Fast Variational Mode-Seeking

Bo Thiesson
Microsoft Research
thiesson@microsoft.com

Jingu Kim
Georgia Institute of Technology
jingu@cc.gatech.edu

Abstract

Mode-seeking algorithms (e.g., mean-shift) constitute a class of powerful non-parametric clustering methods, but they are slow. We present VMS, a dual-tree based variational EM framework for mode-seeking that greatly accelerates performance. VMS has a number of pleasing properties: it generalizes across different mode-seeking algorithms, it does not have typical homoscedasticity constraints on kernel bandwidths, and it is the first truly sub-quadratic acceleration method that maintains provable convergence for a well-defined objective function. Experimental results demonstrate acceleration benefits over competing methods and show that VMS is particularly desirable for data sets of massive size, where a coarser approximation is needed to improve the computational efficiency.

1 Introduction

Clustering is a common task in data analysis and has been successfully applied for knowledge discovery across many application areas. In this paper, we discuss mode-seeking clustering algorithms, which is a class of algorithms that define clusters via an iterative process that associates each data point with a local mode in an underlying kernel density function. Mean-shift [9, 3, 4] is arguably the best known of the mode-seeking algorithms.

Compared to parametric clustering methods, such as mixture modeling, mode-seeking algorithms have a number of pleasing properties: 1) no limitations are imposed on the geometric shape of clusters, 2) explicit a priori knowledge about the number of clusters is not required, but is more loosely controlled via a kernel-bandwidth parameter, and 3) they are more robust to outliers. On the other hand, mode-seeking algorithms also have acknowledged shortcomings. In particular, their time complexities are at least quadratic in sample size, making them computationally impractical for data sets of significant size.

Several approximation schemes have over the years been proposed to address the scalability problem for, in particular, the mean-shift algorithm (e.g., [20, 21, 10, 1, 5]). Many of these approximation schemes offer guaranteed error bounds on each iterative mean-shift update or on computationally demanding parts of the update, but none of them offer guaranteed convergence due to the lack of a well-defined objective function. Although convergence is highly likely for a tight approximation bound, convergence becomes more of an issue if a coarser approximation is needed for improved computational efficiency.

We introduce the first approximation scheme for mean-shift that is truly sub-quadratic in sample size while at the same time enjoying provable convergence for a well-defined objective function at any level of approximation. In addition, without depending on tuning parameters (beyond the desired level of approximation), our approach offers a very tight control over the approximation error, resulting in it generally being faster than its competitors at reaching a solution of same quality, as we demonstrate in Section 6. We target relatively low dimensionality (up to ten), where mean-shift clustering has proven most successful.

Furthermore, it is a demonstrated fact (see, e.g., [6, 10]) that when the feature space differs significantly across data, a global bandwidth parameter used across all kernels will introduce problems for the mean-shift algorithm. In areas where data are sparse, a small bandwidth will not allow data points to move, creating individual clusters for each of them. On the other hand, in dense data areas, a larger bandwidth may cause data points to converge incorrectly to modes of otherwise separate clusters. In contrast to some strong competitors, our approach allows different local bandwidths at individual kernels, hence also enabling the acceleration of the so-called adaptive mean-shift [6].

The elegant interpretation of mean-shift as a (generalized) EM algorithm in [1] provides important groundwork for our acceleration method. This interpretation splits a mean-shift update into explicit E- and M-steps. The E-step is the computationally expensive part responsible for the quadratic complexity. We alleviate

Appearing in Proceedings of the 15^{th} International Conference on Artificial Intelligence and Statistics (AISTATS) 2012, La Palma, Canary Islands. Volume 22 of JMLR: W&CP 22.

this computational burden by a variational approximation relying on two partition trees: one for data that are shifted towards local density modes and one for kernels that define the underlying density function. As in a standard dual-tree framework [12], we maintain certain statistics in the nodes of these trees, allowing us to efficiently compute the variational distribution in the E-step. It is key to the efficient variational approximation that these statistics factor into separate kernel and data parts. To simplify the presentation, we use Gaussian kernels for demonstration in this paper, although any kernel from the exponential family of distributions will satisfy the separation property.

Finally, we demonstrate that alternative mode-seeking algorithms, such as medoid-shift [16] and quick-shift [19], can also be represented in the (generalized) EM view of mode-seeking. In this view, they all share the same E-step and only differ in their respective M-steps. By targeting the computational burden of the E-step, our variational mode-seeking framework can therefore be used across all of these algorithms. To avoid distracting from the general idea behind the variational mode-seeking framework, we will focus on the mean-shift algorithm in the main paper and refer the reader to the supplementary Appendix A for specific algorithmic details on the mode-seeking alternatives.

2 Related acceleration work

There has been much work on accelerating mode-seeking algorithms. Most of this work explores different ways of reducing the number of terms involved in the weighted kernel sums that define the update for a data point (see, e.g., the sums over m involved in the mean-shift update (4) or (6)). There are two basic principles behind these approaches: One is to argue that most of the contribution to a sum comes from kernels located within a neighborhood of the data point and therefore ignore the contribution from kernels further away. Another is to pre-compute statistics that can be used to approximate parts of a weighted kernel sum across the update for many different data points.

The work in [20] falls into the second category and is arguably the most similar to our approach. Both approaches involve a dual-tree framework [12], where separate space partitioning trees (e.g., a kd-tree [8] or ball-tree [2]) are constructed for, respectively, the data and the kernels. By recursing on the nodes in the two trees, smaller parts of the data and kernels are selected together, which tightens the approximation. The stopping criterion that [20] provides for the recursion is based on an error bound tailored to a mean-shift update and further restricted to homoscedastic kernels. Our approach does not have these restrictions and differs further in the following two ways: 1) The approximation at data and kernels nodes is based on a variational approach that is better suited for approximations at higher levels in the trees. 2) We recast a shift update into a log-likelihood optimization problem and provide a provably convergent approach that tightens a lower bound of the log-likelihood as we recurse in the trees. It allows us to control the recursive expansion of the trees in a tighter way than [20], which becomes important because the computational efficiency (in both approaches) is jeopardized if the recursive expansion continues to a very granular level.

In [21, 15], a very successful acceleration of mean-shift draws efficiency from both of the above mentioned basic principles, but it is restricted to homoscedastic Gaussian kernels by applying an improved fast Gauss transform (IFGT) to efficiently evaluate (weighted) sums in the mean-shift update. The IFGT approach first partitions the kernels using a k-center flat clustering algorithm (e.g. [7]) and caches statistic for each cluster based on a truncated set of decaying basis functions. The weighted sums involved in the mean-shift update for each data point can now be efficiently computed by collecting the contribution from kernels in neighboring clusters, using the cached statistics.

In [10], mean-shift is combined with local sensitive hashing (LSH) [11]. The approach falls into the first category of acceleration principles by using LSH to find the neighborhood of kernels that contribute the most to a shift update for a given data point. LSH involves a computationally intensive tuning phase, and on that account, [20] reports significantly better performance at higher accuracy for both of the above dual-tree and IFGT based approaches on standard image segmentation tasks and comparable results on higher dimensional experiments ($d = 16$).

Relating to the variational aspect of our approach, Carreira-Perpiñán [1] provides the interpretation of mean-shift as an EM algorithm and touches on a variational view, known as sparse EM [14]. This interpretation of mean-shift partially applies the second acceleration principle by infrequently using all kernels in an update and otherwise performing updates based on a partial set only. However, due to the (infrequent) use of all kernels in an update, the computational complexity is still quadratic in sample size, and is therefore not appropriate for very large data sets.

Finally, among other more heuristic acceleration approaches, down-sampling [5] is a popular pragmatic way of reducing the computational complexity. This approach is obviously orthogonal to the above acceleration principles and can be used in combination.

3 Mode-seeking Clustering

Let us start by defining the normalized kernel function

$$p(x|\mu, \Sigma) = \frac{1}{Z_\Sigma} K(D_\Sigma(x, \mu)), \tag{1}$$

where $x \in \mathbb{R}^d$, K is the kernel profile, D is a squared distance metric depending on the kernel location $\mu \in$

$\mathbb{R}^d$ and positive definite bandwidth matrix $\Sigma \in \mathbb{R}^d \times \mathbb{R}^d$, and Z is a normalization constant depending on Σ only. For a standard Gaussian kernel, the profile $K(z) = e^{-\frac{1}{2}z}$ is defined with the squared Mahalanobis distance $z = D_\Sigma(x,\mu) = (x-\mu)^T\Sigma^{-1}(x-\mu)$ and normalization $Z_\Sigma = |2\pi\Sigma|^{\frac{1}{2}}$.

Given M data points $\boldsymbol{\mu} = \{\mu_1, \ldots, \mu_M\} \subset \mathbb{R}^d$, we can express the kernel-density estimate as the M component mixture model

$$p(x) = \sum_{m=1}^{M} p(m)p(x|m) = \frac{1}{M}\sum_{m=1}^{M} p(x|\mu_m, \Sigma_m), \quad (2)$$

where $p(m) = \frac{1}{M}$ and for later notational convenience $p(x|m) = p(x|\mu_m, \Sigma_m)$. Notice how we allow kernels with different bandwidth matrices, giving us a *heteroscedastic* density estimator. It simplifies to a *homoscedastic* density estimator if $\Sigma_m = \Sigma$ for all m.

Mode-seeking clustering embraces a class of different methods, which all are characterized by iteratively moving data points upward towards the modes in the kernel density estimate $p(x)$. During this mode-seeking process, each data point moves along a trajectory x_n^t, $t \geq 0$, starting from $x_n^0 = \mu_m$. (Notice that we index moving data by n and the fixed data–i.e. kernel locations–by m.) Clusters are finally defined as the data that have converged to the same mode.

Mean shift, as first introduced in [9] and later formalized in [3], is arguably the best known among the mode-seeking algorithms, and many others have later contributed to theoretical aspects hereof. Let $K'(z) = \frac{dK(z)}{dz}$. Iterating the mean-shift update

$$x_n^{t+1} = \arg\max_{x\in\mathbb{R}^d} \sum_m D_{\Sigma_m}(x,\mu_m)\tfrac{1}{Z_m}K'\left(D_{\Sigma_m}(x_n^t,\mu_m)\right) \quad (3)$$

will, for kernels with monotonically decreasing and convex profile, guarantee convergence to a mode of $p(x)$ [6]. The representational form in (3) is due to Sheikh *et al.* [16]. Setting the first derivative to zero and solving for x will result in the well-known update for heteroscedastic kernels from [6], which for standard Gaussian kernels can be expressed as

$$x_n^{t+1} = \left(\textstyle\sum_m p(x_n^t|m)\Sigma_m^{-1}\right)^{-1}\sum_m p(x_n^t|m)\Sigma_m^{-1}\mu_m \quad (4)$$
$$= \left(\textstyle\sum_m p(m|x_n^t)\Sigma_m^{-1}\right)^{-1}\sum_m p(m|x_n^t)\Sigma_m^{-1}\mu_m \quad (5)$$

The last equality follows by recalling that $p(m) = \frac{1}{M}$ and using Bayes' theorem $p(m|x_n^t) = \frac{p(x_n^t|m)}{Mp(x_n^t)}$. This representation of the mean-shift update has also been discussed in [1] and later becomes important for the intuition behind our variational approach. For homoscedastic kernels, updates (4) and (5) simplify as

$$x_n^{t+1} = \frac{\sum_m p(x_n^t|m)\mu_m}{\sum_m p(x_n^t|m)} = \sum_m p(m|x_n^t)\mu_m. \quad (6)$$

The complexity of the mean-shift algorithm is $\mathcal{O}(M^2)$.

4 Variational Mode-seeking

The general idea behind the variational mode-seeking framework is to replace the posteriors used in the mode-seeking update formula (see (5) and (6)) with variational approximates that have low computational cost and at the same time provide a good approximation. We accomplish this task by applying a variational view on mode-seeking inspired by the EM interpretation of mean-shift in [1].

A trick behind the EM interpretation is to reverse the typical maximum likelihood estimation for the mixture in (2) in the sense that we optimize with respect to an observation x instead of the parameters in the distribution. We will call this view the *reverse likelihood* view. In our framework, we handle all the data points $\boldsymbol{x}^t = \{x_1^t, \ldots, x_M^t\}$ simultaneously and therefore consider the joint reverse log-likelihood

$$\mathcal{L}(\boldsymbol{x}) = \sum_{n=1}^{M}\log p(x_n) = \sum_{n=1}^{M}\log\sum_{m=1}^{M} p(m)p(x_n|m).$$

Let us introduce variational (conditional) probabilities $q(m|n)$, $n, m \in \{1, \ldots, M\}$, where q denotes the joint variational distribution that factorizes with respect to each $q(\cdot|n)$, i.e. $q = \prod_n q(\cdot|n)$. By applying Jensen's inequality, we can lower bound the reverse likelihood:

$$\begin{aligned}
\mathcal{L}(\boldsymbol{x}) &= \sum_n \log\sum_m q(m|n)\frac{p(m)p(x_n|m)}{q(m|n)} \\
&\geq \sum_n\sum_m q(m|n)\log\frac{p(m)p(x_n|m)}{q(m|n)} \quad (7)\\
&= \sum_n\sum_m q(m|n)\log p(x_n) \\
&\quad - \sum_n\sum_m q(m|n)\log\frac{q(m|n)}{p(m|x_n)} \\
&= \mathcal{L}(\boldsymbol{x}) - \sum_n KL[q(\cdot|n)||p(\cdot|x_n)] \quad (8)\\
&\triangleq \mathcal{F}(\boldsymbol{x}, q),
\end{aligned}$$

where $KL[\cdot||\cdot]$ denotes the Kullback-Leibler divergence between two distributions.

For a well-behaved (upper bounded) $\mathcal{L}(\boldsymbol{x})$, the variational generalization [14] of the EM algorithm can be used to maximize a lower bound by alternating the following two steps, i.e. coordinate ascent, on $\mathcal{F}(\boldsymbol{x}, q)$ until guaranteed convergence:

$$\text{E-step:} \quad q^{t+1} = \arg\max_{q\in\mathcal{Q}} \mathcal{F}(\boldsymbol{x}^t, q) \quad (9)$$

$$\text{M-step:} \quad \boldsymbol{x}^{t+1} = \arg\max_{\boldsymbol{x}\in\mathcal{X}} \mathcal{F}(\boldsymbol{x}, q^{t+1}), \quad (10)$$

where $\mathcal{Q}$ and $\mathcal{X}$ denote, respectively, the family of variational distributions for q and the space for $\boldsymbol{x}$.

Importantly, we have flexibility to restrict the sets $\mathcal{Q}$ and $\mathcal{X}$ over which the maximization problems are

solved. Restricting $\mathcal{Q}$ and $\mathcal{X}$ has two different purposes. The restrictions on $\mathcal{Q}$ alleviate the computational burden of the E-step across the mode-seeking algorithms, whereas restrictions on $\mathcal{X}$ determine the choice of the mode-seeking algorithm that we apply.

Regarding the choice of $\mathcal{Q}$, let us consider the E-step with a decomposition of $\mathcal{F}(\boldsymbol{x}^t, q)$ as in (8). Since only the KL-divergence term depends on q, it follows that if $\mathcal{Q}$ is not constrained in any way, the E-step optimization produces $q(m|n) = p(m|x_n^t)$; that is, the variational probabilities $q(\cdot|\cdot)$ are the same as the posterior membership probabilities $p(\cdot|\cdot)$ used in the update formula for the mode-seeking algorithm. In Section 5.2, we show how to design a restricted family $\mathcal{Q}$ in a principled way that allows us to tie the dual-tree framework into a variational approximation and thereby achieve a very efficient algorithm for solving the E-step.

Variational mean shift is achieved by not imposing any constraints on $\mathcal{X}$. Taking the derivative for the expression of $\mathcal{F}(\boldsymbol{x}, q^{t+1})$ in (7) and setting to zero will for heteroscedastic Gaussian kernels result in the following M-step optimization

$$x_n^{t+1} = \left(\sum_m q^{t+1}(m|n)\Sigma_m^{-1}\right)^{-1} \sum_m q^{t+1}(m|n)\Sigma_m^{-1}\mu_m. \tag{11}$$

Notice that this expression is equivalent to the mean-shift update in (5) except that the variational probabilities $q(m|n)$ replace the posterior probabilities $p(m|x_n)$. When $\mathcal{Q}$ is also unrestricted, the variational EM framework for mean-shift is equivalent to the interpretation in [1] as standard EM.

5 Dual-tree based variational acceleration

The class of variational distributions $\mathcal{Q}$ is the key component that significantly affects computational efficiency in our variational mode-seeking framework. The restrictions on $\mathcal{Q}$ are designed to simplify the process of solving the E-step in (9) while maintaining $KL[q(\cdot|n)||p(\cdot|x_n)]$ as small as possible. Efficiency is achieved by starting from a very restricted $\mathcal{Q}$ and then gradually loosening the restrictions until a desired accuracy is obtained for the E-step.

Algorithm 1 shows the main flow of an update iteration in variational mode-seeking. The actual M-step (in Line 11) is the only difference between variational mean-shift and the alternatives in the Appendix A. Details for the algorithm are provided in the following subsections–indicated by comments–and the algorithm should for now just be considered as a point of reference to return to, as the basic concepts are introduced.

Algorithm 1: VARIATIONAL MODE-SEEKING (one iteration)

```
1:  Build kernel & data partition trees        //Sec. 5.1
2:  Set sufficient statistics in the trees     //Sec. 5.4
3:  Initial B-STEP                             //Sec. 5.3 (Algo. 2)
4:  E-STEP                                     //Sec. 5.4 (Algo. 3)
5:  F = Eq.(14), F_0 = F                       //Sec. 5.4
6:  repeat
7:      Refining B-STEP                        //Sec. 5.3
8:      E-STEP
9:      F' = F, F = Eq. (14)
10: until (F - F')/(F - F_0) < ε
11: M-STEP                                     //Sec. 5.5
```

5.1 Partition trees

We maintain two partition trees: one for the data points $\boldsymbol{x} = \{x_1, \ldots, x_M\}$, which are shifted towards the modes, and one for the kernels, which we for convenience will identify by just their fixed locations $\boldsymbol{\mu}=\{\mu_1, \ldots, \mu_M\}$. We will refer to these two trees as, respectively, the *data partition tree* and the *kernel partition tree*. The goal behind the partition trees is to be able to reason about the effect of a group of kernels (represented by a node in the kernel tree) on a group of data (represented by a node in the data tree) without having to access individual data or kernel elements repeatedly. This is achieved by pre-computing and storing sufficient statistics with every node in a tree, as discussed further in Sections 5.4 and 5.5.

Any hierarchical decomposition of data (kernels) that ensures some degree of similarity between data (kernels) in a node is suitable for a partition tree. We exemplify our work by using ball trees, constructed in $\mathcal{O}(M \log M)$ expected time via the anchor method [13]. It should be noted that during this construction, all distance computations use the same distance metric. This assumption is appropriate for constructing the data partition tree, because data simply specify a location in a given space. However, in contrast to data, each kernel in addition imposes its own distance metric, which in principle should be accounted for during the construction. For homoscedastic kernels, all distance metrics are the same and therefore not an issue. For heteroscedastic kernels, it suffices to ignore the heteroscedasticity, because only a rough hierarchical clustering is needed at this stage of our approach – heteroscedasticity will be accounted for later in the mode-seeking clustering process.

The data partition tree is reconstructed for every iteration in the mean-shift clustering process, because the data shift. In contrast, the underlying kernel density is fixed and the kernel partition tree will therefore not change during the clustering process.

5.2 The Block-constrained Variational Distribution

Let us now describe how the data and kernel partition trees tie into the variational approximation by enabling us to define what we call *block-constraints* on the variational distribution. Let $\mathcal{P} = \{(A_1, B_1), \ldots, (A_P, B_P)\}$ define a mutually exclusive partition of the product space of data and kernels, where A_p is a group of the data and B_p is a group of the kernels, corresponding to nodes in the two re-

spective trees. The idea behind this partition is to group data and kernels together for which kernels are unable to differentiate between the data in any significant way. We denote (A_p, B_p) as a *block* and, hence, the partition as a *block partition*.

Our variational distribution q will for all the data in a block assign the same variational membership probability for all the kernels in the block. That is,

$$q(m|n) = q(B_p|A_p) \text{ for all } (n,m) \in (A_p, B_p), \quad (12)$$

where $(n,m) \in (A_p, B_p)$ is a convenient notation for $(x_n, p(\cdot|\mu_m, \Sigma_m)) \in (A_p, B_p)$. Figure 1a) shows a block partition where, e.g., $q(3|5)$=$q(3|6)$=$q(4|5)$=$q(4|6)$.

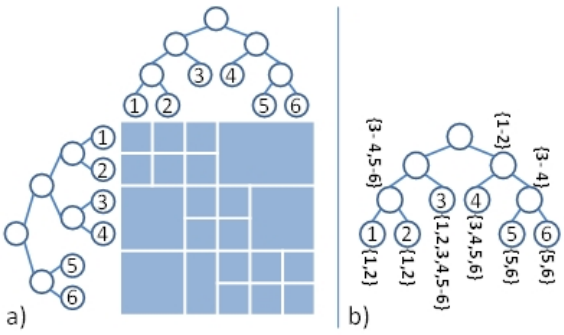

Figure 1: a) A block partition (table) for data partition tree (top) and kernel partition tree (left). b) MPT representation of the block partition.

The variational distributions $q(\cdot|n)$ are standard probability distributions and must therefore satisfy the sum-to-one constraint $\sum_m q(m|n) = 1$. By integrating the sum-to-one constraints into the block constraints in (12) we obtain the following M constraints

$$\sum_{p:n\in A_p} |B_p| q(B_p|A_p) = 1 \text{ for all } n \in \{1, \ldots, M\}. \quad (13)$$

Notice that the constraint associated with data point n involves exactly those $q(B_p|A_p)$ for which $n \in A_p$ in the block partition. In Figure 1a), the variational probabilities involved in the n'th constraint are therefore those $q(B_p|A_p)$ encountered at blocks in the column under data leaf n in the block partition table. For example, for the data partition leaf with data point $n = 1$, the corresponding constraint is $q(1|1)+q(2|1)+2q(3\text{–}4|1\text{–}2)+2q(5\text{–}6|1\text{–}2)=1$, where $q(m_1\text{–}m_2|n_1\text{–}n_2)$ denotes the probability assigned to the block with kernels m_1 through m_2 and data points n_1 through n_2.

The constraints in (13) complicates the E-step optimization of q, because they are intertwined in the sense that some variational probabilities involved in one constraint may also appear as a part of another constraint. Fortunately, this complication can be alleviated, because it is possible to represent the block constraints in the form of a so-called *marked partition tree*, MPT [18]. A MPT is similar to the data partition tree, but with an additional encoding of which data groups relate to which kernel groups in the block partition: For each $(A_p, B_p) \in \mathcal{P}$, a tree node corresponding to A_p is marked with B_p. As an illustration, Figure 1b) shows the MPT corresponding to the partition in Figure 1a). It should be noted that our representation of the MPT is slightly different from the one defined in [18]. Our MPT allows a group of kernels (mixture components in [18]) in a block to be marked together instead of insisting on individual marks for each of the kernels in the block. (E.g., the marks $B_p \in \{3\text{–}4, 5\text{–}6\}$ instead of the marks $B_p \in \{3, 4, 5, 6\}$ for $A_p = 1\text{–}2$ in Figure 1b).) This difference improves computational efficiency.

By construction, each path from a leaf n to the root in the MPT has the important property that it marks all B_p for which $n \in A_p$ and $(A_p, B_p) \in \mathcal{P}$. The MPT therefore defines all the constraints in (13). However, the MPT has the additional benefit of representing the constraints in an explicit tree structure, which enables the efficient estimation of q in the E-step to be described in Section 5.4.

5.3 The B-step – Initial & Refining

The important question is now how to choose a good block partition and hence the constraints we impose on $\mathcal{Q}$. It is important, because the number of blocks in the partition determines the complexity for an iteration in the variational mode-seeking algorithm. We cannot get below the expected $\mathcal{O}(M \log M)$ required to build a partition tree, but if we are not careful the partition could have many more blocks. Worst case is M^2 blocks, which happens if the partition strategy is completely ignorant to any grouping of data or kernels.

Lines 3-10 in Algorithm 1 address the question of restricting $\mathcal{Q}$. An *initial B-step* restricts $\mathcal{Q}$ by constructing the coarsest block-partition, such that any node in the data partition tree does not overlap with any node in the kernel partition tree. The space that a node occupies is here defined by its center and a radius, determined as the distance $D_\Sigma(\cdot,\cdot)$ to the node element furthest away from the center. (We still ignore heteroscedasticity by using the average Σ in these distance computations.) If nodes overlap, it is highly likely that different kernels in a kernel node will have very different affect on data in a data node, and they should therefore not be blocked together. A generic recursive structure for finding an initial partition is sketched in Algorithm 2 and starts from the roots of the two trees. Here, $lch(A)$ and $rch(A)$ are graph properties denoting the left and right child for node A, and cen_A, rad_A, and mkd_A are the attributed center, radius, and set of MPT-marks that decorate the node.

Let L denote the number of refinable blocks in the initial partition. That is, blocks containing more than one data or kernel element. Given the initial block partition, *refining B-steps* now further refine the L most promising blocks at each step. A block is considered

Algorithm 2: RecBlock(A, B): Initial B-step

```
if NoOverlap(A, B) then
    Append B to mkd_A                    //Block (A, B) in MPT
else if Leaf(A) and Leaf(B) then
    Append B to mkd_A                    //Block (A, B) in MPT
else
    if rad_A > rad_B then
        RecBlock(lch(A), B), RecBlock(rch(A), B)
    else
        RecBlock(A, lch(B)), RecBlock(A, rch(B))
    end if
end if
```

more promising for a refinement if it has a larger value of $K(D_{min}(A,B)) - K(D_{max}(A,B))$, where

$$D_{min}(A,B) = D_\Sigma(cen_A, cen_B) - rad_A - rad_B$$
$$D_{max}(A,B) = D_\Sigma(cen_A, cen_B) + rad_A + rad_B$$

are the minimum and maximum possible distances between elements in the two nodes.

Importantly, any refinement enlarges the family of variational distributions by loosening some of the constraints imposed by the block partition structure. A refinement will therefore always result in a tightened lower bound for the log-likelihood. We can measure the improvement by evaluating $\mathcal{F}(\boldsymbol{x}, q)$ and stop refining when the relative improvement is less than a small threshold ϵ (Line 10 of Algorithm 1). An efficient evaluation of $\mathcal{F}(\boldsymbol{x}, q)$ basically runs an E-step to obtain q and then uses (14) to efficiently evaluate $\mathcal{F}(\boldsymbol{x}, q)$ from sufficient statistics stored in the MPT – as described in detail in the next section.

5.4 The Efficient E-step

With the representation of block constraints on the variational distribution, as in (12), the decomposition of $\mathcal{F}(\boldsymbol{x}, q)$ in (7) reduces to

$$\mathcal{F}(\boldsymbol{x}, q) = \sum_{p=1}^{P} |A_p||B_p| q(B_p|A_p) \times \left[-\log q(B_p|A_p) + H(B_p) + G(B_p|A_p) \right], \quad (14)$$

where, for $p(m) = 1/M$,

$$H(B_p) = \frac{1}{|B_p|} \sum_{m \in B_p} \log p(m) = -\log M$$
$$G(B_p|A_p) = \frac{1}{|A_p||B_p|} \sum_{n \in A_p} \sum_{m \in B_p} \log p(x_n|m).$$

Notice that the block-specific $G(B|A)$ is the only part of $\mathcal{F}(\boldsymbol{x}, q)$ that depends on specific data and kernels.

We integrate the constraints in (13) into the maximization of $\mathcal{F}(\boldsymbol{x}, q)$ with respect to q by use of the Lagrange multiplier method. Introducing the Lagrange multipliers $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_M)$, we construct the Lagrangian

$$\mathcal{F}(\boldsymbol{x}, q, \boldsymbol{\lambda}) = \mathcal{F}(\boldsymbol{x}, q) + \sum_{n=1}^{M} \lambda_n \left(\sum_{p: n \in A_p} |B_p| q(B_p|A_p) - 1 \right)$$

By setting $\partial \mathcal{F}(\boldsymbol{x}, q, \boldsymbol{\lambda}) / \partial q(B_p|A_p) = 0$ and solving for $q(B_p|A_p)$, we obtain

$$q_{\boldsymbol{\lambda}}(B_p|A_p) = \exp\left(\frac{1}{|A_p|} \sum_{n \in A_p} \lambda_n - 1\right) \times \exp\left(H(B_p) + G(B_p|A_p)\right), \quad (15)$$

where the $\boldsymbol{\lambda}$ subscript in $q_{\boldsymbol{\lambda}}(B_p|A_p)$ indicates that this solution is still a function of $\boldsymbol{\lambda}$.

A closed-form solution to the constrained optimization problem can now be found by inserting $q_{\boldsymbol{\lambda}}(B_p|A_p)$ into the constraints in (13) and then exploit the tree structure of these constraints, as represented by the MPT, to gradually solve for $\boldsymbol{\lambda}$. Following, by inserting this $\boldsymbol{\lambda}$ back into (15), we finally obtain the optimal value for the variational probabilities $q(B_p|A_p)$, $p = 1, \ldots, P$.

The way $\boldsymbol{\lambda}$ is resolved relates to a solution in [18, Appendix] for a similar problem, where the basic structure of $q_{\boldsymbol{\lambda}}$ is the same as in (15) and constraints have also been organized into a MPT. An important difference is that the $G(B_p|A_p)$ in our $q_{\boldsymbol{\lambda}}$ expression, besides data statistics, also includes statistics for kernel parameters. We will refer to Appendix B for a detailed derivation of this solution, but will account for the derivation of the $G(B_p|A_p)$ statistics in detail below. To ease implementation, we also include a detailed algorithmic procedure for finding the solution in Algorithm 3. The complexity of the E-step is $\mathcal{O}(P)$, where P is the number of blocks in the block partition.

Algorithm 3: E-step

```
Define:
    C(A) = Σ_{B∈mkd_A} (|B|/M) exp(G(B|A))     //G(B|A) = Eq.(16)
    K(A) = |rch(A)| log(D_lch(A) / D_rch(A)) + K_lch(A) + K_rch(A)
    D(A) = C(A) exp(K_A / |A|) + D_lch(A)

//Collect-up
for all data nodes A ∈ MPT; traversed bottom-up do
    if Leaf(A) then
        K_A = 0, C_A = C(A), D_A = C_A
    else
        K_A = K(A), C_A = C(A), D_A = D(A)
    end if
end for

//Distribute-down
for all data nodes A ∈ MPT; traversed top-down do
    if Root(A) then
        λ_A = 1 − log(D_A)
    else if not Leaf(A) then
        λ_lch(A) = λ_A
        λ_rch(A) = λ_A + log(D_lch(A)) − log(D_rch(A))
    end if
    for all kernel nodes B ∈ mkd_A do
        q(B|A) = exp(λ_A − 1 + K_A/|A| + G(B|A)) (1/M)
    end for
end for
```

Sufficient statistics. It is key to the computational efficiency of the variational E-step that sufficient statistics for $G(B_p|A_p)$ factorize into data-specific and kernel-specific statistics and in this way avoids the computation of statistics on the product space of data

and kernels. These statistics can therefore be pre-computed separately and stored at the nodes in the respective partition trees prior to the actual mode-seeking process. For heteroscedastic Gaussian kernels

$$\begin{aligned} G(B_p|A_p) &= \tfrac{1}{|A_p||B_p|}\sum_{n\in A_p}\sum_{m\in B_p}\log p(x_n|\mu_m,\Sigma_m) \\ &= c - \tfrac{1}{2}\big[\tfrac{1}{|B_p|}\sum_{m\in B_p}\log|\Sigma_m| \\ &\quad + \tfrac{1}{|A_p||B_p|}\sum_{n\in A_p}\sum_{m\in B_p}(x_n-\mu_m)^T\Sigma_m^{-1}(x_n-\mu_m)\big] \\ &= c - \tfrac{1}{2}\big[<\log|\Sigma_m|>_{B_p} + <\mu_m^T\Sigma_m^{-1}\mu_m>_{B_p} \\ &\quad + Tr(<\Sigma_m^{-1}>_{B_p}<x_nx_n^T>_{A_p}) - 2<\mu_m^T\Sigma_m^{-1}>_{B_p}<x_n>_{A_p}\big], \end{aligned} \tag{16}$$

where $c = -\frac{1}{2}d\log 2\pi$, d is dimensionality, $<\cdot>_{A_p}$ and $<\cdot>_{B_p}$ denote averages over $n \in A_p$ and $m \in B_p$, respectively, and $Tr(\cdot)$ denotes the trace of a matrix. Importantly, we see that the sufficient statistics in (16) factorize into data-specific and kernel-specific parts.

For the homoscedastic kernels, $G(B_p|A_p)$ simplifies to

$$\begin{aligned} G(B_p|A_p) &= c - \frac{1}{2}\big[\log|\Sigma| + Tr(\Sigma^{-1}<\mu_m^T\mu_m>_{B_p}) \\ &\quad + Tr(\Sigma^{-1}<x_nx_n^T>_{A_p}) - 2<\mu_m^T>_{B_p}\Sigma^{-1}<x_n>_{A_p}\big]. \end{aligned}$$

5.5 The Efficient M-step

The M-step shifts each data point x_n^t according to an optimization, specific to each mode-seeking algorithm. (See Appendix A for alternatives to mean-shift.)

Mean-shift. Recall that after the E-step, the nodes in the MPT contain the computed $q^{t+1}(B_p|A_p)$, and that $q^{t+1}(m|A_p) = q^{t+1}(B_p|A_p)$ for all $m \in B_p$. Using the variational update formula in (11) the update for heteroscedastic kernels can therefore be expressed as

$$\begin{aligned} x_n^{t+1} &= \left(\sum_{p:n\in A_p}|B_p|q^{t+1}(B_p|A_p)<\Sigma_m^{-1}>_{B_p}\right)^{-1} \\ &\quad \times \sum_{p:n\in A_p}|B_p|q^{t+1}(B_p|A_p)<\Sigma_m^{-1}\mu_m>_{B_p}. \end{aligned}$$

For homoscedastic kernels, this expression simplifies to

$$x_n^{t+1} = \sum_{p:n\in A_p}\frac{|B_p|q^{t+1}(B_p|A_p)<\mu_m>_{B_p}}{\sum_{p:n\in A_p}|B_p|q^{t+1}(B_p|A_p)}.$$

We can compute the above updates efficiently, because the statistics $<\Sigma_m^{-1}\mu_m>_{B_p}$, $<\Sigma_m^{-1}>_{B_p}$, and $<\mu_m>_{B_p}$ have already been pre-computed and stored in the kernel partition tree prior to the mode seeking. Conceptually, one can imagine the M-step as following the path from leaf n to the root in the MPT, while computing the update for each x_n^t. A more efficient implementation will traverse the MPT top-down, avoiding recalculations of the shared terms for the updates. This traversal has complexity $\mathcal{O}(P)$.

6 Experiments

The first group of experiments compare our variational mode-seeking (VMS) acceleration with the standard (STD) un-accelerated approach, the dual-tree (DT) acceleration from [20], and the IFGT acceleration from [21] with parameter tuning as in [15]. We compare performance on the mean-shift algorithm since this is the only algorithm for which all three acceleration methods are defined. Still, comparisons are complicated by the fact that they all have non-comparable types of error bounds, and both the acceleration and cluster quality is affected by choices made for each method's bound. To enable meaningful comparisons among VMS, DT, and IFGT, we therefore fix the quality of a mean-shift update while measuring acceleration. This is done by running VMS to a certain value of its error bound and then calibrate the bounds for the alternative methods to obtain an update of same quality as VMS. Here, quality is measured by the *approximation error* defined as the average Euclidian distance to the STD update for each data point. In these first experiments, we report results for one update step and not at final convergence in order not to compound with other factors (e.g., different number of update steps before convergence).

We generated synthetic data with varying sample size and dimensionality. The data were in all cases drawn randomly from 100 random-shaped Gaussians with random location in the unit-hypercube, and truncated to the unit-hypercube. In all experiments, we use a homoscedastic spherical Gaussian kernels with bandwidth set as the average distance to the k'th nearest neighbor for each data point, where k is a fraction of the sample size. We varied one of four experimental default settings at a time: sample size $M = 40,000$, number of dimensions $d = 2$, number of nearest neighbors $k = M/1000$, and the approximation level $\epsilon = 0.01$.

Table 1 shows the results. Entries with an 'X' denote that the algorithm exhausted RAM, hence corrupting timing results by page swapping, and entries with an '∞' denote that experiments were slower than STD and therefore stopped. First of all, we see that VMS significantly improves the computational efficiency over STD in all experiments while keeping the approximation error very low ($\leq 10^{-3}$). As expected, sample size has a positive effect on the speedup, with a computational complexity for VMS dramatically lower than $\mathcal{O}(M^2)$ and almost (but not quite) approaching $\mathcal{O}(M\log M)$. The dimensionality has a negative effect on the speedup, but still at dimension 10 we see a moderate speedup for $40,000$ samples. Surprisingly, and in sharp contrast to the DT algorithm, the kernel bandwidth (determined by k) did not affect the performance of VMS. We discovered the reason by investigating the detailed experiment logs: The variational approximation that VMS uses to handle kernels and data that are blocked together is much better than

Table 1: Running time (in seconds) and approximation error for synthetic experiments.

100 Gaussians, d =2, $k = M/1000$, ϵ =0.01					
Algo / M	5,000	10,000	20,000	40,000	80,000
STD	146	590	2165	8433	35590
IFGT	∞	∞	486	738	1520
DT	8	29	99	366	1381
VMS	7	19	61	145	313
Error	6E-5	2E-4	4E-4	5E-4	4E-4

100 Gaussians, M = 40,000, $k = M/1000$, ϵ =0.01					
Algo / d	2	4	6	8	10
STD	8433	8501	8972	10221	10186
IFGT	738	∞	∞	∞	∞
DT	336	2350	4726	5196	6815
VMS	145	1199	1761	2010	2276
Error	5E-4	8E-4	7E-4	5E-4	4E-4

100 Gaussians, M = 40,000, d =2, ϵ =0.01					
Algo / k	4	40	400	4000	8000
STD	8567	8433	8457	8677	8672
IFGT	∞	738	101	34	16
DT	90	366	2120	X	X
VMS	57	145	118	122	125
Error	3E-5	5E-4	8E-4	1E-3	9E-4

100 Gaussians, M = 40,000, d =2, $k = M/1000$					
Algo / ϵ		0.1	0.01	0.001	
STD		8433	8433	8433	
IFGT		512	738	1002	
DT		339	366	631	
VMS		49	145	642	
Error		1E-3	5E-4	8E-5	

the approximation in DT. Starting from a very course block partition, MPT only needs a few refining B-steps (<10 for our 2-d experiments, but increasing with dimensionality) to achieve a good approximation, which is not the case for DT. Since the block granularity dominates the complexity in the dual-tree based algorithms, VMS does much better. Finally, the table shows that VMS is better than the competing approximation algorithms, except for extremely high bandwidth kernels, where IFGT is best–still VMS does very well in that situation. Finally, lowering ϵ will favor DT at the cost of speedup over STD. However, our experiments are run for relatively small data sets, because of the necessity to run STD for error bound alignment across methods. Data sets of massive size will demand a coarser approximation (with higher ϵ), which we can see will highly favor VMS.

In the second group of experiments, we compared the acceleration methods on image segmentation tasks. We used the first ten images from the Berkeley segmentation data set [17] transformed to the CIE LUV color space and normalized to the unit cube. Although the target of our VMS method is data sets of much larger size, we reduced all images to 85×128 (or 128×85) pixels, again to enable the error bound alignment necessary for comparisons between the acceleration methods (in reasonable time). We used the same default settings as in the synthetic experiments, and ran mean-shift to convergence (with error bounds calibrated according to the first update step). Figure 2 shows the speedup factor for DT and VMS over STD (IFGT is slower) and the approximation error after the first update and at convergence. We see that VMS is faster in all cases and with a lower error in all but two cases. Figure 3 offers a visual examination of the segmentation results for one of the average performing images–without any post-processing of the final clusters. The color segmentation for STD seems well approximated by both acceleration methods.

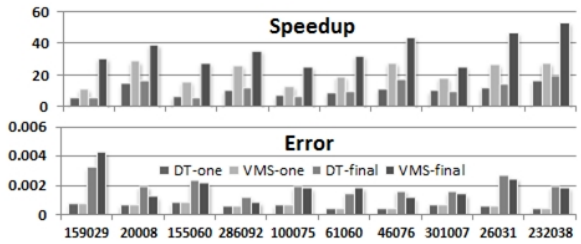

Figure 2: Speedup (top) and error (bottom) for image experiments. Labels at bottom are image identifiers. The bars are in order DT,VMS after one iteration and DT, VMS at final convergence.

Figure 3: Image 100075. Original image followed by STD, DT, and VMS color segmentations.

In a final group of experiments, we demonstrate that VMS can, in fact, handle heteroscedastic kernels, as opposed to the two competing acceleration methods. (The importance of this property is already demonstrated in [6, 10].) We used the same setup as in the previous image experiments, except that each kernel now has an individual bandwidth set as the distance to its k'th nearest neighbor. Being constrained on space, we only summarize the results here. Compared to VMS with homoscedastic kernels, VMS learned with heteroscedastic kernels on average improved the approximation error to a heteroscedastic STD update by almost 50%, at the small cost of being 5% slower.

7 Conclusion and future work

We have presented VMS, a fast dual-tree based variational EM view on mode-seeking. The variational EM view targets the E-step only, thereby allowing generality across different mode-seeking algorithms. By applying a dual-tree data structure we obtain speed while the variational approximation maintains the quality of results. VMS works well across a wide range of bandwidths and is the fastest existing way (we know of) to accelerate large-sample mode-seeking for lower bandwidths. In future work, we plan to further explore kernelization of the distance measure in hope of extending VMS to manifold learning in higher dimensions.

Acknowledgment

We thank Dongryeol Lee for helpful discussions on the error bound for the dual-tree acceleration from [20]. The work of Jingu Kim was done during an internship at Microsoft Research.

References

[1] M. A. Carreira-Perpiñán. Gaussian mean-shift is an EM algorithm. *IEEE Trans. Pattern Anal. and Mach. Intell.*, 29(5):767–776, 2007.

[2] E. Chavez, G. Navarro, R. Baeza-Yates, and J. L. Marroqun. Proximity searching in metric spaces. *ACM Computing Surveys*, 33:273–321, 2001.

[3] Y. Cheng. Mean shift, mode seeking, and clustering. *IEEE Trans. Pattern Anal. Mach. Intell.*, 17:790–799, 1995.

[4] D. Comaniciu and P. Meer. Mean shift analysis and applications. In *Proc. Int'l Conf. Computer Vision*, volume 2, pages 1197–1203, 1999.

[5] D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Trans. Pattern Anal. and Mach. Intell.*, 24(5):603–619, 2002.

[6] D. Comaniciu, V. Ramesh, and P. Meer. The variable bandwidth mean shift and data-driven scale selection. In *Proc. 8th IEEE Int'l Conf. on Computer Vision*, pages 438–445, 2001.

[7] T. Feder and D. Greene. Optimal algorithms for approximate clustering. In *ACM Symp. Theory of Computing,*, pages 434–444, 1988.

[8] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.*, 3:209–226, September 1977.

[9] K. Fukunaga and L. Hostetler. The estimation of the gradient of a density function, with applications in pattern recognition. *IEEE Trans. Inform. Theory*, 21:32–40, 1975.

[10] B. Georgescu, I. Shimshoni, and P. Meer. Mean shift based clustering in high dimensions: A texture classification example. In *Proc. 9th IEEE Int'l Conf. Computer Vision*, pages 456–463, 2003.

[11] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *Very Large Data Bases*, pages 518–529, 1999.

[12] A. Gray and A. Moore. N-body problems in statistical learning. In *Neural Inf. Process. Syst. 13*. MIT Press, 2001.

[13] A. W. Moore. The anchors hierarchy: Using the triangle inequality to survive high dimensional data. In *Proc. 16th Conf. Uncertainty in Artificial Intell.*, pages 397–405, 2000.

[14] R. Neal and G. E. Hinton. A view of the EM algorithm that justifies incremental, sparse, and other variants. In *Learning in Graphical Models*, pages 355–368, 1998.

[15] V. C. Raykar, C. Yang, R. Duraiswami, and N. Gumerov. Fast computation of sums of Gaussians in high dimensions. Technical Report CS-TR-4767, University of Maryland, 2005.

[16] Y. A. Sheikh, E. A. Khan, and T. Kanade. Mode-seeking by medoidshifts. In *Proc. 11th IEEE Int'l Conf. Computer Vision*, 2007.

[17] The Berkeley Segmentation Dataset and Benchmark. http://www.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/.

[18] B. Thiesson and C. Wang. Fast large-scale mixture modeling with component-specific data partitions. In *Neural Inform. Process. Syst. 22*. MIT Press, 2010.

[19] A. Vedaldi and S. Soatto. Quick shift and kernel methods for mode seeking. In *Proc. European Conf. Computer Vision*, pages 705–718. Springer, 2008.

[20] P. Wang, D. Lee, A. Gray, and J. Rehg. Fast mean shift with accurate and stable convergence. In *Proc. 11th Int'l Conf. Artificial Intell. and Statistics*, volume 2, pages 604–611, 2007.

[21] C. Yang, R. Duraiswami, N. A. Gumerov, and L. S. Davis. Improved fast gauss transform and efficient kernel density estimation. In *Int'l Conf. Comp. Vision*, pages 464–471, 2003.

Appendix A—SUPPLEMENTARY MATERIAL

We demonstrate that alternative mode-seeking algorithms that include medoid-shift [16] and quick-shift [19] can also be accelerated by our variational mode seeking approach. A key understanding is that medoid-shift and quick-shift can be viewed as *generalized* EM algorithms. In this view, they share the same E-step with mean-shift and only differ in their respective M-steps. The computationally demanding part is the E-step, and it can be accelerated by the dual-tree based variational approximation described in Section 5. We hence can achieve variational medoid-shift and variational quick-shift by employing the dual-tree based E-step and making corresponding changes to their M-steps.

A.1 Medoid-shift and quick-shift

Let us first introduce the basic forms of medoid-shift and quick-shift. In contrast to mean-shift, medoid-shift and quick-shift constrain the update to be one of the original data points.

Medoid-shift. Instead of maximizing the expression in (3), medoid-shift [16] constrains the maximization to original data points. That is, x_n^t moves to the weighted medoid given by

$$x_n^{t+1} = \arg\max_{x \in \boldsymbol{\mu}} \sum_m D_{\Sigma_m}(x, \mu_m) \frac{1}{Z_m} K'\left(D_{\Sigma_m}(x_n^t, \mu_m)\right). \tag{17}$$

For Gaussian kernels, the medoid-shift update simplifies as

$$\begin{aligned} x_n^{t+1} &= \arg\min_{x \in \boldsymbol{\mu}} \textstyle\sum_m D_{\Sigma_m}(x, \mu_m) p(x_n^t | m) \\ &= \arg\min_{x \in \boldsymbol{\mu}} \textstyle\sum_m D_{\Sigma_m}(x, \mu_m) p(m | x_n^t), \end{aligned} \tag{18}$$

where $p(m) = \frac{1}{M}$ and the Bayes' theorem $p(m|x_n^t) = \frac{p(x_n^t|m)}{Mp(x_n^t)}$ are used to obtain the last equality. Since the trajectories x_n^t, $t \geq 0$ are constrained to pass through points in $\boldsymbol{\mu}$, the updates in medoid-shift need to be computed *only once* for each initial x_n^0, $n = 1, \ldots, M$. (Recall that $x_n^0 \in \boldsymbol{\mu}$.) This first step links each data point to its medoid-shift update, and a trajectory to a mode can therefore be found by following these links until $x_n^{t+1} = x_n^t$. Medoid-shift finally repeats this process on the identified modes until no further changes occurs in the set of modes.

The complexity of the medoid-shift algorithm can be as high as $\mathcal{O}(M^2 d + M^3)$ for simple implementations. However, in [19], it is shown that medoid-shift can reach a complexity as low as $\mathcal{O}(M^2)$ when the underlying distance is Euclidean.

Quick-shift. The quick-shift algorithm [19] simply updates each x_n^0 with the nearest neighbor in $\boldsymbol{\mu}$ for which there is an increment in the kernel-density estimate.

Similarly to medoid-shift, once the first shift update is computed, the following shifts are automatically defined for all trajectories, and convergence for each trajectory is guaranteed at a point for which $x_n^{t+1} = x_n^t$. Different from other mode-seeking algorithms, quick-shift connects the trajectories of all the points into a single tree, and hence defines a single mode. Breaking branches between points with a distance greater than a certain threshold recovers multiple modes in the kernel-density estimate. The complexity of the quick-shift algorithm is $\mathcal{O}(M^2)$ [19].

A.2 Variational medoid-shift and Variational quick-shift

The EM view explained in Section 4 enables a conceptual separation of the E-step and the M-step of a mode-seeking algorithm. Continuing the discussion in Section 4, this view can further be given a variational interpretation represented as the E-step in (9) and the M-step in (10). The E-step is generic across the mode-seeking algorithms, whereas the M-step differs by applying different (generalized) maximization strategies, with the set $\mathcal{X}$ restricted in different ways.

Variational medoid-shift. Variational medoid-shift is achieved by restricting $\mathcal{X}$ to contain only the original set of data points at the M-step maximization. That is, $\mathcal{X} = \boldsymbol{\mu} = \{\mu_1, \ldots, \mu_M\}$. Given a representation of $\mathcal{F}(\boldsymbol{x}, q^{t+1})$ as in (7), the maximization under this restriction can now be expressed as

$$x_n^{t+1} = \arg\max_{x \in \boldsymbol{\mu}} \textstyle\sum_m q^{t+1}(m|n) \log p(x|m) + C,$$

where $C = \sum_m q(m|n) log \frac{p(m)}{q(m|n)}$ is independent of x. For Gaussian kernels, this update simplifies as

$$x_n^{t+1} = \arg\min_{x \in \boldsymbol{\mu}} \textstyle\sum_m q^{t+1}(m|n) D_{\Sigma_m}(x, \mu_m). \tag{19}$$

When $\mathcal{Q}$ is unconstrained, $q(m|n) = p(m|x_n^t)$, and we can see that (19) in this case reduces to the standard expression for medoid-shift in (18).

Variational quick-shift. Variational quick-shift is achieved by updating each x_n^0 with the nearest neighbor in $\boldsymbol{\mu}$ that increases the lower bound $\mathcal{F}(x_n, q)$ instead of the kernel density estimate $\mathcal{L}(x_n)$.

Note that an increase in $\mathcal{F}(x_n, q)$ guarantees an increase in $\mathcal{L}(x_n)$, which is the objective for selecting a nearest neighbor in the original quick-shift. Although likely, the implication is not guaranteed the other way around, and therefore the original quick-shift is not strictly the same as our variational quick-shift, but in spirit the two algorithms are similar.

A.3 Dual-tree based acceleration

Since variational versions of mean-shift, medoid-shift, and quick-shift share the same E-step, the dual-tree

based acceleration of the E-step presented in Section 5 is immediately applicable to the variational versions of all three methods. That is, Lines 1-10 in Algorithm 1 remain the same in the dual-tree accelerated medoid-shift and quick-shift. The last Line 11, which is the M-step, can now be efficiently performed utilizing the dual-tree structure as follows.

Note that before arriving at Line 11, partition trees have been constructed, and a final block partition has been determined together with the construction of a corresponding marked partitioning tree (MPT). Furthermore, sufficient statistics are stored at each node in the MPT and $q^{t+1}(B_p|A_p)$ has been calculated for all $(A_p, B_p) \in \mathcal{P}$.

Dual-tree accelerated medoid-shift For heteroscedastic kernels, efficient computation of the update in (19) utilizes the following equality

$$\begin{aligned}\sum_m q^{t+1}(m|n) D_{\Sigma_m}(x, \mu_m) \\ = \sum_{p:n\in A_p} |B_p| q^{t+1}(B_p|A_p) \big(<\mu_m \Sigma_m^{-1} \mu_m^T>_{B_p} \\ + Tr(<\Sigma_m^{-1}>_{B_p} x x^T) - 2 <\mu_m^T \Sigma_m^{-1}>_{B_p} x\big).\end{aligned}$$

For homoscedastic kernels, the equality simplifies as

$$\begin{aligned}\sum_m q^{t+1}(m|n) D_{\Sigma_m}(x, \mu_m) \\ = \sum_{p:n\in A_p} |B_p| q^{t+1}(B_p|A_p) \big(Tr(\Sigma^{-1} <\mu_m \mu_m^T>_{B_p} \\ + Tr(\Sigma^{-1} x x^T) - 2 <\mu_m^T>_{B_p} \Sigma^{-1} x\big).\end{aligned}$$

The precomputed sufficient statistics $<\mu_m \Sigma_m^{-1} \mu_m^T>_{B_p}$, $<\Sigma_m^{-1}>_{B_p}$, $<\mu_m^T \Sigma_m^{-1}>_{B_p}$, $<\mu_m \mu_m^T>_{B_p}$, and $<\mu_m^T>_{B_p}$ that have been stored in the kernel partition tree prior to the mode-seeking can now be used to efficiently evaluate above quantities. Further efficiency can be gained by using the MPT to approximate the search of the weighted medoid for a data point x_n by only investigating candidates covered by an internal node above x_n in the MPT.

Dual-tree accelerated quick-shift The efficient computation of the M-step update for quick-shift is achieved by realizing that for a single observation x_n, the expression for the lower bound in (14) simplifies as

$$\begin{aligned}\mathcal{F}(x_n, q) &= \sum_{p:x_n\in A_p} |B_p| q(B_p|A_p) \\ &\quad \times \left[-\log q(B_p|A_p) + H(B_p) + G(B_p|x_n)\right],\end{aligned}$$

where

$$G(B_p|x_n) = \tfrac{1}{|B_p|} \sum_{m\in B_p} \log p(x_n|m),$$

and $p : x_n \in A_p$ are all the blocks from the block partition that can be encountered on the path in the MPT from the root to the leaf that represents x_n.

Furthermore, following the same type of algebraic manipulations as in (16), we see that $G(B_p|x_n)$ can be evaluated efficiently from pre-computed sufficient statistics, as used in the E-step. For heteroscedastic kernels

$$\begin{aligned}G(B_p|x_n) &= \tfrac{1}{|B_p|} \sum_{m\in B_p} \log p(x_n|\mu_m, \Sigma_m) \\ &= c - \tfrac{1}{2}\big[<\log|\Sigma_m|>_{B_p} + <\mu_m^T \Sigma_m^{-1} \mu_m>_{B_p} \\ &\quad + Tr(<\Sigma_m^{-1}>_{B_p} x_n x_n^T) - 2 <\mu_m^T \Sigma_m^{-1}>_{B_p} x_n\big],\end{aligned}$$

which for homoscedastic kernels simplifies to

$$\begin{aligned}G(B_p|x_n) &= c - \frac{1}{2}\big[\log|\Sigma| + Tr(\Sigma^{-1} <\mu_m^T \mu_m>_{B_p}) \\ &\quad + Tr(\Sigma^{-1} x_n x_n^T) - 2 <\mu_m^T>_{B_p} \Sigma^{-1} x_n\big].\end{aligned}$$

We can therefore efficiently calculate $\mathcal{F}(x_n, q)$ for all x_n by traversing the MPT top-down while avoiding recalculation of shared terms for the different x_n. The final nearest neighbor search can now for each x_n be efficiently conducted by considering the leaf in the MPT that represents x_n and then backtrack through the tree until a data point with a larger value of $\mathcal{F}$ is found. During this backtracking, knowledge about the center and radius of a node can be used to disregard some parts of the tree, where a closer candidate to the current best candidate cannot be found.

Appendix B—SUPPLEMENTARY MATERIAL

This Appendix contains a detailed description of how the Algorithm 3 resolves the constrained optimization problem in the variational E-step.

Let us first summarize some notations. Let A be a node in the MPT. The left and right child of A are denoted by $lch(A)$ and $rch(A)$, respectively. For two nodes A and A', where A is an ancestor of A', there exists a unique path in the MPT that connects A and A' found by iteratively traversing children. Let us denote this path by $A \to A'$ and the set of nodes that participate in this path by $\{A \to A'\}$. Let $lvs(A)$ be the set of all leaf descendants of A. If A is a leaf node, $lvs(A) = \{A\}$. The left-most leaf among the descendants of A is special, and we denote this leaf by $ll(A)$. If A is a leaf node, $ll(A) = A$. An important observation is that all nodes on the path from a node to its left-most leaf share the same left-most leaf. That is, $ll(V) = ll(A)$ for all $V \in \{A \to ll(A)\}$.

Let mkd_A represent the set of kernel nodes that are marked on the data node A in the MPT. Hence, $B \in mkd_A$ if and only if $(A, B) \in \mathcal{P}$. For each $(A, B) \in \mathcal{P}$, there is an associated variational probability $q(B|A)$. We now define q_A to be the weighted sum of all these probability values at node A, as follows

$$q_A = \sum_{B\in mkd_A} |B|\, q(B|A), \tag{20}$$

and further define

$$q_{A\to A'} = \sum_{V\in\{A\to A'\}} q_V.$$

With these definitions in place, consider an arbitrary leaf node L and the root node R of the MPT. The sum-to-one constraint from (13) associated with L can now be written in the following simple way

$$q_{R\to L} = 1. \tag{21}$$

Similarly, for a non-leaf node A, one of its child nodes A', and a leaf descendant L of A', the sum-to-one constraint can be written as

$$q_{R\to A} + q_{A'\to L} = 1. \tag{22}$$

The representations in (21) and (22) are very useful in deriving the details of the dual-tree based E-step.

The goal of the E-step is to find probability assignments $q(B|A)$ for all $(A,B)\in\mathcal{P}$ that will optimize $\mathcal{F}(\boldsymbol{x},q)$ under $|lvs(R)|$ sum-to-one constraints in the form of (21). Algorithm 3 implements the Lagrange multiplier method for this task. In this solution, a *collect-up phase* first traverses the MPT bottom-up, level by level, and gradually reduces the $|lvs(R)|$ constraints to a single equation involving only $\lambda_{ll(R)}$, the Lagrange multiplier for the constraint associated with the left-most leaf in the MPT. After solving this equation, a *distribute-down phase* now traverses the MPT top-down, level by level, and gradually resolves the remaining Lagrange multipliers $\lambda_L, L\in lvs(R)$, by back-substitution of previously resolved λ_L. The resolved Lagrange multipliers can then be used to find the desired variational probability assignments $q(B|A)$. The details for the collect-up and distribute-down phases are described next.

B.1 Collect-up

In (15) of the main text, we have shown that the optimal variational probability satisfies

$$q(B|A) = \exp\left(\tfrac{1}{|A|}\sum_{L\in lvs(A)}\lambda_L - 1\right)\frac{1}{M}\exp G(B|A). \tag{23}$$

Recall, also, that Algorithm 3 keeps track of three attribute values for each node A in the MPT, namely

$$\begin{aligned} K_A &= |rch(A)|\log\left(\frac{D_{lch(A)}}{D_{rch(A)}}\right) \\ &\quad + K_{lch(A)} + K_{rch(A)} && (24)\\ C_A &= \sum_{B\in mkd_A}\frac{|B|}{M}\exp G(B|A) && (25)\\ D_A &= C_A\exp\left(\frac{K_A}{|A|}\right) + D_{lch(A)} && (26) \end{aligned}$$

which are computed recursively as the MPT is traversed bottom-up. At the leaf nodes, these three attributes are initialized as $K_A = 0$, $D_A = C_A$.

We claim the following proposition.

Proposition 1. *After the collect-up phase of Algorithm 3, for any node A of the MPT, the Lagrange multipliers satisfy*

$$\sum_{L\in lvs(A)}\lambda_L = |A|\,\lambda_{ll(A)} + K_A. \tag{27}$$

In addition, the optimal variational probability satisfy, for any node A,

$$q_A = C_A\exp\left(\lambda_{ll(A)} + \tfrac{K_A}{|A|} - 1\right). \tag{28}$$

Proof. Our proof is by induction. As a base case, suppose A is a leaf node. Then, because $lvs(A) = \{A\}$, $|A| = 1$, and $K_A = 0$, (27) trivially holds. Substituting (23) into (20) with $lvs(A) = \{A\}$ and $|A| = 1$, (28) also holds.

Now, suppose A is a non-leaf node and that (27) and (28) hold for all the descendants of A. Using induction hypothesis (27),

$$\begin{aligned} \sum_{L\in lvs(A)}\lambda_L &= \sum_{L\in lvs(lch(A))}\lambda_L + \sum_{L\in lvs(rch(A))}\lambda_L \\ &= |lch(A)|\,\lambda_{ll(lch(A))} + K_{lch(A)} \\ &\quad + |rch(A)|\,\lambda_{ll(rch(A))} + K_{rch(A)}. \end{aligned} \tag{29}$$

Let A' denote a child node of A. Using the induction hypothesis (28),

$$\begin{aligned} &q_{A'\to ll(A')} \\ &= \sum_{V\in\{A'\to ll(A')\}} q_V \\ &= \sum_{V\in\{A'\to ll(A')\}} \exp\left(\lambda_{ll(V)} - 1\right) C_V \exp\left(\frac{K_V}{|V|}\right) \\ &= \exp\left(\lambda_{ll(A')} - 1\right) \sum_{V\in\{A'\to ll(A')\}} C_V \exp\left(\frac{K_V}{|V|}\right) \\ &= \exp\left(\lambda_{ll(A')} - 1\right) D_{A'}. \end{aligned} \tag{30}$$

The second equality follows from the fact that $ll(V) = ll(A')$ for all $V\in\{A'\to ll(A')\}$, and the third equality is based on (26).

The sum-to-one constraints in (22) imply that

$$\begin{aligned} &q_{R\to A} + q_{lch(A)\to ll(lch(A))} \\ &\qquad = \; q_{R\to A} + q_{rch(A)\to ll(rch(A))} = 1, \end{aligned}$$

and therefore

$$q_{lch(A)\to ll(lch(A))} = q_{rch(A)\to ll(rch(A))}. \tag{31}$$

Consider the two children $lch(A)$ and $rch(A)$ for the node A and substitute the expression in (30) for these two nodes into (31). In this case

$$\begin{aligned}\exp\left(\lambda_{ll(lch(A))}-1\right)D_{lch(A)} \\ = \quad \exp\left(\lambda_{ll(rch(A))}-1\right)D_{rch(A)},\end{aligned}$$

which leads to

$$\lambda_{ll(rch(A))} = \lambda_{ll(lch(A))} + \log\left(\frac{D_{lch(A)}}{D_{rch(A)}}\right). \qquad (32)$$

Substitute (32) into (29), and we have

$$\begin{aligned}\sum_{L\in lvs(A)}\lambda_L &= \left(|lch(A)|+|rch(A)|\right)\lambda_{ll(lch(A))}+K_A \\ &= |A|\,\lambda_{ll(A)}+K_A, \qquad (33)\end{aligned}$$

where $ll(lch(A)) = ll(A)$ and (24) are used. We have now shown (27). Finally, using (23), (33), and the definition (20),

$$\begin{aligned}&q_A \\ &= \sum_{B\in mkd_A}|B|\,q(B|A) \\ &= \sum_{B\in mkd_A}\frac{|B|}{M}\exp G(B|A)\exp\left(\tfrac{1}{|A|}\sum_{L\in lvs(A)}\lambda_L-1\right) \\ &= \sum_{B\in mkd_A}\frac{|B|}{M}\exp G(B|A)\exp\left(\lambda_{ll(A)}+\tfrac{K_A}{|A|}-1\right) \\ &= C_A\exp\left(\lambda_{ll(A)}+\tfrac{K_A}{|A|}-1\right),\end{aligned}$$

which proves (28). □

B.2 Distribute-down

The root node R in the MPT starts the distribute-down phase. Using the fact that after the collect-up phase the property (28) of Proposition 1 holds for any nodes in the MPT, we can repeat the derivation in (30) for R to obtain

$$q_{R\to ll(R)} = \exp\left(\lambda_{ll(R)}-1\right)D_R = 1,$$

where the last equality is due to the the sum-to-one constraint (21). Solving this equation for $\lambda_{ll(R)}$ gives us

$$\lambda_{ll(R)} \quad = \quad 1-\log D_R.$$

Since $ll(lch(A)) = ll(A)$, traversing the MPT top-down will, for the left child of a node A, imply

$$\lambda_{ll(lch(A))} = \lambda_{ll(A)}$$

and using (32), we can for the right child compute

$$\lambda_{ll(rch(A))} = \lambda_{ll(A)} + \log\left(\frac{D_{lch(A)}}{D_{rch(A)}}\right).$$

As each node A is visited, we can now compute the optimal variational probabilities $q(B|A)$ for each $B \in mkd_A$. Using (23) and (27),

$$q(B|A) \quad = \quad \exp\left(\lambda_{ll(A)}+\frac{K_A}{|A|}-1+G(B|A)\right)\tfrac{1}{M}.$$

Finally, to ease notation in the main paper, notice that Algorithm 3 uses a node attribute denoted by λ_A to keep track of the actual value of $\lambda_{ll(A)}$ during the distribute down phase.