *Beta-function* [34] and is expressed in terms of Γ-*function* as $B(u_1, \cdots, u_K) = \prod_{k=1}^{K} \Gamma(u_k) / \Gamma(\sum_{k=1}^{K} u_k)$. For an update in the value of the maximum partition score at level j, we stop the BSP algorithm if the score does not improve within a further fixed number of partitioning levels, Δj. We have used a value of $\Delta j = 10$ in this work. At this point, partition with maximum score, *i.e. maximum a-posterior* (), is chosen as the best partition.

1) BSP EXAMPLE

Consider the Gaussian mixture distribution in D-dimensions,

$$X \sim \sum_{r=1}^{R} c_r \mathcal{N}_r(\boldsymbol{\mu}_r, \boldsymbol{\Sigma}_r) \tag{7}$$

where $\mathcal{N}_r(\boldsymbol{\mu}_r, \boldsymbol{\Sigma}_r)$ is a normal distribution with the D-dimensional mean vector $\boldsymbol{\mu}_r = (\mu_1, \mu_2, \ldots, \mu_D)_r$; $\boldsymbol{\Sigma}_r = \mathrm{Cov}[X_i, X_j]_r$ is a $D \times D$ covariance matrix with $i, j = 1, 2, \cdots, D$, and c_r are the mixture weights. Fig. 2 presents the sample data points for $N = 20,000$ for the case of $R = 3$ and $D = 2$, with the following parameters [1]:

$$\boldsymbol{\mu_1} = [2.25, 5.40], \quad \boldsymbol{\mu_2} = [2.60, 5.65], \quad \boldsymbol{\mu_3} = [2.8, 5.15],$$
$$\boldsymbol{\Sigma}_1 = \begin{bmatrix} 0.04^2 & 0 \\ 0 & 0.04^2 \end{bmatrix}, \quad \boldsymbol{\Sigma}_2 = \begin{bmatrix} 0.07^2 & 0 \\ 0 & 0.07^2 \end{bmatrix},$$
$$\boldsymbol{\Sigma}_3 = \begin{bmatrix} 0.04^2 & 0 \\ 0 & 0.04^2 \end{bmatrix}$$
$$c_1 = 0.25, c_2 = 0.4, c_3 = 0.35$$

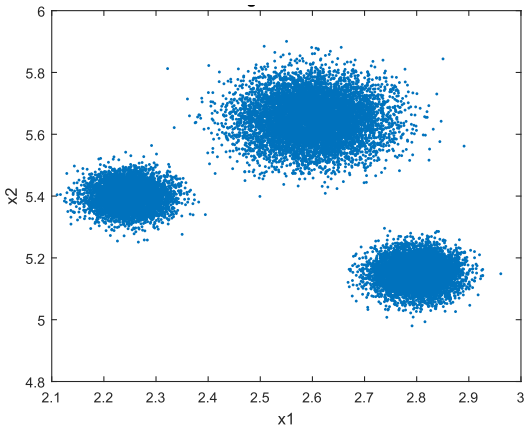


FIGURE 2. Sample data ($N = 20,000$) from a trimodal bivariate normal distribution.

Fig. 3 presents the actual density of the data in Fig. 2, obtained from (7). Fig. 4 presents the result of applying BSP on the sample space of Fig. 2, with $M = 200$ sample partitions, and $j = 182$ cuts.

We used the dataset shown in Fig. 2 to do a comparison with the Histogram Transform (HT) method proposed in [49]. Performances of the two methods are compared using KLD and Hellinger distance (HLGR). The results presented in Table 1 show that performance of BSP is significantly higher than the HT method.

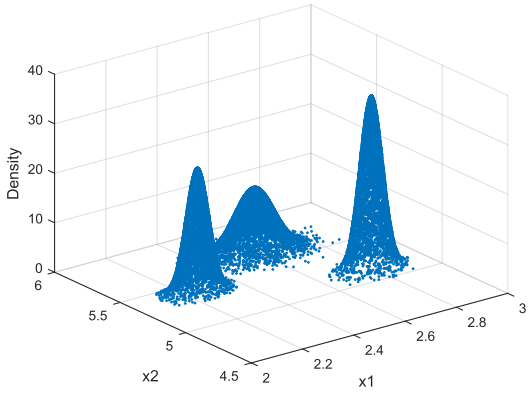


FIGURE 3. Actual joint density of the data in Fig. 2.

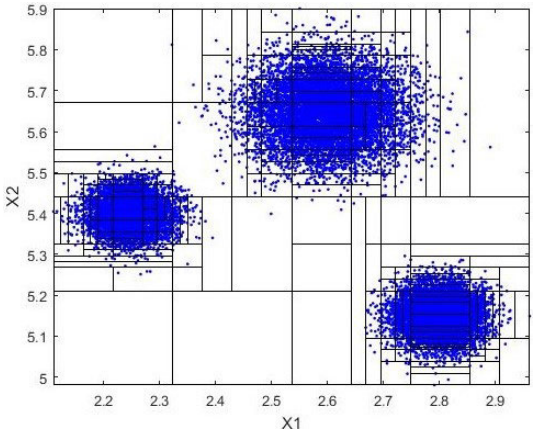


FIGURE 4. BSP cuts on the sample space of Fig. 2, with $N = 20,000$ and $M = 200$. The number of BSP cuts is 182.

TABLE 1. KL divergence (KLD) and Hellinger distance (HLGR) values for density estimation on the data shown in Fig. 2, obtained form BSP and HT methods.

	BSP	HT
KLD	**0.018**	0.0375
HLGR	**0.006**	0.015

III. COMPLEXITY REDUCTION THROUGH COPULA TRANSFORM

Further the true distribution is from uniform, the BSP requires more cuts to capture the structure of the data. This effect results in dramatic increase in the number of cuts and the deterioration of KLD values for high-dimensional data. The complexity arises due to the fact that in BSP the marginal distributions are learned together with the joint one. To reduce the number of time consuming cuts in the density estimation in high dimension space, this section presents our work on using copula transformation [24], [35], [36] as a method to map a D-dimensional density estimation problem into the

product of D one-dimensional marginal densities and a copula density. Each marginal density is estimated separately, and the results, along with density estimate of the copula, are used to estimate the joint density of the original dataset. The advantage of presenting the density in copula-transformed domain is that data has uniform marginal distributions in the interval [0, 1] [24], which leads to a significant reduction in the number of cuts in BSP in high dimensions and much better KLD [1].

For a random vector $\boldsymbol{X} = (X_1, \cdots, X_D)$, BSP is applied to each of the dimensions, separately, to estimate the marginal PDFs $f_1(x_1), \cdots, f_D(x_D)$. The estimated PDFs are then used to build the marginal cumulative distribution functions (CDFs), $F_1(x_1), \cdots, F_D(x_D) \in [0, 1]$, where $F_d(x_d) = P(X_d \leq x_d)$. The resulting random variables $F_1(X_1), \cdots, F_D(X_D)$ form the new multivariate dataset. As a result, the joint CDF of the sample dataset $F_{\boldsymbol{X}}(x)$ can be expressed as a standard copula C [24] as,

$$F_{\boldsymbol{X}}(x) = C(F_1(x_1), \cdots, F_D(x_D)) \tag{8}$$

In copula domain, instead of using N samples of $\boldsymbol{X} = (X_1, \cdots, X_D)$ to perform the BSP, we use N samples of the generated marginal CDFs as a copula-transformed dataset $(F_1(X_1), \cdots, F_D(X_1))$ (an $N \times D$ dataset). The method of BSP is then applied to this new D-dimensional dataset, to estimate the joint PDF $c(F_1(x_1), \cdots, F_D(x_D))$, where $c(u) = \frac{\partial^D}{\partial u_1 \ldots \partial u_D} C(u)$. The PDF for the original dataset, $f(x)$, can then be calculated as [24],

$$f(x) = c(F_1(x_1), \cdots, F_D(x_D)) \prod_{d=1}^{D} f_d(x_d) \tag{9}$$

Our simulations show that in high dimensions, use of copula transform reduces the total number of cuts. More importantly, it reduces the number of computationally complex cuts required by the BSP algorithm in high dimensional space by as much as 98%, and substitutes them by computationally cheaper cuts in the marginal distributions. Table 2 presents the number of cuts, the execution times as well as KLD and HLGR values of the direct and copula-transformed BSP for various dimensions, for a multivariate normal distribution. The synthetic dataset used in these simulations is a bivariate trimodal normal distribution for the first two dimensions, as shown in Figure 3. In 32 and 64-dimensional datasets, the third dimension is a unimodal normal distribution, and for dimensions four and higher, a bimodal normal distribution. This choice of datasets ensures adequate level of diversity in the marginal distributions. Significant from the data in Table 2 is the vast disparity between the KLD values for the direct and copula techniques for the high dimensional datasets.

Fig. 5 to Fig. 8 illustrate BSP through the application of copula transform for the same $N = 20,000$ data samples generated by (7). Fig. 5 presents the estimated marginal PDFs and CDFs from BSP. Discontinuities in the PDF plots correspond to cuts from BSP process. Fig. 6 presents two random variables $F_1(X_1)$ and $F_2(X_2)$ in the transformed domain, which

TABLE 2. BSP with direct and copula-transformed cuts on D-dimensional space for various dimensions, for $N = 20,000$, $M = 200$. Time unit is seconds. The number of cuts are shown as the sum of total marginal and copula cuts, as well as a pair in the parenthesis corresponding to each component.

	$D = 2$		$D = 32$		$D = 64$	
	Direct	**Copula**	**Direct**	**Copula**	**Direct**	**Copula**
Cuts	182	183 (96+87)	2938	1694 (1614+80)	3648	3338 (3266+72)
Time	13	19	2200	278	6018	552
KLD	0.045	0.018	33.5	0.19	72.5	0.31
HLGR	0.013	0.006	1.0	0.044	1.0	0.093

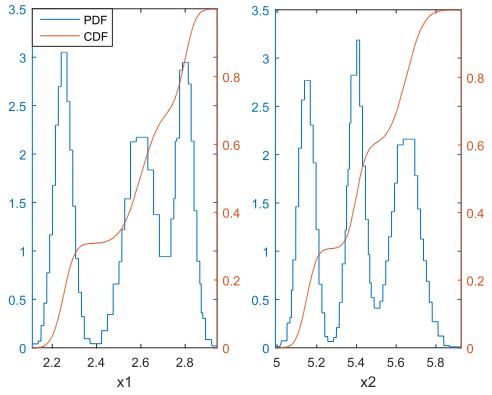


FIGURE 5. Estimated marginal PDF and CDF, using copula transform, for $N = 20,000$ and $M = 200$. The number of BSP cuts for the two marginals X_1 and X_2 are 49 and 47, respectively.

are in fact marginal CDFs of X_1 and X_2. Fig. 7 shows the copula-transformed sample space and the cuts made by BSP process. For $N = 20,000$ and $M = 200$ the BSP has made a total of 183 cuts with 49 and 47 cuts for the two marginals X_1 and X_2, respectively. The number of cuts for estimating the copula-transformed density in the 2-dimensional space is 87, a significant reduction from 182 cuts in the direct method. However, as expected, and seen in Table 2, for a low dimensional problem of $d = 2$ the overhead associated with the copula transformation far outweighs the reduction in the number of cuts in 2-dimensional space, which results in a higher execution time. Further, notice the difference between the cuts in Fig. 4 with a higher number of cuts in the high density areas, and Fig. 8, where most cuts are made away from high density areas. As we will see later, the cuts in the high density areas involve much higher computational complexity. The estimated joint density from the BSP algorithm is shown in Fig. 8. The KLD between the true density in Fig. 4 and the estimated density in Fig. 8 is only 0.018 [1].

A. COPULA AND SAMPLE PARTITION DIVERSITY

One important effect of copula transformation is that for each of the marginals, cuts are made in one-dimensional space, meaning that at each level j, there are only $j - 1$ possible ways to make the next cut; alleviating the need for selecting a typically large value of M to improve density estimation through increased path diversity. Simulation results for the 2-dimensional example in Fig. 5 for marginals and Fig. 7 for

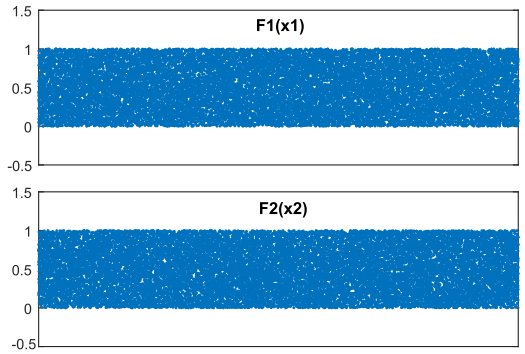


FIGURE 6. Distribution of the transformed random variables $F_1(X_1)$ and $F_2(X_2)$, with $N = 20,000$.

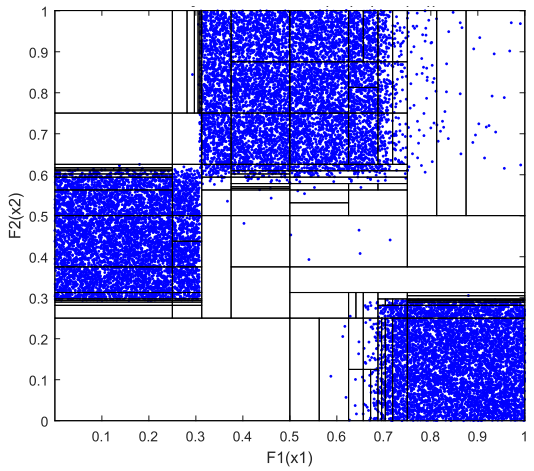


FIGURE 7. BSP cuts on copula-transformed sample space with $N = 20,000$ and $M = 200$. The number of BSP cuts is 87.

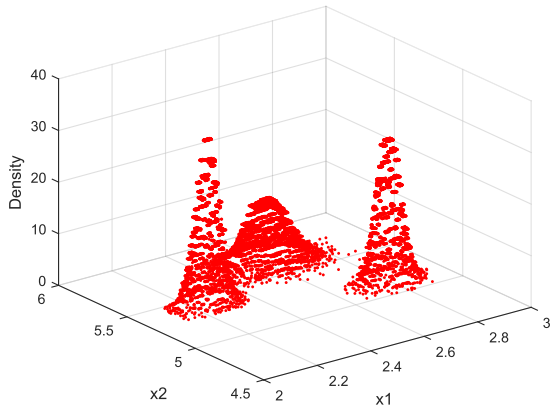


FIGURE 8. Estimated joint density, using copula transform with $N = 20,000$ and $M = 200$.

copula-transformed partitions are shown in Fig. 9 (a) and (b), respectively. As seen from Fig. 9 (a) after a relatively small number of cuts, the partition scores for all $M = 200$ sample partitions merge towards a single value, indicating that all M independent paths eventually lead to the same or similar partitions. Plots in Fig. 9 (a) also show that for two choices

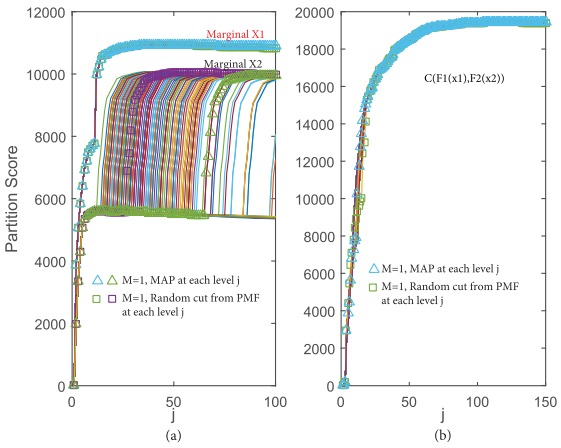


FIGURE 9. Partition scores with $N = 20,000$ and $M = 200$ sample partitions for examples for (a) marginals in Fig. 5, (b) copula in Fig. 7. In addition to 200 plots for all $M = 200$ sample partitions, the figures also show the results for two choices of $M = 1$, (Δ for MAP, and $\square$ for random cut from PMF) at each level j.

of $M = 1$, (Δ for MAP, and $\square$ for random cut from PMF), the scores are very close to, (and for most cuts even better than), the best score obtained with $M = 200$. Thus, for all marginals, we can apply the BSP with only one sample partition, $M = 1$ with for all j cuts [1].

For the case of BSP in copula-transformed multidimensional space, in Fig. 9 (b) despite the fact that most of the sample partitions tend to converge at relatively small number of cuts j, the algorithm needs to process a large number of sample partitions to maintain a good diversity. The scores for two cases of $M = 1$ are very close to the best score obtained with $M = 200$, but generally less than the best score obtained with $M = 200$.

To further investigate the effect of M on the density estimation, we performed a set of simulations with dimensions and distributions identical to Table 2, with two different dataset sizes. Simulation results are shown in Table 3, for a range of dimensions, from 2 to 256.

In the basic setup (*Option 1*) M is 200 for all marginals and the copula-transformed density estimations. A variation of the basic set up is *Option 2*, where for all marginals we only maintain one sample partition from beginning to the end of BSP. We set $M = 1$ with MAP to choose the location of the cut for every value of j. Finally, in *Option 3*, we set $M = 1$ for all marginals as well as the copula-transformed estimation. For the marginals we employ the MAP technique to pick up the best cut. For copula-transformed estimation, the location of the next subregion cut is decided based on a random draw from the generated PMF. However, similar to the marginals, it is also possible to employ the MAP technique for the copula-transformed estimation.

As can be seen in Table 3, for small dimension of $D = 2$ the variations in the accuracy (KLD) is small across the three options, even when the dataset is relatively small. Therefore, we can set $M = 1$ for both marginals and the copula to gain one to two orders of magnitude reduction in the

TABLE 3. BSP with copula-transformed cuts on D-dimensional space, for three different options: *Option 1*: $M = 200$ for marginals, $M = 200$ for copula, *Option 2*: $M = 1$ with MAP for marginals, $M = 200$ for copula, *Option 3*: $M = 1$ with MAP for marginals, $M = 1$ for copula; and two datasets with $N = 20,000$ and $N = 100,000$. The values reported in this table are mean values obtained from multiple runs of the code: 32 runs for $D = 2$, 16 runs for $D = 32$, 8 runs for $D = 64$, and 4 runs for higher dimensions. The number of cuts are shown as the sum of marginal and copula cuts, with individual components presented as the pair in the parenthesis. The standard deviation corresponding to each KLD value is reported below it, in parenthesis.

		$D=2$			$D=32$			$D=64$			$D=128$			$D=256$		
		Option 1	**Option 2**	**Option 3**	**Option 1**	**Option 2**	**Option 3**	**Option 1**	**Option 2**	**Option 3**	**Option 1**	**Option 2**	**Option 3**	**Option 1**	**Option 2**	**Option 3**
N=20k	Cuts	183 (96+87)	185 (79+106)	**173** (79+94)	1694 (1614+80)	1678 (1543+135)	**1625** (1517+108)	3338 (3266+72)	**3262** (3152+110)	3279 (3147+132)	6666 (6569+97)	**6474** (6249+225)	6485 (6302+183)	13314 (13237+77)	**12914** (12649+265)	12939 (12693+246)
	Time	19	13	**0.1**	278	58	**1**	552	87	**2.6**	842	197	**10**	1680	427	**21**
	KLD	**0.018**	0.024	0.033	**0.19**	0.93	1.33	**0.31**	1.62	2.02	**1.51**	4.22	3.90	**1.59**	6.84	6.59
	(std)	(0.0034)	(0.0116)	(0.0202)	(0.0727)	(0.4629)	(0.5576)	(0.0102)	(0.4665)	(0.7129)	(0.4599)	(0.7237)	(0.7129)	(0.3445)	(1.0132)	(0.7772)
N=100k	Cuts	293 (156+137)	294 (152+142)	**284** (150+134)	2674 (2554+120)	2654 (2526+128)	**2598** (2531+67)	5277 (5155+122)	5244 (5130+114)	**5153** (5115+38)	10462 (10345+117)	10400 (10277+123)	**10328** (10277+51)	20837 (20734+103)	20723 (20618+105)	**20621** (20618+3)
	Time	111	44	**0.5**	1262	110	**5**	2475	169	10	4254	293	**50**	8857	485	**99**
	KLD	**0.007**	0.008	0.01	**0.068**	0.077	0.75	**0.13**	0.14	0.88	**0.31**	0.39	1.10	**0.52**	0.57	1.56
	(std)	(0.0014)	(0.0018)	(0.00508)	(0.004)	(0.0049)	(0.4119)	(0.0044)	(0.0043)	(0.4499)	(0.1079)	(0.1224)	(0.4221)	(0.0068)	(0.0084)	(0.4499)

computational complexity. For larger dimensions (32 and 64), with $N = 20,000$, the degradation in KLD is significant and use of options 2 and 3 become less attractive. However, for large dataset of $N = 100,000$, with no appreciable increase in KLD, an order of magnitude reduction in computation complexity can be obtained by setting $M = 1$ for the marginals. A further one order magnitude reduction in computation time can be achieved, by setting $M = 1$ for both marginals and copula BSP, if an increase in KLD to more than 1.0 can be accepted.

The trend remains the same as we increase the dimensions to 128 and 256. It can be seen that even for high-dimensional cases, the BSP method (with *Option 1* or *2*) is still able to provide good results, as long as the sample size is increased accordingly. In the following subsections, we will present an extended version of the simulation results for a wider range of values of N, for the case of $D = 64$, to show how the estimation error is decreased by increasing the sample size.

As far as the number of cuts are concerned, we have investigated the marginal and copula cuts separately. First considering the marginal cuts, with *Option 1*, for each marginal 200 different sample partitions (with potentially different cut patterns) are generated in parallel. However, with *Option 2*, for each dimension only one sample partition (the one with the highest *score*) is retained. This means that with *Option 1* there is a good chance that some of those sample partitions proceed to a higher number of cuts (compared to only one sample partition in *Option 2*). Thus, as can be seen in Table 3, the number of marginal cuts in *Option 1* is higher than *Option 2*. Although the difference is not large.

The quality of the partitions in the marginal part influences the number of cuts in the copula part. Both *Options* 1 and 2 in Table 3 have same number of copula sample partitions ($M = 200$). However, *Option 2* has (slightly) higher number of copula cuts, than *Option 1*, due to the fact that its marginals were produced with only one sample partition, which may not have been the best possible partition for the marginal. This results in the situation where for the copula (joint density estimation) part, the partitioning algorithm will likely have to create more number of cuts before it converges. Regarding computation time, it is observed that the time increases almost linearly with increasing dimensions. We will further discuss the computational complexity of the algorithm in Sections V and VI.

B. MARGINALS DENSITY ESTIMATION WITH THE KDE METHOD

With the separation of high dimensional density estimation into one-dimensional and copula parts we are able to perform the density estimation for the marginals using the KDE method in (2). However, unlike the method of BSP, the choice the bandwidth h and kernel function in (2) becomes critical in the accurate estimation of the density. We use the Gaussian kernel function and rule of thumb bandwidth estimator of $h = 1.06\sigma n^{-\frac{1}{5}}$ [13] in this work. Table 4 compares the density estimation results, using the BSP and KDE for the same 64-dimensional dataset used before. Three observations can be made from the data in the table. First, for the BSP, the KLD improves by a factor of up to 18.20 (Option 2) and the execution time increases by up to 22 (Option 1) when the size of dataset increases by a factor of 20, from $20k$ to $400k$. For the KDE, the improvement in KLD is no more than 1.61 (nGrid = 1000), while the increase in the execution time goes by a factor of up to 64 (nGrid = 250). The second observation is that the BSP is a more flexible technique. The execution times and the KLD values for the BSP change by up to two orders magnitude for the three BSP options. This is not the case, however, for the KDE where the changes in both execution times and the KLD values are much less significant. The third observation is the fact that for small data size of $N = 20k$, similar KLD and execution times can be obtained for the BSP and KDE techniques. However, KDE loses to BSP in the execution time by two orders magnitude for a similar KLD performance. For example, BSP Option 1 is 136 times faster than KDE with nGrid of 500 for the similar KLD values [1].

To further evaluate the relative performance of BSP and KDE, we experimented with a 64-dimensional dataset with a skewed distribution. We maintained the same distribution for the first three marginals. Dimensions 4 to 64, however, come from the following mixture of Beta distribution [34]:

$$X \sim 0.6 \times B(2, 8) + 0.4 \times B(120, 14)$$

It is a bimodal distribution, with the $B(2, 8)$ having rather a wide PDF with a positive skew, and the $B(120, 14)$ mode

TABLE 4. Comparison of density estimations using the methods of BSP, KDE and median-based cuts for the marginals ($D = 64$). For KDE, $M = 200$ has been set for the copula part. Results are average of 10 runs. (5 runs for $N = 400k$ cases.)

		BSP			Median			KDE		
		Option 1	**Option 2**	**Option 3**	**Option 1**	**Option 2**	**Option 3**	**nGrid=250**	**nGrid=500**	**nGrid=1000**
N=20k	Cuts	3338	**3262**	3279	3194	**2196**	3021	**16086**	32094	64124
		(3266+72)	(3152+110)	(3147+132)	(3085+109)	(2076+120)	(3019+2)	(16000+86)	(32000+94)	(64000+124)
	Time	397	43	**5**	361	75	**5**	86	104	**74**
	KLD	**0.37**	0.42	1.24	**0.39**	1.74	1.45	0.99	0.72	**0.61**
	(std)	(0.0092)	(0.0056)	(0.33)	(0.018)	(0.073)	(0.099)	(0.68)	(0.54)	(0.32)
N=100k	Cuts	5277	5244	**5153**	5065	**4268**	4895	**16165**	32172	64150
		(5155+122)	(5130+114)	(5115+38)	(4887+178)	(4088+180)	(4831+64)	(16000+165)	(32000+172)	(64000+150)
	Time	1770	147	**25**	1621	204	**24**	645	660	**500**
	KLD	**0.15**	0.18	1.01	**0.15**	0.22	1.02	0.49	**0.38**	0.45
	(std)	(0.0058)	(0.0043)	(0.45)	(0.01)	(0.042)	(0.20)	(0.46)	(0.44)	(0.32)
N=200k	Cuts	6606	6496	**6356**	6151	6807	**5909**	**16209**	32198	64189
		(6447+159)	(6346+150)	(6281+75)	(5934+217)	(5822+5)	(5863+46)	(16000+209)	(32000+198)	(64000+189)
	Time	3846	281	**21**	3311	371	**48**	1805	**1428**	1469
	KLD	**0.10**	0.12	1.03	**0.098**	0.10	1.07	0.71	0.42	**0.35**
	(std)	(0.0047)	(0.0031)	(0.16)	(0.0024)	(0.0027)	(0.15)	(0.80)	(0.47)	(0.37)
N=400k	Cuts	7920	7881	**7756**	7610	7415	**7169**	**16246**	32255	64279
		(7736+184)	(7683+198)	(7683+73)	(7272+263)	(7146+269)	(7024+45)	(16000+246)	(32000+255)	(64000+279)
	Time	8495	531	**42**	7157	671	**98**	5582	5727	**4719**
	KLD	**0.067**	0.083	1.02	**0.065**	0.067	1.04	0.87	0.66	**0.63**
	(std)	(0.0055)	(0.0011)	(0.11)	(0.002)	(0.0017)	(0.014)	(0.63)	(0.12)	(0.19)

having a much sharper and almost symmetric PDF. Table 5 presents the results. Comparing the results with those in Table 4, the execution times have gone up for the KDE case due to higher number of cuts in copula distribution. For BSP, being a more adaptable method, the changes in the number of cuts have been by much smaller margins. However, while the KLD values for the BSP have remained unchanged, they have deteriorated for the KDE by an order of magnitude [1].

C. MARGINALS DENSITY ESTIMATION WITH MEDIAN-BASED CUTS

In an attempt to make the sequential cuts a better fit to the data density pattern, we have tried the idea of replacing the previously described mid-point binary cuts with *median-based* cuts. As explained in Section II-B, in the original BSP method, sequential cuts are made in the sample space using the mid-point BP scheme, i.e., at each level, one of the existing subregions is cut into two equal halves. So, the cut location is always at the center point of the selected subregion, regardless of distribution of data in that subregion. In median-based approach, on the other hand, the location of the cut is at the median point of the data samples in that subregion. The decision on which subregion to cut at level j is made using a conditional probability, similar to Eq. 5, with the difference that with median-based cuts, the volumes of the subregions, v_p, $v_{pd}^{(1)}$ and $v_{pd}^{(2)}$ also appear in the equation [1]:

$$s_{j_{pd}}(cut_{j_{pd}}|g_{j-1}) = C_{j-1}\Big(\frac{\Gamma(n_{pd}^{(1)})\Gamma(n_{pd}^{(2)})}{\Gamma(n_p)}\Big)\Big(\frac{v_p}{v_{pd}^{(1)}v_{pd}^{(2)}}\Big)^{n_p} \quad (10)$$

Also, the partition score calculation will be slightly different from Eq. 6.

We have tried this idea with the 64-dimensional data described before. As the results presented in Tables 4 and 5 show, for all 3 options, and for all different values of N covered in this simulation, using the median-based method for estimating the marginals reduces the number of marginal cuts, compared to the original method of mid-point binary cuts. This is because firstly, the median-based approach tries to use the information about the distribution of data in the subregions to decide which subregion to cut; and secondly, once the subregion to cut is picked, it makes the cut in a more data-adaptive fashion. For *Option 1*, where the marginal cuts are the dominant part of the computation time, using the median-based cuts method for marginals leads into saving the overall computation time. Regarding estimation accuracy, both methods have similar performance in almost all cases presented in the tables. While the number of cuts for the method of the median is always smaller than the mid-point method, and both methods exhibit similar measures of accuracy, the computational complexity, except for *Option 1*, is not better. That is because searching for the median points incurs significant computational overhead [1]. Therefore, for the rest of this paper, we continue to use the mid-point binary cuts method for further discussions, simulations and analyses.

We have also tried the idea of combining diffusion-based KDE for marginals and copula transform approach for the joint density in [37].

IV. ALGORITHM AND DATA STRUCTURES FOR THE BAYESIAN SEQUENTIAL PARTITIONING (BSP)

A. ALGORITHM

When dealing with large, high-dimensional datasets, BSP method of density estimation results in high computational complexity. Fig. 10 illustrates our efficient implementation of the BSP through repeated evaluation of (5) and (6) in a loop. Similarly, the flowchart in Fig. 11 illustrates the process of BSP using copula transformation through evaluation of (8)

TABLE 5. Comparison of density estimations using the methods of BSP, KDE and median-based cuts, for the marginals for a bimodal skewed Beta distribution ($D = 64$). For KDE, $M = 200$ has been set for the copula part. Dimensions 4 to 64 each have a bimodal Beta distribution.

		BSP			Median			KDE		
		Option 1	**Option 2**	**Option 3**	**Option 1**	**Option 2**	**Option 3**	**nGrid=250**	**nGrid=500**	**nGrid=1000**
N=20k	Cuts	2825 (2735+90)	2804 (2729+75)	**2731** (2729+2)	2725 (2616+109)	2316 (2196+120)	**2087** (2081+6)	**16114** (16000+114)	32096 (32000+96)	64142 (64000+142)
	Time	357	40	**5**	312	75	**5**	**32**	127	172
	KLD	**0.35**	0.36	1.20	**0.65**	1.74	2.84	7.7	**7.6**	7.6
	(std)	(0.0043)	(0.0073)	(0.30)	(0.08)	(0.073)	(0.084)	(0.61)	(0.29)	(0.24)
N=100k	Cuts	4526 (4405+121)	4520 (4397+123)	**4399** (4397+2)	4414 (4236+178)	4448 (4268+180)	**4102** (4095+7)	**16229** (16000+229)	32201 (32000+201)	64223 (64000+223)
	Time	1641	149	**25**	1531	204	**23**	**722**	741	805
	KLD	**0.13**	0.14	1.10	**0.14**	0.22	1.22	4.9	**4.8**	4.8
	(std)	(0.0023)	(0.0021)	(0.15)	(0.008)	(0.042)	(0.11)	(0.74)	(0.38)	(0.27)
N=200k	Cuts	5633 (5490+143)	5597 (5449+148)	**5461** (5450+11)	5348 (5138+210)	5209 (4988+221)	**5007** (4988+19)	**16230** (16000+230)	32221 (32000+221)	64229 (64000+229)
	Time	3460	280	**51**	3070	364	**47**	**1478**	1499	1577
	KLD	**0.0867**	0.092	1.06	**0.091**	0.11	1.13	3.78	**3.73**	3.78
	(std)	(0.0036)	(0.0013)	(0.039)	(0.0018)	(0.009)	(0.11)	(0.67)	(0.60)	(0.51)
N=400k	Cuts	7349 (7169+180)	6990 (6802+188)	**6805** (6802+3)	6522 (6263+259)	6331 (6073+258)	**6150** (6072+78)	**16274** (16000+274)	32283 (32000+283)	64261 (64000+261)
	Time	7764	547	**100**	6625	661	**97**	**4716**	4746	4848
	KLD	**0.054**	0.060	1.04	**0.067**	0.078	0.91	3.05	**2.72**	2.94
	(std)	(0.0044)	(0.0008)	(0.0009)	(0.0064)	(0.005)	(0.22)	(1.00)	(0.80)	(0.39)

and (9). In the flowchart for BSP, parameter j represents the partitioning *level*, similar to the notation used in Fig. 1. The outer loop in this flowchart is over j, starting with $j = 1$ (*i.e.,* the original non-partitioned space), and continues until the algorithm converges, or it hits j_{max} without converging. Note that j_{max} is only used to impose an upper limit on the running time of the algorithm, beyond which it can be assumed that the algorithm does not converge. The value of j_{max} is set arbitrarily as to not affect the accuracy of the algorithm. This value is typically in the order of 2-3 times larger than the rough estimate for the required number of cuts.

B. SUBREGION EVALUATION REDUCTION

To improve the speed of the algorithm in the flowchart of Fig. 10, we make an important observation. With reference to (5) at each level j, there is no need to calculate $s_{j_{pd}}$ for all possible pd cuts. This is because the $s_{j_{pd}}$ values corresponding to all cuts, except the $2D$ potential cuts in the two recently modified subregions, are already calculated and stored in `sjLog` structure in the previous level, $j - 1$. This reduces the number of $s_{j_{pd}}$ evaluations from $(j - 1)D$ to only $2D$; which is a huge reduction in computation time, when level j becomes large, in high-dimensional problems. This simplification eliminates the loop over p in the flowchart of Fig. 10. Fig. 12 illustrates this computational simplification in 2D sample space. At each level $j > 1$ there are only two possible cuts (marked in blue and red dashed lines) for each subregion, with a total of $2(j - 1)$ possible cuts. The red dashed line identifies the location earmarked for the actual cut at level j. The resulting two subregions separated by a solid black line, due to a cut at level j are marked in beige. At each level $j \geq 3$, we only need to calculate four new values of $s_{j_{pd}}$ that belong to these two new subregions. All remaining possible $s_{j_{pd}}$ values for cuts in subregions marked white are carried over from level $j - 1$. In Fig. 12, the left column shows all the possible cuts, with red lines marking the actual cut. This leads to the new partition shown in the right column. For example, at level $j = 5$, there are $(5 - 1) \times 2 = 8$ possible ways to make the next cut. But the probabilities associated with the possible cuts in subregions 1 and 3 (cut numbers 1, 2,5, and 6) are already available from the previous level $j = 4$. So we only need to evaluate the $s_{j_{pd}}$ values for cut numbers 3, 4, 7, and 8 [1].

C. DATA STRUCTURES

Main data structures for implementing the described BSP algorithm are listed in Table 6. The data structures are designed for efficient memory access and minimum data movement. They are also designed for the ease of mapping into an efficient parallel processing paradigms of OPENMP® and MPI [1]. All data structures are organized for optimal column-major memory access, the method of choice in MATLAB®. Some of the data structures listed in Table 6 are for the purpose of estimation of one-dimensional marginal densities, while some others are exclusively designed for copula density estimation, and some structures are common for both estimations.

At each level j, subsequent to a cut in one of the existing subregions, the data points in each of the two new subregions need to be identified and their total number counted and stored in the structure `np`. This is the most time consuming part of BSP algorithm for large and high-dimensional datasets. To reduce the computational complexity associated with this operation, we augmented the $N \times D$ input dataset structure, `data`, with an additional column p to store the corresponding subregion number for each of the data points.

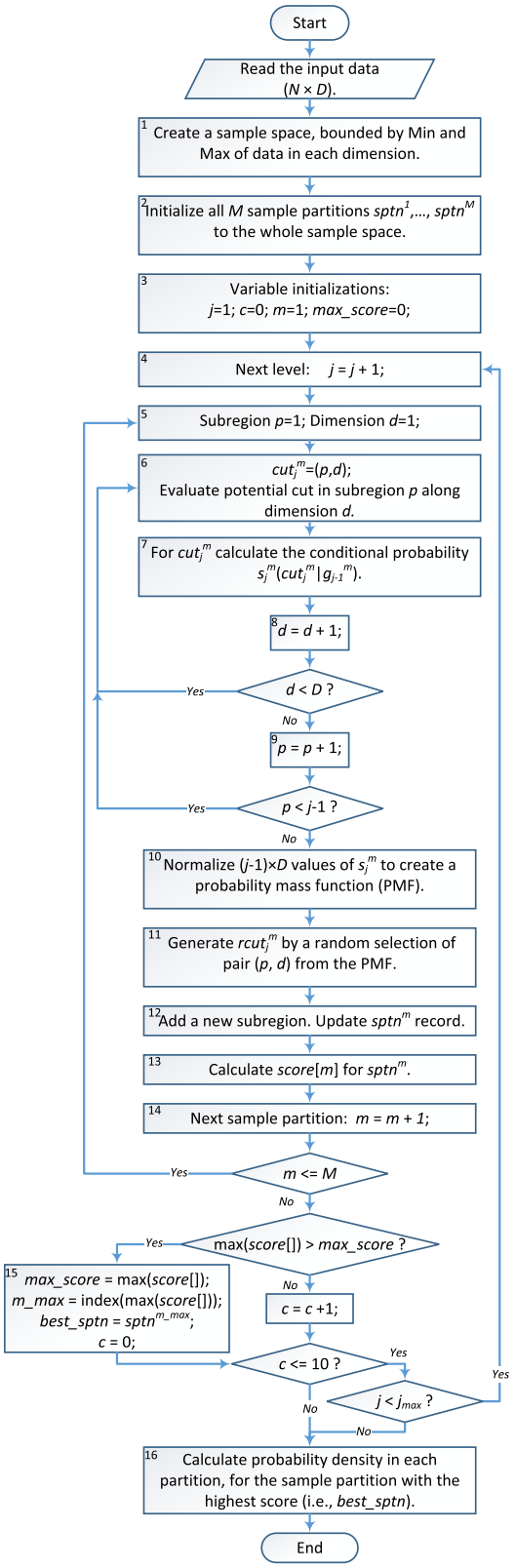


FIGURE 10. Flowchart for Bayesian sequential partitioning.

Then the whole structure is rearranged (partially sorted) such that all the points with the same subregion identity are stored in a contiguous block. The implemented scheme is

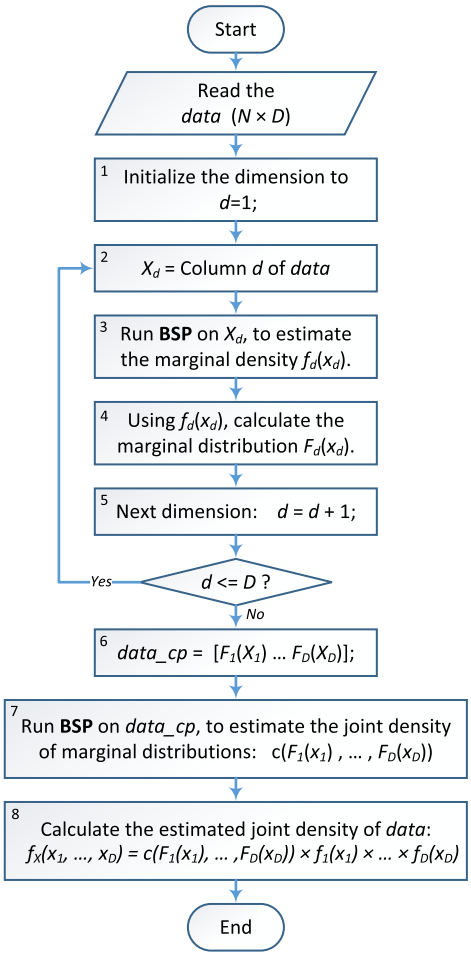


FIGURE 11. Flowchart for density estimation using copula transform.

shown in Fig. 13. A pointer structure, `dataDistLimits`, stores the indices for the first (marked green) and the last element (marked red) in each subregion. The efficiency of the proposed structure comes from the fact that when at level j, subregion t is cut, we only need to sort the subset of the `data` structure marked as t. This partitioning scheme of dataset structure, `data` works well for small dimensions less than 5. However, for larger dimensions, even the limited subregion sorting process requires movement of a large amount of multi-dimensional data. To reduce the data movement for dimensions larger than five, we adopted the structure shown in Fig. 14 (a) where an intermediate data structure `dataDist` stores the indices to `data`. This way, the subregion sorting process involves a much simpler 2-column structure, instead of a D-column structure. In the optimized implementation of BSP using copula transform, for each of the marginals, a simpler structure of in Fig. 14 (b) is used. Before processing each marginal d its corresponding column is copied from `data` to `dataMarginal`. After processing, `dataMarginal` will contain the CDF for that marginal, which is copied back to the corresponding column of `data` for copula processing, using the structure in Fig. 14 (a).

TABLE 6. Main data structures for high-dimensional density estimation using BSP algorithm.

Name	Dimensions	Description
`data`	$N \times D$	Contains the whole input dataset or copula-transformed data, i.e. N rows of D-dimensional data.
`dataMarginal`	$N \times 3 \times M$	For each of the M sample partitions, `dataMarginal` stores distribution of the marginal data for dimension d in partition subregions. The first column contains an index to the original dataset stored in `data`. The second column contains a copy of column d of `data`. The third column stores the corresponding subregion numbers for the data from column 2. After processing of each marginal d, the second column of `dataMarginal` will contain its marginal CDF, and is copied back to the corresponding column of `data` for copula processing.
`dataDist`	$N \times 2 \times M$	For each of the M sample partitions, `dataDist` stores distribution of the data in the partition subregions. The first column contains an index to the original dataset stored in `data`, and the second column stores the subregion numbers corresponding to the indices from column 1.
`dataDistLimits`	$2 \times j_{max} \times M$	Maintains pointers to the start and end entries of each subregion in `dataDist`.
`subCoords`	$2D \times j_{max} \times M$	Stores coordinates of the subregions. Each subregion is specified by its lower and higher boundaries in each dimension.
`sjLog`	$D \times j_{max} \times M$	Stores the calculated probabilities for cutting each of the existing subregions along any of dimensions, at each level j. All values are in *log* domain.
`sj`	$D \times j_{max}$	Stores the *log* normalized probabilities from `sjLog`.
`scores`	$M \times 1$	Stores the scores for sample partitions.
`nSub`	$j_{max} \times M$	Stores the number of data points in each of the subregions for each of M sample partitions.
`nSubMAP`	$j_{max} \times 1$	Records the state of one column of `nSub` with the highest partition score, when the BSP meets the stopping criteria.
`vSub`	$j_{max} \times M$	Stores the volume of the subregions for each of the M sample partitions.
`vSubMAP`	$j_{max} \times 1$	Records the state of one column of `vSub` with the highest partition score, when the BSP meets the stopping criteria.
`bestDistMAP`	$N \times 2$	Stores a subset of `dataDist` structure for the sample partition with the highest partition score.
`rCuts`	$M \times 1$	Stores the subregion number that is randomly cut at each level j.
`CoordSubMarMAPs`	$2 \times j_{max} \times D$	Each columns of this structure corresponds to one of the marginal densities and stores the final coordinates of the subregions for the marginal sample partition with the highest partition score.
`densSubMarMAPs`	$j_{max} \times D$	Each columns of this structure corresponds to one of the marginal densities and stores the final estimated subregion marginal densities.
`CoordCopulaMAP`	$2D \times j_{max}$	Stores the final coordinates of the subregions, taken from the partition with the highest partition score.
`densCopulaMAP`	$j_{max} \times 1$	Stores the final values of the joint probability densities, for the copula-transformed density estimation.

V. DENSITY ESTIMATION SIMULATION RESULTS

This section presents the results obtained from MATLAB® simulation runs [1] on a single Intel® Core i7-3820, 3.60 GHz, with 32 GB RAM, for the performance evaluations of the algorithm.

Density estimation is performed for the range of sample data sizes from $N = 10,000$ up to $N = 1,000,000$, and range of dimensions, from $D = 2$ up to $D = 64$. The simulations aim at examining the impact of different parameters on computational efficiency, measured in execution time, and estimation accuracy measured as KLD. Fig. 15 presents the execution time and KLD for a 64-dimensional dataset versus the sample sizes N, with the number of sample partitions M as a parameter. Clearly, a larger sample dataset improves accuracy, at the cost of increased execution time. It can be seen that the execution time grows almost linearly with the sample size N. The KLD between the actual density and the estimated density, on the other hand, decreases exponentially with sample size. Increasing the sample size over $N = 300,000$ has little impact in improving the estimation accuracy. While the execution time increases linearly with M, the KLD exhibits small sensitivity to M. It is also seen that the choice of $M = 1$ for the marginals and $M = 200$ for the copula transformation results in no appreciable increase in KLD, with a six-fold reduction in computational complexity, when compared with the next closest case of $M = 50$ for both marginals and copula. A further 12-fold improvement in the execution time can be seen for the choice of $M = 1$ for both marginals and the copula transformation. However, KLD on average remains higher than the case of $M = 1$ for the marginals and $M = 200$ for the copula transformation, by about a little more than 1.0. It also can be seen that for this case, due to the lack of diversity in the selection of the sample partitions, the KLD is relatively unstable across the range of N values.

We have performed all our simulations on synthetic data to show the effectiveness of the algorithm in density estimation for multivariate data. For real data, where the true density values are not available, the estimation accuracy can not be evaluated using the KLD analysis with the true density values. As the results in the Tables 3 to 5 and in Figure 15 show, the KLD value converges after a sufficiently large number of data points are processed. We have used this fact to find a way to have an estimate of the density estimation accuracy for real data, in [38] on online density estimation. The estimated densities obtained from a sufficiently large dataset are used as a close approximation of the true densities, to evaluate the density estimation accuracy for various sample sizes.

VI. COMPLEXITY ANALYSIS AND PARALLELIZATION

A. COMPLEXITY ANALYSIS

The flowcharts in Figures 10 and 11 and the results presented in Figure 15, suggest that the complexity changes almost linearly with number of samples N, the number of sample

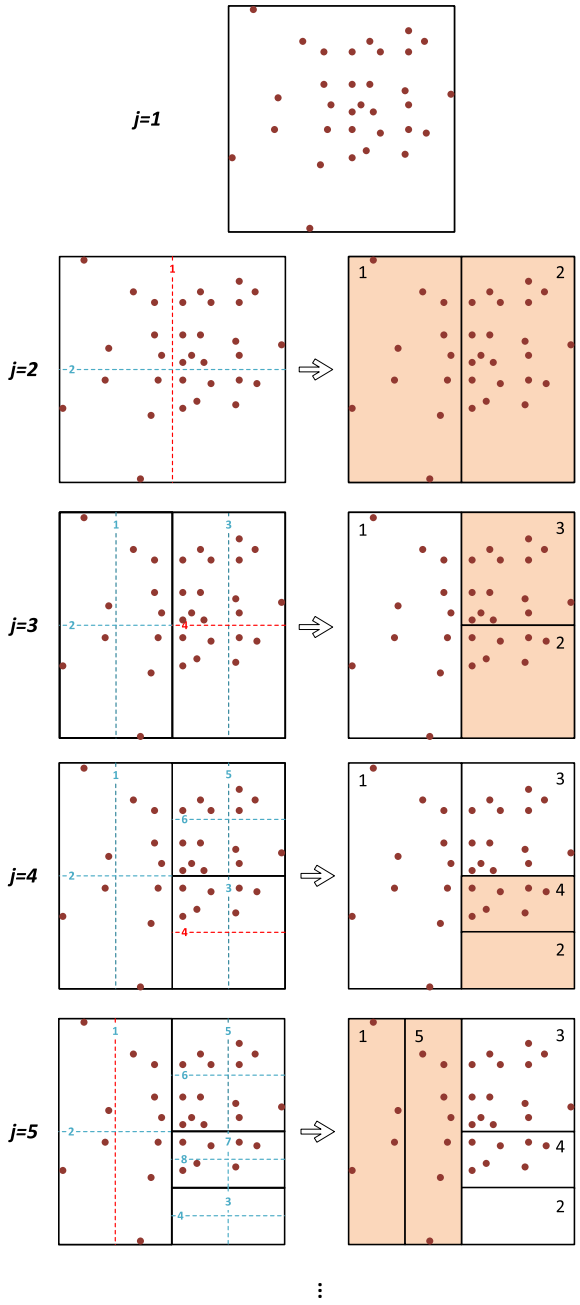


FIGURE 12. Calculating four new cut probabilities at each level, in 2D space.

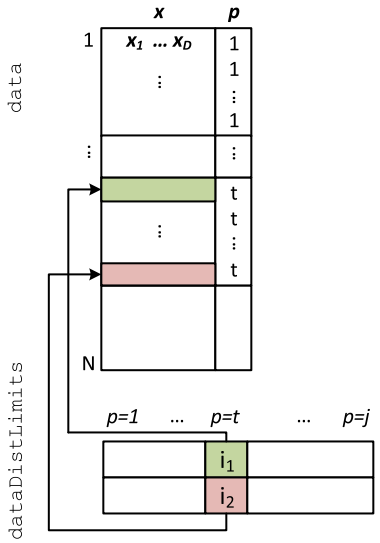


FIGURE 13. Data structure extended, to store distribution of the data points in subregions.

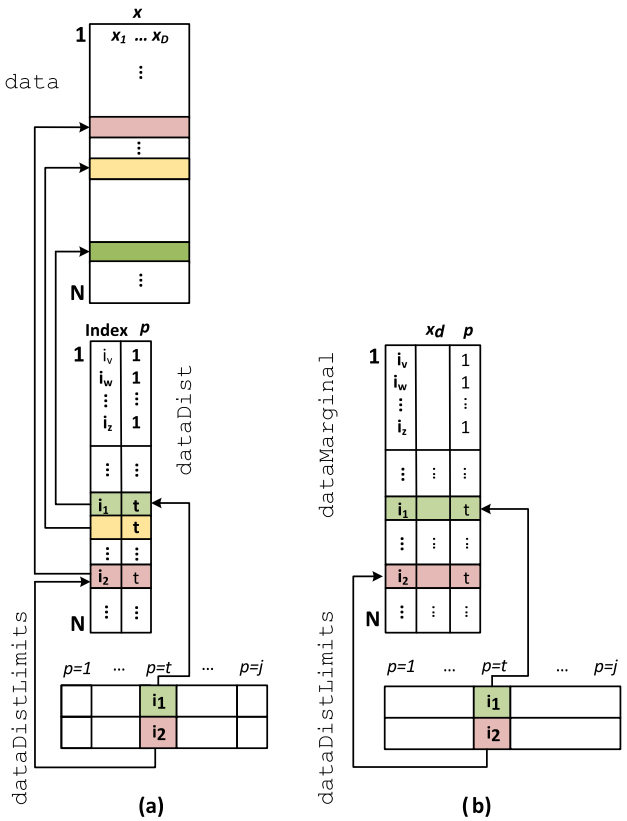


FIGURE 14. Optimized data structure for storing distribution of data points in subregions for (a) copula where intermediate data structure `dataDist` stores the indices to `data`, and (b) for each of the marginals, where the intermediate step is not required.

partitions M, and sample dimension D. However a closer examination of different steps of the flowcharts reveals that the computational complexity of the algorithms is actually $O(MDNlog(N))$. This can be justified based on the 3 loops over N, M and D, in Figure 10. The outer loop (steps 5 to 14) is over M sample partitions ($O(M)$ complexity). The other loop consisting of steps 6 and 7 goes through all D dimensions ($O(D)$ complexity). For complexity with respect to N, we need to look at steps 6-14. In step 6, evaluation of potential cuts requires a search over sample points to get the count of points in each side of the potential cut ($Nlog(N)$ complexity). In step 7, calculating the conditional probability of each potential cut using Gamma function has a complexity of N. Finally, steps 10-14 correspond to a constant computation time ($O(1)$ complexity). As a result, the overall algorithm has an asymptotic computational complexity of $O(MDNlog(N))$.

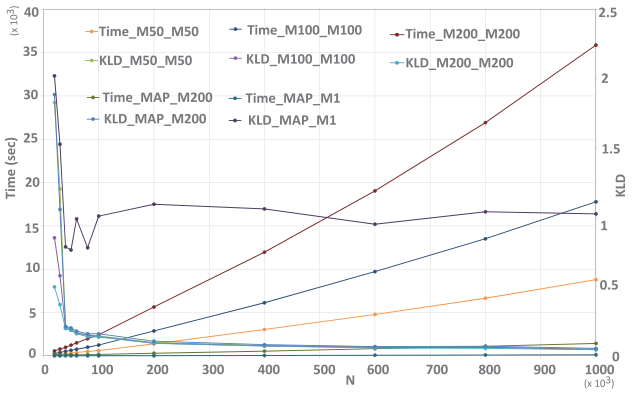


FIGURE 15. Execution time and KLD versus sample size (N) for different values of M for a 64-dimensional dataset. In the figure, _Mp_Mq refers to the case where $M = p$ is used for marginal densities and $M = q$ is used for the copula part. MAP refers to when maximum a-posteriori probability is used.

The profiling of the algorithm presented in the flowcharts of Fig. 10 and Fig. 11 for *Option 1*, ($M = 200$ for marginals, $M = 200$ for copula), with $N = 100,000$ and $D = 64$ in Table 3 reveals that only 5% of the execution time is spent on the copula part. One reason is that only 2% of 5277 cuts are attributed to the copula. However, since a copula cut goes over all the 64 dimensions we should expect it to have a much higher complexity (close to 64 times higher) compared to the complexity of a marginal cut that goes over a single dimension. But the profiling results reveal that the average computational complexity for a copula cut is only about 2 times larger than that of the marginal cuts! There are two reasons for this ratio to be low. First, referring to the flowchart in Fig. 10, we note that steps 10 to 14 in the flowchart are outside the loop for d, and therefore, executed only once for each iteration of m irrespective of the value of d. The profiling results further reveal that over 81% of the overall execution time is spent on the execution of these steps, with step 12 being the overwhelming contributor (over 78%). From the number of cuts in Table 3, it can be observed that on average every round of execution of steps 10 to 14 for a copula cut, there are 40 execution rounds of the same steps for a marginal cut. However, the overall complexity of one round of execution of these steps (in most part due to step 12) is more than three times less for a copula cut compared to a marginal cut. This is because in the copula case the fractions of cuts in the high density regions requiring time consuming count and sort operations across a large number of data points, and update of data structures in Fig. 13 and 14 is far less than that for the marginals. This can be observed from Fig. 7, where large blocks of high density areas in blue have no cuts. Further, for by the same token, the complexity of one round of execution of steps in the d loop, step 6 (involving the count of data points in each newly created subregion) to 9 is much less for a copula cut than for a marginal cut [1].

From the preceding discussion, due to the fact that copula cuts form only a small fraction of the overall time, attempts to parallelize this part of the algorithm yields no benefit.

TABLE 7. Impact of correlation on computation time and estimation accuracy ($N = 100,000$).

R	Time			Cuts			KLD
	Marginal	Copula	Total	Marginal	Copula	Total	
0	2315 (93.7%)	160 (6.3%)	2475	5155	122	5277	0.13
0.2	2288 (92.8%)	178 (7.2%)	2466	5106	178	5284	0.13
0.4	2282 (90.5%)	239 (9.5%)	2521	5078	246	5324	0.14
0.6	2282 (89.9%)	256 (10.1%)	2538	5133	269	5402	0.23
0.8	2280 (85.0%)	402 (15.0%)	2682	5133	464	5597	0.15
0.9	2352 (82.6%)	494 (17.4%)	2846	5133	579	5712	0.16
0.95	2302 (80.0%)	574 (20.0%)	2876	5133	671	5804	0.18
1.0	2322 (25.1%)	6930 (74.9%)	9252	5148	4177	9325	14.71

Therefore, we only focus our efforts on parallelization over the marginals.

B. EFFECT OF COVARIANCE MATRIX ON PERFORMANCE OF BSP

For illustration simplicity in Section II, for the Gaussian mixture distribution in (7), we assumed $\boldsymbol{\Sigma}_r[i,j] = 0$ for $i \neq j$. This is equivalent to $\boldsymbol{R}_r[i,j] = 0$ for $i = j$ and $\boldsymbol{R}_r[i,i] = 1$, where $\boldsymbol{R}_r = (\boldsymbol{\Sigma}_r)^{-\frac{1}{2}} \boldsymbol{\Sigma}_r (\boldsymbol{\Sigma}_r)^{-\frac{1}{2}}$ is the correlation matrix. The results in Table 7 present the KLD and execution time performance of BSP for a range of correlation coefficients between 0 and 1. From the data in the table, the KLD values remain small for correlation coefficients as high as 0.95. This shows that the performance of the algorithm is not negatively affected by increased correlation between the dimensions. The contribution of copula to the total computation time increases linearly from about 6% to about 20% for the change in the correlation coefficient from 0 to 0.95. The increase in the number of cuts follows the same trend [1].

However, $R = 1.0$ is a special case, not typically observed with real data. As an example, in a 2-dimensional space, the joint density is reduced to a straight line. As can be seen in Table 7, the number of cuts and computation time for the marginals are not changing significantly, from R = 0.95 to R = 1.0. The sudden rise in the computation time is due to the big jump in the number of copula cuts (from 671 for $R = 0.95$ to 4177 for $R = 1.0$).

It should be noted that in the multivariate analysis, where there is high correlation between the marginals we can use the method of principal component analysis (PCA) [39] to transform the dataset into a new set of variables in smaller number of dimensions (principal components) that are uncorrelated. The method of BSP can then be applied to the transformed dataset [1].

In another experiment, we investigated the performance of the algorithm in density estimation for a 12-dimensional correlated data, similar to the data employed in [40]. It is a 4-component Gaussian mixture, with the mean and variance values shown below. $\boldsymbol{\mu_1}, \boldsymbol{\mu_2}, \boldsymbol{\mu_3}, \boldsymbol{\mu_4}, \boldsymbol{\Sigma}_1, \boldsymbol{\Sigma}_2, \boldsymbol{V}_1, \boldsymbol{V}_2, \boldsymbol{V}_3$, and $\boldsymbol{V}_4$, as shown at the bottom of the next page.

The fractions of data assigned to components 1 to 4 is: 5%, 75%, 5% and 15%, respectively. That is:

$$X \sim 0.05\mathcal{N}(\mu_1, V_1) + 0.75\mathcal{N}(\mu_2, V_2) + 0.05\mathcal{N}(\mu_3, V_3) + 0.15\mathcal{N}(\mu_4, V_4)$$

The results of the density estimation are reported in Table 8. In both marginal and copula parts of the algorithm, $M = 200$ is used. As the results show, the algorithm is able to provide good estimation accuracy for this complex multimodal dataset. Similar to the previous experiments, the computation time is changing linearly with the sample size N. Also, as the sample size is increased, the estimation error KLD decreases, until it almost reaches a saturation level. Unlike the results previously shown in Tables 3 to 5, the number of cuts made in the copula domain is more than the overall marginal cuts. This is because in this new dataset, most of the dimensions are correlated, and the dataset is a mixture of 4 Gaussian components, which creates a complex multidimensional and multimodal structure in the 12-dimensional copula transformed sample space. Thus, a lot of cuts will be required in the multidimensional copula domain.

C. PARALLELIZATION

The proposed algorithm in the flowcharts of Fig. 10, and 11, and the data structures in Table 6, Fig. 13 and Fig. 14, have been designed suitably for the ease of parallelization around BSP parameters; datatset elements $1 \leq n \leq N$, sample partitions $1 \leq m \leq M$, dimensions $1 \leq d \leq D$ and subregions $1 \leq p \leq j-1$ [1]. In this section, we limit the discussion on the coarse-grain parallelization around m and d. Fine-grain parallelization over data sample n is best achieved through massively parallel architectures of graphical processing unit (GPU) and is limited to the counting and sorting of data samples for the purpose of density estimation in the subregions. Due to significant overhead of data transfer between the host CPU and the device GPU, and the non-coalesced memory access pattern of the data structures in Fig. 14, it is not possible to take full advantage of the GPU computing fabric to efficiently accelerate the counting and sorting operations on the GPU. Following the discussion in Section IV and with reference to Fig. 12, parallelization over p is no more needed due to elimination of the loop over p due to simplification presented in Section IV-B. Therefore, we have focused our efforts on the coarse-grain parallelization over d and m, where we expect to obtain the maximum speedup gains. The chosen platform for parallelization is a four-core CPU (Intel i7-3820, 3.60 GHz, with 32 GB RAM).

1) PARALLELIZATION OVER d

For the BSP over the marginals, since the dimensions are decoupled from each other, we expect to gain the maximum benefit in parallelization over d. We have used OPENMP® [41] programming model ("`parfor`" in MATLAB®) for parallelization over d. The reason for this choice is that iteration over the marginals are completely independent of each other and there is no exchange of data between the parallel execution threads. With a 4-core CPU, we can achieve a four-way paralellization for a 64-dimensional problem, with each worker core being allocated the workload for 16 marginals. Unfortunately, due to uneven workload across the marginals, the overhead of setting up the parallel streams, and non-parallelized execution portion for copula cuts, the maximum achievable speedup, as seen form Table 9, is no more than 3.3 for *Option 1* ($M = 200$ for marginals, $M = 200$ for copula) and *Option 3* (for marginals, $M = 1$ for copula), with $N = 100,000$

$$\boldsymbol{\mu_1} = [1.41, 0.81, 0.49, 0.80, 1.07, 0.30, 2.11, 1.21, 0.73, 1.20, 1.61, 0.45]$$
$$\boldsymbol{\mu_2} = [-14.07, -8.09, -4.86, -7.97, -10.74, -2.98, 14.07, 8.09, 4.86, 7.97, 10.74, 2.98]$$
$$\boldsymbol{\mu_3} = [14.07, 8.09, 4.86, 7.97, 10.74, 2.98, 7.04, 4.05, 2.43, 3.99, 5.37, 1.49]$$
$$\boldsymbol{\mu_4} = [5.63, 3.24, 1.95, 3.19, 4.30, 1.19, -21.11, -12.14, -7.30, -11.96, -16.11, -4.48]$$

$$\boldsymbol{\Sigma}_1 = \begin{bmatrix} 0.885 & -0.0023 & 0.0077 & 0.0041 & -0.0229 & -0.0025 \\ -0.0023 & 0.55 & 0.0015 & 0.0028 & 0.0013 & 0.0001 \\ 0.0077 & 0.0015 & 0.31 & 0.0018 & -0.0011 & -0.0002 \\ 0.0041 & 0.0028 & 0.0018 & 0.29 & 0.0004 & -0.0012 \\ -0.0229 & 0.0013 & -0.0011 & 0.0004 & 0.169 & 0.0004 \\ -0.0025 & 0.0001 & -0.0002 & -0.0012 & 0.0004 & 0.24 \end{bmatrix},$$

$$\boldsymbol{\Sigma}_2 = \begin{bmatrix} 0.1365 & -0.0009 & 0.0063 & 0.0075 & -0.0119 & -0.0101 \\ -0.0009 & 0.82 & 0.0021 & 0.0048 & 0.0045 & -0.0049 \\ 0.0063 & 0.0021 & 0.20 & 0.0018 & 0.0011 & -0.0018 \\ 0.0075 & 0.0048 & 0.0018 & 0.137 & 0.0042 & -0.0078 \\ -0.0119 & 0.0045 & 0.0011 & 0.0042 & 0.153 & -0.0051 \\ -0.0101 & -0.0049 & -0.0018 & -0.0078 & -0.0051 & 0.96 \end{bmatrix}$$

$$V_1 = \begin{bmatrix} \Sigma_1 & 0 \\ 0 & \Sigma_1 \end{bmatrix}, \quad V_2 = \begin{bmatrix} \Sigma_1 & 0 \\ 0 & \Sigma_2 \end{bmatrix}, \quad V_3 = \begin{bmatrix} \Sigma_2 & 0 \\ 0 & \Sigma_1 \end{bmatrix}, \quad V_4 = \begin{bmatrix} \Sigma_2 & 0 \\ 0 & \Sigma_2 \end{bmatrix}$$

TABLE 8. Density estimation results for a 12-dimensional Gaussian mixture. Results are the average of 5 runs. The number of cuts are shown as the sum of total marginal and copula cuts, as well as a pair in the parenthesis corresponding to each component. For KL divergence (KLD) and Hellinger distance (HLGR), the standard deviation is also shown inside parenthesis.

	N=20k	N=100k	N=200k	N=400k	N=600k	N=800k	N=1M
Cuts	2197 (744+1453)	5065 (1156+3909)	7273 (1426+5847)	10169 (1762+8407)	12589 (1983+10606)	14261 (2177+12084)	16119 (2314+13805)
Time	603	2244	4232	8068	15638	20231	24605
KLD (std)	1.4610 (0.0637)	1.0334 (0.0487)	0.8110 (0.0232)	0.6917 (0.0145)	0.6513 (0.0047)	0.6186 (0.0073)	0.6010 (0.0048)
HLGR (std)	0.4344 (0.0110)	0.3116 (0.0067)	0.2482 (0.0049)	0.2088 (0.0034)	0.1922 (0.0016)	0.1816 (0.0022)	0.1756 (0.0019)

TABLE 9. Speedup rates for parallelization over d and m, on a 4-core machine ($N = 100,000$, $D = 64$).

Scenario	Speedup		
	Option 1	Option 2	Option 3
Parallelization over d	3.3	1.4	3.3
Parallelization over m	2.2	-	-

and $D = 64$. Table 9 also presents the speedup for *Option 2* (for marginals, $M = 200$ for copula). The low speedup factor of 1.4 for *Option 2* is in keeping with the fact that with $M = 1$, the execution time for the marginals has reduced by a large factor and is of the similar order to the non-parallelized copula.

2) PARALLELIZATION OVER m

In parallelization over m, in each level j, evaluations of M sample partitions are only partially independent of each other. The decision diamond in the flowchart of Fig. 10, where `max(score[])` is evaluated outside the parallel loop, forms the synchronization barrier for the computing worker cores. We have used MPI [42] programming model for parallelization over m. The reason for this choice is that iterations of m are only partially independent of each other and there is a need for the exchange of data between the parallel execution threads through the MPI message passing constructs ("`spmd`" in MATLAB®). Considering that step 15 in the flowchart forms about 10% of the execution time, and the significant overhead of synchronization in the diamond decision box, overhead of distribution of data among the workers in step 15, and initial overhead of distribution of large data structures across the workers, plus the overhead of non-prallellized copula cuts, the maximum speedup for four workers on a 4-core machine is no more than 2.2 as seen form Table 9.

VII. RELATED WORK

Sequential adaptive partitioning of the sample space has been used in other studies for non-parametric multivariate density estimation. In this section we briefly review some of those studies.

In [43], a partition-based technique similar to BSP is used for density estimation. The algorithm starts with a d-dimensional hyper-rectangle and makes sequential binary partitions, to learn a piecewise constant density function at the end. In each step, in order to decide which one of the existing sub-rectangles should be cut next, the algorithm examines how uniformly the data points in each sub-rectangle are distributed. They use *star discrepancy* [44] to measure the uniformity of points in each sub-rectangle.

The work presented in [45] is also based on the same idea of adaptive binary partitions, done in a sequential way, to obtain piecewise constant density functions. However, the decision on which sub-rectangle to cut next is made via a maximum likelihood estimator.

Star discrepancy is also used in [46] as a measure for testing the uniformity of sample points, along each dimension in all D-dimensional sub-cubes. However, in this study, once a non-uniform sub-cube is selected for further splitting, each dimension of the chosen sub-cube is temporarily divided into m equal bins. Then, the $(m-1)D$ possible cuts are examined, so that the one leading into maximum non-uniformity is selected for splitting the sub-cube.

In a more recent work [47], another partitioning-based non-parametric density estimation method is presented. This method is also based on adaptive sequential partitioning of a hyper-rectangular domain. However, they use Wasserstein distance [48] for testing the uniformity of data distribution in each partition element.

In [49] a nonparametric density estimator based on multivariate histograms is presented. Their aim was to smooth out the discontinuities caused by histograms bins, via applying some affine transformation on the data. This transformation shifts the multidimensional histogram bins and changes their shape and size. Following that, in order to deal with the discontinuities, they simply do an averaging over the piece-wise constant estimated densities.

The related studies reviewed in this section are mostly limited to low dimensional cases of two, to six. The work in [43] presents the results for a 10-dimensional case. The reason for inability of these sequential adaptive partitioning methods is the difficulty of learning joint and marginal distributions

together– the difficulty that we overcome through the use of Copula transform.

The other issue with these works is that their computational complexities grow rapidly with dimension. The computational complexity in [43] is bound by $O(N\ log^{D-1}N)$. For the method in [45], the worst computational complexity is $O(N\ log^{D}N)$. The work in [49] presents the simulation results for dimensions less than six, where the computational complexity grows in a quadratic way with the number of dimensions ($O(D^2NH)$).

In this paper, we have provided simulation results for a range of dimensions from 2 to 256, where the computational complexity grows linearly with dimension: $O(MDNlog(N))$. As an example for a case of $N = 400k$, $D = 256$ and $M = 200$, the computational complexity of the method presented in this work is lower than the technique presented in [45] by 185 orders of magnitude!

VIII. CONCLUSION AND FUTURE WORKS

In this paper, we presented the computational details of BSP algorithm. We also presented our method of copula transform to reduce the complexity of BSP. Further, we designed a set of efficient data structures for deploying BSP method for accelerated density estimation in high dimensions. All the data structures have been suitably designed for efficient implementation on parallel computing paradigms. Use of copula transform has been shown to be effective in saving significant amount of computation time, in high dimensions. It reduces the number of required cuts in the high dimensional sample space, by mapping the random variables to a copula domain, in which the data have uniform marginal distributions. The performance of the BSP algorithm, in terms of the estimation error and computation time, was investigated for a wide range of sample sizes N, and a number of sample partitions M.

From the simulation results, computation time changes in a linear manner with N and M. Careful choice of M results in a great deal of saving in computation time. With the use of copula transform, increasing the number of sample partitions beyond $M = 1$ in estimating the marginal densities does not result in appreciable decrease in estimation error.

With separation of marginal and joint density estimation using copula transform, we tried the idea of using KDE method or median-based cuts for computing the marginal densities. Neither of these methods showed a consistent advantage over the regular BSP method.

Further, it was demonstrated we can take the advantage of independent nature density estimation across the marginals to fully parallelize the computation across the marginals.

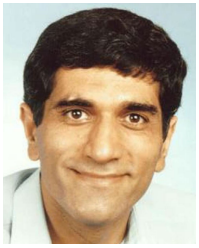
Possible directions for future works include exploration of other possible options to replace the current binary partitioning method with a more effective data-driven partitioning scheme, to improve the performance of the algorithm with more complex high-dimensional data structures with high correlation, specially when the number of available data points is small.

REFERENCES

[1] A. Majdara, "Offline and online density estimation for large high-dimensional data," Ph.D. dissertation, Dept. Elect. Comput. Eng., Michigan Technol. Univ., Houghton, MI, USA, 2018.
[2] P. Simon, *Too Big to Ignore: The Business Case for Big Data*. Hoboken, NJ, USA: Wiley, 2013.
[3] C. Heinz and B. Seeger, "Cluster kernels: Resource-aware kernel density estimators over streaming data," *IEEE Trans. Knowl. Data Eng.*, vol. 20, no. 7, pp. 880–893, Jul. 2008.
[4] B. W. Silverman, *Density Estimation for Statistics and Data Analysis*. London, U.K.: Chapman & Hall, 1986.
[5] M. Bressan and J. Vitrià, "Nonparametric discriminant analysis and nearest neighbor classification," *Pattern Recognit. Lett.*, vol. 24, no. 15, pp. 2743–2749, Nov. 2003.
[6] Y. Motai and H. Yoshida, "Principal composite kernel feature analysis: Data-dependent kernel approach," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 8, pp. 1863–1875, Aug. 2013.
[7] J. H. Friedman and N. I. Fisher, "Bump hunting in high-dimensional data," *Statist. Comput.*, vol. 9, no. 2, pp. 123–143, Apr. 1999.
[8] M. P. Wand and M. C. Jones, *Kernel Smoothing*. London, U.K.: Chapman & Hall, 1995.
[9] L. Lu, H. Jiang, and W. H. Wong, "Multivariate density estimation by Bayesian sequential partitioning," *J. Amer. Stat. Assoc.*, vol. 108, no. 504, pp. 1402–1410, Dec. 2013.
[10] D. W. Scott, *Multivariate Density Estimation: Theory, Practice, and Visualization*. Hoboken, NJ, USA: Wiley, 1992.
[11] P. P. B. Eggermont and V. N. LaRiccia, *Maximum Penalized Likelihood Estimation: Density Estimation* (Springer Series in Statistics), vol. 1. New York, NY, USA: Springer, 2001.
[12] R. E. Bellman, *Dynamic Programming*. New York, NY, USA: Dover, 2003.
[13] A. W. Bowman and A. Azzalini, *Applied Smoothing Techniques for Data Analysis*. Oxford, U.K.: Oxford Univ. Press, 1997.
[14] Y. Zheng, J. Jestes, J. M. Phillips, and F. Li, "Quality and efficiency for kernel density estimates in large data," in *Proc. Int. Conf. Manage. Data (SIGMOD)*, 2013, pp. 433–444.
[15] C. Yang, R. Duraiswami, N. A. Gumerov, and L. Davis, "Improved fast Gauss transform and efficient kernel density estimation," in *Proc. 9th IEEE Int. Conf. Comput. Vis.*, Oct. 2003, pp. 664–671.
[16] J. Chacón and T. Duong, *Multivariate Kernel Smoothing and Its Applications*. Boca Raton, FL, USA: CRC Press, 2018.
[17] A. Gramacki, *Nonparametric Kernel Density Estimation and Its Computational Aspects*. Cham, Switzerland: Springer, 2018.
[18] S. Wang, A. Li, K. Wen, and X. Wu, "Robust kernels for kernel density estimation," *Econ. Lett.*, vol. 191, pp. 109–138, Jun. 2020.
[19] Y. Jin, Y. He, and D. Huang, "An improved variable kernel density estimator based on L2 regularization," *Mathematics*, vol. 9, no. 16, p. 2004, Aug. 2021.
[20] J. Jiang, Y.-L. He, D.-X. Dai, and J. Z. Huang, "A new kernel density estimator based on the minimum entropy of data set," *Inf. Sci.*, vol. 491, pp. 223–231, Jul. 2019.
[21] Y. Chen, "A tutorial on kernel density estimation and recent advances," *Biostatist. Epidemiol.*, vol. 1, no. 1, pp. 161–187, 2017.
[22] S. Chen, "Optimal bandwidth selection for kernel density functionals estimation," *J. Probab. Statist.*, vol. 2015, pp. 1–21, Jan. 2015.
[23] Y.-L. He, X. Ye, D.-F. Huang, J. Z. Huang, and J.-H. Zhai, "Novel kernel density estimator based on ensemble unbiased cross-validation," *Inf. Sci.*, vol. 581, pp. 327–344, Dec. 2021.
[24] J. Klemelä, J. S. Marron, and S. J. Sheather, *A Brief Survey of Bandwidth Selection for Density Estimation*. Hoboken, NJ, USA: Wiley, 2009.
[25] J. Klemelä, "Multivariate histograms with data-dependent partitions," *Statistica Sinica*, vol. 19, no. 1, pp. 159–176, 2009.
[26] G. Lugosi and A. Nobel, "Consistency of data-driven histogram methods for density estimation and classification," *Ann. Statist.*, vol. 24, no. 2, pp. 687–706, Apr. 1996.
[27] J. Silva and S. S. Narayanan, "Information divergence estimation based on data-dependent partitions," *J. Stat. Planning Inference*, vol. 140, no. 11, pp. 3180–3198, Nov. 2010.
[28] A. J. Izenman, *Modern Multivariate Statistical Techniques: Regression, Classification, and Manifold Learning*. New York, NY, USA: Springer, 2003.
[29] J. S. Liu, *Monte Carlo Strategies in Scientific Computing* (Springer Series in Statistics). New York, NY, USA: Springer, 2001.

[30] A. Kong, J. S. Liu, and W. H. Wong, "Sequential imputations and Bayesian missing data problems," *J. Amer. Statist. Assoc.*, vol. 89, no. 425, pp. 278–288, Mar. 1994.
[31] E. Barat, T. Dautremer, and T. Montagu, "Nonparametric Bayesian inference in nuclear spectrometry," in *Proc. IEEE Nucl. Sci. Symp. Conf. Rec. (NSS)*, Honolulu, HI, USA, Oct. 2007, pp. 880–887.
[32] D. W. Meyer, "Density estimation with distribution element trees," *Statist. Comput.*, vol. 28, no. 3, pp. 609–632, May 2018.
[33] G. E. Andrews, R. Askey, and R. Roy, *Special Functions*. Cambridge, U.K.: Cambridge Univ. Press, 2001.
[34] M. Hazewinkel, Ed., *Encyclopedia of Mathematics*. New York, NY, USA: Springer, 2001.
[35] A. Bayestehtashk and I. Shafran, "Parsimonious multivariate copula model for density estimation," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2013, pp. 5750–5754.
[36] M.-S. Chang and X. Wu, "Transformation-based nonparametric estimation of multivariate densities," *J. Multivariate Anal.*, vol. 135, pp. 71–88, Mar. 2015.
[37] A. Majdara and S. Nooshabadi, "Nonparametric density estimation using copula transform, Bayesian sequential partitioning, and diffusion-based kernel estimator," *IEEE Trans. Knowl. Data Eng.*, vol. 32, no. 4, pp. 821–826, Apr. 2020.
[38] A. Majdara and S. Nooshabadi, "Online density estimation over high-dimensional stationary and non-stationary data streams," *Data Knowl. Eng.*, vol. 123, Sep. 2019, Art. no. 101718.
[39] I. T. Jolliffe, *Principal Component Analysis*. New York, NY, USA: Springer, 2002.
[40] S. Chib and S. Ramamurthy, "Tailored randomized block MCMC methods with application to DSGE models," *J. Econometrics*, vol. 155, no. 1, pp. 19–38, Mar. 2010.
[41] (2016). *The OpenMP API Specification for Parallel Programming, OPenMP Architecture Review Board*. [Online]. Available: http://openmp.org/wp/
[42] (2016). *MPI: A Message-Passing Interface Standard, Message Passing Interface Forum*. [Online]. Available: http://www.mpi-forum.org/
[43] K. Yang, H. Su, and W. H. Wang, "Density estimation via discrepancy," 2015, *arXiv:1509.06831*.
[44] W. Chen, A. Srivastav, and G. Travaglini, *A Panorama of Discrepancy Theory* (Lecture Notes in Mathematics), vol. 2107. New York, NY, USA: Springer, 2014.
[45] L. Liu and W. H. Wong, "Multivariate density estimation based on adaptive partitioning: Convergence rate, variable selection and spatial adaptation," Dept. Statist., Stanford Univ., Stanford, CA, USA, Tech. Rep. 2014-01, 2014.
[46] K. Yang and W. H. Wong, "Discovering and visualizing hierarchy in multivariate data," 2014, *arXiv:1403.4370*.
[47] E. Luini and P. Arbenz, "Density estimation of multivariate samples using Wasserstein distance," *J. Stat. Comput. Simul.*, vol. 90, no. 2, pp. 181–210, Jan. 2020.
[48] L. Rüschendorf. *Encyclopedia of Mathematics*. Accessed: Oct. 2021. [Online]. Available: http://encyclopediaofmath.org/index.php?title=Wasserstein_metric&oldid=50083
[49] E. López-Rubio, "A histogram transform for probability density function estimation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 4, pp. 644–656, Apr. 2013.

AREF MAJDARA (Member, IEEE) received the B.S. degree in electrical engineering from Shahid Beheshti University, Iran, in 2002, the Ph.D. degree in management science and technology from Tohoku University, Japan, in 2009, and the M.S. and Ph.D. degrees in electrical engineering from Michigan Technological University, Houghton, MI, USA, in 2018. He is currently a Lecturer at the Department of Electrical and Computer Engineering, Michigan Technological University. His main research interests include analysis of high-dimensional big data, machine learning, high-performance computing, and parallel processing.



SAEID NOOSHABADI (Senior Member, IEEE) received the M.Tech. and Ph.D. degrees in electrical engineering from the India Institute of Technology, Delhi, India, in 1986 and 1992, respectively. He is currently a Professor of computer systems engineering with the Department of Electrical and Computer Engineering and the Department of Computer Science, Michigan Technological University, Houghton, MI, USA. Prior to his current appointment, he has held multiple academic and research positions. His last two appointments were with the Gwangju Institute of Science and Technology, Republic of Korea, from 2007 to 2010, and with the University of New South Wales, Sydney, NSW, Australia, from 2000 to 2007. His research interests include big data parallel information processing and low-power embedded processors.