Fast Mean Shift with Accurate and Stable Convergence

Ping Wang pingwang@cc.gatech.edu
Dongryeol Lee dongryel@cc.gatech.edu
Alexander Gray agray@cc.gatech.edu
James M. Rehg rehg@cc.gatech.edu

College of Computing
Georgia Institute of Technology
Atlanta, GA 30332

Abstract

Mean shift is a powerful but computationally expensive method for nonparametric clustering and optimization. It iteratively moves each data point to its local mean until convergence. We introduce a fast algorithm for computing mean shift based on the dual-tree. Unlike previous speed-up attempts, our algorithm maintains a relative error bound at each iteration, resulting in significantly more stable and accurate convergence. We demonstrate the benefit of our method in clustering experiments with real and synthetic data.

1 Introduction

This paper presents a fast algorithm for computing mean shift (MS). MS is a nonparametric, iterative method for unsupervised clustering and global/local optimization. It has a wide range of applications in clustering and data analysis. For example, the computer vision community has utilized MS for (1) its clustering property in image segmentation, feature analysis (Comaniciu & Meer, 2002) and texture classification (Georgescu et al., 2003); and for (2) its quadratic optimization property in visual tracking (Collins, 2003; Comaniciu et al., 2000). MS is attractive for clustering and optimization problems due to its ability to adapt to the data distribution. However, it suffers from high computational cost - $O(NM)$ operations in each iteration (see the pseudo code in algorithm 1), where N is the size of reference data and M is the size of query data.[1] Therefore, applications of MS have either used fairly small datasets (Comaniciu & Meer, 2002; Comaniciu et al., 2000), or avoided updating all of the points in the query set (e.g. a local optimization process is started from a single query). Alternatively, some fast approximations of MS have been proposed (Georgescu et al., 2003; Yang et al., 2003). While these methods have been shown experimentally to have high efficiency, they suffer from three major limitations: 1) Improved Fast Gauss Transform-based MS (Yang et al., 2003) (IFGT-MS) can use only the Gaussian kernel; 2) Both IFGT-MS and Locality Sensitive Hashing-based MS (Georgescu et al., 2003) (LSH-MS) have many tuning parameters; 3) Both methods lack an explicit error bound for the vector approximation required in each iteration of MS.

We believe speedup techniques should ensure both the *accuracy* and the *stability* of the approximation. "Accuracy" means that the approximation has a guaranteed error bound. "Stability" means that the approximation should return almost identical results over different runs. Nondeterminism typically stems from randomized initialization, and approximation methods which lack reliable error control mechanisms can be sensitive to these initial values, resulting in a significant variation in their outputs for a fixed input. In this paper, we introduce an acceleration technique that achieves both accuracy and stability – Dual-tree (Gray & Moore, 2001) based mean shift (DT-MS). DT-MS can use any kernel, has a user-specified *relative* error tolerance on each computation of $m(x_q)$ (eq. 1) and requires no other parameter tuning. Our experiments on datasets with dimensionality ranging from 2 to 16 and size ranging from 6000 to 68040 demonstrate the superiority of DT-MS over IFGT-MS and LSH-MS in terms of speed, accuracy, and stability. This paper makes three contributions:

1. Introduction of DT-MS, a novel approximation method for MS which is fast, accurate, and stable
2. An extension of the dual-tree method (introduced in (Gray & Moore, 2001) for positive scalar targets) to the signed mean vector case. To achieve this extension, we have developed (i) A new global error bound (Theorem 1) for pruning nodes, (ii) A novel finite difference approximation for the

[1] We follow the terminology used in (Gray & Moore, 2001; Lee & Gray, 2006)

signed mean vector, and (iii) A new algorithm for updating bounds on the L1 norm.

3. The first experimental comparison of fast MS algorithms on a standardized dataset. We highlight for the first time the issue of stability in MS approximation.

1.1 Mean shift

Algorithm 1 Mean shift

Input: X_Q, X_R, ϵ (the pre-defined distance threshold)
Output: The converged query set

MEAN-SHIFT$(X_Q, X_R, w(\cdot), K_h(\cdot), \epsilon)$
  $dist = 100 * ones(N_Q, 1)$ {initialize distance vector}
  while max(dist)$\geq \epsilon$ **do**
    for each $x_q \in X_Q$ **do**
      $m(x_q) = \frac{\sum_{x_r \in X_R} K_h(x_r - x_q) w(x_r) x_r}{\sum_{x_r \in X_R} K_h(x_r - x_q) w(x_r)}$
      $dist(x_q) = \|m(x_q) - x_q\|_2$ {distance can be of any norm}
      $x_q \leftarrow m(x_q)$
  Return X_Q

Mean shift (Cheng, 1995; Fukunaga & Hostetler, 1975) moves each query to its local mean until convergence (see algorithm 1). Let X_R denote the reference data set, and X_Q denote the query data set. $X_R \subset R^D, X_Q \subset R^D, x_r \in X_R, x_q \in X_Q$. The mean of query x_q is defined as:

$$m(x_q) = \frac{h(x_q)}{f(x_q)} = \frac{\sum_{x_r \in X_R} K_h(x_r - x_q) w(x_r) x_r}{\sum_{x_r \in X_R} K_h(x_r - x_q) w(x_r)} \quad (1)$$

where $w : R^D \rightarrow R$ is a weight function which can vary with x_r and time. In this paper we set $w(x_r) = 1$ for all x_r. The kernel function $K_h : R^D \rightarrow R$ has profile $k : [0, \infty] \rightarrow R$, such that $K_h(x) = k(\|\frac{x}{h}\|^2)$, where h is the bandwidth and k is monotonically nonincreasing, nonnegative and piecewise continuous (Cheng, 1995). Cheng (Cheng, 1995) proves that MS is a step-varying gradient ascent optimization. (Fashing & Tomasi, 2005) shows MS is equivalent to Newton's method with piecewise constant kernels, and is a quadratic bound maximization for all kernels.

1.2 Previous acceleration methods

The denominator of Equation 1 is a kernel density estimate (KDE) while the numerator is a weighted vector sum. The key challenge in accelerating MS is to approximate this ratio. Since MS is closely related to KDE, most speedup methods focus on fast approximation of $f(x_q)$ or fast search of the neighborhood around x_q (defined by the bandwidth). The two most important related works are the Improved Fast Gauss Transform-based MS (IFGT-MS) (Yang et al., 2003) and Locality Sensitive Hashing-based MS (LSH-MS) (Georgescu et al., 2003).

IFGT-MS is only applicable to the Gaussian kernel. IFGT-MS first clusters the reference points using the k-center algorithm, and loops over each query point/reference cluster pair, evaluating the precomputed (truncated) Taylor coefficients for clusters that are within the distance threshold from the query point. IFGT-MS requires a significant amount of manual parameter tuning for good performance.[2]

LSH (Gionis et al., 1999) has been popular recently for k-nearest neighbor (k-NN) search in high dimensions. It performs L random partitions of the data set. For each partition, a boolean vector of size K is generated for each datum, thus the data set are indexed into 2^K cells. Each query x_q belongs to L cells simultaneously. The union of the L cells is returned as the neighborhood of x_q. The choice of (K, L) is critical. The training process (Georgescu et al., 2003) selects the (K, L) that minimizes the query time on a subset of X_Q and satisfies a user-specified k-NN distance approximation error bound, which unfortunately is not directly related to the approximation of $m(x_q)$.

Unlike these two previous speedup techniques, our dual-tree based mean shift method imposes a relative error bound on the entire mean vector $m(x_q)$. Achieving this stronger accuracy guarantee requires more computation than other approaches, but our experimental results demonstrate that DT-MS achieves much more stable convergence results while still providing a significant speedup. In particular, DT-MS is faster than IFGT-MS, LSH-MS and naive MS in speed and convergence when using the Epanechnikov kernel.

2 Dual-tree based mean shift

Dual-tree methodology and Dual-tree KDE. The dual-tree framework (Gray & Moore, 2001) generalizes all of the well-known node-to-node algorithms (Barnes & Hut, 1986; Greengard & Rokhlin, 1987; Appel, 1985; Callahan & Kosaraju, 1995). Dual-tree based algorithms have the computational advantage of considering a region of query points with a region of reference points, whereas IFGT-based and LSH-based methods consider one query point at a time. The DT framework can use adaptive data structures such as *kd*-trees (Freidman et al., 1977) and ball-trees (Chavez et al., 2001), and are bichromatic (can specialize for differing query and reference sets). The idea is to represent both the query points and the reference points with separate trees, denoted as Qtree (Query tree) and

[2]The important parameters are: p-polynomial order, K_c-number of partitions, e-ratio of the cutoff radius to the bandwidth, which determines the absolute error bound. We follow the authors' suggestion: $K_c = \sqrt{N_R}$; we gradually increase e and p as we tune them to achieve comparable result to DT-MS(Epan.), though the authors recommend $e = 3$ and $p \leq 3$.

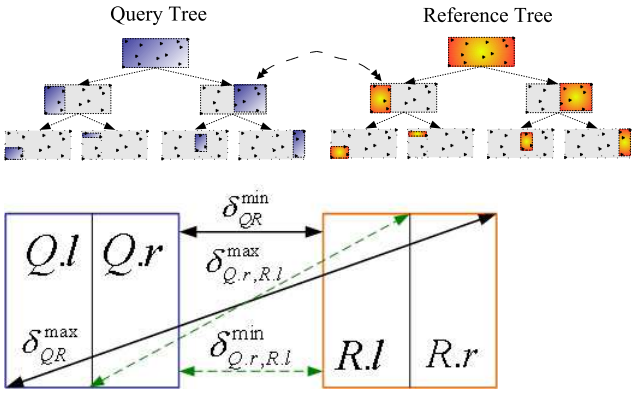


Figure 1: Dual-tree illustration. Top: A kd-tree partitions 2-dimensional points. Each node in the kd-tree records the bounding hyper-rectangle for the subset of the data it contains (highlighted in color). In dual-tree recursion, a pair of nodes chosen from the query tree and the reference tree is compared at each step. Bottom: Zoom-in on the two nodes (Q, R) which are linked by a dashed arc in the top figure. Minimum and maximum distance bounds are illustrated.

Dualtree(Q, R)
  if **Can-approximate**(Q, R, τ),
    Approximate(Q, R), return
  if leaf(Q) and leaf(R), **DualtreeBase**(Q, R)
  else **Dualtree**$(Q.l, R.l)$, **Dualtree**$(Q.l, R.r)$,
    Dualtree$(Q.r, R.l)$, **Dualtree**$(Q.r, R.r)$

Figure 2: Generic structure of dual-tree.

Rtree (Reference tree). Then node R's (a node from Rtree) kernel sum contribution to node Q (a node from Qtree) is recursively updated by first comparing Q and R, and then possibly comparing the pairs of their children. The method can be viewed as a simultaneous traversal of two trees.

Figure 2 shows the generic structure of a dual-tree algorithm. τ denotes the user-specified error tolerance. We will explain the method in the context of KDE, while the readers should keep in mind that all the functions can be modified to suit other N-body problems. **Can-approximate**(Q,R,τ) computes the maximal and minimal distances between Q and R, δ_{QR}^{max} and δ_{QR}^{min} (illustrated by Figure 1), and checks whether the kernel values $K_h(\delta_{QR}^{max})$ and $K_h(\delta_{QR}^{min})$ are close to each other. If they are, **Approximate** assumes the midpoint kernel value for all points in R, i.e., R's contribution to Q is linearly approximated as $N_R\bar{K}_h$, where $\bar{K}_h = (K_h(\delta_{QR}^{max}) + K_h(\delta_{QR}^{min}))/2$ and N_R is the number of reference points in R. When τ is chosen as a relative error bound ($|\hat{f}(x_q) - f(x_q)|/|f(x_q)| \leq \tau$), **Can-approximate** in Dual-tree KDE returns true if $K_h(\delta_{QR}^{min}) - K_h(\delta_{QR}^{max}) \leq 2\tau f_Q^{min}/N$ (which we call the *pruning criterion*), where $N = |X_R|$ is the size of the reference dataset. Otherwise, the algorithm recurses on each possible child-pair of Q, R to consider a smaller subset of points in both datasets, until it either encounters a prunable node-pair or computes the non-prunable leaf-leaf pair using **DualtreeBase**.

The geometric intuition behind dualtree algorithms is that the distance between Q and R is bounded by δ_{QR}^{max} and δ_{QR}^{min}. Therefore the lower and upper bounds for $f(x_q)$ are $f_Q^{min} = N_R K_h(\delta_{QR}^{max})$ and $f_Q^{max} = N_R K_h(\delta_{QR}^{min})$. The bounds are tightened in every recursion when smaller subsets of the query and the reference datasets are compared. The following relationship always holds:

$$\delta_{QR}^{min} \leq \delta_{Q.l/r,R.l/r}^{min} \leq \delta_{Q.l/r,R.l/r}^{max} \leq \delta_{QR}^{max}$$

where $(Q.l/r, R.l/r)$ represents any combination of a child node from Q and a child node from R. This inequality guarantees that f_Q^{min} increases and f_Q^{max} decreases, until pruning occurs. The distance bounds between the two roots (i.e. the bounds for the query set and reference set) are used for f_Q^{max} and f_Q^{min}'s initialization. The error in the approximation of $f(x_q)$ is due to the pruning.

2.1 Dual-tree mean shift

As in the case of KDE, the DT methodology can be applied to the mean shift computation because it computes $m(x_q) = h(x_q)/f(x_q)$ (which involves summations of weighted pairwise kernel values) in every iteration of MS. In every iteration of MS, a query tree is rebuilt because x_q is updated as $m(x_q)$, while the reference tree remains fixed. In contrast to KDE, mean shift involves the numerator $h(x_q)$ which is a weighted vector sum and $f(x_q)$ which is in the form of KDE. Here we ensure a relative error bound in L_1 norm (other norms are applicable too) on the mean vector $m(x_q)$ directly: $|\hat{h}(x_q)/\hat{f}(x_q) - h(x_q)/f(x_q)|_1/|h(x_q)/f(x_q)|_1 \leq \tau$. $\hat{h}(x_q)$ and $\hat{f}(x_q)$ denote approximations to the numerator and the denominator, respectively.

This error bound brings up three questions: 1) How to distribute the global error bound τ into the local node-node pruning? 2) How to maintain the bounds for the vector? 3) How to apply these bounds in approximation? We answer these questions below.

Maintaining the bounds. The distance bounds between Q and R, and hence the bounds on $f(x_q)$ and $h(x_q)$, are used in the linear approximation and error bounds distribution. Unlike KDE, the vector term takes on both positive and negative values, so we need to keep track of them separately. For each query point x_q and for each query node Q, we maintain dimension-wise lower and upper bounds for the

numerator: $h^{min}_{q,d}$ and $h^{max}_{q,d}$ for x_q, and $h^{min}_{Q,d}$ and $h^{max}_{Q,d}$ for Q for $1 \le d \le D$, denoted h^{min}_q, h^{max}_q, h^{min}_Q, and h^{max}_Q collectively as a vector. Similarly, the lower and the upper bounds for the denominator can be maintained: f^{max}_q, f^{max}_q, f^{min}_Q, and f^{max}_Q. We define the following sums of directional coordinate values for all reference points: $S^A_d = \sum_{x_r \in X_R} |x_r(d)|$, $S^+_d = \sum_{x_r \in X_R, x_r(d)>0} x_r(d)$, $S^-_d = \sum_{x_r \in X_R, x_r(d)<0} x_r(d)$, and the sums for reference points belonging to a given reference node R: $S^+_{R,d} = \sum_{x_r \in R, x_r(d)>0} x_r(d)$, $S^-_{R,d} = \sum_{x_r \in R, x_r(d)<0} x_r(d)$, $S_{R,d} = S^+_{R,d} + S^-_{R,d}$, $S^A_{R,d} = \sum_{x_r \in R} |x_r(d)|$, where $x_r(d)$ is the d^{th} coordinate of x_r, $d = 1, ..., D$. After the query and the reference trees are built, we initialize the lower and the upper bounds for the numerator and the denominator for all x_q's and all Q's as follows:

$$h^{min}_{Q,d} = h^{min}_{q,d} = S^-_d K_h(\delta^{min}_{root}) + S^+_d K_h(\delta^{max}_{root})$$
$$h^{max}_{Q,d} = h^{max}_{q,d} = S^+_d K_h(\delta^{min}_{root}) + S^-_d K_h(\delta^{max}_{root})$$
$$f^{min}_Q = f^{min}_q = N K_h(\delta^{max}_{root})$$
$$f^{max}_Q = f^{max}_q = N K_h(\delta^{min}_{root})$$

where δ^{min}_{root} and δ^{max}_{root} denote the min/max distances between the root node of the query tree and the root node of the reference tree. The bounds above will be maintained and updated at all times, such that for any query node Q, we have: $h^{min}_{Q,d} \le h_q(d) \le h^{max}_{Q,d}$ for $1 \le d \le D$ and $f^{min}_Q \le f(x_q) \le f^{max}_Q$ for any $x_q \in Q$.

Specifying the Approximate function. Given a query node Q and a reference node R, we can approximate R's contribution to the numerator as $h_R(x_q)$ and to the denominator as $f_R(x_q)$ for all $x_q \in Q$ by the linear finite difference approximation with the bounds: $\hat{h}_{R,d}(x_q) = (h^{min}_{R,d} + h^{max}_{R,d})/2 = ((S^-_{R,d} K_h(\delta^{min}_{QR}) + S^+_{R,d} K_h(\delta^{max}_{QR})) + (S^+_{R,d} K_h(\delta^{min}_{QR}) + S^-_{R,d} K_h(\delta^{max}_{QR})))/2 = S_{R,d}\bar{K}_h, d = 1, ..., D$ and to the denominator $f_R(x_q)$ by: $f_R(x_q) = N_R \bar{K}_h$. During recursion, the bounds are tightened as:

$$\begin{aligned} h^{min}_{Q,d} +&= S^-_{R,d}(K_h(\delta^{min}_{QR}) - K_h(\delta^{min}_{root})) \\ &\quad + S^+_{R,d}(K_h(\delta^{max}_{QR}) - K_h(\delta^{max}_{root})) \\ h^{max}_{Q,d} +&= S^+_{R,d}(K_h(\delta^{min}_{QR}) - K_h(\delta^{min}_{root})) \\ &\quad + S^-_{R,d}(K_h(\delta^{max}_{QR}) - K_h(\delta^{max}_{root})) \\ f^{min}_Q +&= N_R(K_h(\delta^{max}_{QR}) - K_h(\delta^{max}_{root})) \\ f^{max}_Q +&= N_R(K_h(\delta^{min}_{QR}) - K_h(\delta^{min}_{root})) \end{aligned}$$

Specifying the Can-approximate function. The global relative error bound τ is satisfied by ensuring a local pruning criterion in the function **Can-approximate**. Simple algebraic manipulation reveals that: $|\hat{h}(x_q)/\hat{f}(x_q) - h(x_q)/f(x_q)|_1 / |h(x_q)/f(x_q)|_1 \le \tau \Leftrightarrow |f(x_q)\hat{h}(x_q) - \hat{f}(x_q)h(x_q)|_1 \le \tau \hat{f}(x_q)|h(x_q)|_1$. Theorem 1 derives the pruning condition based on the triangle inequality, which shows how to satisfy the right hand side of the above relationship. The condition specifies the **Can-approximate** function for DT-MS to guarantee the global error bound. Multipole expansion is also used for more pruning (Lee & Gray, 2006). We define some notations first. Given a query node Q, the bounds for $h(x_q)$ in L_1 norm for any $x_q \in Q$ are defined as: $L_Q = \sum_{d=1}^{D} I(h^{min}_{Q,d}, h^{max}_{Q,d})$, $U_Q = \sum_{d=1}^{D} \max(|h^{min}_{Q,d}|, |h^{max}_{Q,d}|)$ where $I(a,b) = \begin{cases} a, a \ge 0 \\ -b, b < 0 \\ 0, otherwise \end{cases}$ for $a, b \in R$, such that $L_Q \le |h(x_q)|_1 \le U_Q$ for all $x_q \in Q$.

Theorem 1. *Given a query node Q and a reference node R, if R's contribution to all $x_q \in Q$ is approximated as $\hat{h}_{R,d}(x_q) = (S_{R,d}(K_h(\delta^{max}_{QR}) + K_h(\delta^{min}_{QR})))/2, d = 1, ..., D$ and $\hat{f}_R(x_q) = N_R \bar{K}_h$, the following local pruning criterion must be enforced to guarantee the global relative error bound τ: $K_h(\delta^{min}_{QR}) - K_h(\delta^{max}_{QR}) \le \min\{\frac{\tau f^{min}_Q L_Q}{N U_Q}, \frac{\tau L_Q}{\sum_d S^A_d}\}$*

Proof: *If the local pruning criterion is met and R's contribution is approximated, we have $|\hat{h}_{R,d}(x_q) - h_{R,d}(x_q)| \le (h^{max}_{R,d} - h^{min}_{R,d})/2 = S^A_{R,d}(K_h(\delta^{min}_{QR}) - K_h(\delta^{max}_{QR}))/2, d = 1, ..., D$ and $|\hat{f}_R(x_q) - f(x_q)| \le N_R(K_h(\delta^{min}_{QR}) - K_h(\delta^{max}_{QR}))/2$. Given $x_q \in Q$, suppose $\hat{h}(x_q)$ and $\hat{f}(x_q)$ were computed using reference nodes $R = \{R_1, R_2, \cdots R_k\}$. By the triangle inequality,*

$$\begin{aligned} &|f(x_q)\hat{h}(x_q) - \hat{f}(x_q)h(x_q)|_1 \\ \le& |\hat{h}(x_q)|_1 |f(x_q) - \hat{f}(x_q)| + \hat{f}(x_q)|(\hat{h}(x_q) - h(x_q))|_1 \\ \le& \sum_R |\hat{h}(x_q)|_1 (f_R(x_q) - \hat{f}_R(x_q))| + \hat{f}(x_q)|\hat{h}_R(x_q) - h_R(x_q)|_1 \\ \le& |\hat{h}(x_q)|_1 \sum_R N_R(K_h(\delta^{min}_{QR}) - K_h(\delta^{max}_{QR}))/2 + \\ &\hat{f}(x_q) \sum_R \sum_d S^A_{R,d}(K_h(\delta^{min}_{QR}) - K_h(\delta^{max}_{QR}))/2 \\ \le& \sum_R \left[|\hat{h}(x_q)|_1 \frac{\tau N_R f^{min}_Q L_Q}{2 N U_Q} + \hat{f}(x_q) \frac{\tau \sum_d S^A_{R,d} L_Q}{2 \sum_d S^A_d} \right] \\ \le& \tau \hat{f}(x_q)|h(x_q)|_1 \quad \square \end{aligned}$$

3 Experiments and discussions

We have two tasks in the experiments. One is to compare the speedup of DT-MS over the naive MS. The other is to compare the speed, accuracy and stability in convergence among DT-MS, IFGT-MS and LSH-MS. We used the IFGT-MS and LSH-MS codes provided by the authors. LSH uses an Epanechnikov-like kernel. So we tested both the Gaussian kernel ($K_h(x_q - x_r) = e^{-\|x_q - x_r\|^2/2h^2}$) and Epanechnikov kernel ($K_h(x_q - x_r) = 1 - \|x_q - x_r\|^2/h^2$ if $\|x_q - x_r\| \le h$,

Algorithm 2

MS-DUALTREE(Q, R)

if !leaf(Q) **then**

for each dimension d **do**

$h^{min}_{Q,d} = \max\left(h^{min}_{Q,d}, \min\left(h^{min}_{Q.l,d}, h^{min}_{Q.r,d}\right)\right)$

$h^{max}_{Q,d} = \min\left(h^{max}_{Q,d}, \max\left(h^{max}_{Q.l,d}, h^{max}_{Q.r,d}\right)\right)$

$f^{min}_{Q} = \max\left(f^{min}_{Q}, \min\left(f^{min}_{Q.l}, f^{min}_{Q.r}\right)\right)$

$f^{max}_{Q} = \min\left(f^{max}_{Q}, \max\left(f^{max}_{Q.l}, f^{max}_{Q.r}\right)\right)$

$\Delta K = K_h(\delta^{min}_{QR}) - K_h(\delta^{max}_{QR})$

$\Delta K_{min} = K_h(\delta^{min}_{QR}) - K_h(\delta^{min}_{root})$

$\Delta K_{max} = K_h(\delta^{max}_{QR}) - K_h(\delta^{max}_{root})$

$dl_f = N_R \Delta K_{max}$, $du_f = N_R \Delta K_{min}$.

if $\Delta K \leq \min\{\frac{\tau f^{min}_Q L_Q}{N U_Q}, \frac{\tau L_Q}{\sum_d S^A_d}\}$ **then**

for each dimension d **do**

$dl_{h_d} = S^-_{R,d}\Delta K_{min} + S^+_{R,d}\Delta K_{max}$

$du_{h_d} = S^+_{R,d}\Delta K_{min} + S^-_{R,d}\Delta K_{max}$

$h^{min}_{Q,d} += dl_{h_d}$, $h^{max}_{Q,d} += du_{h_d}$

$f^{min}_Q += dl_f$, $f^{max}_Q += du_f$

else if leaf(Q) and leaf(R) **then**

MS-DUALTREEBASE(Q, R)

else

MS-DUALTREE$(Q.l, R.l)$, MS-DUALTREE$(Q.l, R.r)$

MS-DUALTREE$(Q.r, R.l)$, MS-DUALTREE$(Q.r, R.r)$

MS-DUALTREEBASE(Q, R)

for each $x_q \in Q$ **do**

for each $x_r \in R$ **do**

$c = K_h(\|x_q - x_r\|)$, $f^{min}_q += c$, $f^{max}_q += c$, $f^{min}_q -= N_R K_h(\delta^{max}_{root})$, $f^{max}_q -= N_R K_h(\delta^{min}_{root})$

for each dimension d **do**

$h^{min}_{q,d} += c \cdot x_r(d)$, $h^{max}_{q,d} += c \cdot x_r(d)$,

$h^{min}_{q,d} -= (S^-_{R,d}K_h(\delta^{min}_{QR}) + S^+_{R,d}K_h(\delta^{max}_{QR}))$,

$h^{max}_{q,d} -= (S^-_{R,d}K_h(\delta^{max}_{QR}) + S^+_{R,d}K_h(\delta^{min}_{QR}))$

$f^{min}_Q = \min_{q\in Q} f^{min}_q$, $f^{max}_Q = \max_{q\in Q} f^{max}_q$

for each dimension d **do**

$h^{min}_{Q,d} = \min_{q\in Q} h^{min}_{q,d}$, $h^{max}_{Q,d} = \max_{q\in Q} h^{max}_{q,d}$

otherwise 0) for DT-MS. [3] X_Q is initialized as X_R for all the datasets.

Speedup of DT-MS over the naive MS. We chose image segmentation as a representative clustering task. The goal of image segmentation is to cluster pixels into several distinct groups. We followed (Yang et al., 2003)'s approach of segmentation, where each datum represents the normalized CIE LUV color space for each pixel and the labels are assigned to the pixels by applying a k-means algorithm to the converged X_Q returned by MS. In other words, one image forms one dataset $X_R \subset R^3$ and the size of X_R equals the number of pixels in the image. We applied DT-MS and the naive MS to 10 test images from the Berkeley segmentation dataset.[4] The image size is 481×321, i.e. $N = 154401$. The speedup is an order of magnitude in 7 images and two orders of magnitude in one image. A summary of running time and speedups for a set of representative images is given in table 1. Segmentation results for these images are shown in figure 3.

Table 1: Running time (in seconds) of DT-MS and naive-MS with the Gaussian kernel. N_{it} is the number of iterations in MS, $\tau = 0.1$, $\epsilon = 0.01$.

Images	Speedup	Time(DT/Naive)	N_{it}	h_g
Fox	44.74	155.22/6944.54	1/1	0.0166
Snake	136.51	39.71/5420.36	1/1	0.0065
Cowboys	1.75	3059.38/5352.24	2/1	0.0172
Vase	19.06	300.66/5729.44	1/1	0.0163
Plane	32.86	187.54/6162.65	1/1	0.0102
Hawk	48.88	127.35/6224.48	1/1	0.0136

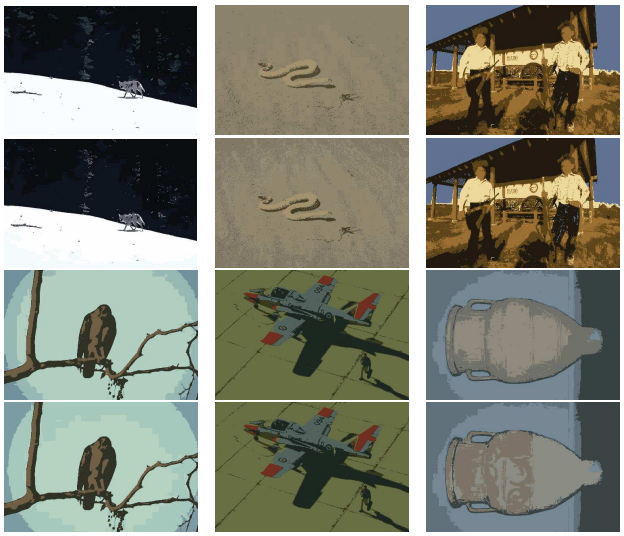


Figure 3: Selected segmentation results. For each image pair, top: DT-MS, bottom: naive-MS.

Comparison among DT-MS, IFGT-MS and LSH-MS. The speed, accuracy and stability in convergence of the three algorithms are empirically evaluated on synthetic and real datasets. The accuracy of convergence is evaluated by the relative error in L_1 norm as $|\hat{m}(x_q) - m(x_q)|_1 / |m(x_q)|_1$, where $m(x_q)$ is the final convergence of x_q using naive MS and $\hat{m}(x_q)$ is produced by the approximation algorithm. Stable algorithms should exhibit low variance in the converged point positions over multiple runs. We demonstrate stability by displaying the results from different runs.

Experiment 1: We first compare the three methods on two typical images for segmentation (figure 4). Table 2 shows the average running time and accuracy of convergence (represented in relative error) for two images. The results over different runs are not shown because the variations are mostly cancelled by apply-

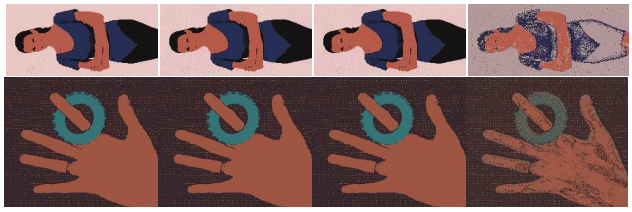
[3] The optimal bandwidth h_g for the Gaussian kernel is automatically selected by DT-KDE using leave-one-out least square cross validation. The optimal bandwidth h_e for the Epanechnikov kernel is determined as $h_e = 2.214 * h_g$ according to the equivalent kernel rescaling in Table 6.3 in (Scott, 1992).

[4] http://www.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/

ing k-means to group the converged pixels. LSH's running time has two parts: MS+(K, L) training. We include the training time because it is required for every dataset, and (K, L) training comprises the majority of the running time. DT-MS(Epan.) is the best in both speed and accuracy. The average number of iterations for IFGT-MS and DT-MS is very small (1 to 2) because the normalized CIE LUV space is sparse for the tested images.[5] Therefore, after a few iterations the query point will have no neighbors within distance h_g or h_e to itself. IFGT-MS is faster than DT-MS(Gauss.), but has a slightly higher relative error. In the image segmentation case, such a difference can be ignored.

Table 2: Running time (in seconds) and relative error averaged over 3 runs. Top row: woman.ppm with $h_g = 0.027$, $h_e = 0.0598$. Bottom row: hand.ppm with $h_g = 0.0186$, $h_e = 0.0412$. $\epsilon = 0.01$, $\tau = 0.1$ for both images. IFGT-MS: $e = 4$, $p = 3$. DT-MS(Epan.) gives the best result in terms of speed and accuracy.

Alg.	Time	Rel. Err.
naive/DT(Epan.)	55.07/0.35	0/0
naive/DT(Gauss.)	194.6/2.08	0/0
IFGT	0.47	0.0093
LSH	$0.21 + 266.95$	0.3154
naive/DT(Epan.)	308.74/0.92	0/0
naive/DT(Gauss.)	1258.4/5.81	0/0
IFGT	1.24	$8.245e-5$
LSH	$0.52 + 621.15$	0.052501



DT(Epan.) IFGT DT(Gauss.) LSH

Figure 4: Image segmentation results. Size of woman:116×261. Size of hand: 303×243.

Experiment 2: The segmentation is obtained by applying k-means to group the converged points. This is potentially a confounding factor, since k-means can compensate for poorly-converged points. Therefore we synthesized a dataset where k-means cannot work well, but MS can still find the correct modes. This experiment and the next one demonstrate MS's ability in noise reduction of the dataset to help reveal its intrinsic dimensionality (Fukunaga & Hostetler, 1975). Testing data containing 6000 2-D points was generated by adding Gaussian noise to sampled points on two intersected half circles (the blue dots in figure 5), viewed as 2 c-shape clusters. Table 3 and figure 5 again show that DT-MS(Epan.) achieves the best overall result among speed, accuracy and stability. IFGT-MS is slightly faster than DT-MS(Epan.) with slightly bigger variations in different runs. Naive-MS(Gauss.) runs faster than the DT-MS(Gauss.) for this dataset. This is because when the data points are not well clustered under certain bandwidth, the pruning does not happen frequently enough to cancel the additional cost for distance computation per each query/reference node pair.

[5]IFGT-MS often returns NaN because absolute error pruning creates zeroes in the numerator and the denominator of $m(x_q)$.

Table 3: Running time (in seconds) and relative error of convergence on 2-C-shape data averaged over 3 runs. $h_e = 8.856, h_g = 4$, $\epsilon = 0.2$, $\tau = 0.01$ for Epanechnikov kernel and $\tau = 0.001$ for Gaussian kernel. IFGT-MS: $e = 8$, $p = 30$. N_{it} is N/A for LSH-MS because it uses a different loop order from IFGT-MS and DT-MS.

Alg.	Time	N_{it}	Rel. Err.
naive/DT(Epan.)	38.56/11.89	22/22	0/1.8e-4
naive/DT(Gauss.)	190.21/207.6	26/26	0/1.16e-2
IFGT	10.74	25	0.015
LSH	0.58+279.73	N/A	0.1174

Table 4: Running time (in seconds) and relative error of convergence on noisyswissroll.ds averaged over 3 runs. $h_e = 4.06, h_g = 1.833$, $\epsilon = 0.02$, $\tau = 0.1$ for Epanechnikov kernel and $\tau = 0.01$ for Gaussian kernel. IFGT-MS: $e = 9$, $p = 20$. DT-MS(Epan.) is best in both speed and accuracy.

Alg.	Time	N_{it}	Rel Err.
naive/DT(Epan.)	992.39/148.16	44/44	0/1.5e-4
naive/DT(Gauss.)	4314.85/3116.9	51/51	0/0.025
IFGT	240.05	20	0.0573
LSH	3.81+713.58	N/A	0.2137

Experiment 3: Swissroll data with additive Gaussian noise($N(0, 4)$) (figure 6).[6] $N = 20000$, $D = 3$. Though the dataset size is larger and the dimension is bigger, DT-MS(Epan.) still achieves best performance in speed, accuracy and stability (table 4 and figure 7).

Experiment 4: High-dimensional data ($N = 68040$, $D = 16$).[7] The running time and relative error of convergence are shown in table 5. DT-MS(Epan.) again achieves the best performance in both speed and accuracy. We could improve IFGT-MS's relative error further by increasing p and e (which will increase the running time), but the algorithm failed due to memory limit. Even at its current level of accuracy, IFGT is slower than DT-MS(Epan.). DT-MS(Gauss.) is slower than the naive case for the same reason as explained in Experiment 2.

[6]http://isomap.stanford.edu/datasets.html

[7]http://www.ics.uci.edu/ kdd/databases/CorelFeatures/CorelFeatures.data.html

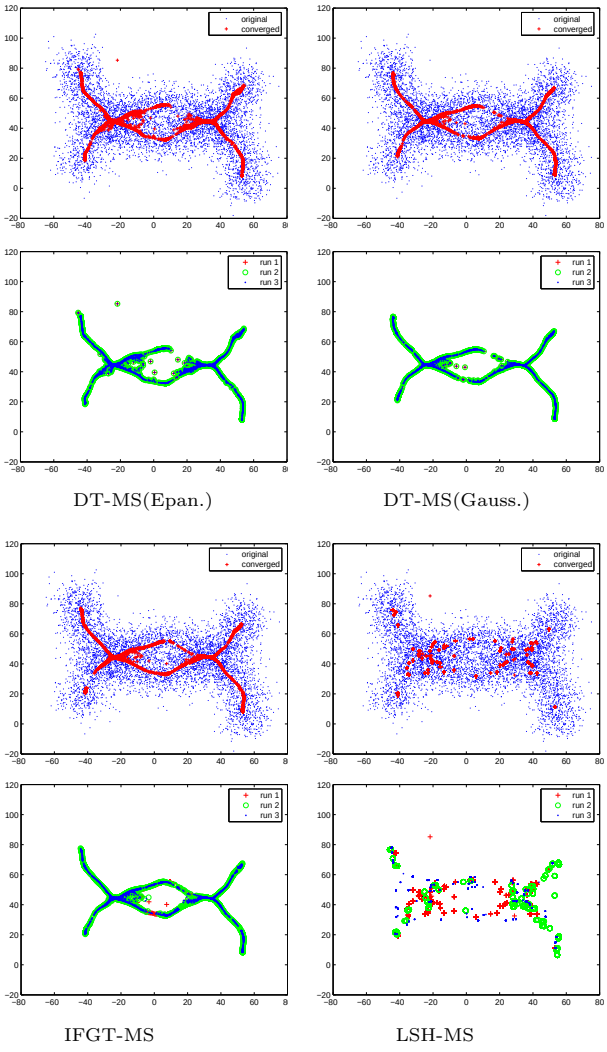


Figure 5: Accuracy/stability of convergence. Converged queries (red) imposed on the original data (blue). Stability illustrated by the converged queries of 3 runs(indicated by 3 colors).

Summary of the Experiments. DT-MS with the Epanechnikov and the Gaussian kernels provides consistent and accurate convergence, and is faster than naive MS (by two orders of magnitude in some cases with both kernels). DT-MS(Epan.) returns almost zero relative error when compared to the naive case. DT-MS(Gauss.) also returns zero relative error for well-clustered data (table 2). For less well-clustered data, DT-MS(Gauss.) returns slightly bigger relative error than DT-MS(Epan.), but the error is small enough to be safely ignored (table 4, 5).

DT-MS(Epan.) is always faster than DT-MS(Gauss.) in our datasets, because the Epanechnikov kernel has finite extent and can be pruned more frequently than the Gaussian kernel with zero approximation error (Gray & Moore, 2001). The Epanechnikov kernel

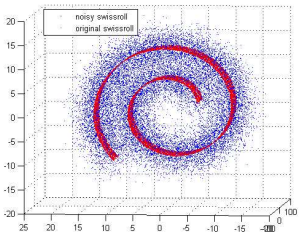


Figure 6: Noisy swissroll (in blue) and the clean swissroll (in red).

Table 5: Running time (in seconds) and relative error of convergence on high-dimensional data averaged over 3 runs. $h_e = 0.49, h_g = 0.2212$, $\epsilon = 0.02$, $\tau = 0.1$ for both the Epanechnikov and Gaussian kernels. IFGT-MS: $e = 9, p = 7$. DT-MS(Epan.) gives the best result in terms of speed and accuracy.

Alg.	Time	N_{it}	Rel Err.
naive/DT(Epan.)	3515.74/516.34	7/7	0/0
naive/DT(Gauss.)	24189.8/39680	17/17	0/7.6e-6
IFGT	1260.56	2	0.2539
LSH	390.7+1026.9	N/A	0.4605

is also optimal in the sense of minimizing asymptotic mean integrated squared error, so it is statistically preferred.

For some datasets the relative error for DT-MS(Gauss.) is bigger than τ (table 3, 4). This is because τ controls the relative error of $\hat{m}(x_q)$ in one iteration of MS, not in the converged result. Thus, the approximated trajectory of a point may not match the one computed by the naive method.

DT-MS(Epan.) always dominates IFGT-MS and LSH-MS in speed, accuracy and stability, and requires no parameter tuning. IFGT-MS can achieve very good speedup and accuracy, if the parameters are set correctly (table 3 and figure 5). LSH-MS with an adequate (K, L) pair is very fast. However, training (K, L) takes much time and is dependent on the dataset and the search range of (K, L). Both IFGT-MS and LSH-MS require trial-and-error, manual tuning of parameters, and also require much more storage than DT-MS.

4 Conclusions

This paper presents a new algorithm DT-MS for accelerating mean shift. It extends the dual-tree method to the fast approximation of the signed mean vector in MS. Our experiments have demonstrated its fast, accurate and stable approximation of MS. Especially with the Epanechnikov kernel, DT-MS scales quite well to larger datasets with higher dimensions. It has the best performance in terms of speed, accuracy and stability in comparison to IFGT-MS, LSH-MS and DT-MS(Gauss.).

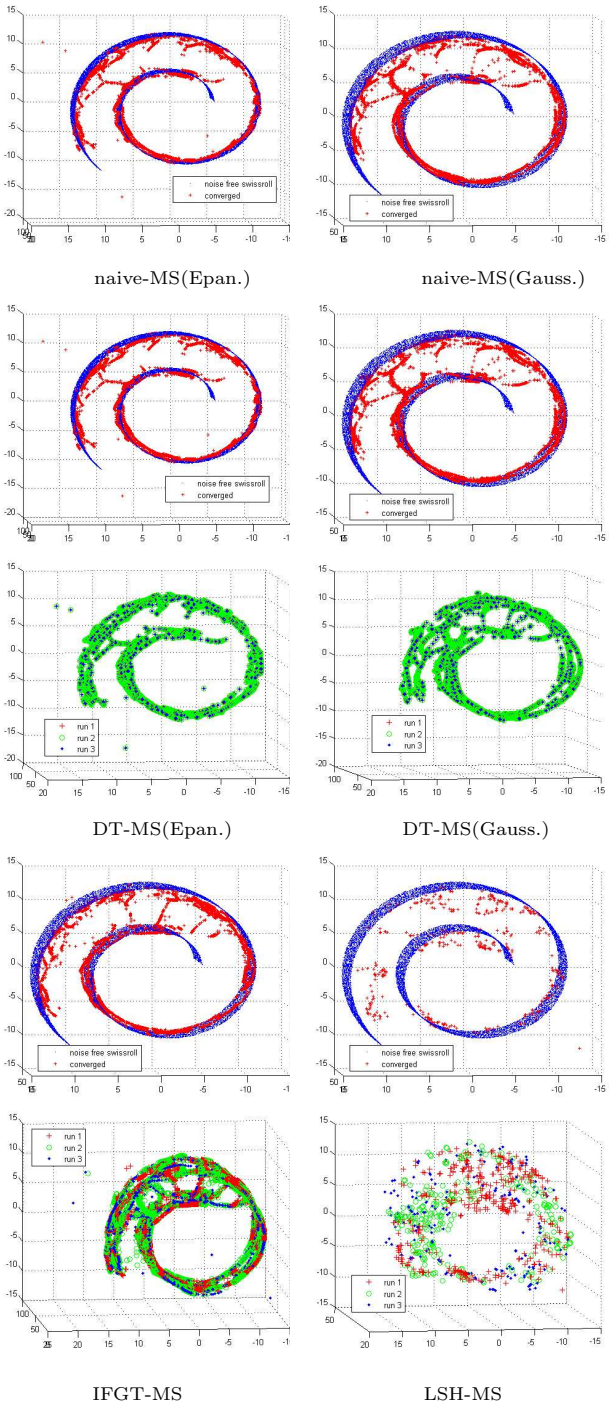


Figure 7: Accuracy and stability of convergence: For clarity all the MS results (red) are imposed on the original swissroll (blue). Stability is illustrated by the converged queries of 3 runs (indicated by 3 colors). For comparison, the results obtained by the naive MS are shown in the top row.

Acknowledgements

Dongryeol Lee is supported by a Dept. of Homeland Security Fellowship. This material is based upon work which was supported in part by the NSF under IIS-0433012.

References

Appel, A. W. (1985). An efficient program for many-body simulations. *SIAM J. Sci. Stat. Comput.*, *6*, 85–103.

Barnes, J., & Hut, P. (1986). A hierarchical $o(n \log n)$ force-calculation algorithm. *Nature 324*, 446–449.

Callahan, P., & Kosaraju, S. (1995). A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *J. of the ACM*, *62*, 67–90.

Chavez, E., Navarro, G., Baeza-Yates, R., & Marroqun, J. L. (2001). Proximity searching in metric spaces. *ACM Computing Surveys*, *33*, 273–321.

Cheng, Y. (1995). Mean shift, mode seeking, and clustering. *IEEE Trans. Pattern Anal. Mach. Intel.*, *17*, 790–799.

Collins, R. (2003). Mean-shift blob tracking through scale space. *Conf. on Computer Vision and Pattern Rec.* (pp. 234–240).

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

Comaniciu, D., Ramesh, V., & Meer, P. (2000). Real-time tracking of non-rigid objects using mean shift. *Conf. on Computer Vision and Pattern Rec.* (pp. 142 – 149).

Fashing, M., & Tomasi, C. (2005). Mean shift is a bound optimization. *IEEE Trans. Pattern Anal. Mach. Intel.*, *27*, 471–474.

Freidman, J. H., Bentley, J. L., & Finkel, R. A. (1977). An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.*, *3*, 209–226.

Fukunaga, K., & Hostetler, L. D. (1975). The estimation of the gradient of a density function, with applications in pattern recognition. *IEEE Trans. on Information Theory*, *21*, 32–40.

Georgescu, B., Shimshoni, I., & Meer, P. (2003). Mean shift based clustering in high dimensions: A texture classification example. *Intl. Conf. on Computer Vision* (pp. 456–463).

Gionis, A., Indyk, P., & Motwani, R. (1999). Similarity search in high dimensions via hashing. *VLDB* (pp. 518–529).

Gray, A., & Moore, A. (2001). N-body problems in statistical learning. *NIPS* (pp. 521–527).

Greengard, L., & Rokhlin, V. (1987). A fast algorithm for particle simulations. *J. of Comp. Physics*, *73*, 325–348.

Lee, D., & Gray, A. (2006). Faster gaussian summation: Theory and empirical experiments. *UAI*.

Scott, D. W. (1992). *Multivariate density estimation: Theory, practice, and visualization*. Wiley.

Yang, C., Duraiswami, R., Gumerov, N. A., & Davis, L. (2003). Improved fast gauss transform and efficient kernel density estimation. *Intl. Conf. on Computer Vision* (pp. 464–471).