Empirical Testing of Fast Kernel Density Estimation Algorithms

Dustin Lang **Mike Klaas** **Nando de Freitas**
{ dalang, klaas, nando }@cs.ubc.ca
Dept. of Computer Science
University of British Columbia
Vancouver, BC, Canada V6T1Z4

Abstract

We present results of experiments testing the Fast Gauss Transform, Improved Fast Gauss Transform, and Dual-Tree methods (using *kd*-tree and Anchors Hierarchy data structures) for fast Kernel Density Estimation (KDE). We examine the performance of these methods with respect to data set size, dimension, allowable error, and data set structure ("clumpiness"), measured in terms of CPU time and memory usage. This is the first multi-method comparison in the literature. The results are striking, challenging several claims that are commonly made about these methods. The results are useful for researchers considering fast methods for KDE problems.

Along the way, we provide a corrected error bound and a parameter-selection regime for the IFGT algorithm.

1 INTRODUCTION

This paper examines several methods for speeding up KDE. This problem arises as a basic operation in many statistical settings, including message passing in belief propagation, particle smoothing, population Monte Carlo, spectral clustering, SVMs and Gaussian processes.

The KDE problem is to compute the values

$$f_j = \sum_{i=1}^{N} w_i \; K_i\left(x_i, y_j\right) \quad , \quad j = 1 \ldots M \quad . \qquad (1)$$

The points $X = \{x_i\}\,,\ i = 1 \ldots N$ each have a *weight* w_i. We call these *source particles.* We call the points $Y = \{y_j\}\,,\ j = 1 \ldots M$ the *target points.* We call f_j the *influence* at point y_j, and K is the *kernel function.* For simplicity, we consider a subset of the KDE problem in this paper. Some of these simplifying assumptions can be easily relaxed for some of the fast methods, but we assume that these will not change the essential properties that we examine. Specifically, we assume:

$$\begin{aligned} K_i\left(x_i, y_j\right) &= \exp\left(-\frac{\|x_i - y_j\|^2}{h^2}\right) \\ w_i &\geq 0 \\ M &= N \quad . \end{aligned}$$

That is, we consider the points to live in a vector space, and use a Gaussian kernel, non-negative weights, and equal numbers of sources and targets.

To be useful to a broad audience, we feel that fast KDE methods must allow the user to set a guaranteed absolute error level. That is, given an allowable error ϵ, the method must return approximations $\hat{f}_j$ such that

$$\left|\hat{f}_j - f_j\right| \leq \epsilon \quad .$$

Ideally, the methods should work well even with small ϵ, so that one can simply "plug it in and forget about it."

2 FAST METHODS

In this section, we briefly summarize each of the methods that we test.

2.1 Fast Gauss Transform

The Fast Gauss Transform (FGT) was introduced by Greengard and Strain [5, 6, 2]. It is a specialization of the Fast Multipole Method to the particular properties of the Gaussian kernel. The algorithm begins by dividing the space into boxes and assigning sources and targets to boxes. For the source boxes, Hermite series expansion coefficients are computed. Since the Gaussian falls off quickly, only a limited number of source

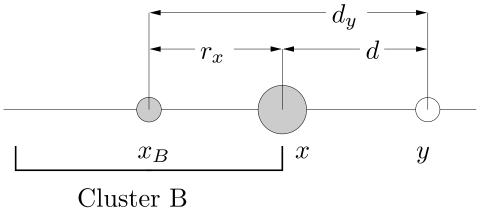

Figure 1: *An illustration of the IFGT error bound due to ignoring clusters outside radius r_y. The cluster center is x_B and its radius is r_x; the cluster is ignored if the distance d_y from the center to y is greater than r_y. In the worst case, a particle x with large weight sits on the cluster boundary nearest to y. The distance d_y is slightly larger than r_y, so the cluster is ignored. The distance between y and x can be as small as $d = r_y - r_x$,* not r_y.

boxes will have non-negligible influence upon a given target box. These neighbouring boxes are collected, and the Hermite expansions are combined to form a Taylor expansion that is valid inside the target box.

The FGT implementation we test was graciously provided by Firas Hamze, and is written in C with a MATLAB wrapper. It uses a fairly simple space-subdivision scheme, so has memory requirements of $O\left(\left(\frac{1}{h}\right)^3\right)$. Adaptive space subdivision or sparse memory allocation could perhaps improve the performance, particularly for small-bandwidth problems.

2.2 Improved Fast Gauss Transform

The Improved Fast Gauss Transform (IFGT) [9, 10] aims to improve upon the FGT by using a space partitioning scheme and series expansion that scale better with dimension.

The space-partitioning scheme is the farthest-point K-centers algorithm, which performs a simple clustering of the source particles into K clusters. As in the FGT, the series expansion coefficients for the source points in each cluster can be summed.

The IFGT does not build a space-partitioning tree for the target points. Rather, for each target point it finds the set of source clusters that are within range, and evaluates the series expansion for each.

The implementation of the IFGT that we test was generously provided by Changjiang Yang and Ramani Duraiswami. It is written in C++ with MATLAB bindings.

2.2.1 IFGT Error Bounds

The original IFGT papers contain an incorrect error bound, which we correct here.

The error bound given is equation 3.13 in [9]:

$$E_C \;\le\; W \exp\left(-\frac{{r_y}^2}{h^2}\right) \quad ,$$

where W is the sum of source weights. This is the error that results from ignoring the influence of source clusters that are outside of range r_y from a target point. That is, if the center of cluster B is x_B, then the influence of the cluster on target point y is ignored if the distance $\|x_B - y_j\| > r_y$. This bound is incorrect, as illustrated in Figure 1. The correct bound is

$$E_C \;\le\; W \exp\left(-\frac{(r_y - r_x)\,2}{h^2}\right) \quad .$$

2.2.2 Choosing IFGT Parameters

The IFGT has several parameters that must be chosen, yet the original papers ([10, 9]) do not suggest a method for choosing these parameters. We developed a protocol for automatically choosing parameters that will satisfy a given error bound ϵ, without seriously degrading the computational complexity of the algorithm. We make no claim that this protocol is optimal.

The parameters that must be chosen include K, the number of source clusters; p, the number of terms in the series expansion; and r_y, the range. As K increases, the radius of the largest source cluster, r_x, decreases.

The IFGT has two sources of error. The first is discussed above, and is due to ignoring clusters that are outside range r_y from a given target point. The second is due to truncation of the series expansion after order p. This is called E_T in equation 3.12 of [9]:

$$E_T \;\le\; W \exp\left(\frac{2\; r_x r_y - {r_x}^2 - {r_y}^2}{h^2}\right) \frac{2^p}{p!} \left(\frac{r_x r_y}{h^2}\right)^p$$

where r_x is the maximum source cluster radius, r_y is the range, and p is the order of the series expansion.

See Figure 2 for the protocol. It is based on four constraints C:

$$\begin{array}{lrcl} C_1: & E_C & \le & \epsilon \\ C_2: & E_T & \le & \epsilon \\ C_3: & K & \le & K^* \\ C_4: & \left(\frac{r_x r_y}{h^2}\right) & \le & 1 \end{array}$$

where C_1 and C_2 are hard constraints that guarantee the error bound, C_3 is a hard constraint that guarantees the algorithmic complexity, and C_4 is a soft constraint that improves convergence. Note that each source cluster contributes to the error either through series expansion (E_T) or by being ignored (E_C), but not both, so it suffices to require $E_C \le \epsilon$ and $E_T \le \epsilon$.

```
INPUT: $r_y$(ideal), $\epsilon$
OUTPUT: $k$, $r_y$, $p$
ALGORITHM:
  for $k = 1$ to $K^*$:
    run $k$-centers algorithm.
    find largest cluster radius $r_x$.
    using $r_y = r_y$(ideal), compute $C_1$, $C_4$.
    if $C_1$ AND $C_4$:
      break
  if $k < K^*$:
    // $C_4$ can be satisfied.
    set $r_y = \min(r_y)$ such that $C_1$ AND $C_4$.
  else:
    // $C_4$ cannot be satisfied.
    set $r_y = \min(r_y)$ such that $C_1$.
  set $p = \min(p)$ such that $C_2$.
```

Figure 2: *Protocol for choosing IFGT parameters. We first try to find $k < K^*$ that will allow all the constraints to be satisfied with the given r_y(ideal). In many cases, this is not possible so we must set $k = K^*$ and increase r_y. Finally, we choose p to satisfy the truncation error bound C_2. In practice, the k-centers algorithm is run iteratively rather than being run anew each time through the loop.*

Note that the IFGT, unlike the FGT, does not cluster the target points y; the distance from each target to each source cluster is computed. The algorithm therefore has $O(KM)$ complexity, where K is the number of source clusters and M is the number of targets. To keep $O(M)$ complexity, K must be bounded above by a constant, K^*. This is constraint C_3. Note that r_x (the maximum cluster radius) decreases as K (the number of clusters) increases. Since K has an upper bound, r_x has a lower bound. Contrary to the claim in [9], r_x cannot be made as small as required by increasing K while still maintaining $O(M)$ complexity.

2.3 DUAL-TREE

We developed an implementation of the dual-tree recursion described by Gray and Moore [4, 3]. We make several implementation-level changes, as detailed in [7].

The dual-tree strategy is based on building space-partitioning trees for both the source and target points. The algorithm proceeds by expanding the ‘cross product’ of the trees in such a way that only areas of the trees that contain useful information are explored. With these trees, it is inexpensive to compute distance bounds between nodes, which allows us to bound the influence of a source node upon a target node. If the influence bound is too loose, it can be tightened by *expanding* the nodes; that is, by replacing the parent nodes with the sum of their children. When the influence bounds have been tightened sufficiently (below ϵ), we are finished.

A major difference between the dual-tree strategy and the series expansion-based methods (FGT and IFGT) is that the expansion-based methods guarantee error bounds based on theoretical bounds on the series truncation error. These bounds are computed *a priori*, and are valid for all data distributions. However, since they are based on worst-case behaviour, they are often quite loose for average-case problems. Conversely, the dual-tree strategy is based solely on error bounds determined at run time, so is fundamentally concerned with a particular data set.

Our implementation of the dual-tree strategy is independent of the space-partitioning strategy. We implemented the classic *kd*-tree and the Anchors Hierarchy [8]. It is written in C with MATLAB bindings.

3 RESULTS

All tests were run on our Xeon 2.4 GHz, 1 GB memory, compute servers. We ran the tests within MATLAB; all the fast algorithms are written in C or C++ and have MATLAB bindings. We stopped testing a method once its memory requirements rose above 1 GB in order to avoid swapping. In all cases we repeated the tests with several data sets. In some of the plots the error bars are omitted for clarity. The error bars are typically very small. Most of the plots have log-log scales.

For the IFGT, we set the upper bound on the number of clusters to be $K^* = \sqrt{N}$. In practice, K^* should be set to a constant, but since we are testing over several orders of magnitude this seems more reasonable.

The curves labelled “KDtree” and “Anchors” are our dual-tree implementation using *kd*-tree and Anchors Hierarchy space-partitioning trees. “Naive” is the straightforward $O\left(N^2\right)$ summation.

Many of the tests below can be seen as one-dimensional probes in parameter space about the point $N = 10,000$, Gaussian bandwidth $h = 0.01$, dimension $D = 3$, allowable error $\epsilon = 10^{-6}$, clumpiness $C = 1$ (ie, uniform) point distribution, with weights drawn uniformly from $[0, 1]$. In all cases the points are confined to the unit D-cube. We occasionally choose other parameters in order to illustrate a particular point. We use $D = 3$ to allow the FGT to be tested.

3.1 Test A: N

Researchers have focused attention on the performance of fast algorithms with respect to N (the number of source and target points). Figure 3 shows that it is crucially important to consider other factors, since these strongly influence the empirical performance.

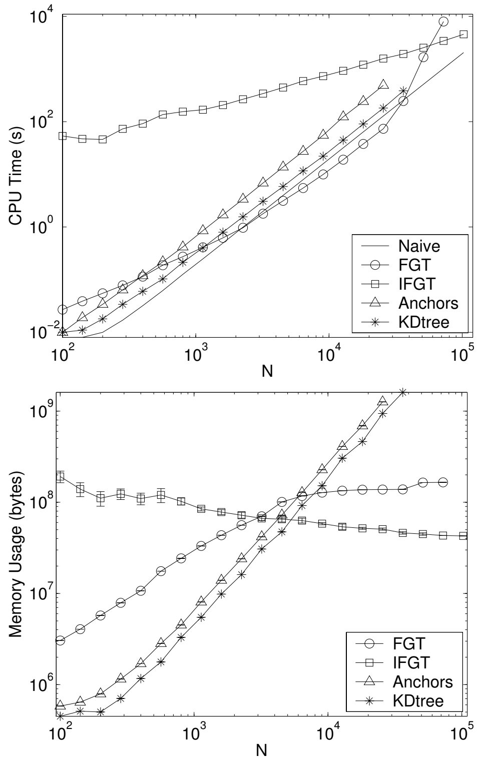

Figure 3: *Test A: $D = 3$, $h = 0.1$, uniform data, $\epsilon = 10^{-6}$.*

In this test, the scale of the Gaussians is $h = 0.1$, so a large proportion of the space has a significant contribution to the total influence. An important observation in Figure 3 is that the dual-tree methods (KDtree and Anchors) are doing about $O\left(N^2\right)$ work. Empirically, they are never faster than Naive for this problem. Indeed, only the FGT is ever faster, and then only for a small range of N. The IFGT appears to be demonstrating better asymptotic performance, but the crossover point (if the trend continues) occurs at about 1.5 hours of compute time.

Another important thing to note in Figure 3 is that the dual-tree methods run out of memory before reaching $N = 50,000$ points; this happens after a modest amount of compute time. Also of interest is the fact that the IFGT has *decreasing* memory requirements. We presume this is because the number of clusters increases, so the cluster radii decrease and the error bounds converge toward zero more quickly, meaning that fewer expansion terms are required.

3.2 Test B: N

In this test, we repeat test A but use a smaller Gaussian scale parameter, $h = 0.01$. The behaviour of the algorithms is strikingly different. We can no longer run the IFGT, since the number of expansion terms required is more than 10^{10} for $N = 100$. The dual-tree methods perform well, though memory usage is still a concern.

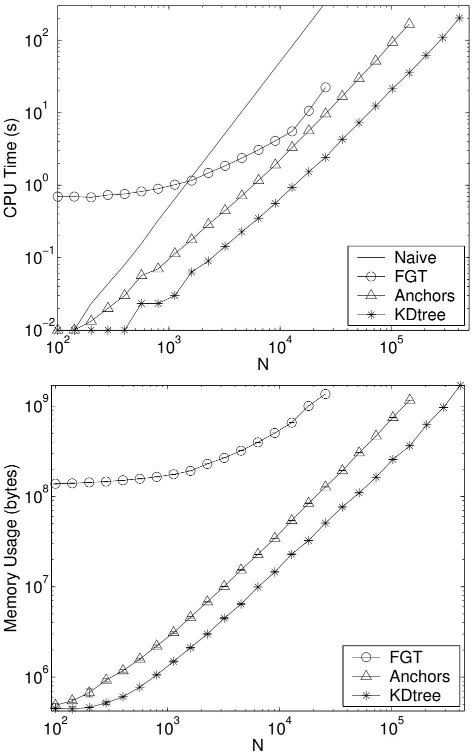

Figure 4: *Test B: $D = 3$, $h = 0.01$, uniform data, $\epsilon = 10^{-6}$.*

3.3 Test C: Dimension D

In this test, we fix $N = 10,000$ and vary the dimension. We set $\epsilon = 10^{-3}$ to allow the IFGT to run in a reasonable amount of time. Surprisingly, the IFGT and Anchors, both of which are supposed to work well in high dimension, do not perform particularly well. The IFGT's computational requirements become infeasibly large above $D = 2$, while Anchors never does better than KDtree. This continues to be true even when we subtract the time required to build the Anchors Hierarchy.

3.4 Test D: Allowable Error ϵ

In this test, we examine the cost of decreasing ϵ. The dual-tree methods have slowly-increasing costs as the accuracy is increased. The FGT has a more quickly increasing cost, but for this problem it is still competitive at $\epsilon = 10^{-11}$.

We find that the dual-tree methods begin to have problems when $\epsilon < 10^{-11}$; while these methods can give arbitrarily accurate approximations given exact arithmetic, in practice they are prone to cancellation error.

The bottom plot in Figure 6 shows the maximum error

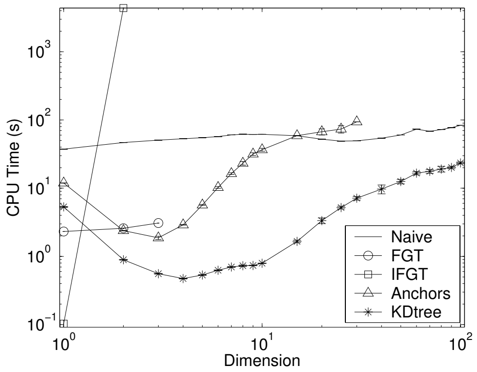

Figure 5: *Test C:* $h = 0.01$, $\epsilon = 10^{-3}$, $N = 10,000$, *uniform data.*

in the estimates. The dual-tree methods produce results whose maximum errors are almost exactly equal to the error tolerance ϵ. One way of interpreting this is that these methods do as little work as possible to produce an estimate that satisfies the required bounds. The FGT, on the other hand, produces results that have real error well below the requirements. Notice the 'steps'; we believe these occur as the algorithm either adds terms to the series expansion, or chooses to increase the number of boxes that are considered to be within range.

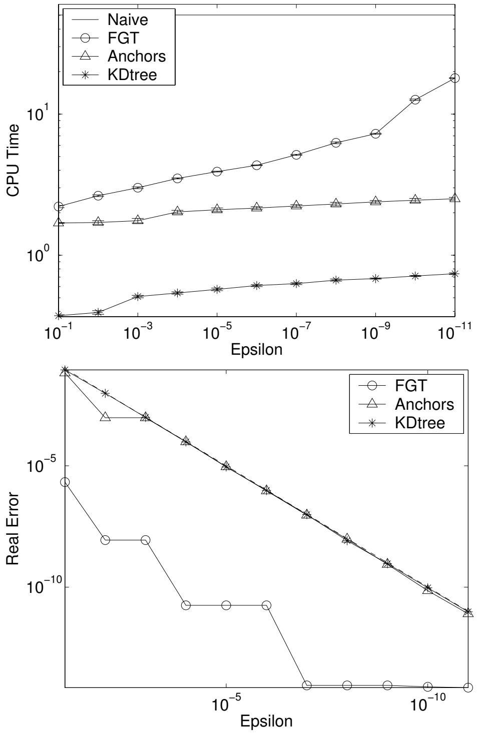

Figure 6: *Test D:* $D = 3$, $h = 0.01$, $N = 10,000$, *uniform data. Top: CPU Time. Bottom: Real error.*

3.5 DATA SET CLUMPINESS

Next, we explore the behaviour of the fast methods on data sets drawn from non-uniform distributions.

We use a method for generating clumpy data that draws on the concept of lacunarity. Lacunarity [1] measures the texture or 'difference from uniformity' of a set of points, and is distinct from fractal dimension. It is a scale-dependent quantity that measures the width of the distribution of point density. Lacunarity at a given scale can be measured by covering the set with boxes of that scale; the distribution of point densities in the boxes is measured, and the lacunarity is defined as the second moment of the distribution divided by the first moment squared.

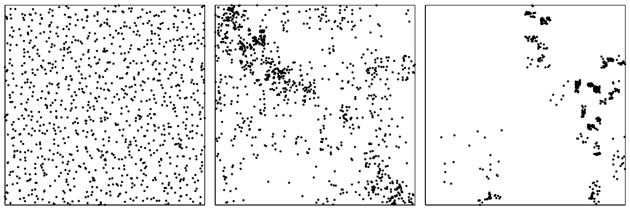

Figure 7: *Example clumpy data sets. The clumpinesses are* $C = 1$ *(left),* $C = 1.5$ *(middle), and* $C = 3$ *(right). Each data set contains* 1000 *points.*

We adapt the notion of the ratio of variance to squared mean. Given a number of samples N and a clumpiness C, our clumpy data generator recursively divides the space into 2^D sub-boxes, and distributes the N samples among the sub-boxes such than

$$\sum_{i=1}^{2^D} N_i = N$$

$$\mathrm{var}\left(\{N_i\}\right) = (C-1)\,\mathrm{mean}\big(\{N_i\}\big)^2 \quad .$$

This process continues until N is below some threshold (we use 10). Some example clumpy data sets are shown in Figure 7.

3.6 Test E: Source Clumpiness

In this test, we draw the source particles X from a clumpy distribution, while the targets are drawn from a uniform distribution.

Figure 8 shows the relative CPU time as clumpiness increases. The dual-tree methods show significant improvements as the source points become clumpy. Anchors improves more than KDtree, although KDtree is still faster in absolute terms. The FGT shows minor improvement as clumpiness increases.

3.7 Test F: Source and Target Clumpiness

In this test, we draw both the sources and targets from clumpy distributions. The dual-tree methods

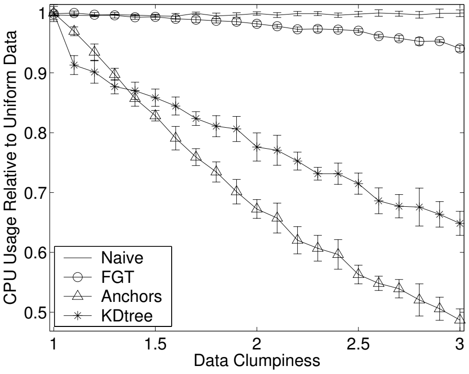

Figure 8: *Test E:* $D = 3$*,* $h = 0.01$*,* $\epsilon = 10^{-6}$*,* $N = 10,000$*, clumpy* X*, uniform* Y*, relative CPU time.*

show even more marked improvement as clumpiness increases. The FGT also shows greater improvement than in the previous test.

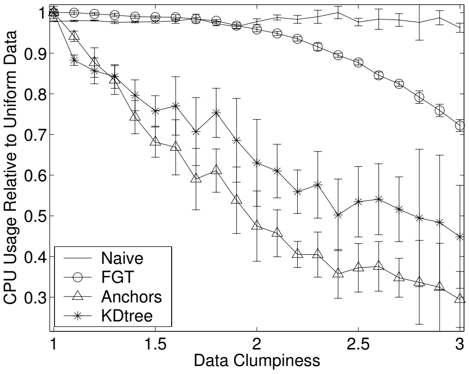

Figure 9: *Test F results:* $D = 3$*,* $h = 0.01$*,* $\epsilon = 10^{-6}$*,* $N = 10,000$*, clumpy* X*, clumpy* Y*; relative CPU time.*

3.8 Test G: Clumpy Data, Dimension D

In this test, we test the performance with dimension, given clumpy data sets. The results are not very different than the uniform case (Test C). This is surprising, since neither Anchors nor IFGT does particularly well, even given clumpy data.

4 CONCLUSIONS

We presented the first comparison between the most widely used fast methods for KDE. In our comparison, we varied not only the number of interacting points N, but also the structure in the data, the required precision and the dimension of the state space. The results indicate that the fast methods can only work well when there is structure in the kernel matrix. They also indicate that dual tree methods are preferable in high dimensions. Surprisingly, the results show that the KDtree works better than the Anchors Hierarchy in our particular experiments. This seems to contradict common beliefs about these methods. Yet, there is a lack of methodological comparisons between these methods in the literature. This makes it clear that further investigation is warranted.

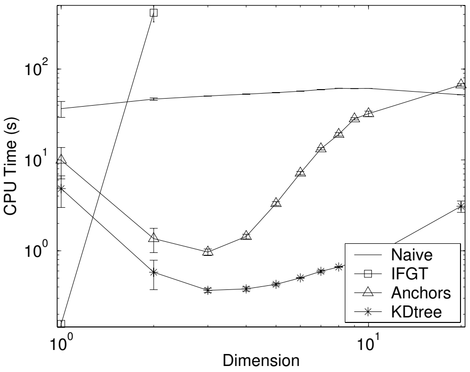

Figure 10: *Test G results:* $h = 0.01$*,* $\epsilon = 10^{-3}$*,* $N = 10,000$*, clumpy* X*, clumpy* Y*.*

References

[1] C Allain and M Cloitre. Characterizing the lacunarity of random and deterministic fractal sets. *Physical Review A*, 44(6):3552–3558, Sep 1991.

[2] B J C Baxter and G Roussos. A new error estimate of the fast Gauss transform. *SIAM Journal of Scientific Computing*, 24(1):257–259, 2002.

[3] A Gray and A Moore. Nonparametric density estimation: Toward computational tractability. In *SIAM International Conference on Data Mining*, 2003.

[4] A G Gray and A W Moore. 'N-Body' problems in statistical learning. In *NIPS 4*, pages 521–527, 2000.

[5] L Greengard and J Strain. The fast Gauss transform. *SIAM Journal of Scientific Statistical Computing*, 12(1):79–94, 1991.

[6] L Greengard and X Sun. A new version of the Fast gauss transform. *Documenta Mathematica*, ICM(3):575–584, 1998.

[7] D Lang. Fast methods for inference in graphical models. Master's thesis, University of British Columbia, 2004.

[8] A Moore. The Anchors Hierarchy: Using the triangle inequality to survive high dimensional data. Technical Report CMU-RI-TR-00-05, Carnegie Mellon University, February 2000.

[9] C Yang, R Duraiswami, and N A Gumerov. Improved fast gauss transform. Technical Report CS–TR–4495, University of Maryland, 2003.

[10] C Yang, R Duraiswami, N A Gumerov, and L S Davis. Improved fast Gauss transform and efficient kernel density estimation. In *ICCV*, Nice, 2003.