Agglomerative Mean-Shift Clustering via Query Set Compression

Xiao-Tong Yuan  Bao-Gang Hu  Ran He

Abstract
Mean-Shift (MS) is a powerful non-parametric clustering method. Although good accuracy can be achieved, its computational cost is particularly expensive even on moderate data sets. In this paper, for the purpose of algorithm speedup, we develop an agglomerative MS clustering method called Agglo-MS, along with its mode-seeking ability and convergence property analysis. Our method is built upon an iterative query set compression mechanism which is motivated by the quadratic bounding optimization nature of MS. The whole framework can be efficiently implemented in linear running time complexity. Furthermore, we show that the pairwise constraint information can be naturally integrated into our framework to derive a semi-supervised non-parametric clustering method. Extensive experiments on toy and real-world data sets validate the speedup advantage and numerical accuracy of our method, as well as the superiority of its semi-supervised version.

1 Introduction
To find the clusters of a data set sampled from a certain unknown distribution is important in many machine learning and data mining applications. Probability density estimator may represent the distribution of data in a given problem and then the modes may be taken as the representatives of clusters. As a non-parametric method, the kernel density estimation is mostly applied in practice. Given a set of \( N \) independent, identically distributed samples \( \mathcal{X} = \{x_1, \ldots, x_N\} \) drawn from a population with density function \( f(x), x \in \mathbb{R}^d \), the kernel density estimator (KDE) with kernel \( k(\cdot) \) is defined by

\[
\hat{f}_k(x) = \sum_{i=1}^{N} p(i)p(x|i) = \sum_{i=1}^{N} w_i \frac{1}{C_i} k(M^2(x, x_i, H_i))
\]

where \( p(i) = w_i \) is the prior weight or mixing proportion of point \( x_i \) (satisfying \( \sum_{i=1}^{N} w_i = 1 \)), \( M^2(x, x_i, H_i) = (x - x_i)^T H_i^{-1} (x - x_i) \) is the Mahalanobis distance from \( x \) to \( x_i \) with covariance \( H_i \) and \( C_i \) is a normalization constant.

The Mean-Shift (MS) algorithm discovered by Fukunaga and Hostetler [12] is a powerful optimization algorithm for KDE [11]. It is expressed as the following fixed-point iteration:

\[
x_{t+1} = \left( \sum_{i=1}^{N} w_i \frac{1}{C_i} g(M^2(x_t, x_i, H_i)) H_i^{-1} \right)^{-1}
\]

where \( g(x) = -k'(x) \) and \( k(\cdot) \) is also called as the shadow of the profile \( g(\cdot) \) [8].

By setting the query set \( Q = \mathcal{X} \) and the reference set \( R = \mathcal{X} \), the Naive-MS clustering is done by grouping points in \( Q \) according to the modes they converge to via MS conducted on \( R \). It has a wide range of applications, such as discontinuity preserving image smoothing [8], image segmentation [21] and texture classification [13]. Obviously, the Naive-MS clustering typically requires \( O(KN^2) \) evaluation (\( K \) is the average number of MS iterations per-query). Even for moderate data set, such an exhaustive querying mechanism will lead to severe requirements for computational time and/or storage. The recent years have witnessed a surge of interests of fast MS clustering methods [4, 5, 12, 20, 21]. One well formulated method is the Gaussian blurring MS (GBMS) [2], which iteratively sharpens the query and reference set by moving each data point according to the Gaussian MS (GMS). Carreira-Perpinan [2] proves that GBMS converges cubically and further provides an accelerated version of GBMS by using an improved iteration stopping criterion. Yang et al. [21] accelerate the speed of GMS to linear running time by using improved fast gaussian transform (IFGT). Although very efficient for large size and high dimensional databases, IFGT-MS is difficult to be generalized for convex kernels other than Gaussian. For image segmentation applications, Carreira-Perpinan [2] has evaluated four kinds of acceleration strategies for GMS, which are based on the spatial structure of images and on the fact that GMS is an Expectation-Maximization (EM) algorithm [6]. The
fastest one is *spatial discretisation*, which can accelerate GMS by one to two orders of magnitude. However, one limitation of this strategy is that it is typically image analysis specified because it requires *spatial* dimensions that can be regularly discreted. Based on the dual-tree technique, Wang et al. [20] present the DT-MS for fast data clustering. Since a relative error bound is maintained at each iteration, DT-MS is proved to be stable and accurate. However, the computational cost saving by DT-MS, with comparison to IFGT-MS, is more impressive under the simple Epanechnikov kernel than under the Gaussian kernel.

Recently, Yuan and Li [22] point out that convex kernel based MS is equivalent to half-quadratic (HQ) optimization for the KDE function \( f^* \). The HQ analysis framework of MS implies the fact that MS is a quadratic bounding (QB) optimization for KDE, which is also discovered by Fashing and Tomasi [11]. Motivated by the QB nature of MS, we develop a highly efficient query set compression mechanism to accelerate MS clustering under general kernels. The basic idea is to construct a family of \( d \)-dimensional hyperellipsoids to cover the current query set \( Q \), promising that points inside each hyperellipsoid will be at the same *basin* of the \( d \)-dimensional KDE surface and hence will converge to the same mode via MS algorithm. For a given query point, the hyperellipsoid is constructed from a lower QB function of KDE defined at this point. Empirically, we have observed that the number of the covering hyperellipsoids is much smaller (in most cases a reduction of one to two orders of magnitude) than the size of \( Q \). We then take the centers of these hyperellipsoids to form a new query set with size dramatically reduced. Such a query set covering procedure can be iteratively run until convergence is attained. Numerical experiments show that the approximation error introduced in such a query set compression mechanism is always limited and acceptable under proper kernel bandwidth. After each iteration of query set compression, the clustering can be done by grouping the points in current query set according to the hyperellipsoids they are assigned with. This naturally leads to an agglomerative clustering framework. We have analyzed the mode-seeking ability and convergence property of the proposed algorithm. A tight upper bound of the convergent query set size is derived, which promises in theory the speedup performance of our algorithm.

Our second contribution is to develop a semi-supervised non-parametric clustering algorithm inside the proposed query set compression framework. Until now, the MS algorithm is typically applied in an unsupervised manner. We point out that supervisory information, e.g., the pairwise constraints, can be naturally integrated into the proposed query set compression framework, which leads to a novel constrained non-parametric clustering algorithm. Comparing to the constrained Kmeans (CKmeans) [13] and its variations [21,9,19], our method has the following advantages: 1) There is no potential assumption on data distribution for input data; 2) The number of output clusters is not necessarily to be known beforehand, and 3) No initialization is required to start the algorithm running. At the same time, if necessary, the CKmeans can be used as a postprocessing step to further group the output clusters by our method into desired number of clusters. The experimental evaluation on UCI and real-world data sets validates the superiority of our method.

The remainder of the paper is structured as follows. In Section 2 we briefly review the QB nature of MS, which founds the base of this work, from the viewpoint of HQ analysis. In Section 3, we develop an agglomerative MS clustering method based on iterative query set compression and evaluate its numerical performance on toy and real-world clustering problems. In Section 4 by utilizing the pairwise constraint information, we further extend our unsupervised algorithm framework into a semi-supervised version. Finally, we conclude this work in Section 5.

2 Quadratic Bounding Nature of MS

The fact that MS is a QB optimization is originally discovered by Fashing and Tomasi in [11], motivated by the relationship between MS and Newton-Raphson method. Actually, the QB nature of MS can be more straightforwardly derived from the HQ optimization viewpoint of MS [22]. When kernel is convex and monotonically decreasing, the MS algorithm can be explained as HQ optimization for \( \hat{f}_k(x) \). This feature can be shown by using the theory of convex conjugated functions [17] to introduce the following augmented energy function with \( d + N \) variables:

\[
(2.3) \quad \hat{F}(x, p) = \sum_{i=1}^{N} w_i \frac{1}{C_i} (-p_i M^2(x, x_i, H_i) + \varphi(p_i))
\]

where \( x \in \mathbb{R}^d \) is the KDE variable and \( p = (p_1, ..., p_N) \) is an \( N \)-dimensional vector of auxiliary variables. \( \varphi(\cdot) \) is the dual function of \( k(\cdot) \) [22]. \( \hat{F}(x, p) \) is quadratic w.r.t. \( x \) while concave w.r.t. \( p \). For a fixed \( x \), the following relationship holds

\[
(2.4) \quad \hat{f}_k(x) = \sup_{p} \hat{F}(x, p)
\]

and thus

\[
\max_{x} \hat{f}_k(x) = \max_{x, p} \hat{F}(x, p)
\]
which means that maximizing $\hat{f}_k(x)$ is equivalent to maximizing the augmented function $\hat{F}(x, \mathbf{p})$ on the extended domain. A local maximizer $(\hat{x}, \mathbf{p})$ of $\hat{F}$ can be calculated in the following alternate maximization way from a starting point $\hat{x}^0$

$$\begin{align}
\hat{p}_i^l & = -k\left(M^2(\hat{x}^{l-1}, x_i, \mathbf{H}_i)\right) , i = 1, ..., N \\
\hat{x}^l & = \left(\sum_{i=1}^{N} w_i \frac{1}{C_i} \hat{p}_i^l \mathbf{H}_i^{-1}\right)^{-1} \left(\sum_{i=1}^{N} w_i \frac{1}{C_i} \hat{p}_i^l \mathbf{H}_i^{-1} x_i\right)
\end{align}$$

which is actually the MS algorithm \[\text{Algorithm 1}\].

At a fixed point $\hat{x}^{l-1} \in \mathbb{R}^d$, a lower QB function for $\hat{f}_k(x)$ is given by

$$\hat{\rho}(x) \triangleq \hat{F}(x, \hat{p}^l)$$

which according to \[\text{Algorithm 1}\] satisfies $\hat{f}_k(\hat{x}^{l-1}) = \hat{\rho}(\hat{x}^{l-1})$ and $\hat{f}_k(x) \geq \hat{\rho}(x)$ for $\forall x$. Obviously the step \[\text{Algorithm 1}\] is equivalent to solving

$$\hat{x}^l = \arg \max_x \hat{\rho}(x)$$

which indicates that MS is a QB optimization for KDE. The QB viewpoint of MS motivates the following acceleration strategy for MS clustering.

3 Agglomerative MS Clustering

The key point of our agglomerative MS algorithm is to construct a family of $d$-dimensional hyperellipsoids to cover the current query set $\mathcal{Q}$, promising that points inside each hyperellipsoid will converge to a common local maximum of KDE via MS. We then use the centers of these hyperellipsoids to form a new query set as the compressor of the original one. We may iteratively run such a set covering mechanism several times until it converges. After each iteration level, the clustering is done by choosing the current query points according to the hyperellipsoids which they are associated with, leading to hierarchically clustering. In the following derivation, we assume that the covariance is homogenous, i.e., $\mathbf{H}_i = \mathbf{H}$.

### 3.1 Query Set Covering

Let’s start with $\mathcal{Q}_0 = \mathcal{R} = \mathcal{X}$. Given a data point $x_i \in \mathcal{Q}_0$, we initialize $\hat{x}_1^1 \leftarrow x_i$. It is known from eq. \[\text{(2.8)}\] that the output $\hat{x}_1^1$ from the first MS iteration is the maximizer of a QB function $\hat{\rho}_1(x)$. Thus we may write down:

$$\hat{\rho}_1(x) = s M^2(x, \hat{x}_1^1, \mathbf{H}) + C'$$

where $s = -\sum_{i=1}^{N} w_i \frac{\hat{p}_i^1}{C_i} < 0$ and $C'$ is the constant term. Taking $\hat{x}_1^1$ as center, we define the following $d$-dimensional hyperellipsoid:

$$\text{HE}(\hat{x}_1^1) = \{x | M^2(x, \hat{x}_1^1, \mathbf{H}) \leq M^2(\hat{x}_1^0, \hat{x}_1^1, \mathbf{H})\}$$

For $\forall x \in \text{HE}(\hat{x}_1^1)$, due to the quadratic bounding property of $\hat{\rho}_1(x)$, we have

$$\hat{f}_k(x) \geq \hat{\rho}_1(x) \geq \hat{\rho}_1(\hat{x}_1^0) = \hat{f}_k(\hat{x}_1^0)$$

which means that any data point $x_j \in \mathcal{Q}_0 \cap \text{HE}(\hat{x}_1^1)$ can be alternatively chosen as $\hat{x}_1^1$ to increase the current QB function value, rather than maximizing it. Therefore it is quite hopeful that $x_j$ will converge to the same mode as $x_i$ does. This is close to the essence behind generalized EM, in which the M step is allowed to merely increase the expectation obtained in E step rather than maximizing it. Now we may reasonably claim that points inside $\mathcal{Q}_0 \cap \text{HE}(\hat{x}_1^1)$ will converge to the same local maximum and can be safely clustered together by MS. Sequentially, we can run such a hyperellipsoid construction procedure on the query set $\mathcal{Q}_0$ for each point not yet being covered by any existing hyperellipsoids, until the whole set is completely scanned. As a result, we obtain a family $\mathcal{S}_0$ of hyperellipsoids which covers the entire query data set, i.e., $\mathcal{Q}_0 \subseteq \bigcup_{\hat{x}^1_j \in \mathcal{S}_0} \text{HE}(\hat{x}_1^1)$. Actually, $\mathcal{S}_0$ can be viewed as a compressor of $\mathcal{Q}_0$. The size of $\mathcal{S}_0$ generally depends on data set distribution, kernel bandwidth and data scanning order for hyperellipsoid construction. When $\mathcal{S}_0$ is relatively dense, we could alternatively apply the well known set covering greedy algorithm \[\text{Algorithm 1}\] to find a subset of $\mathcal{S}_0$, to cover $\mathcal{Q}_0$. A formal description of our query set covering mechanism is given in a form of algorithm function in Algorithm 1.

### Algorithm 1: The Query Set Covering Function

```plaintext
1: Function: Query_Set_Covering(Query data set $\mathcal{Q}$, Reference data set $\mathcal{R}$)
2: Initialization: $\mathcal{S} = \emptyset$
3: for each $x_i \in \mathcal{Q}$ do
4: if $\exists HE(\hat{x}_1^1) \in \mathcal{S}$ such that $x_i \in HE(\hat{x}_1^1)$ then
5: Associate $x_i$ with $HE(\hat{x}_1^1)$
6: else
7: Run one iteration of MS from $x_i$ with reference set $\mathcal{R}$ and construct the $HE(\hat{x}_1^1)$, as stated in section \[\text{Algorithm 1}\]
8: $S = S \cup \{HE(\hat{x}_1^1)\}$
9: end if
10: end for
11: Return $S$
```

### 3.2 Iterative Query Set Compression

Given currently constructed hyperellipsoid set $\mathcal{S}_0$, we may take the centers of the hyperellipsoids in it to form a compressed query set, i.e., $\mathcal{Q}_1 = \{\hat{x}_1^1 | HE(\hat{x}_1^1) \in \mathcal{S}_0\}$. The above presented set covering operation can be directly applied on $\mathcal{Q}_1$. After sufficient times of iteration until convergence, we will obtain a sparse enough query set
For each cluster \( C \) of \( Q \) that currently compressed query set Naive-MS does. Actually, each point survives in the Agglo-MS generally will find no more modes than of Agglo-MS, and 2) The convergence property of query algorithm analysis for the Agglo-MS. The discussion we give in this section some

3.3 Algorithm Study

3.3.1 On Mode-Seeking Ability

First, we tell that the Agglo-MS generally will find no more modes than Naive-MS does. Actually, each point survives in the currently compressed query set \( Q_l \) is the \( l \)-th MS iteration output from some initial point in \( Q_0 \). Therefore the final obtained \( Q_\infty \) is a subset of the convergent modes returned by Naive-MS on \( Q_0 \). This also implies that \( L \approx K \). Second, we show that the Agglo-MS will find no less modes than the Naive-MS does if the query set covering is done with some proper data scanning orders. Suppose that \( m \) different modes are located by the Naive-MS on \( Q_0 \) and let \( C_1, \ldots, C_m \) be the corresponding \( m \) clusters. For each cluster \( C_j \), let \( x_j^* = \arg \max_{x \in C_j} f_k(x) \). It is easy to know that a hyperellipsoid constructed from \( x_j^* \) will cover no other points in \( Q_0 \) than \( x_j^* \). We may scan from \( x_j^* \), \( j = 1, \ldots, m \), to do set covering on \( Q_0 \) and compress it into \( Q_1 \). Obviously the Naive-MS will find at least these same \( m \) modes on \( Q_1 \). Such a way of scanning order generation can be repeated on \( Q_l, l \geq 1 \), until it converges, and the \( Q_\infty \) will surly contain at least these \( m \) modes returned by the Naive-MS.

Based on the above two points, we claim that the Agglo-MS possesses the same mode-seeking ability as that of the Naive-MS if the query set covering is done with some proper data scanning orders. Our numerical observation shows that such a property of the Agglo-MS always holds under most data scanning orders.

3.3.2 Convergence Property of \(|Q_l|\)

We discuss here the convergence property of \(|Q_l|\), which is important in algorithm speedup performance analysis. First, due to the fact that \(|Q_1| \geq |Q_2| \geq \ldots\), and \(|Q_l| \geq 1\), we have the following convergence proposition:

**Proposition 3.1.** The sequence \(|Q_l|, l = 1, 2, \ldots\) generated in the Agglo-MS algorithm converges.

We now focus on the value of \(|Q_\infty|\), which is more interested in by end-users. It is known that the Naive-MS clustering is done by grouping the queries according to the modes they separately converge to. In practice, the convergent modes correspond to one common KDE maximum may not be exactly the same due to computational error. We take those modes fall inside a sphere with small enough radius \( \rho \) as identical, and nonidentical otherwise. Here \( \rho \) serves as the resolution of Naive-MS clustering, i.e., the smaller \( \rho \) is, the more clusters will be output. For the clarity of description, we introduce the following concept of \( \rho \)-Mode Number for Naive-MS:

**Definition 1.** \((\rho\text{-Mode Number})\) Given a resolution parameter \( \rho \), the number of nonidentical modes located by Naive-MS algorithm on data set \( \mathcal{X} \) under bandwidth \( \mathbf{H} \) is referred as \( \rho \)-Mode Number, which is denoted by \( M(\mathcal{X}, \mathbf{H}, \rho) \).

We aim to establish the relationship between \(|Q_l|\) and \( M(\mathcal{X}, \mathbf{H}, \rho) \). To do this, we slightly modify the set covering mechanism in algorithm 1 as follows: The if condition in line \#4 is revised as: \( \exists HE(x_j^*) \in S \) such that \( x_j^* \in HE(x_j^*) \cup HS(x_j^*) \). Here \( HS(x_j^*) \) is a hypersphere centered at \( x_j^* \) with radius \( 4\rho \). We refer algorithm 1 after this modification as the modified query set covering, with which we have the following proposition on an upper bound of \(|Q_\infty|\).

**Proposition 3.2.** Given a resolution parameter \( \rho \), the sequence \(||Q_l|, l = 1, 2, \ldots\) generated in the Agglo-MS algorithm with the modified query set covering mechanism satisfies

\[
\lim_{{|Q_\infty|}} |Q_l| \leq M(\mathcal{X}, \mathbf{H}, \rho)
\]
Proof. For each \( x_i^l \in Q_l \), denote \( x_i^\infty \) be its corresponding MS convergent point. It is known that \( \exists L > 0 \) such that if \( l > L \) then \( \| x_i^l - x_i^\infty \| < \rho, \forall i \). Also, we have that the convergent set \( \{ x_i^\infty \mid i = 1, \ldots, |Q_l| \} \) is covered by \( M(\mathcal{X}, \mathbf{H}, \rho) \) hyperspheres with radius \( \rho \). Based on these preliminary knowledge, we now prove the proposition with reduction to absurdity.

Assume that \( \lim_{l \to \infty} |Q_l| = T > M(\mathcal{X}, \mathbf{H}, \rho) \). Due to the discreteness of cardinality we get that \( \exists L' > 0 \) such that if \( l > L' \) then \( |Q_l| = T \). Now we fix the iteration level \( l > \max(L, L') \). Since \( l > M(\mathcal{X}, \mathbf{H}, \rho) \), from the drawer principle we know that there exist at least two points \( x_i^l \) and \( x_j^l \) whose convergent points \( x_i^\infty \) and \( x_j^\infty \) will fall inside the same covering hypersphere. Now, we consider the \((l + 1)\)th iteration of query set compression from \( x_i^l \) (note that hyperellipsoids will be constructed for each point in query set when it converges). According to the triangle inequality, we have \( \| x_i^{l+1} - x_j^l \| < \| x_i^{l+1} - x_i^\infty \| + \| x_i^\infty - x_j^\infty \| + \| x_j^\infty - x_j^l \| < \rho + 2\rho + \rho = 4\rho \). Therefore, according to the modified set covering mechanism \( x_i^{l+1} \) shall be associated with the hyperellipsoid constructed from \( x_i^{l+1} \) in the \((l + 1)\)th level of set compression. This indicates that \( |Q_{l+1}| < |Q_l| = T \), which leads to contradiction. 

Proposition 3.2 tells us that when \( \rho \) is properly chosen and \( l \) is large enough, \( Q_l \) will be rather sparse. Therefore significant speedup of Agglo-MS over Naive-MS can be achieved.

3.4 Summary of Method To summarize the algorithm development so far, an accelerated MS clustering method based on iterative query set compression is derived. The computational complexity, mode-seeking ability and convergence property of the proposed algorithm are analyzed. In methodology, our Agglo-MS algorithm differs itself from the existing acceleration strategies, including IFGT-MS [21], LSH-MS [4], ms1-MS [4] and DT-MS [20]. The IFGT-MS approximates MS calculation at each query point by taking its nearby points as reference set and adopting the fast multipole method. It is only applicable to Gaussian kernel. The LSH-MS also makes a fast search of the neighborhood around a query point to approximately compute MS iteration. For image segmentation tasks, the ms1-MS smartly stops the current MS iteration if the ascent-path intersects with a previously visited pixel. This is close to the trick used in [8]. The DT-MS achieves speedup by recursively considering a region of query points and a region of reference points, which are represented by nodes of a query tree and a reference tree respectively. The proposed Agglo-MS, on the other hand, iteratively compress the query set until convergence is attained, and the query points are clustered in an agglomerative way during this procedure.

In the following experiments, we choose the Naive-MS as baseline algorithm and make performance comparison among the Agglo-MS, IFGT-MS and ms1-MS (for image segmentation). In our implementation of the Agglo-MS, we use the modified query set covering function.

3.5 A Walkthrough Example We apply in this section the Agglo-MS on a 2D toy data set (shaped as shown in Figure 1(b) size 1,821) to illustrate its working mechanism. In this case, isotropic covariance \( \mathbf{H} = \sigma^2 I \) is used, hence the hyperellipsoids are degenerated into disks.

Let’s start from the point \( A(-1.5,0.6) \) (the green dot in Figure 1(b)) with bandwidth \( \sigma = 0.4 \) to perform the query set covering. After one iteration of MS, point \( A \) shifts to point \( B(-1.29,0.62) \) (the red dot in Figure 1(b)). Now, we construct the first disk, taking \( B \) as its center and \( \| AB \| \) as the radius. In Figure 1(a) we plot the mesh of KDE (in black) and the QB function \( \hat{\rho}_1(x) \) (in blue) at \( A \). Obviously, any point lies inside this disk will increase \( \hat{\rho}_1(x) \), hence is hopeful to converge to the same mode as that of \( A \) via MS. After one pass of scanning, a total of 223 disks are obtained to cover the initial query data set, as illustrated in Figure 1(c). One worrying aspect of our set covering mechanism is the possible intersections between disks from different clusters. To more clearly illustrate such a phenomenon, we have selected a local window (in green) located around one cluster boundary area in Figure 1(c) and magnified it in Figure 1(d). Those disks in blue straddle clusters. This is also where the numerical error of Agglo-MS generates from. The detailed quantitative evaluation on this aspect will be addressed later on. We now enter the iterative query set compression phase. After \( l = 5 \) times of iteration a query set with 104 points is obtained, and the corresponding compressed query set and clustering result are shown in Figure 1(e)&1(f). The convergent \((l = 60)\) compressed query set and the corresponding clustering result are shown in Figure 1(g)&1(h). The query set size vs. iteration number curves for different bandwidths are given in Figure 2(a). We evaluate the numerical performance of accelerated MS algorithms using the CPU running time and the following defined \( \varepsilon \)-error rate (\( \varepsilon \)-ER):

\[
\varepsilon \text{-ER} = \frac{1}{N} \sum_{i=1}^{N} \delta \left( \frac{\| \tilde{x}_i^\infty, X-MS \|}{\| \tilde{x}_i^\infty, \text{Naive-MS} \|} \geq \varepsilon \right)
\]

where \( \delta(x) \) is the delta function that equals one if
Boolean variable $x$ is true and equals zero otherwise, while $\hat{x}_\infty, \text{X-MS}$ and $\hat{x}_\infty, \text{Naive-MS}$ are convergent modes returned by X-MS (X stands for 'Agglo', 'IFGT' or 'ms1' in this work) and Naive-MS respectively from an initial query point $x_i$ in $Q_0$. We set the parameters $\rho = \varepsilon = 10^{-3}$, and choose the Gaussian kernel throughout the experiments in this paper. The algorithm codes are basically written in C++, and run on a P4 Core2 2.4G Hz CPU with 2G RAM. For this example, the quantitative results under bandwidth $\sigma = 0.4$ are listed in table 1. The optimal clustering results ($|Q_\infty| = 5$) by Agglo-MS are achieved under $\sigma \in (0.25, 0.54)$. The speedup vs. bandwidth curves of the Agglo-MS and IFGT-MS over the Naive-MS are given in Figure 2(b), from which we can see that Agglo-MS outperforms IFGT-MS in speedup performance. We also show the $\varepsilon$-ER vs. bandwidth curves of the Agglo-MS and IFGT-MS in Figure 2(c), from which we can see that the approximation error are comparable between these two algorithms. Note that when bandwidth is small ($\sigma<0.3$) or large ($\sigma>0.6$), relatively high $\varepsilon$-ER is introduced in Agglo-MS. This is because small bandwidth tends to lead to over clustering by MS method, which makes the Agglo-MS sensitive to the intersection between disks from different clusters, while too large bandwidth will lead to large covering disks, which also increase the chance of involving points belonging to different clusters.

---

The IFGT-MS code is implemented using the IFGT library provided by the authors at [http://www.umiacs.umd.edu/~vikas/Software/IFGT/IFGT_code.htm](http://www.umiacs.umd.edu/~vikas/Software/IFGT/IFGT_code.htm).

The important turnable parameters, e.g., $p_{\text{max}}$-polynomial order, $K$-number of cells and $r$-ratio of the cut off radius, can be automatically chosen by the program.

We further carefully tune the parameters $p_{\text{max}}$ and $r$ to achieve comparable result to Naive-MS.
Table 1: CPU running time (in milliseconds) and $\varepsilon$-ER of Agglo-MS, IFGT-MS ($p_{\text{max}} = 5$, $r = 2$) and Naive-MS on the 2D synthetic dataset. $\sigma = 0.4$.

| Methods      | $|Q_0|$ | CPU Time | $\varepsilon$-ER |
|--------------|--------|----------|------------------|
| Agglo-MS     | $l = 1$ | 223      | 40               |
|              | $l = 5$ | 104      | 53               |
|              | $l = 60$| 5        | 117, 0.0016      |
| IFGT-MS      | —      | 975      | 0                |
| Naive-MS     | —      | 1,759    | —                |

3.6 Real-World Experiments We now assess the performance of Agglo-MS on some real-world clustering tasks, including image segmentation and high dimensional data set clustering.

3.6.1 Image Segmentation The most important application of MS clustering algorithm is for unsupervised image segmentation. We follow the approach in [21], where each datum is represented by spatial and range features $(i, j, L^*, u^*, v^*)$, where $(i, j)$ is the pixel location in the image and $(L^*, u^*, v^*)$ is the normalized LUV color feature. Figures 3(b), 4(f) show the results by Agglo-MS on the color image *hand*. The speedup vs. bandwidth curves of Agglo-MS, IFGT-MS and ms1-MS over Naive-MS on this image are given in Figure 4(a) from which we can see that the Agglo-MS is always much faster than the other two. The $\varepsilon$-ER vs. bandwidth curves of the Agglo-MS, IFGT-MS and ms1-MS are plotted in Figure 3(b) from which we can see that the approximation error are comparable among these three algorithms. The reasonable segmentations are achieved under the bandwidth interval $\sigma \in (0.1, 0.22)$. We give in Figure 4 the selected segmentation results under bandwidths $\sigma = 0.1, 0.2$ by different MS clustering methods.

Figure 3: Quantitative evaluation curves for the *hand* image.

Some images from [21] and the Berkeley segmentation dataset [3] are also used for evaluation. Four selected groups of segmentation results are given in Figure 5. The quantitative comparison between Agglo-MS, IFGT-MS and ms1-MS on these images are listed in Table 2. As expected, the Agglo-MS significantly outperforms the other two in speedup performance on these four test images. The $\varepsilon$-ER achieved by Agglo-MS is overall less than 3% and comparable to the other two.

3.6.2 High Dimensional Cases To evaluate the speedup and numerical performance of the Agglo-MS in high dimensional data clustering tasks, we have applied it on several real-world databases, including the CMU PIE face database [4], the MNIST handwritten digit database [5] and the TDT2 document database[6]. We first briefly describe these data sets and then give the quantitative results on them.

CMU PIE Data set The CMU PIE face database contains 68 subjects with 41,368 face images as a whole. We follow [7] to use 170 face images for each individual in our experiment. The total count of points in the data set is 11,554. The size of each cropped gray scale image is $32 \times 32$ pixels. We do clustering in a subspace embedded by spectral regression (SR) [8], with dimension reduced from 1024 into 67.

MNIST Document Data set The MNIST database of handwritten digits has a training set of 60,000 examples, and a test set of 10,000 examples. The digits have been size-normalized and centered in a fixed-size (28 × 28) bilevel image. Our clustering is done on the training set embedded by SR, with dimension reduced from 784 to 9.

TDT2 Document Data set The TDT2 corpus con-
Figure 5: Selected image segmentation results. For each image group, from left to right: Naive-MS, Agglo-MS, IFGT-MS and ms1-MS.

Table 2: Quantitative results by Agglo-MS, IFGT-MS ($p_{max} = 3, r = 2$), ms1-MS and Naive-MS on four test images.

<table>
<thead>
<tr>
<th>Images</th>
<th>House</th>
<th>Base Dive</th>
<th>Hawk</th>
<th>Cowboy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sizes</td>
<td>255 × 192</td>
<td>432 × 294</td>
<td>481 × 321</td>
<td>481 × 321</td>
</tr>
<tr>
<td>$\sigma$</td>
<td>0.1</td>
<td>0.1</td>
<td>0.06</td>
<td>0.1</td>
</tr>
<tr>
<td>CPU</td>
<td>Agglo-MS</td>
<td>IFGT-MS</td>
<td>ms1-MS</td>
<td>Naive-MS</td>
</tr>
<tr>
<td>Time (s)</td>
<td>1.780</td>
<td>3.927</td>
<td>9.313</td>
<td>5.888</td>
</tr>
<tr>
<td></td>
<td>5.628</td>
<td>23.863</td>
<td>28.753</td>
<td>27.82</td>
</tr>
<tr>
<td></td>
<td>116.587</td>
<td>169.290</td>
<td>155.829</td>
<td>717.521</td>
</tr>
<tr>
<td>Speedup</td>
<td>Agglo-MS</td>
<td>IFGT-MS</td>
<td>ms1-MS</td>
<td>Naive-MS</td>
</tr>
<tr>
<td></td>
<td>65.50</td>
<td>43.11</td>
<td>16.73</td>
<td>121.86</td>
</tr>
<tr>
<td></td>
<td>2.64</td>
<td>5.16</td>
<td>2.85</td>
<td>3.66</td>
</tr>
<tr>
<td></td>
<td>14.55</td>
<td>11.59</td>
<td>12.60</td>
<td>39.59</td>
</tr>
<tr>
<td></td>
<td>0.029</td>
<td>0.025</td>
<td>0.001</td>
<td>0.005</td>
</tr>
<tr>
<td></td>
<td>0.021</td>
<td>0.004</td>
<td>0.046</td>
<td>0.015</td>
</tr>
<tr>
<td></td>
<td>0.027</td>
<td>0.024</td>
<td>0.001</td>
<td>0.005</td>
</tr>
<tr>
<td>$\varepsilon$-ER</td>
<td>Agglo-MS</td>
<td>IFGT-MS</td>
<td>ms1-MS</td>
<td>Naive-MS</td>
</tr>
<tr>
<td></td>
<td>0.092</td>
<td>0.065</td>
<td>0.002</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.092</td>
<td>0.065</td>
<td>0.002</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5/5</td>
<td>7/7</td>
<td>3/3</td>
<td>5/5</td>
</tr>
<tr>
<td>$</td>
<td>Q_\infty</td>
<td>/M(\mathcal{X}, H, \rho)$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

consists of 11,201 on-topic documents which are classified into 96 semantic categories. In this experiment, we use the top 30 categories, thus leaving us with 9,394 documents. The clustering is also performed in a subspace embedded by SR, with dimension reduced from 36,771 to 29.

The provided IFGT-MS code fails on these datasets. The quantitative results by Agglo-MS and Naive-MS are listed in Table 2, from which we can see that the Agglo-MS also significantly improves the clustering speed with acceptable approximation error in high dimensional space.

Table 3: Performance of Agglo-MS vs. Naive-MS on high dimensional datasets.

<table>
<thead>
<tr>
<th>Datasets</th>
<th>PIE</th>
<th>MNIST</th>
<th>TDT2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimensions</td>
<td>67</td>
<td>9</td>
<td>29</td>
</tr>
<tr>
<td>Classes</td>
<td>68</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>Sizes</td>
<td>11,554</td>
<td>60,000</td>
<td>9,394</td>
</tr>
<tr>
<td>$\sigma$</td>
<td>15</td>
<td>10</td>
<td>0.8</td>
</tr>
<tr>
<td>CPU</td>
<td>Agglo-MS</td>
<td>IFGT-MS</td>
<td>ms1-MS</td>
</tr>
<tr>
<td>Time (s)</td>
<td>52.748</td>
<td>13,273</td>
<td>4.789</td>
</tr>
<tr>
<td></td>
<td>505.111</td>
<td>381.646</td>
<td>125.729</td>
</tr>
<tr>
<td>Speedup</td>
<td>9.58</td>
<td>28.75</td>
<td>26.25</td>
</tr>
<tr>
<td>$\varepsilon$-ER</td>
<td>0.092</td>
<td>0.065</td>
<td>0.002</td>
</tr>
<tr>
<td></td>
<td>76/76</td>
<td>113/113</td>
<td>22/22</td>
</tr>
<tr>
<td>$</td>
<td>Q_\infty</td>
<td>/M(\mathcal{X}, H, \rho)$</td>
<td></td>
</tr>
</tbody>
</table>

4 Extension for Constrained Clustering

Our Agglo-MS can be naturally extended into a constrained non-parametric clustering framework, with cluster purity much improved. Typically, the constrained clustering focused on the use of background information in the form of instance level must-link and cannot-link constraints. A must-link constraint enforces that two instances must be placed in the same cluster while a cannot-link constraint enforces that two instances must not be placed in the same cluster. The previous work on constrained clustering can be divided into two categories: 1) where the constrains help the algorithm construct a distortion/distance function for optimization [1][16][19] and 2) where the constraints are used as hints to guide the algorithm to search a feasi-
A constrained clustering algorithm is referred to as ML-feasible. The first type of work makes the assumption that points surrounding a pair of must-link/cannot-link points should be close to far from each other, while the second type just requires that the two points be in the same/different clusters. Our constrained Agglo-MS (CAgglo-MS) method falls into the second category. The must-link and cannot-link constraints are used to guide query set covering at each iteration level to promise the feasibility of the output clusters. At the same time, the constraints themselves are automatically updated as query set evolves. To further get the desired number of clusters, the Ckmeans-like methods [2][6][19] can be alternatively applied on the final compressed query set with updated constraints.

4.1 Definition of Constraints We consider the problem of clustering the set \( \mathcal{X} \) under the following types of constraints:

1. **Must-Link Constraints:** Each must-link constraint involves a pair of points \( x_i \) and \( x_j \) \((i \neq j)\). In any feasible clustering, points \( x_i \) and \( x_j \) must be in the same cluster. Denote \( \mathcal{M} \) be the set of must-link constraint pairs.

2. **Cannot-Link Constraints:** Each cannot-link constraint also involves a pair of distinct points \( x_i \) and \( x_j \). In any feasible clustering, points \( x_i \) and \( x_j \) must not be in the same cluster. Denote \( \mathcal{C} \) be the set of cannot-link constraint pairs.

A constrained clustering algorithm is referred to as ML-feasible and CL-feasible if the constraints \( \mathcal{M} \) and \( \mathcal{C} \) are satisfied respectively.

4.2 The Algorithm and Feasibility The CAgglo-MS clustering method is formally given in Algorithm 4. We show through the following detailed construction procedure that CAgglo-MS is both ML-feasible and CL-feasible. As is well known, must-link constraints are transitive. Therefore, a given collection \( \mathcal{M} \) of must-link constraints can be transformed into an equivalent collection \( \mathcal{M} = \{M_1,...,M_r\} \) of constraints, by computing the transitive closure of \( W \). In the CAgglo-MS, we take \( M_1,...,M_r \) as \( r \) singletons, and together with the complemental set \( \mathcal{X} - \bigcup_{i=1}^r M_i \) to form the initial query set. Such a way of initialization promises that the output clusters at each iteration level \( l \) is ML-feasible due to the fact that Agglo-MS is an agglomerative clustering algorithm. To promise the CL-feasibility, we revise the algorithm 1 into a cannot-link constrained query set covering mechanism, as is formally described in Algorithm 3. The key point is that each cannot-link pair is not allowed to be associated with the same hyperellipsoid during the set covering. This will make sure that, at each iteration level \( l \), the obtained clusters are CL-feasible under current cannot-link constraints \( C_l \). Here, the cannot-link constraint \( C_l \) is updated according the following rule: Given a pair of cannot-link points in \( C_{l-1} \), the centers of the two hyperellipsoids they separately associated with in \( S_l \) are labeled as cannot-link in \( C_l \). Therefore, through simple induction on \( l \) we get that the output clusters at each iteration level are CL-feasible under the initial cannot-link constraint \( C_0 = C \). The following proposition summarizes the feasibility issues discussed above.

Proposition 4.1. The output clusters at each level \( l \) by CAgglo-MS algorithm are ML-feasible and CL-feasible under the constraints \( \mathcal{M} \) and \( \mathcal{C} \).

Typically, CAgglo-MS will output more clusters than Agglo-MS does on the same data set. This can also be verified by the following knowledge: Since CAgglo-MS is CL-feasible, we know from [10] that \( |\mathcal{Q}_\infty| \) is bounded below by a number \( k_{\min} \), which is upper bounded by one plus the maximum degree of a node in an undirected graph \( G_c = \{V_c,E_c\} \) whose vertices set \( V_c = \mathcal{Q}_0 \) and edges set \( E_c = C \). If necessary, as what is done in the following experiments, we may further group the output clusters of CAgglo-MS so that more compact results can be obtained.

4.3 Performance Evaluation

4.3.1 Data Preparation The data sets used in our constrained clustering experiments include 4 data sets from the UCI repository: Iris, Wine, Sonar and Ionosphere; and two subsets from the MNIST and TDT2. For MNIST we choose the digits \{3, 6, 8, 9\} to form the MNIST.Multi4 while for TDT2 we chose the top ten documents to form the TDT2.Multi10. We then randomly sample 5\% from the MNIST.Multi4 and TDT2.Multi10. Table 3 summarizes the properties of these data sets. The constraints are generated as follows: for each constraint, we pick out one pair of data points randomly from the input data sets (the labels of which are available for evaluation purpose but unavailable for clustering). If the labels of this pair of points are the same, then we generate a must-link. If the labels are different, a cannot-link is generated. The amounts of constraints are determined by the size and difficulty of data set.

Algorithm 3: Cannot-Link Constrained Query Set Covering

1: Function: 
   \( CL_{\text{Query Set Covering}}(\text{Query data set } Q, \text{Reference data set } R, \text{Cannot-link constraints } C) \)
2: Initialization: \( S = \emptyset \).
3: for each \( x_i \in Q \) do 
4: if \( \exists HE(\hat{x}_i^1) \in S \) such that \( x_i \in HE(\hat{x}_i^1) \lor HS(\hat{x}_i^1), 4p \) and Violate\_Cannot\_Link(\( x_i, HE(\hat{x}_i^1), C \)) is false then 
5: Associate \( x_i \) with \( HE(\hat{x}_i^1) \).
6: else 
7: Run one iteration of MS from \( x_i \) with reference set \( R \) and construct the \( HE(\hat{x}_i^1) \), as stated in section 3.1.
8: \( S = S \cup \{ HE(\hat{x}_i^1) \} \)
9: end if 
10: end for 
11: Return \( S \)

Function: 
\( \text{Violate\_Cannot\_Link}(\text{Data point } x_i, \text{Hyperellipsoid } HE, \text{Cannot-link constraints } C) \)

if \( \exists x_j \) such that \( (x_i, x_j) \in C \) and \( x_j \) is already associated with \( HE \) then 
13: Return true
14: else 
15: Return false
16: end if

Algorithm 4: Constrained Agglo-MS Clustering

1: \( \text{Constrained Initial Query Set Covering:} \)
2: Let \( X' = X - \bigcup_{i=1}^{\infty} M_i \). Set query set \( Q_0 = X' \bigcup \{ x_i \in M_i, i = 1, ..., r \} \) and reference set \( R = X \). Set the constraints \( C_0 = C \).
3: \( S_0 = CL_{\text{Query Set Covering}}(Q_0, R, C_0) \)
4: \( \text{Constrained Iterative Set Compression Phase:} \)
5: Set \( l = 1 \)
6: while Convergence is not attained do 
7: Let query set \( Q_l = \{ \hat{x}_i^1 | HE(\hat{x}_i^1) \in S_{l-1} \} \).
8: Construct cannot-link constraints \( C_l \) so that:\n9: \( \forall (x_m, x_n) \in C_{l-1} \) and \( x_m \in HE(\hat{x}_i^1), x_n \in HE(\hat{x}_j^1) \), we have \( (\hat{x}_i^1, \hat{x}_j^1) \in C_l \).
10: \( S_l = CL_{\text{Query Set Covering}}(Q_l, R, C_l) \)
11: Group each point in the query set \( Q_l \) according to the hyperellipsoid it is associated with in \( S_l \).
12: \( l \leftarrow l + 1 \)
13: end while

4.3.2 Experimental Design In the first group of experiments, we aim to compare the clustering performance of CAgglo-MS and the original Agglo-MS. We use the Precision [2] to evaluate the purity of output clusters by CAgglo-MS and Agglo-MS. The reason to adopt Precision as measurement is that the number of output clusters by Agglo-MS and CAgglo-MS may differ from the underlying class numbers.

Our second group of experiments is designed to compare CAgglo-MS with two popular constrained K-means clustering methods MPCKmeans [2] and CKmeans [15], as well as the traditional K-means. To output the desired number of clusters in our algorithm, we alternatively apply MPCKmeans to further group the output clusters from the CAgglo-MS with the updated CL-Constraints. We refer to such a combined method as CAgglo-MS-Kmeans. The astute reader might notice that during the query set compression in CAgglo-MS, the CL-Constraints \( C_l \) will be stronger than \( C_{l-1} \) since the cannot-link property is propagated from point pairs to cluster pairs. The final obtained CL-Constraint \( C_\infty \) will then always be over strengthened and go against the compact clustering of \( Q_\infty \). To alleviate such an inherent drawback, in our implementation, we just carry out the MPCKmeans on the first five \( Q_i \) \((l \leq 5)\) with the corresponding CL-Constrained \( C_l \) and pick the one that performs the best as final clustering output. The F-Measure score [2] is adopt as quantitative measurement for clustering performance evaluation.

4.3.3 Results The Precision curves of the CAgglo-MS and Agglo-MS on the test data sets are plot in Figure [9] from which we can see that integrating semi-supervision information into query set compression does help to significantly improve the purity of the output clusters. We illustrate in Figure [7] the evolution of cannot-link constraints during the running of CAgglo-MS on the Iris data set with 200 constraints. From Figure [7(c)] we can obviously see that the CL-Constraint is much strengthened and at least six clusters are required on \( Q_{150} \) to fulfill the corresponding \( C_{150} \).

The F-Measure curves of CAgglo-MS-Kmeans, MPCKmeans, CKmeans and the traditional K-means on these data sets are given in Figure [8]. Our CAgglo-MS-Kmeans achieves the best performance on the four UCI data sets and the TDT2.Multi10 subset. On the MINIST.Multi4 subset, CAgglo-MS-Kmeans and MPCKmeans perform comparably, and both are superior to CKmeans when constraints are less than 650 while inferior to it as constraints increase above 650.
Figure 6: The Precision vs. number of constraints curves of CAgglo-MS and Agglo-MS.

Figure 7: Evolution of cannot-link constraints during query set compression in CAgglo-MS, on the UCI Iris data set. The points from the three classes are differently colored as dots. The cannot-link constraints are represented by red lines. The black stars represent the current query points. Here, the first two dimensions of input feature are plotted for the sake of visualization.

Figure 8: The F-Measure vs. number of constraints curves of CAgglo-MS-Kmeans, MPCKmeans, CKmeans and traditional K-means.
5 Conclusion

In this paper, we report our research progress on algorithm improvement for the popularly used MS non-parametric clustering method. As the first contribution, we develop the so called Agglo-MS algorithm based on a highly efficient hyperellipsoid query set covering mechanism. The mode-seeking ability and convergence property of the Agglo-MS are analyzed. The Agglo-MS is applicable to general convex kernels. Another advantage of Agglo-MS over some existing ones, e.g., IFGT-MS and LSH-MS, lies in that it is free of parameter tuning, hence is more flexible when applied in practice. Extensive evaluation on several toy and real-world clustering tasks validates the time efficiency and numerical accuracy of Agglo-MS in both low and high dimensional spaces. The second contribution of this work is to elegantly integrate the pairwise constraint information into the Agglo-MS to develop a semi-supervised non-parametric clustering algorithm, namely CAgglo-MS. The experimental evaluation on UCI and real-world data sets validates the superiority of CAgglo-MS. We expect that, through combining with dimensionality reduction techniques, one can achieve competitive solutions via Agglo-MS and CAgglo-MS in many clustering tasks.

References