Nonparametric Density Estimation: Toward Computational Tractability

Alexander G. Gray
Department of Computer Science
Carnegie Mellon University
agray@cs.cmu.edu

Andrew W. Moore
Robotics Inst. and Dept. Comp. Sci.
Carnegie Mellon University
awm@cs.cmu.edu

January 24, 2003

Abstract

Density estimation is a core operation of virtually all probabilistic learning methods (as opposed to discriminative methods). Approaches to density estimation can be divided into two principal classes, parametric methods, such as Bayesian networks, and nonparametric methods such as kernel density estimation and smoothing splines. While neither choice should be universally preferred for all situations, a well-known benefit of nonparametric methods is their ability to achieve estimation optimality for ANY input distribution as more data are observed, a property that no model with a parametric assumption can have, and one of great importance in exploratory data analysis and mining where the underlying distribution is decidedly unknown. To date, however, despite a wealth of advanced underlying statistical theory, the use of nonparametric methods has been limited by their computational intractibility for all but the smallest datasets. In this paper, we present an algorithm for kernel density estimation, the chief nonparametric approach, which is dramatically faster than previous algorithmic approaches in terms of both dataset size and dimensionality. Furthermore, the algorithm provides arbitrarily tight accuracy guarantees, provides anytime convergence, works for all common kernel choices, and requires no parameter tuning. The algorithm is an instance of a new principle of algorithm design: multi-recursion, or higher-order divide-and-conquer.

Keywords: kernel density estimation, nonparametric statistics, algorithms, divide-and-conquer, space-partitioning trees.

1 Introduction: Data Analysis Without Assumptions

In this section we'll briefly review the fundamental problem of density estimation and the reasons that nonparametric density estimation approaches are particularly well-suited to exploratory data mining. We'll then describe the severe computational obstacles posed by nonparametric density estimation, the main factor limiting their use in large-scale data analysis.

1.1 The fundamental problem of density estimation. In any probabilistic learning method (as opposed to discriminative methods such as support vector machines) the task of estimating a probability density from data is a fundamental one, upon which subsequent inference and decision-making procedures are based. For example, in classification one must find $\widehat{P}(C|\underline{X}) = \frac{\hat{p}(\underline{X}|C)P(C)}{\hat{p}(\underline{X})}$, where C is one of K classes and $\hat{p}(\underline{X}|C)$ is the (class-conditional) density of the data $\underline{X}$. Direct density estimation provides a principled way to formulate many common types of analyses, for example outlier detection (as low-density points), or more generally scoring of points according to how 'common' they are. In general, density estimation provides a classical basis across statistics for virtually any kind of data analysis in principle, including clustering, classification, regression, time series analysis, active learning, and so on [7, 1].

1.2 Methods of estimating a density. The task of estimating a probability density from data is a fundamental one, upon which subsequent inference, learning, and decision-making procedures are based. Density estimation has thus been heavily studied, under three primary umbrellas: parametric, semi-parametric, and nonparametric. *Parametric* methods are useful when the underlying distribution is known in advance or is simple enough to well-modeled by a standard distribution. *Semi-parametric* models (such as mixtures of simpler distributions) are more flexible and more forgiving of the user's lack of the true model, but usually require significant computation in order to fit the resulting nonlinear models (such as the EM iterative re-estimation method). *Nonparametric* methods assume the least structure of the three, and take the strongest stance of letting the data speak for themselves [22]. They are useful in the setting of arbitrary-shape distributions coming from complex real-world data sources. They are generally the method of choice in exploratory data analysis for this reason, and can be used, as the other types of models, for the entire range of statistical settings, from supervised learning to unsupervised learning to reinforcement learning. However, they apparently often come at the heaviest computational cost of the three types

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited

of models. This has, to date, been the fundamental limitation of nonparametric methods for density estimation. It prevents practitioners from applying them to the increasingly large datasets that appear in modern real-world problems, and even for small problems, their use as a repeatedly-called basic subroutine is limited.

Neither parametric nor nonparametric estimators are universally preferable in all situations, however. For example, when the sample size is small, there are sometimes statistical reasons to prefer parametric methods - though in the data mining setting we are generally dealing with large if not massive datasets. If compression is desired, nonparametric methods are entirely inappropriate. On the other hand, often strong parametric assumptions are inappropriate, perhaps nowhere more so than in exploratory data analysis. In practical terms, incorrect assumptions generally lead to incorrect inferences.

1.3 Density estimation without assumptions. Nonparametric methods make minimal or no distribution assumptions and can be shown to achieve asymptotic estimation optimality for ANY input distribution under them. For example using KDE (detailed below), with no assumptions at all on the true underlying distribution, given only that the scale $h_N \to 0$ and $N h_N \to \infty$, and that the kernel $K(\cdot)$ is a non-negative Borel function whose integral is 1 (easily satisified by all commonly-used kernels), then with probability 1,

$$\int |\hat{p}(\underline{x}) - p(\underline{x})| d\underline{x} \to 0 \quad as \quad N \to \infty \tag{1.1}$$

i.e. as more data are observed, the estimate converges to the true density [5]. This is clearly a property that no particular parametrization can achieve. [1] For this reason nonparametric estimators are the focus of a considerable body of advanced statistical theory [19, 6].

1.4 Kernel density estimation. The task is to estimate the density $\hat{p}(\underline{x}_q)$ for each point $\underline{x}_q$ in a query (test) dataset $\underline{X}_{\mathcal{Q}}$ (having size $N_{\mathcal{Q}}$), from which we can also compute the overall log-likelihood of the dataset $\widehat{L}_{\mathcal{Q}} = \sum_{q=1}^{N_{\mathcal{Q}}} \log \hat{p}(\underline{x}_q)$.

[1] 'Semi-parametric' approaches provide a certain middle-ground between parametric and nonparametric approaches, utilizing a number of components/hidden units/basis functions which is smaller than the training set, but are generally characterized by nonlinear optimization procedures which either find only locally-optimal solutions dependent on starting conditions (*e.g.* EM, gradient descent) or are acutely expensive (*e.g.* quadratic programming). Semi-parametric methods such as mixtures of Gaussians or sigmoidal neural networks can be shown to to approximate any density, BUT only if the number of components/hidden units is allowed to grow to infinity with the data: at this point they are by definition nonparametric methods. When viewed this way, as universal approximators with a very large number of components, such models suffer from a similar computational problem to the one we address here— and as it turns out similar methods apply to such models.

Kernel density estimation (KDE) is the most widely analyzed and used nonparametric density estimation method, hence the focus this paper. (There exist many elaborations upon the basic model presented here, but most of them do not pose any particular problems for our computational approach.) The 'model' is the training dataset $\underline{X}_{\mathcal{T}}$ (having size $N_{\mathcal{T}}$) itself, in addition to a local kernel function $K(\cdot)$ centered upon each training datum, and its scale parameter h (the 'bandwidth'). The density estimate at the q^{th} test point $\underline{x}_q$ is

$$\hat{p}(\underline{x}_q) = \frac{1}{N_{\mathcal{T}}} \sum_{t=1}^{N_{\mathcal{T}}} \frac{1}{V_{D\,h}} K\left(\frac{\|\underline{x}_q - \underline{x}_t\|}{h}\right) \tag{1.2}$$

where D is the dimensionality of the data and $V_{D\,h}$ is the volume encompassed by $K(\cdot)$.

1.5 Practical issues of KDE. For good estimates, the exact form of $K(\cdot)$ turns out to be relatively unimportant, while the correct choice of h is critical [22, 20]. Common choices for $K(\cdot)$ are the spherical, Gaussian, or Epanechnikov kernels; our method works efficiently with any such standard kernel. The spherical kernel ($K(\underline{x}_q, \underline{x}_t) = 1$ if $\|\underline{x}_q - \underline{x}_t\| < h$, otherwise 0) is simplest but can introduce sharp discontinuities for small datasets. The Epanechnikov kernel ($K(\underline{x}_q, \underline{x}_t) = \frac{D+2}{2V'_{D\,h}}(1 - \|\underline{x}_q - \underline{x}_t\|^2)$ if $\|\underline{x}_q - \underline{x}_t\| < h$, otherwise 0, where $V'_{D\,h}$ is the volume of the sphere in D dimensions) has the property of asymptotically-optimal efficiency among all possible kernels [8], and so is the default used in our studies.

While a multitude of analytical, or 'plug-in' methods exist for selecting the optimal bandwidth for a dataset, their derivation usually depends on asymptotic assumptions which may or may not hold for the data at hand. The alternative, for reliable bandwidth selection, is cross-validation [2, 22, 15]. However, such procedures entail performing multiple density estimates, one for each of the B candidate bandwidths considered, which multiplies the core cost of estimation by another factor of B. This makes the cost of estimation even more acutely felt in practice.

1.6 Computational cost of KDE. The basic cost of KDE (for a given bandwidth) poses two severe computational obstacles, which are the primary problems addressed by this paper.

- **Quadratic time complexity.** The main difficulty is that evaluating a density naively (by summing over each training point for each of the query points) is $O(N_{\mathcal{Q}} N_{\mathcal{T}})$ (or simply $O(N^2)$; much of the time it will turn out that $\underline{X}_{\mathcal{Q}} = \underline{X}_{\mathcal{T}}$, so for notational convenience we may also use $N = N_{\mathcal{Q}} = N_{\mathcal{T}}$).
- **Curse of dimensionality.** The general learning setting is necessarily multivariate, and thus the assumption in

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited

this paper. Algorithms which scale exponentially in D are said to suffer from the curse of dimensionality. This behavior tends to quickly render problems beyond even one or two dimensions intractable, so thought must be given to avoid this.

2 Existing Approaches

Though interest in the computational problem of kernel density estimation is nearly as old as the statistical method itself, effective algorithmic solutions have been developed only for the univariate case. These methods do not extend feasibly to the general multivariate setting.

2.1 Gridding the data. The idea of gridding is to approximate the training data by chopping each dimension into a fixed number of intervals M, then assigning the original data to neighboring grid points to obtain grid counts, representing the amount of data in its neighborhood. The kernel function is evaluated at grid points rather than actual points. The main problem with gridding is that the number of grid points required is M^D, exponential in the number of dimensions. The cost of estimating a density given N training and test points is $O((M^D)^2)$, where presumably M is somehow set proportionally to N if accuracy is to be maintained. Test points falling in the regions in-between are linearly interpolated, yielding another source of error. Further, the true error resulting from the reduced representation is not provided by such a method, and available insight into the error of such procedures is limited [23]. Ironically, most of this computational expense is wasted, since in higher dimensions most of the grid cells will be empty. A recently-proposed method of this kind is the 'fast-binning' method [9].

2.2 FFT on gridded data. An elaboration uses the fast Fourier transform [21, 22], performing discrete convolutions to combine the grid counts and kernel weights. However, because a grid still underlies the method (the FFT was intended for a regularly-spaced time-varying signal, which explains its awkwardness in this context), it still suffers from similarly explosive scaling and error limitations. Its cost is $O(M^D \log(M^D))$. For these obvious reasons, these grid-based methods were presented for the univariate setting and hardly considered for D higher than 2 or perhaps 3. [2]

2.3 Naive method. For the general multivariate case, then, the only available method is the simple looping described earlier. Its cost is $O(DN^2)$.

[2] In a thorough study by Wand[23], in tests of dimensionality up to 3 and data size up to 10,000 points, it was concluded empirically that this method can give speedups up at most 5 over the naive method, and in many cases incurs about the same computational cost as the quadratic method. Not surprisingly, available implementations by the authors are hard-coded for the univariate case.

3 Space-partitioning Trees

From the outset, our approach departs from the fundamentally exponential nature of grids in the multivariate setting.

3.1 From grids to *kd*-trees. A tree can be understood in this context as a more powerful generalization of a grid - specifically, as a set of linked grids built at different resolutions This departure from the simplistic grid to the tree suddenly yields a basis for divide-and-conquer, which can integrate local information to obtain an accurate global solution.

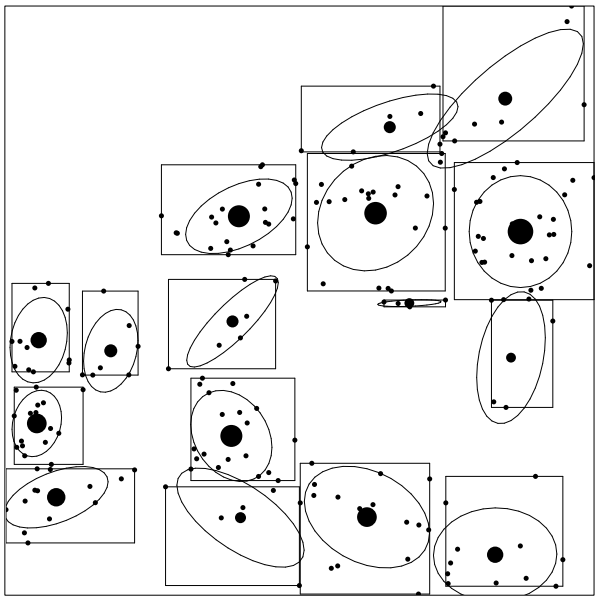

Figure 1: Fourth level of an *mrkd*-tree. The dots are the individual data points. The sizes and positions of the disks show the node counts and centroids. The ellipses and rectangles show the covariances and bounding boxes.

kd-trees [10, 18, 4] are simple yet effective space-partitioning trees, where each node in the tree defines a distinct region in the data space using a bounding hyperrectangle (which is maximally tight in each coordinate), each split is made along a single coordinate (the one having largest variance), and leaves contain one or a small number of points. The tree may be built all the way down to individual points at the leaves, or alternatively such that leaves contain up to some small predefined number of points. We use an extension called *multi-resolution* kd-*trees* [4] which additionally contain local statistics such as the mean and covariance within each node, depending on the problem. Besides generalizing the flat grid to higher levels of resolution, *kd*-trees place hyperrectangles in a manner which is sensitive to the shape of the density, allow us to escape much of the inaccuracy caused by the data-blind nature of the simplistic grid.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited

3.2 From rectangles to spheres: Anchor hierarchies. By partitioning in each dimension separately, *kd*-trees have some of the same property which causes the grid representation to have exponential growth in the dimension. If we instead partition by selecting centroid *points* to define the left and right sets of the partitioning, we can escape this representation problem as well. It can also be seen that there is nothing in the construction, storage, or traversal of such a data structure, called a *ball-tree*, which grows exponentially with the dimension of the data. Ball-trees can be built efficiently and with high quality using the anchors hierarchy algorithm [16], and have been demonstrated to be effective in up to thousands of dimensions in [16]).

In general *kd*-trees are still effective in low dimensions and faster to build than ball-trees. Thus we have two choices for our divide-and-conquer substrate: *kd*-trees for low dimensions (less than about 10) or ball-trees for higher dimensions (demonstrated to be effective in up to thousands of dimensions in [16]).

4 Divide-and-Conquer: Single-Tree Algorithm

We'll now develop an algorithm where a single tree partitions the training set $\underline{X}_{\mathcal{T}}$. It builds upon the method of [4]: though developed for kernel regression, it straightforwardly leads to a method for KDE, which we'll augment with *bounded and prioritized approximation*. While new and quite effective in its own right, we present it only as a stepping stone for understanding the more powerful dual-tree method to come.

4.1 Exclusion and inclusion. For each query point, we traverse the tree in standard depth-first fashion. At each node T encountered during the traversal, using the boundary of the data $\underline{X}_T$ in the node (having size N_T), we can easily compute lower and upper bounds on the distance between the query point and any point in $\underline{X}_T$. Using this we can compute bounds on the mass contribution of $\underline{X}_T$ to the density at the query point, $\hat{p}(\underline{x}_q)$. If the maximum density contribution of T is zero within the floating precision of the machine, it can be pruned from the search (*i.e.* we do not need to recurse on its children). The opposite of exclusion is also possible: we can prune a node when its minimum possible mass contribution is 1 within machine-precision. Note that exclusion does not necessarily introduce any error: in the case of a finite-extent kernel (such as the spherical or optimal-efficiency Epanechnikov kernel), exclusion when the minimum distance is greater than the kernel extent preserves the result exactly. A similar property holds for inclusion with the spherical kernel.

4.2 Constant-mass centroid approximation. If the percentage difference between these bounds is smaller than some predetermined small δ, we can prune the node by approximating its mass contribution by its centroid $\underline{\mu}_T$, *i.e.* we add $N_T K(\underline{\mu}_T)$. Exclusion and inclusion are actually both special cases of this more general pruning rule. Up to this point we have described the algorithm of [4] adapted to KDE, albeit in new terms and allowing a new type of tree; now we add a few more things.

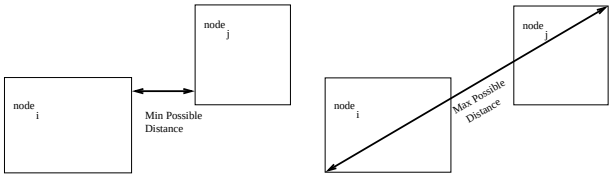

Figure 2: Distance bounds. The lower and upper bound on pairwise distances between the points contained in each of two *kd*-tree nodes.

4.3 Guaranteed point-wise precision. While in practice quite judicious, the approximation pruning criterion described does not provide an obvious guarantee on the accuracy of the density estimate $\hat{p}(\underline{x}_q)$. We can maintain lower and upper bounds l_q and u_q on this quantity itself: we begin with maximally pessimistic bounds and tighten them as we recurse and observe training points at increasingly finer granularity. We start by agnostically setting the lower bound to assume that no training points contribute any mass, and the upper bound to assume that all training points contribute maximum mass. If we perform exclusion only when $\frac{|(u_q+du)-(l_q+dl)|}{|l_q+dl|} < \epsilon$ (where dl and du represent the contribution of the node in question), we impose a hard guarantee that the maximum error in $\hat{p}(\underline{x}_q)$ is no greater than ϵ. However, the resulting behavior is much less aggressive than that of constant-mass pruning.

4.4 Locally prioritized approximation. Observing that the order of nodes encountered affects the efficiency of the procedure (the earlier we obtain tight bounds on a node, the more pruning we can do), we replace the depth-first traversal with a local best-first traversal according to a simple priority function (choose the node having minimum distance to $\underline{x}_q$).

5 Higher-order Divide-and-Conquer: Generalizing to a Dual-tree Algorithm

Extending the ideas in the augmented single-tree algorithm just described, we now develop a *dual*-tree algorithm: Now, a second tree is also built to partition the query set $\underline{X}_{\mathcal{Q}}$. Rather than compare a single point with chunks of data, we will now compare chunks with chunks. This will allow us to exploit additional structure in the overall computational problem and to reason globally about its parts. The algorithm is shown with the developments in this section in Figure 3.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited

5.1 Multi-recursion: Dual-tree traversal. We must now, however, have a way of traversing two trees simultaneously: we achieve this by means of a four-way recursive step, corresponding to each possible node-pair comparison. [3] The single-tree algorithm is actually a special case of the dual-tree algorithm, where the query set consists of one point and the algorithm is simply called once for each query.

We'll also need to generalize some of our other notions to the dual-tree case. Now the priority function chooses the node-pair having smallest distance between bounding regions. The point-wise mass bounds l_q and u_q are generalized to l_Q and u_Q, holding for all the query points in query node Q.

5.2 Guaranteed global precision. By considering all the query points simultaneously rather than individually, we are now able to achieve approximation on the overall log-likelihood $\widehat{L}_{\mathcal{Q}}$: this is often the true quantity of interest, particularly when performing multiple density estimates within cross-validation for model selection. We now have the ability to spend computation more wisely, possibly allowing extra point-wise error in some query points as long as the overall density's error is still within our requirement.

Now a challenge is introduced, since we need to maintain bounds on a *global* quantity, $\widehat{L}_{\mathcal{Q}}$, but each pruning or approximation step in the algorithm is *local* to a single query node. The key to maintaining global log-likelihood bounds $\widehat{L}^l_{\mathcal{Q}}$ and $\widehat{L}^u_{\mathcal{Q}}$ during the search is the additive composition of $\widehat{L}_{\mathcal{Q}}$, *e.g.* the contribution of node Q to $\widehat{L}^l_{\mathcal{Q}}$ is the addition of $N_Q \log l_Q$. To remove ambiguity across levels, a separate pair of bounds is maintained for each level in the query tree. The global error constraint is met if at any level $\frac{|\widehat{L}^u_{\mathcal{Q}} - \widehat{L}^l_{\mathcal{Q}}|}{|\widehat{L}^l_{\mathcal{Q}}|} < \epsilon$.

5.3 Globally prioritized approximation. It is now useful to generalize the search control beyond locally-recursive traversal using a global *priority queue* $\mathcal{P}$, which can influence the search toward node-pairs where the query node's bounds are currently least tight, with a simple modification to the priority function. Search begins by pairing the two root nodes on $\mathcal{P}$.

Anticipating that the search will end at $\epsilon > 0$, we can effectively defer the computational cost of maintaining the minimum-priority property of the underlying heap structure as long as possible, ending the search with as many unpaid-for queue insertions as possible. This can be done by implementing the priority queue $\mathcal{P}$ using the asymptotically-optimal *Fibonacci heap*, which has amortized insertion cost $O(1)$ and dequeueing cost $O(\log n)$ for n items on the queue.

[3] Interestingly, this progression can be extended to an arbitrary number of trees (if done non-naively), to increasingly dramatic computational advantage for the appropriate problems [12].

Dualtree($\mathcal{P}$)
`while` !empty($\mathcal{P}$),
  `if` $\frac{|\widehat{L}^u_{\mathcal{Q}} - \widehat{L}^l_{\mathcal{Q}}|}{|\widehat{L}^l_{\mathcal{Q}}|} < \epsilon$, `return`.
  $\{Q, T, dl, du, p\}$ = minpriority($\mathcal{P}$).
  `if` $p < p_{\max}$,
    $dl' = N_T K(\text{maxdist}(Q, T))$.
    $du' = N_T K(\text{mindist}(Q, T))$.
    $dl = dl'$, $du = du' - N_T$.
    `if` $\frac{|du' - dl'|}{|l_Q + dl'|} < \delta$,
      `foreach` $\underline{x}_q \in Q$, l_q += dl, u_q += du.
      $\widehat{L}^l_{\mathcal{Q}}$ -=$N_Q log(l_Q)$, $\widehat{L}^u_{\mathcal{Q}}$ -= $N_Q log(u_Q)$.
      l_Q += dl, u_Q += du.
      $\widehat{L}^l_{\mathcal{Q}}$ += $N_Q log l_Q$, $\widehat{L}^u_{\mathcal{Q}}$ += $N_Q log u_Q$.
      enqueue($\{Q, T, dl, du,$priority(Q, T)+$P\}$).
      `continue`.
  `else`,
    $dl = -dl$, $du = -du$.
    `foreach` $\underline{x}_q \in Q$, l_q += dl, u_q += du.
    $\widehat{L}^l_{\mathcal{Q}}$ -=$N_Q log(l_Q)$, $\widehat{L}^u_{\mathcal{Q}}$ -= $N_Q log(u_Q)$.
    l_Q += dl, u_Q += du.
    $\widehat{L}^l_{\mathcal{Q}}$ += $N_Q log l_Q$, $\widehat{L}^u_{\mathcal{Q}}$ += $N_Q log u_Q$.
  `if` leaf(Q) `and` leaf(T), **Dualtree_base**(Q,T).
  enqueue(Q.left,T.left,dl, du,priority(Q, T)).
  enqueue(Q.left,T.right,dl, du,priority(Q, T)).
  enqueue(Q.right,T.left,dl, du,priority(Q, T)).
  enqueue(Q.right,T.right,dl, du,priority(Q, T)).

Dualtree_base(Q,T)
`foreach` $\underline{x}_q \in Q$,
  `foreach` $\underline{x}_t \in T$,
    $c = K(||\underline{x}_q - \underline{x}_t||)$, l_q += c, u_q += c.
  u_q -= N_T.
$\widehat{L}^l_{\mathcal{Q}}$ -= $N_Q log l_Q$, $\widehat{L}^u_{\mathcal{Q}}$ -= $N_Q log u_Q$.
$l_Q = \min_{q \in Q} l_q$, $u_Q = \max_{q \in Q} u_q - N_T$.
$\widehat{L}^l_{\mathcal{Q}}$ += $N_Q log l_Q$, $\widehat{L}^u_{\mathcal{Q}}$ += $N_Q log u_Q$.

Figure 3: Dual-tree algorithm, basic form. For simplification, the algorithm shown does not explicitly add the centroid mass to the estimate, but simply tightens the lower and upper bounds; at the end of the computation, the estimate $\hat{p}(\underline{x}_q)$ is based on the midpoint between l_q and u_q. In the pseudocode `continue` has the same meaning as in C, skipping all subsequent code and sending execution directly to the next `while` loop iteration; also `a += b` means `a = a + b`. P is the maximum possible value of the priority function. If a node is a leaf, its left or right child is defined to be itself.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited

5.4 Anytime pproximation property. Ensuring a global error guarantee requires that a previously approximated (pruned) node can have its mass contributions undone so that its children can be recursed upon for further accuracy if needed. This is achieved by storing an approximated node on $\mathcal{P}$ for possible future revisiting, with a penalty to its priority so that all unvisited nodes are chosen first. Importantly, this allows us to bring back the aggressive constant-mass pruning, since it can now coexist with the maintenance of either a global or point-wise guarantee. Unfortunately, this undoing capability slows the algorithm significantly once it has reached the point of expanding only previously-observed nodes, so that it is still more efficient to choose δ to be in the right ball park. The reversibility mechanism removes the absolute need to make this choice, however, if desired by the user. The algorithm is shown in this fully reversible form.

The algorithm monotonically tightens the global bounds with every node expansion (all undone tightenings are immediately replaced with tighter ones) and has no limit on the accuracy it can achieve, unlike previous approximation methods: it is a truly *anytime* algorithm. Unlike before, the user need not specify ϵ in advance, and can elect to instead stop the convergence when satisified.

6 Optimization of Upper and Lower Bounds

We can enhance performance by maximizing the tightness of the bounds, which allows approximation pruning as early in the search as possible. These techniques (not shown in the algorithm for simplicity of presentation) are not strictly necessary in order to realize the primary gain in efficiency yielded by the dual-tree structure of the algorithm.

6.1 Cross-scale information maximization: Up-down mass propagation. Each local bounds update due to a prune can be regarded as a new piece of information which is known only locally; information in the tree as a whole can be maximized by upward and downward propagation. Downward propagation recursively passes dl and du to the entire subtree below Q. Upward propagation can be done similarly by taking the min/max of the children's bounds. This technique can be seen as an instance of tree-based dynamic programming.

6.2 Deferred asynchronous propagation. To avoid the cost of full propagation down to the leaves upon every prune, we can employ a frugal asynchronous dynamic programming method: dl and du are passed only to Q's immediate children, in temporary holding slots for 'owed' mass. Whenever any node is considered during the search, it first checks these slots for any mass owed to it, integrates that mass into its bounds, and passes the mass to its immediate children's 'owed' slots. Propagation with this technique now has cost $O(1)$ rather than scaling with the number of nodes in the tree.

7 Empirical Study

We now empirically compare and evaluate the naive, single-tree, and dual-tree multivariate density estimation schemes.

7.1 Experimental methodology. We measure seconds of actual runtime on a modern Pentium-Pro Linux desktop workstation with 2Gb of RAM. Asterisks denote times estimated from smaller problem sizes using the known algorithm complexity. All density estimates were performed at the optimal bandwidth h^* as found by likelihood cross-validation [14], chosen over the set $\{0.25, 0.5, 0.75, 1\} \times 10^i$ for $-5 \leq i \leq 1$. The greatest computation requirement almost always occurred at the optimal bandwidth, as evidenced by the third table (in fact we conjecture that this may be a provable property of the algorithm), and so represents a worst case in terms of bandwidth. In all experiments a leave-one-out computation is measured, so the training set and test set have the same size N. The approximation parameter is set in all cases so that the maximum possible error in the overall log-likelihood was no more than 10^{-3}, *i.e.* one one-hundredth of one percent away from the true value, and in most cases was no more than 10^{-6}. The kernel function used is the Epanechnikov kernel, which has optimal efficiency among all kernel functions. In all of the experiments only ball-trees are used.

7.2 Datasets. Most experiments are on a segment of the Sloan Digital Sky Survey—a dataset of current scientific interest, and the active subject of ongoing nonparametric density estimation studies. It contains spatial coordinates in the first two dimensions and includes an additional 30 color attributes from various instruments. The other datasets are a 2-dimensional astrophysical point-spread function parameter dataset, a 5-dimensional biological screening dataset, the 38-dimensional Covtype dataset from the KDD repository, and the MNIST 768-dimensional noisy character image database.

7.3 Scaling with Dataset Size. The single-tree algorithm scales as $O(N \log N)$. We believe that the dual-tree algorithm scales as $O(N)$, as supported by empirical observation, but the proof will appear in a future publication. Both improve greatly upon the $O(N^2)$ scaling of the naive exhaustive method. Tree construction scales as $O(N \log N)$. Note that tree construction needs to be performed only once for the life of a dataset, and is thereafter amortized over all density estimation operations done using it.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited

Data = SDSS, $D = 2$				
N	h^*	Naive Time	Single Tree Time	Dual Tree Time
12.5K	.0025	7	.45	.12
25K	.0025	31	1.4	.31
50K	.001	123	2.1	.46
100K	.00075	494	5	1.0
200K	.0005	1976*	10	2
400K	.0005	7904*	27	5
800K	.00025	31616*	49	10
1600K	.00025	126465*	127	23

Table 1: Scaling with dataset size: numerical values

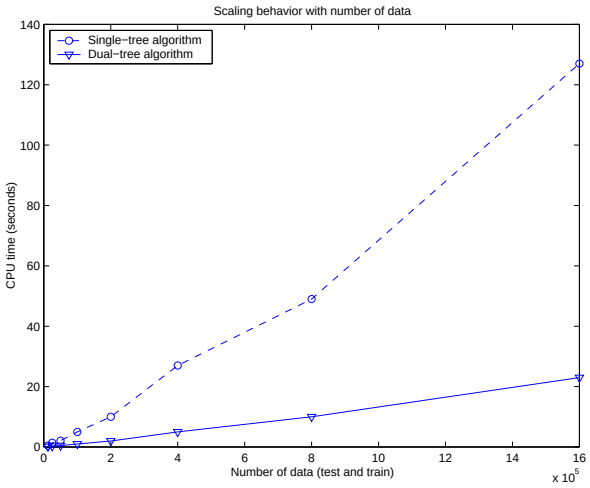

Figure 4: Scaling with dataset size: plot of values

7.4 Effect of kernel function. The infinite-extent Gaussian kernel yields approximately double the cost of the other two, finite-extent kernels.

Data = SDSS, $D = 2$				
N	Tree Build Time	Dual Tree Spher.	Dual Tree Epan.	Dual Tree Gauss.
12.5K	.3	.11	.12	.32
25K	.6	.31	.31	.70
50K	1	.45	.46	1.1
100K	3	1.0	1.0	2
200K	6	2	2	5
400K	15	5	5	11
800K	33	10	10	22
1600K	70	23	23	51

Table 2: Effect of kernel function: numerical values

7.5 Effect of bandwidth. As noted earlier, estimation at bandwidths larger or smaller than the optimal bandwidth is typically much less expensive.

SDSS, $D = 2$ $N = 1.6M$	
h	Dual Tree
$.001h^*$	9
$.01h^*$	9
$.1h^*$	10
h^*	23
$10h^*$	18
$100h^*$	2
$1000h^*$	1

Table 3: Effect of bandwidth: numerical values

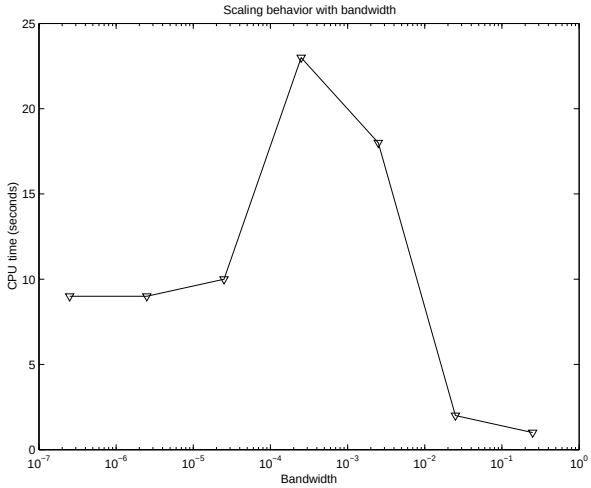

Figure 5: Effect of bandwidth: plot of values

7.6 Effect of approximation. Studying the effect of approximation for the Gaussian kernel (the finite-extent kernels yield near-exact estimates almost irrespective of the approximation level), we see a dramatic drop in runtime as the ϵ tolerance is increased. Note that the maximum possible error bounds provided by the algorithm are typically about two orders of magnitude looser than the actual error in log-likelihood.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited

Data = SDSS, $N = 1.6M$ Kernel = Gaussian		
ϵ	Max. Poss. Error	Dual Tree Time
.1	0.00084595	51
1	0.0149808	41
10	0.0753523	32
100	0.155863	21

Table 4: Effect of approximation: numerical values

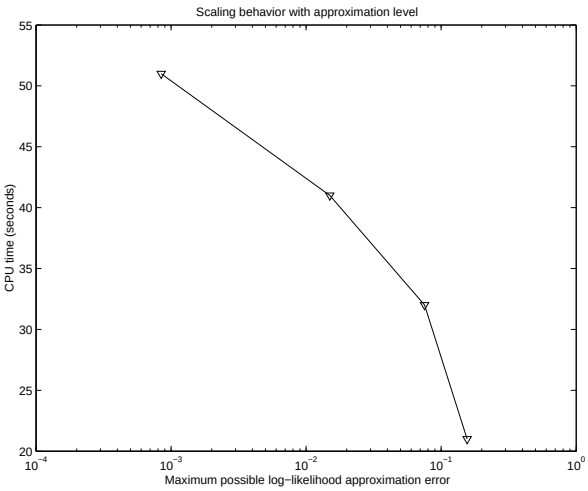

Figure 6: Effect of approximation: plot of values

7.7 Effect of dimensionality. As we add more of the dimensions of the SDSS dataset, the space becomes more complex and datasets are harder for the trees to localize. Note, however, that the growth is polynomial rather than exponential in D.

Data = SDSS, $N = 100,000$			
D	h^*	Naive Time	Dual Tree Time
2	.00075	494	1
3	.0075	543	6
4	.025	579	18
8	.05	945	41
16	.025	1424	43
32	.1	3326	57

Table 5: Effect of dimensionality: numerical values

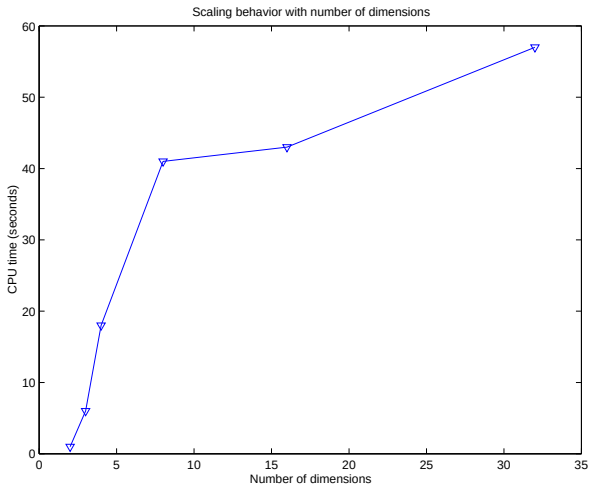

Figure 7: Effect of dimensionality: plot of values

7.8 Other kinds of data. To exhibit the algorithm's behavior under varying settings, we explore a sampling of datasets generated by various different kinds of processes.

Other Datasets				
Dataset	N	D	h^*	Dual Tree Time
BIO5	103,016	5	10^{-2}	10
CovType	136,081	38	10^{-4}	8
MNIST	10,000	784	10^{-5}	24
PSF2d	3,056,092	2	10^{-3}	9

Table 6: Other types of data: numerical values

8 Past, Present, and Future

8.1 Previous related algorithms based on space-partitioning trees. Both tree-based methods presented represent new extensions along a continuing line of research in statistical algorithms [4, 17, 16, 11]. For the closely-related problem of kernel regression, a single-*kd*-tree-based method was presented in [4]. We presented an early bare-bones dual-tree method for the KDE problem in [11], which used the exclusion/inclusion idea and an *ad hoc* approximation method. Here we greatly extend that algorithm with several powerful techniques (ball-trees, guaranteed pointwise precision, local and global priority search, anytime reversible approximation, dynamic programming, asynchronous propagation).

8.2 Multi-recursion and generalized N-body problems. The dual-tree method is additionally an instance of a new algorithmic design principle we refer to as higher-order divide-and-conquer, or multi-recursion [12], which is appropriate

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited

for a wide range of problems which includes what we call generalized N-body problems: those involving distances or potentials between points in a multi-d space. The famous Greengard's algorithm [13] for gravitational and electostatic simulation can be seen as a special case of multi-recursive algorithm specialized for the setting of Coulombic potentials, utilizing the multipole expansion as its approximation workhorse. The same can be said of the well-separated pair decomposition [3], which was designed for computational geometry problems. Though related in this broad sense to the dual-tree algorithm presented here, these methods are not algorithms for solving the KDE problem, and no reasonable adaptation of them for KDE would have the necessary properties for the general KDE problem. Comparison of these approaches with algorithms like the dual-tree algorithm presented in this paper is far afield of the topic of this paper but can be found in [12].

8.3 Making nonparametric techniques feasible. We believe there exist no other KDE algorithms which approach the efficiency (with accuracy) we have demonstrated in terms of both dataset size and dimensionality. Note that in the finite-kernel case our algorithms are exact, and in the infinite-kernel case they provide hard lower and upper bounds on their approximation error.

We hope that this new capability to perform kernel density estimation tractably, including solutions for many of the practicalities of the task (large test and query sets, possibly large dimension, parameter-less operation, hard error bounds), will open new frontiers for the use of nonparametric density estimation in general, as well as inspire algorithms for other nonparametric estimators. The code will be available for download at `http://www.cs.cmu.edu/~agray`. More generally, the we will pursue the application of multi-recursive methods to other major classes of nonparametric statistics, including nonparametric hypothesis-testing and nonparametric classification (*e.g.* SVM's), as well as common semi-parametric models such as mixtures of Gaussians.

References

[1] C. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Oxford, 1995.

[2] A. Bowman. A Comparative Study of Some Kernel-Based Nonparametric Density Estimators. *Journal of Statistical Computation and Simulation*, 21:313–327, 1985.

[3] P. Callahan and S. Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *Journal of the ACM*, 62(1):67–90, January 1995.

[4] K. Deng and A. W. Moore. Multiresolution Instance-based Learning. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pages 1233–1239, San Francisco, 1995. Morgan Kaufmann.

[5] L. Devroye and L. Gyorfi. *Nonparametric Density Estimation: The L_1 View*. Wiley, 1985.

[6] L. Devroye and G. Lugosi. *Combinatorial Methods in Density Estimation*. Springer-Verlag, 2001.

[7] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. John Wiley & Sons, 1973.

[8] V. A. Epanechnikov. Nonparametric Estimation of a Multidimensional Probability Density. *Theory of Probability and its Applications*, 14:153–158, 1969.

[9] J. Fan and J. Marron. Fast Implementations of Nonparametric Curve Estimators. *Journal of Computational and Graphical Statistics*, 3:35–56, 1994.

[10] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3(3):209–226, September 1977.

[11] A. Gray and A. W. Moore. N-Body Problems in Statistical Learning. In T. K. Leen, T. G. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems 13 (December 2000)*. MIT Press, 2001.

[12] A. G. Gray. *Higher-Order Divide-and-Conquer and Generalized N-Body Problems*. PhD. Thesis, Carnegie Mellon University, Computer Science Department, expected 2003.

[13] L. Greengard and V. Rokhlin. A Fast Algorithm for Particle Simulations. *Journal of Computational Physics*, 73, 1987.

[14] J. D. F. Habbema, J. Hermans, and K. van der Broek. A Stepwise Discrimination Program Using Density Estimation. In G. Bruckman, editor, *Computational Statistics*, pages 100–110. Physica Verlag, 1974.

[15] M. Jones, J. Marron, and S. Sheather. A Brief Survey of Bandwidth Selection for Density Estimation. *Journal of the American Statistical Association*, 91:401–407, 1996.

[16] A. W. Moore. The Anchors Hierarchy: Using the Triangle Inequality to Survive High-Dimensional Data. In *Twelfth Conference on Uncertainty in Artificial Intelligence*. AAAI Press, 2000.

[17] A. W. Moore and M. S. Lee. Cached Sufficient Statistics for Efficient Machine Learning with Large Datasets. *Journal of Artificial Intelligence Research*, 8, March 1998.

[18] F. P. Preparata and M. Shamos. *Computational Geometry*. Springer-Verlag, 1985.

[19] B. P. Rao. *Nonparametric Functional Estimation*. Academic Press, 1983.

[20] D. W. Scott. *Multivariate Density Estimation*. Wiley, 1992.

[21] B. Silverman. Kernel Density Estimation using the Fast Fourier Transform. *Journal of the Royal Statistical Society Series C: Applied Statistics*, 33, 1982.

[22] B. W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman and Hall/CRC, 1986.

[23] M. P. Wand. Fast Computation of Multivariate Kernel Estimators. *Journal of Computational and Graphical Statistics*, 1994.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited