# Fast maximum *a posteriori* inference in Monte Carlo state spaces

**Mike Klaas** **Dustin Lang** **Nando de Freitas**

Computer Science Department
University of British Columbia
{klaas,dalang,nando}@cs.ubc.ca

## Abstract

Many important algorithms for statistical inference can be expressed as a weighted max-kernel search problem. This is the case with the Viterbi algorithm for HMMs, message construction in maximum *a posteriori* BP (max-BP), as well as certain particle-smoothing algorithms. Previous work has focused on reducing the cost of this procedure in discrete regular grids [4]. Monte-Carlo state spaces, which are vital for high-dimensional inference, cannot be handled by these techniques. We present a novel dual-tree based algorithm that is appliable to a wide range of kernels and shows substantial performance gains over naïve computation.

## Introduction

Max-kernel problems arise at the heart of many powerful and widely-used statistical inference algorithms. Examples include the message computation in max belief propagation, sequence recursion in the Viterbi algorithm, and classes of maximum *a posteriori* sequence estimation algorithms based on particle methods. This operation is expensive—requiring $O(N^2)$ operations, where $N$ is the size of the state space of a random variable. As a result, applications with large state spaces either must artificially coarsen the state space or simply choose to use less powerful inference techniques. Recent work by Felzenszwalb *et al.* addresses the computational burden when the state space can be embedded in a regular discrete grid [4, 5]. This technique, based on the *distance transform*, is extremely powerful in its domain, but has two major limitations:

- It is limited to kernels of the form $K(x, y) = \exp\left\{\frac{1}{\sigma^2}\|x - y\|\right\}$ or $\exp\left\{\frac{1}{\sigma^2}\|x - y\|^2\right\}$
- It is only applicable to state spaces embedded in a regular grid of parameters.

Monte Carlo methods, such as MCMC and particle filters, have been shown to effectively adapt to examine interesting regions of the state space, and can achieve better results than regular discretizations using fewer support points [1, 13]. Problems requiring high-dimensional inference are ubiquitous in machine learning, and are best attacked with Monte Carlo techniques as regular discretizations grow exponentially and quickly become intractable. In this paper, we address the need of fast algorithms for computing weighted max-kernel on Monte Carlo grids by demonstrating how the quadratic cost can be reduced to $N \log N$ by adopting and extending powerful algorithms proposed for $N$-body simulation [7, 8].[1]

In particular, we develop a new efficient dual-tree recursion to exactly solve the max-kernel problem. We derive the method in the context of kernels parameterized by a distance function,[2] which represent a broad class of frequently used kernel functions, including Gaussians, Epanechnikov, spherical, and linear kernels, as well as thresholded versions of the same. Our method can also be used to accelerate other spatial-based kernels (such as $K(x, y) = x \cdot y$), and problems that have multiple kernels over different regions of the state space, but we restrict our attention to the simpler and more common case in this paper.

Our empirical results show that our algorithm provides a speedup of several orders of magnitude over the naïve method, becoming more efficient after as little as 10ms of compute time. The dual-tree algorithm still compares favorably to naïve computation on discrete grids where the distance transform can be applied, but we

[1] We note that there are techniques for dealing with KDE on Monte Carlo grids (fast Gauss Transform), but these are inapplicable in the max-kernel setting.

[2] By *distance functions* we mean functions that are similar to a metric but need not obey the triangle inequality.

find that the latter algorithm is superior in this case.

The performance of algorithms based on dual-tree recursion as $N$ grows is relatively well-understood; see Gray and Moore [8] and Ihler [9]. However, we have found that the performance of this family of techniques also depends heavily on other variables, such as the data distribution, the dimensionality of the problem, and the choice of spatial index and kernel. We present several experiments to investigate these effects, and we believe that the conclusions can be generalized to other pruning-based dual-tree algorithms.

# 1 Problem setting

The algorithms we discuss in this paper are designed to solve the following problem: We are given points (which we will call *particles*) $X \triangleq \{x_j\}$ and $Y \triangleq \{y_i\}$, and weights $\{w_j\}$ corresponding to the $X$ particles. The source ($X$) particles exert an *influence* on the target ($Y$) particles given by $\text{infl}(x_j, y_i) = w_j K(x_j, y_i)$, where $K(\cdot)$ is an affinity kernel. We wish to compute, for each $y$, the maximum influence attained and the $x$ particle corresponding to it,[3] ie.

$$f_i = \max_{j=1}^{N} w_j K(y_i, x_j) \quad i = 1, 2, \ldots, M \tag{1}$$

This procedure's $O(MN)$ cost dominates the runtime of many important algorithms such as max-BP and MAP sequence estimation, which limits their use to settings of small order (correponding to a coarse discretization of a continuous state space or choosing a small number of particles).

In the following section, we detail how the max-kernel algorithm arises in common inference methods.

## 1.1 Maximum *a posteriori* belief propagation

Given a graphical model with latent variables $\mathbf{u}_{1:n}$[4], observations $\mathbf{z}_{1:n}$, and potentials $\psi_{kl}$, $\phi_k$, a joint probability distribution is admitted:

$$p(\mathbf{u}_{1:n}, \mathbf{z}_{1:n}) = \frac{1}{Z} \prod_{k,l} \psi_{kl}(\mathbf{u}_k, \mathbf{u}_l) \prod_k \phi_k(\mathbf{u}_k, \mathbf{z}_k)$$

We are interested in computing the maximum *a posteriori* estimate, $\mathbf{u}_{1:n}^{MAP} = \arg\max_{\mathbf{u}_{1:n}} p(\mathbf{u}_{1:n}|\mathbf{z}_{1:n})$. We can use the standard max-product belief propagation equations for message passing and marginal (belief) computation [12]. The message from node $l$ to node $k$ is given by:

$$m_{lk}(u_{ki}) = \max_{j=1}^{|\mathbf{u}_l|} \phi(u_{lj}, \mathbf{z}_l)\psi(u_{ki}, u_{lj}) \prod_{r \in \mathcal{N}(l)-k} m_{rl}(u_{lj}) \tag{2}$$

where $\mathcal{N}(l)-k$ denotes the neighbours of node $l$ excluding $k$. We can re-write equation (2) as a max-kernel problem by setting

$$\begin{aligned} \{x_j\} &= \{u_{lj}\}, \\ \{y_i\} &= \{u_{ki}\}, \\ w_j &= \phi(u_{lj}, \mathbf{z}_l) \prod_{r \in \mathcal{N}(l)-k} m_{rl}(u_{lj}), \\ K(y_i, x_j) &= \psi(u_{ki}, u_{lj}) \end{aligned}$$

## 1.2 MAP sequence estimation using particle methods

Consider the Markovian time-series model with latent state $\mathbf{u}_t$ and observations $\mathbf{z}_t$ given by

$$\begin{aligned} \mathbf{u}_t &\sim p(\mathbf{u}_t|\mathbf{u}_{t-1}) \\ \mathbf{z}_t &\sim p(\mathbf{z}_t|\mathbf{u}_t) \end{aligned}$$

In standard particle filtering [3], we draw a set of samples $\left\{u_{1:n}^{(i)}\right\}_{i=1}^{N}$ using sequential importance sampling in order to approximate the filtering distribution with a Monte Carlo estimator $\widehat{p}(\mathbf{u}_n|\mathbf{z}_{1:n}) = \frac{1}{N}\sum_{i=1}^{N} \delta_{u_n^{(i)}}(d\mathbf{u}_n)$, where $\delta_{u_n^{(i)}}(d\mathbf{u}_n)$ denotes the delta Dirac function. This is typically done in a chain (or tree) with $n$ nodes, at a cost of $O(nN)$. However, our goal is to obtain an estimate of the maximum *a posteriori* sequence

$$\mathbf{u}_{1:n}^{MAP}(n) \triangleq \arg\max_{\mathbf{u}_{1:n}} p(\mathbf{u}_{1:n}|\mathbf{z}_{1:n}). \tag{3}$$

As introduced by Godsill *et al.* in [6], equation (3) can be estimated by performing a Viterbi-like algorithm on the Monte Carlo state space induced by the filtered particles at time $t$. At the heart of this algorithm lies the following recursion:

$$\begin{aligned} \delta_k(j) = &\log p\left(\mathbf{z}_k|u_k^{(j)}\right) \\ &+ \max_i \left[\delta_{k-1}(i) + \log p\left(u_k^{(j)}|u_k^{(i)}\right)\right] \end{aligned} \tag{4}$$

This must be computed for each particle, thus incurring a $O(N^2)$ cost. It is straightforward to show that the maximization in (4) is equivalent to the max-kernel problem in equation (1) transformed to log space.

[3] We will subsequently refer to this procedure as *weighted maximum-kernel*, or simply *max-kernel*.

[4] In describing these algorithms, we use $\mathbf{u}$ and $\mathbf{z}$ rather than the traditional $x$ and $y$ to highlight the distinction between the variables in the inference algorithms and the variables in the max-kernel computation.

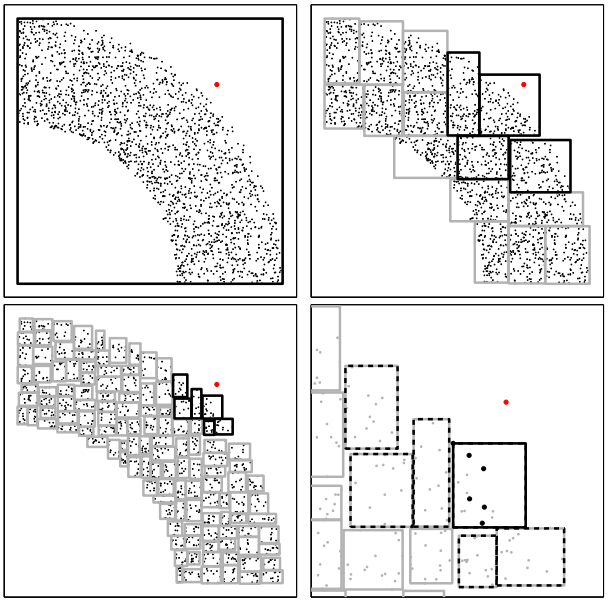

Figure 1: Example of pruning the max-kernel algorithm (for a single $y$ particle). The candidate (dark) and non-candidate (light) nodes are shown. In the bottom-right plot, a close-up of the six final candidate nodes is shown (dashed). The single box whose particles are examined is shown in black. The subset of the particles that are examined individually is shown in black. There were 2000 particles in $X$, of which six nodes (containing 94 particles total) were candidate leaf nodes. Of these, only six particles from the first node were examined individually.

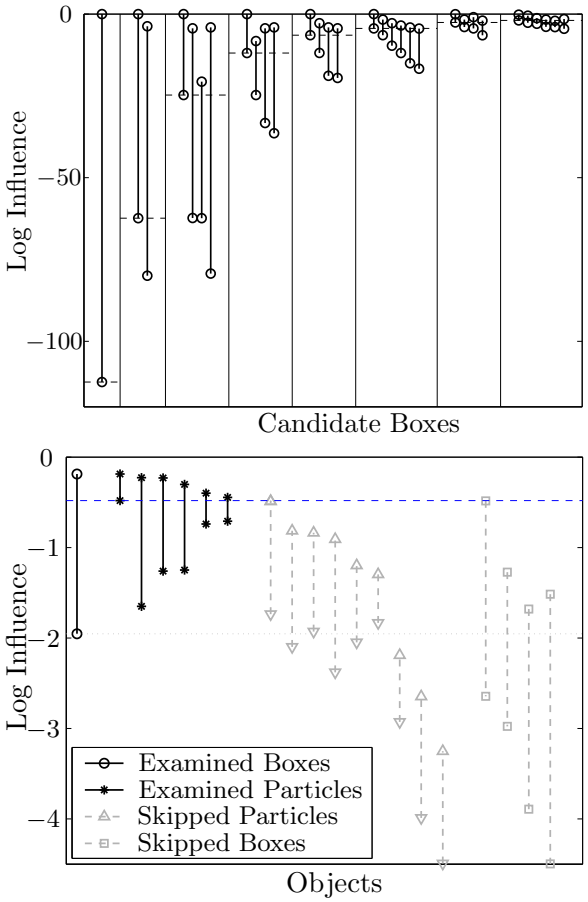


Figure 2: Dual-tree max-kernel example. *Top:* the influence bounds for the nodes shown in figure 1. The pruning threshold at each level is shown (dashed line), along with the bounds for each candidate node. *Bottom:* pruning at the leaf level: in the example, six leaf nodes are candidates. We begin examining particles in the first box. As it happens, the first particle we examine is the best particle (the correct answer). Pruning by particle weight (the upper marker) allows us to ignore all but the first six particles. The pruning threshold is then sufficiently high that we can prune the remaining candidate nodes without having to examine any of their particles.

## 2 Fast methods for computing max-kernel

### 2.1 The distance transform

In [4], Felzenszwalb and Huttenlocher derive a fast algorithm for a class of max-kernel problems by observing that the maximization in equation (1) is solvable by applying the distance transform. This achieves $O(N)$ cost and is very efficient in practice. Additionally, the problem is separable in dimensionality, so a $d$-dimensional transform of $N^d$ points costs $O(dN^d)$.

#### 2.1.1 Extension to Monte Carlo grids in 1-D

While the distance transform was designed to work exclusively on regular grids, it is easily extended to irregular grids in the one-dimensional case, for a small increase in cost. This observation is novel, to our knowledge, although it represents a rather direct extension to the original algorithm.

Assume we are given source particles $\{x_1, \ldots, x_N\}$ and target particles $\{y_1, \ldots, y_M\}$. The first step of the algorithm is to compute the lower envelope of the parabolas anchored at $\{x_i\}$. This step is unchanged, save that the $x$ particles need to be pre-sorted at a cost of $O(N \log N)$. The second step is to calculate the value of the lower envelope at each $y$ particle. This can be done by either pre-sorting the $y$ particles, or employing binary search on the lower-envelope, which costs $O(M \log M)$ or $O(M \log N)$ respectively.

Unfortunately, this extension only applies to the one-dimensional case. Other means must be used to compute higher-dimensional max-kernel problems on Monte Carlo grids.

```
INPUTS: root nodes of X and Y trees: X_r, Y_r.
ALGORITHM:
  leaves = {}, candidates = {X_r}
  max_recursive(Y_r, leaves, candidates, −∞)
FUNCTION  max_recursive(Y, leaves, candidates, τ)
  if (leaf(Y)  AND  candidates = {})
    // Base Case: reached leaves (see figure 4).
    max_base_case(Y, leaves)
  else // Recursive case: recurse on each Y child.
    foreach y ∈ children*(Y)
      τ_y = τ, valid = {}
      foreach p ∈ candidates
        // Check if we can prune parent node p.
        if (w(p) K(d^l(p, y)) < τ_y)
          continue
        foreach x ∈ children(p)
          // Compute child bounds.
          f^{u,l}(x) = w(x) K(d^{l,u}(x, y))
        // Set pruning threshold.
        τ_y = max(τ_y , max_x(f^l(x)))
        valid = valid ∪ {x ∈ children(p) : f^u(x) ≥ τ_y}
      valid = {x ∈ valid : f^u(x) ≥ τ_y}
      leaves_y = {x ∈ valid : leaf(x)}
      candidates_y = {x ∈ valid : NOT(leaf(x))}
      sort(leaves_y by f^l)
      max_recursive(y, leaves_y, candidates_y, τ_y)
```

Figure 3: Dual-tree max-kernel algorithm, part 1.

```
FUNCTION  max_base_case(Y, leaves)
  foreach x ∈ leaves
    f^{u,l}(x) = w(x) K(d^{l,u}(x, Y))
  τ = max_x(f^l(x))
  leaves = {x ∈ leaves : f^u(x) ≥ τ}
  sort(leaves by f^l)
  // Examine individual y points.
  foreach y ∈ Y
    τ_y = τ
    foreach x ∈ leaves
      // Prune nodes by Y (cached), then by y.
      if (f^u(x) < τ_y OR w(x) K(d^l(x, y)) < τ_y)
        continue
      // Examine individual x particles.
      foreach i ∈ x
        // Prune by weight.
        if (f^u(x) w(i)/w(x) < τ_y)
          break
        f(i) = w(i) K(d(i, y))
        if (f(i) > τ_y)
          // i is the new best particle.
          τ_y = f(i), x*(y) = i
```

Figure 4: Dual-tree max-kernel algorithm, part 2.

### 2.2 Dual-tree max-kernel

In this section we present a novel dual-tree algorithm for solving the weighted max-kernel problem. Our algorithm is based on bounding the distance and weight, hence the influence, of subtrees of $X$ particles upon subtrees of $Y$ particles. We begin by constructing space-partitioning trees for the $X$ particles and $Y$ points (see Section 2.3). The leaf nodes of these trees can contain multiple points. We also cache at each node in the $X$ tree the maximum particle weight in the node ($w(X)$). At leaf nodes, we sort the particles in order of decreasing weight.

The algorithm proceeds by doing a depth-first recursion down the $Y$ tree. For each node, we maintain a list of $X$ nodes that could contain the best particle (candidates). We know the particle of maximum weight in a given node $X$. Thus, we can bound the influence of $X$ by considering the cases when that particle is as close (or far) from $Y$ as possible.

For each $X$ node we compute the lower and upper bounds of the influence of the maximum particle in the node on all points in the $Y$ node ($f^{\{l,u\}}$) by evaluating the kernel at the upper and lower bound on the distances to particles in the node ($d\{i, l\}$). The largest lower bound on influence is the *pruning threshold* ($\tau$): any candidate node whose upper bound is less than this threshold cannot possibly contain the best particle, and hence need not be considered. See Figures 1 and 2 for an example.

In each recursive step, we choose one $Y$ child on which to recurse. Initially, the set of $X$ candidates is the set of candidates of the parent. We sort the candidates by lower bound, which allows us to explore the most promising nodes first. For each of the candidates' children, we compute the lower bound on distance and hence the upper bound on influence. Any candidates that have upper bound less than the pruning threshold are pruned. For those that are kept, the lower influence bound is computed; these nodes have the potential to become the new best candidate.

The influence bounds tighten as we descend the tree, allowing an increasingly number of nodes to be pruned. Once we reach the leaf nodes, we begin looking at individual particles. The candidate nodes are sorted by lower influence bound, and the particles are sorted by weight, so we examine the most promising particles first and minimize the number of individual particles examined. In many cases, we only have to examine the first few particles in the first node, since the pruning threshold often increases sufficiently to prune the remaining candidate nodes. Figures 3 and 4 contain pseudo-code for the algorithm.

For a given node in the $Y$ tree, the list of candidate nodes in the $X$ tree is valid for all the points within the $Y$ node, which is the secret behind the efficiency of dual-tree recursion. In this way, pruning decisions are shared among $Y$ points when possible.

### 2.3 Spatial indices

*Spatial indices* (sometimes called *spatial access methods*) intelligently subdivide a set into regions of high locality given some concept of distance. We briefly review two commonly-used spatial indices.

#### 2.3.1 Kd-trees

A kd-tree operates on a vector field, and recursively chooses a dimension and split point to localize particles The dimension of largest spread is typically chosen as splitting dimension. Kd-trees are effective in low dimensional settings; a 2-D example is given in figure 1 (not all levels are shown).

#### 2.3.2 Anchors hierarchy and metric trees

Metric trees are more relaxed in their requirements than kd-trees; they need only a defined distance metric. Nodes in a metric tree consist of a *pivot* (a point lying at the centre of the node), and *radius*. All points belonging to the node must have a distance to the pivot smaller than the radius of the node.[5] The *Anchors hierarchy* was introduced by Moore in [11] and is an efficient means of constructing a metric tree. Unlike kd-trees, metric tree construction and access costs do not have factors that explicitly depend on dimension.

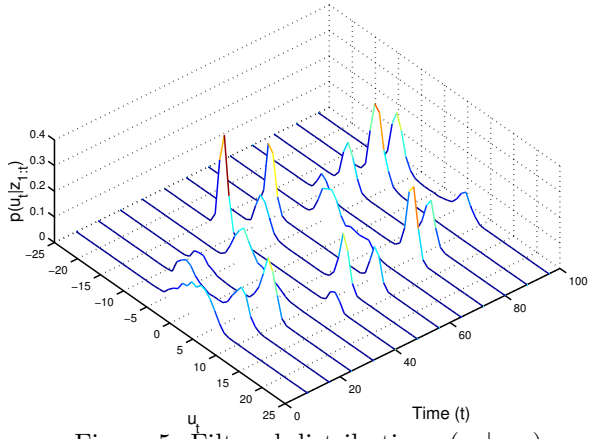

Figure 5: Filtered distribution $p(\mathbf{u}_t|\mathbf{z}_{1:t})$

## 3 Performance in $N$

We turn to empirical evaluation of the dual-tree algorithm. In this section, we focus on performance in synthetic and real-world settings as $N$ grows; comparisons are made both in settings where the distance transform is applicable and where it is not. We present results in terms of both CPU time and number of distance computations (kernel evaluations) performed. This is important as in some applications the kernel evaluation is extremely expensive and thus dominates the runtime of the algorithm.

[5] Note: it is not the case that all points within the radius of the pivot belong to the node.

### 3.1 Multi-modal non-linear time series

Consider the following standard reference model [3, 6]:

$$u_{t+1} = \frac{1}{2}u_t + 25\frac{u_t}{1+u_t^2} + 8\cos 1.2t + v_{t+1}$$

$$z_{t+1} = \frac{u_{t+1}^2}{20} + w_{t+1}$$

where $v_t \sim \mathcal{N}(0, \sigma_v)$ and $w_t \sim \mathcal{N}(0, \sigma_w)$. The filtered distribution is bimodal and highly non-linear (figure 5), meaning the standard particle filter produces significant error even with a high particle count. After running a standard SIR particle filter, we implemented the MAP sequence estimation described in Section 1.2. Figure 6 demonstrates the accuracy gained by calculating the MAP solution. We chose a one-dimensional setting so that the dual-tree algorithm could be directly compared against the distance transform (using the modified algorithm from Section 2.1.1). Figures 7 and 8 summarize the results. It is clear that the distance transform is superior in this setting, although the dual-tree algorithm is still quite usable, being several orders of magnitude faster than the naïve method.

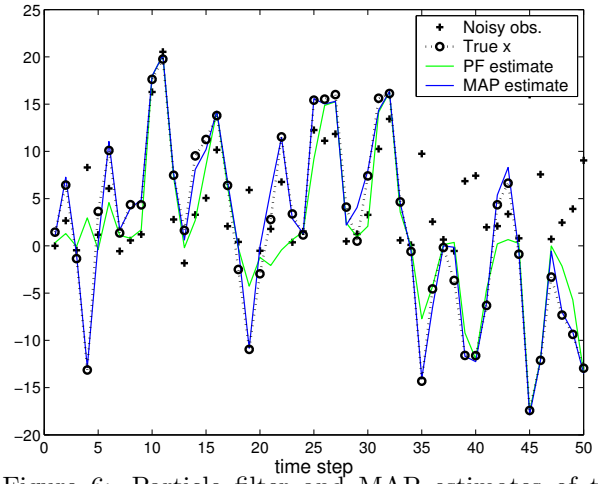

Figure 6: Particle filter and MAP estimates of the latent state in the 1-D time series experiment. Mean error for the particle filter was 4.05, while the MAP solution achieved a mean error of 1.74.

### 3.2 Beat-tracking

Beat-tracking is the process of determining the time slices in a raw song file that correspond to musical beats. This is a challenging problem: both the tempo and phase of the beats must be estimated throughout

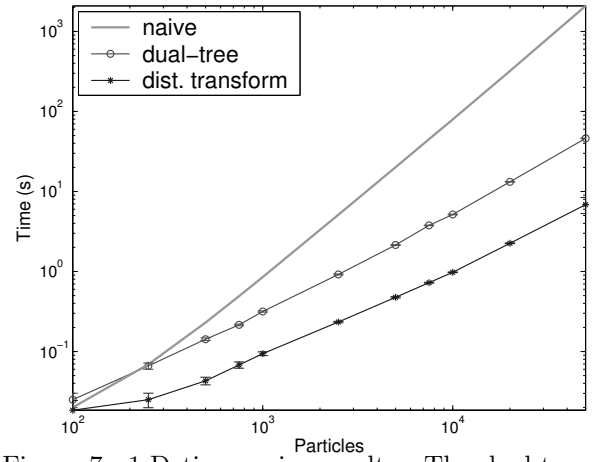


Figure 7: 1-D time series results. The dual-tree algorithm became more efficient than naïve computation after approximately 70ms of compute time. Both dual-tree and distance transform methods show similar asymptotic growth, although the constants in the distance transform are approximately three times smaller.

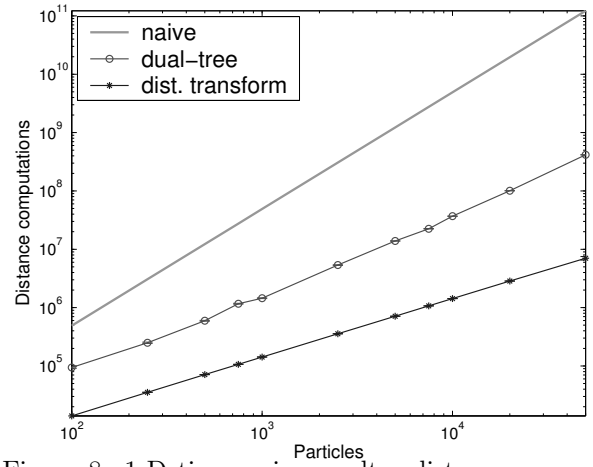


Figure 8: 1-D time series results: distance computations v. particle count.

the song. MAP sequence estimation after particle filtering has achieved impressive results in the literature. We omit the details of the probability model for the sake of brevity, but a full explanation is found in [10]. The algorithm used was the forward pass particle filter, backward pass Viterbi algorithm described in Section 1.2. Since the state space of this model is a three-dimensional Monte Carlo grid, the distance transform cannot be used. Figures 9 and 10 summarize the results: songs can be processed in seconds rather than hours with this method. Using the fast method also enables more particles to be used, which results in a better solution: the probability of the MAP sequence with 50000 particles was $p = 0.87$, while using 1000 particles resulted in a MAP sequence of probability $p = 0.53$.

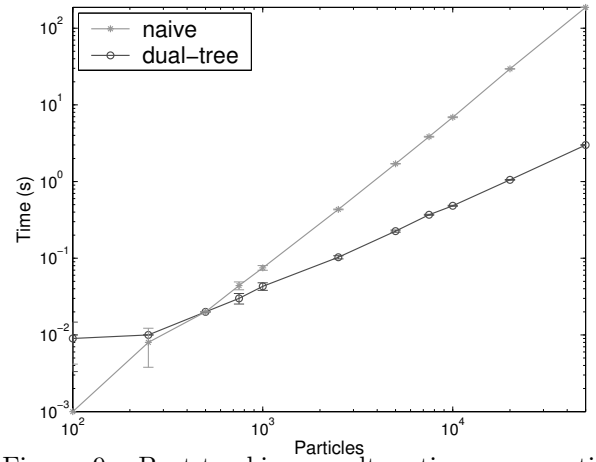


Figure 9: Beat-tracking results: time v. particle count. The dual-tree method becomes more efficient at $t = 10$ms, and thereafter dominates the naïve method.

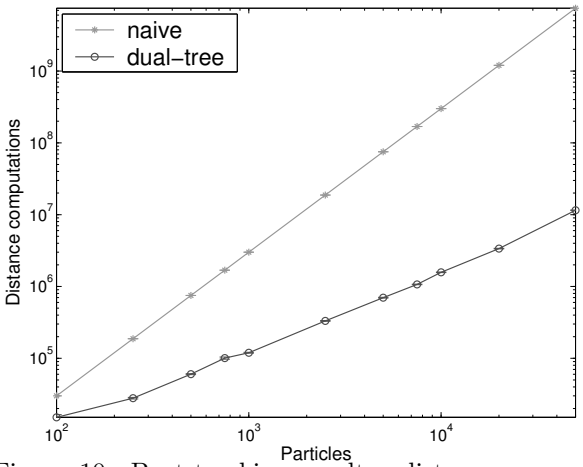


Figure 10: Beat-tracking results: distance computations v. particle count.

# 4 The effect of other parameters

## 4.1 Distribution and dimensionality

To examine the effects of other parameters on the behaviour of the dual-tree algorithm, we ran several experiments varying dimensionality, distribution, and spatial index while keeping $N$ constant. We used two spatial indices: kd-trees and metric trees (built using the Anchors hierarchy) as described in Section 2.3. We generated synthetic data by drawing points from a mixture of Gaussians distributed evenly in the space. Figure 11 shows a typical clustered data distribution. In all runs the number of particles was held constant at $N = 20,000$, and the dimension was varied to a maximum of $d = 40$. Figures 12 and 13 show the results for CPU time and distance computations, respectively. In these figures, the solid line represents a uniform distribution of particles, the dashed line represents a 4-cluster distribution, the dash-dot line has

20 clusters, and the dotted line has 100. We expect methods based on spatial indexing to fare poorly given uniform data since the average distance between points quickly becomes a very peaked distribution in high dimension, reducing the value of distance as a measure of contrast. The results are consistent with this expectation: for uniform data, both the kd-tree and anchors methods exceeded $O(N^2)$ distance computations when $d \geq 12$. More surprising is that the kd-tree method consistently outperformed the anchors method on uniform data even up to $d = 40$. The depth of a balanced binary kd-tree of 20000 particles and leaf size 25 is ten, so for $d > 10$ there are many dimensions that are not split even a single time!

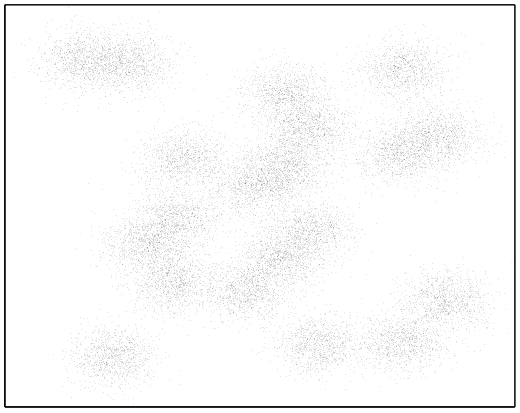

Figure 11: Synthetic data set with $c = 20$ clusters.

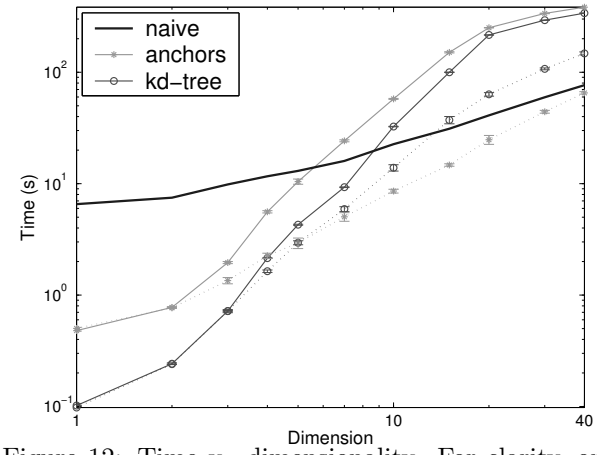


Figure 12: Time v. dimensionality. For clarity, only the uniform distribution and one level of clustered data are shown. This experiment demonstrates that some structure is required to accelerate max-kernel in high dimensions.

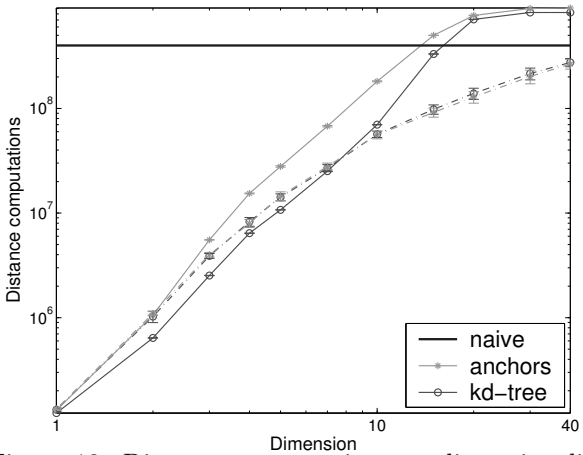


Figure 13: Distance computations v. dimensionality. The level of clustering shown is less than in figure 12. For kd-trees, clustering hurts performance when $d \leq 8$.

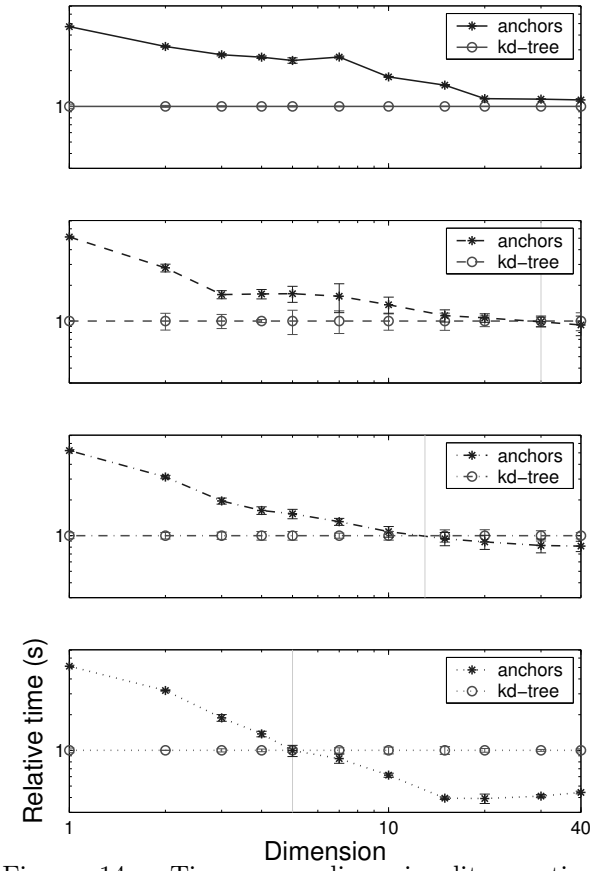


Figure 14: Time v. dimensionality; ratio to kd-tree = 1. Metric trees are better able to properly index clusters: the more clustered the data, the smaller dimensionality required for the anchors method to outperform kd-trees ($d = 30$ for somewhat-clustered data, $d = 15$ for moderately-clustered data, and $d = 6$ for significantly-clustered data).

Of more practical interest are the results for clustered data. It is clear that the distribution vastly affects the runtime of dual-tree algorithms; at $d = 20$, performing max-kernel with the anchors method was six times faster on clustered data compared to uniform. We expect this effect to be even greater on real data sets, as the clustering should exist on many scales rather than

simply on the top level as is the case with our synthetic data. It is also interesting to note the different effect that clustering had on kd-trees compared to metric trees. For the anchors method, clustering always improved the runtime, albeit by a marginal amount in low dimensions. For kd-trees, clustering *hurt* performance in low dimensions, only providing gains after about $d = 8$. The difference in the two methods is shown in figure 14.

### 4.2 Effect of kernel width

To measure the effect of different kernels, we test both methods on a 1-D uniform distribution of $200,000$ points, and use a Gaussian kernel with bandwidth ($\sigma$) varying over several orders of magnitude. The number of distance computations required was reduced by an order of magnitude over this range (figure 15). Wider kernels allow the weights to have more contrast, hence affording more opportunities for pruning.

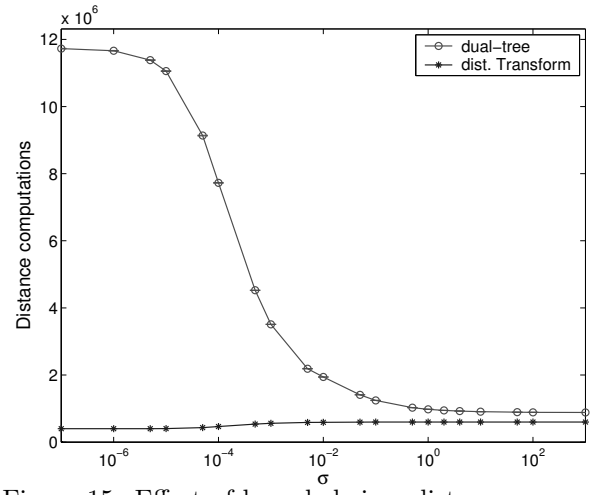


Figure 15: Effect of kernel choice: distance computations v. bandwidth of a Gaussian kernel.

## 5 Conclusion

Weighted maximum-kernel problems are common in statistical inference, being used, for instance, in belief propagation and MAP particle filter sequence estimation. We develop an exact algorithm based on dual-tree recursion that substantially reduces the computational burden of this procedure for a wide variety of kernels. It is particularly important when the state space lies on a multi-dimensional Monte Carlo grid, where, to our knowledge, no existing acceleration methods can be applied.

The method we present speeds up the inner loop of belief propagation, which means that it can be combined with other acceleration methods such as node pruning and dynamic quantization [2] to achieve even faster results, albeit at the expense of the loss of accuracy that those methods entail. The techniques we present could also be integrated seamlessly into hierarchical BP [4].

We also look at the other variables that affect the performance of dual-tree recursion, such as dimensionality, data distribution, spatial index, and kernel. These parameters have dramatic effects on the runtime of the algorithm, and our results suggest that more exploration is warranted into these effects—behaviour as $N$ varies is only a small part of the story.

## References


[1] N Bergman, *Recursive Bayesian estimation: Navigation and tracking applications*, Ph.D. thesis, Department of Electrical Engineering, Linköping University, Sweeden, 1999.

[2] J M Coughlan and H Shen, *Shape matching with belief propagation: Using dynamic quantization to accomodate occlusion and clutter*, GMBV, 2004.

[3] A Doucet, N de Freitas, and N J Gordon (eds.), *Sequential Monte Carlo methods in practice*, Springer-Verlag, 2001.

[4] P Felzenszwalb and D Huttenlocher, *Efficient belief propagation for early vision*, CVPR, 2004.

[5] P Felzenszwalb, D Huttenlocher, and J Kleinberg, *Fast algorithms for large-state-space HMMs with applications to web usage analysis*, NIPS (2003).

[6] S J Godsill, A Doucet, and M West, *Maximum a posteriori sequence estimation using Monte Carlo particle filters*, Ann. Inst. Stat. Math. **53** (2001), no. 1, 82–96.

[7] A Gray and A Moore, *'N-Body' problems in statistical learning*, NIPS, 2000, pp. 521–527.

[8] A Gray and A Moore, *Rapid evaluation of multiple density models*, AISTATS, 2003.

[9] A T Ihler, E B Sudderth, W T Freeman, and A S Willsky, *Efficient multiscale sampling from products of Gaussian mixtures*, NIPS 16, 2003.

[10] D Lang and N de Freitas, *Beat tracking the graphical model way*, NIPS 17, 2004.

[11] A Moore, *The Anchors Hierarchy: Using the triangle inequality to survive high dimensional data*, UAI 12, 2000, pp. 397–405.

[12] J Pearl, *Probabilistic reasoning in intelligent systems: networks of plausible inference*, Morgan-Kaufmann, 1988.

[13] C P Robert and G Casella, *Monte Carlo statistical methods*, Springer-Verlag, New York, 1999.