Fast Algorithm for Affinity Propagation

Yasuhiro Fujiwara[†], **Go Irie**[‡], **Tomoe Kitahara**[*]
[†]NTT Cyber Space Laboratories, [‡]NTT Cyber Solutions Laboratories, [*]Nihon University
{fujiwara.yasuhiro, irie.go}@lab.ntt.co.jp, csto10004@g.nihon-u.ac.jp

Abstract

Affinity Propagation is a state-of-the-art clustering method recently proposed by Frey and Dueck. It has been successfully applied to broad areas of computer science research because it has much better clustering performance than traditional clustering methods such as k-means. In order to obtain high quality sets of clusters, the original Affinity Propagation algorithm iteratively exchanges real-valued messages between all pairs of data points until convergence. However, this algorithm does not scale for large datasets because it requires quadratic CPU time in the number of data points to compute the messages. This paper proposes an efficient Affinity Propagation algorithm that guarantees the same clustering result as the original algorithm after convergence. The heart of our approach is (1) to prune unnecessary message exchanges in the iterations and (2) to compute the convergence values of pruned messages after the iterations to determine clusters. Experimental evaluations on several different datasets demonstrate the effectiveness of our algorithm.

1 Introduction

Cluster analysis is an important component of scientific and industrial data analysis, and many clustering methods have been proposed [Jain *et al.*, 1999]. When a set of data points is given, clustering assigns each data point into a group (cluster). By using clustering, we can extract representative examples from the dataset or discover the structures present in the raw data.

Recently, Frey and Dueck proposed a novel clustering method, named Affinity Propagation [Frey and Dueck, 2007]. Affinity Propagation recursively transmits real-valued messages between all pairs of data points until message values converge. And it determines a set of clusters and their representative data points called 'exemplars', based on the converged message values. Affinity Propagation has a number of noteworthy advantages. For instance, (1) it achieves much lower clustering error than existing clustering methods such as k-means clustering [Jain *et al.*, 1999], (2) it can support similarities that are not symmetric or do not satisfy the triangle inequality, and (3) it is deterministic, i.e., its clustering results do not depend on initialization, unlike most clustering methods such as k-means. Therefore, it has been successfully used in many applications in various disciplines such as initial training set selection in active learning [Hu *et al.*, 2010], keyphrase extraction [Liu *et al.*, 2009], image clustering [Jia *et al.*, 2008] [Dueck and Frey, 2007], representative image extraction [Zha *et al.*, 2009], and supervised dimensionality reduction [Chang and Zheng, 2009].

Although Affinity Propagation has been receiving a lot of attention for use in many applications, one of the most important research issues is its speed, especially for large scale datasets [Jia *et al.*, 2008]. This is because the original Affinity Propagation algorithm requires quadratic CPU time in the number of data points to compute the messages. For resolving this issue, Jia et al. recently proposed FSAP, a fast algorithm for Affinity Propagation [Jia *et al.*, 2008]. However, their algorithm achieves its efficiency at the expense of clustering result accuracy; its clustering results are not the same as those of the original Affinity Propagation algorithm. Sacrificing clustering accuracy makes it difficult to realize truly effective applications.

In this paper, we propose an efficient clustering algorithm for Affinity Propagation that gives the same clustering result as the original algorithm after convergence. In order to reduce computation cost without loss of clustering accuracy, the proposal (1) prunes unnecessary message exchanges between data pairs in the iterations to compute the convergence values, and (2) computes the convergence values of pruned messages from those of unpruned messages. Moreover, our algorithm does not require users to set any inner-parameters, unlike FSAP. FSAP has a specific inner-parameter that must be carefully set if the clustering results are not to be negatively impacted. We conduct evaluation experiments on several different datasets to compare our algorithm against existing algorithms. The results demonstrate the superiority of our algorithm. It is much faster than the existing algorithms while it guarantees exactness of the clustering results. This confirms the practicality of our algorithm for real world applications. With our method, many applications can be processed more efficiently.

The remainder of this paper is organized as follows. Section 2 gives a brief review of the original Affinity Propagation

Table 1: Definition of main symbols.

Symbol	Definition
N	Number of data points
T	Number of iterations
$s[i,j]$	Similarity between data point i and j
$r[i,j]$	Responsibility between data point i and j
$a[i,j]$	Availability between data point i and j
$\rho[i,j]$	Propagating responsibility between data point i and j
$\alpha[i,j]$	Propagating availability between data point i and j
λ	Damping factor, $0 \leq \lambda < 1$

algorithm and FSAP. Section 3 introduces the main ideas and details of our algorithm. Section 4 reviews the results of our experiments. Section 5 provides our conclusions.

2 Preliminary

In this section, we briefly review the original Affinity Propagation algorithm proposed by Frey and Dueck [Frey and Dueck, 2007], as well as FSAP [Jia *et al.*, 2008], a state-of-the-art fast algorithm for Affinity Propagation. Table 1 lists the main symbols and their definitions.

2.1 Affinity Propagation

Affinity Propagation is a clustering algorithm that identifies a set of 'exemplars' that represents the dataset [Frey and Dueck, 2007]. The input of Affinity Propagation is the pair-wise similarities between each pair of data points, $s[i,j](i,j = 1,2,\ldots,N)$ [1]. Any type of similarities is acceptable, e.g. negative Euclidean distance for real valued data and Jaccard coefficient for non-metric data, thus Affinity Propagation is widely applicable.

Given similarity matrix $s[i,j]$, Affinity Propagation attempts to find the exemplars that maximize the net similarity, i.e. the overall sum of similarities between all exemplars and their member data points. The process of Affinity Propagation can be viewed as a message passing process with two kinds of messages exchanged among data points: *responsibility* and *availability*. Responsibility, $r[i,j]$, is a message from data point i to j that reflects the accumulated evidence for how well-suited data point j is to serve as the exemplar for data point i. Availability, $a[i,j]$, is a message from data point j to i that reflects the accumulated evidence for how appropriate it would be for data point i to choose data point j as its exemplar. All responsibilities and availabilities are set to 0 initially, and their values are iteratively updated as follows to compute convergence values:

$$\begin{aligned} r[i,j] &= (1-\lambda)\rho[i,j] + \lambda r[i,j] \\ a[i,j] &= (1-\lambda)\alpha[i,j] + \lambda a[i,j] \end{aligned} \tag{1}$$

where λ is a damping factor introduced to avoid numerical oscillations, and $\rho[i,j]$ and $\alpha[i,j]$ are, we call, *propagating responsibility* and *propagating availability*, respectively. $\rho[i,j]$ and $\alpha[i,j]$ are computed by the following equations:

[1] $s[i,i]$ for each data point is called preference and impacts the number of clusters.

$$\rho[i,j] = \begin{cases} s[i,j] - \max_{k\neq j}\{a[i,k]+s[i,k]\} & (i\neq j) \\ s[i,j] - \max_{k\neq j}\{s[i,k]\} & (i=j) \end{cases} \tag{2}$$

$$\alpha[i,j] = \begin{cases} \min\{0, r[j,j] + \sum_{k\neq i,j}\max\{0, r[k,j]\}\} & (i\neq j) \\ \sum_{k\neq i}\max\{0, r[k,j]\} & (i=j) \end{cases} \tag{3}$$

That is, messages between data points are computed from the corresponding propagating messages. The exemplar of data point i is finally defined as:

$$\arg\max\{r[i,j] + a[i,j] : j = 1,2,\ldots,N\} \tag{4}$$

As described above, the original algorithm requires $O(N^2T)$ time to update massages, where N and T are the number of data points and the number of iterations, respectively. This incurs excessive CPU time, especially when the number of data points is large. Therefore, a fast Affinity Propagation algorithm is demanded as pointed out in [Jia *et al.*, 2008].

2.2 FSAP

For resolving the computation time issue of the original Affinity Propagation algorithm, Jia et al. recently proposed FSAP [Jia *et al.*, 2008]. One promising idea for improving the speed of Affinity Propagation is to reduce the number of message values that need to be computed. FSAP aims to reflect this idea as follows. The first stage of FSAP constructs a K-nearest neighbor graph. If data point i is among the K data points that have the largest similarity with data point j, then data point i and j are connected by an edge, otherwise not. Since FSAP performs message transmissions on the K-nearest neighbor graph, too many exemplars (at least N/K) might be generated. Therefore, in order to merge multiple exemplars into one cluster, the second stage adds further edges based on the following three criteria:

1. If data point i is the exemplar of data point j, then data point i and j are connected by an edge;
2. For two data points i and j, if there exists two data points m and n that take data point i and j as their exemplar, respectively, and data point m and n are K-nearest neighbor to each other, and so data point i and j are connected by an edge; and
3. For two data points i and j, if they are connected by criterion 2, then all data points that choose data point i as exemplar are connected to data point j, and vise versa.

After convergence, the exemplar of data point i is finally determined by Equation (4).

They showed that their approach is much faster than the original algorithm described in Section 2.1. However, FSAP is based on heuristic ideas, i.e., the linked edges are determined based on K-nearest neighbor approximation and heuristic criteria. Therefore, FSAP does not guarantee the same result as the original Affinity Propagation algorithm. As we will show in Section 4, this fact seriously degrades the clustering accuracy. Our algorithm presented in this paper is faster than FSAP while it still theoretically guarantees the exactness of the clustering results after convergence.

3 Proposed method

In this section, we present our fast Affinity Propagation algorithm which gives the same clustering result as the original algorithm after convergence. First, we give an overview of the ideas underlying the proposed algorithm and then a full description. We also give some theoretical analyses of its performance.

3.1 Ideas

In order to improve running speed, we construct a sparse graph structure $G = [V, E]$ where node set V and edge set E correspond to data points and a part of the messages between any two of the data points, respectively. This allows us to compute the message values efficiently. This is because the propagating responsibility and propagating availability that need to be computed are limited to those on existing edges. However, one may easily raise the following questions: how can we identify a set of necessary edges that yields the same clustering result as the original algorithm?

We can identify the edges whose convergence values do not need to be computed by estimating their upper and lower bounds. Based on this idea, we can construct a sparse graph that has no unnecessary edges. Moreover, as we describe later, the convergence values of pruned (unnecessary) messages can be computed from that of unpruned (necessary) messages. Therefore, after the iterations, we can compute the convergence values of pruned messages. This is our answer to the question. As a result, we can efficiently obtain the same clustering result as the original algorithm.

These new ideas provide a further advantage. Our upper/lower bound estimation does not need any user-defined inner-parameters, whereas FSAP has one parameter, K, for determining the K-nearest neighbor graph. The value of K impacts the clustering result. That is, our algorithm is user friendly.

3.2 Approach

We first describe how to construct the sparse graph structure, and give some theoretical highlights of the sparse graph structure. And then, we detail our clustering algorithm.

Sparse graph structure

We reduce the number of messages between data points in the iterations in order to compute the convergence values at low cost. To realize efficient computation, we construct graph $G = [V, E]$ where node set V consists of data points. We prune unnecessary messages by estimating upper/lower values of convergence values. We show the following lemma of convergence message values:

Lemma 1 (Convergence message) *After convergence, message values of responsibility and availability are* $r[i, j] = \rho[i, j]$ *and* $a[i, j] = \alpha[i, j]$*, respectively.*

Proof. It is easily found from Equation 1. □
This lemma implies that if the Affinity Propagation algorithm converges, its convergence values can be computed from the definition of propagating messages (Equation (2) and (3)).

By utilizing the above lemma, upper/lower values of convergence messages can be obtained from *just* pair-wise similarities. Formally, the following equations give upper/lower values of convergence message values:

Definition 1 (Upper/lower bound) *The following definitions give upper/lower bounding estimations* $\underline{a}[i, j]$*,* $\overline{r}[i, j]$*, and* $\overline{a}[i, j]$ *for convergence messages:*

$$\underline{a}[i,j] = \begin{cases} \min\{0, r[j,j]\} & (i \neq j) \\ 0 & (i = j) \end{cases} \tag{5}$$

$$\overline{r}[i,j] = \begin{cases} s[i,j] - \max_{k\neq j}\{\underline{a}[i,k] + s[i,k]\} & (i \neq j) \\ s[i,j] - \max_{k\neq j}\{s[i,k]\} & (i = j) \end{cases} \tag{6}$$

$$\overline{a}[i,j] = \begin{cases} \min\{0, \overline{r}[j,j] + \sum_{k\neq i,j} \max\{0, \overline{r}[k,j]\}\} & (i \neq j) \\ \sum_{k\neq i} \max\{0, \overline{r}[k,j]\} & (i = j) \end{cases} \tag{7}$$

Note that $\underline{a}[i, j]$ can be computed just from pair-wise similarities $s[i, j] (i, j = 1, 2, \dots, N)$ since $r[j, j] = s[j, j] - \max_{k\neq j}\{s[j, k]\}$ (Equation (2)). In addition, $\overline{r}[i, j]$ and $\overline{a}[i, j]$ can be computed from $\underline{a}[i, j]$ and $\overline{r}[i, j]$, respectively. Therefore, we do not need any iterations to compute these upper/lower bounding values.

We show the following lemmas to introduce the upper/lower properties of $\underline{a}[i, j]$, $\overline{r}[i, j]$, and $\overline{a}[i, j]$.

Lemma 2 (Upper/lower bound) *For any node pairs i and j in graph G,* $\underline{a}[i, j] \leq a[i, j]$*,* $\overline{r}[i, j] \geq r[i, j]$*, and* $\overline{a}[i, j] \geq a[i, j]$ *hold after convergence.*

Proof. We first prove that $\underline{a}[i, j] \leq a[i, j]$. From Lemma 1, $a[i, j] = \alpha[i, j]$ holds after convergence. If $i \neq j$, then $a[i, j] \geq \min\{0, r[j, j]\} = \underline{a}[i, j]$ holds since $\sum_{k\neq i,j} \max\{0, r[k, j]\} \geq 0$. If $i = j$, then $\underline{a}[i, j] \leq a[i, j]$ holds since $\sum_{k\neq i} \max\{0, r[k, i]\} \geq 0$. We next prove that $\overline{r}[i, j] \geq r[i, j]$. This is obvious from Lemma 1, Equation (2), and $\underline{a}[i, j] \leq a[i, j]$. We finally prove that $\overline{a}[i, j] \geq a[i, j]$. This is similarly obvious given Lemma 1, Equation (3), and $\overline{r}[i, j] \geq r[i, j]$. □

We construct the sparse graph structure by pruning unnecessary messages by using upper/lower bound estimations. Formally, we obtain the edge between node i and j as follows:

Definition 2 (Sparse graph) *The sparse graph G, where node set V consists of data points, has an edge between node i and j if and only if:*

$$\begin{aligned} &(1)\overline{r}[i,j] \geq 0, \text{ or} \\ &(2)\overline{a}[i,j] + s[i,j] \geq \max_{k\neq j}\{\underline{a}[i,k] + s[i,k]\} \end{aligned} \tag{8}$$

We introduce the following two lemmas to describe the property of our graph structure:

Lemma 3 (Convergence availability values) *For any node pair i and j, convergence values of availabilities can be exactly computed just from the convergence responsibilities that satisfy* $\overline{r}[i, j] \geq 0$.

Proof. From Lemma 1, $a[i, j] = \alpha[i, j]$ holds after convergence. If $\overline{r}[i, j] < 0$ holds for a pair, the responsibility of the pair does not affect the values of $\sum_{k\neq i,j} \max\{0, r[k, j]\}$ and $\sum_{k\neq i} \max\{0, r[k, j]\}$ in Equation (3) after convergence. □

Algorithm 1 Proposed algorithm

```
Input: pair-wise similarities
Output: exemplars of each data point
 1: for each data point pair $[i,j]$ do
 2:     compute $\underline{a}[i,j]$, $\overline{r}[i,j]$, and $\overline{a}[i,j]$ by Equation (5-7);
 3: end for
 4: for each data point pair $[i,j]$ do
 5:     if $\overline{r}[i,j] \geq 0$ or $\overline{a}[i,j]+s[i,j] \geq \max_{k\neq j}\{\underline{a}[i,k]+s[i,k]\}$ then
 6:         link data point pair $[i,j]$;
 7:     end if
 8: end for
 9: for $t = 1$ to $T$ do
10:     for each linked data point pair $[i,j]$ do
11:         update $r[i,j]$ and $a[i,j]$ by Equation (1);
12:     end for
13: end for
14: for each unlinked data point pair $[i,j]$ do
15:     compute $r[i,j] = \rho[i,j]$ and $a[i,j] = \alpha[i,j]$;
16: end for
17: for each data point $i$ do
18:     compute exemplar by Equation (4);
19: end for
```

Lemma 4 (Convergence responsibility values) *For any node pair i and j, convergence values of responsibilities can be exactly computed just from the convergence availabilities that satisfy $\overline{a}[i,j] + s[i,j] \geq \max_{k\neq j}\{\underline{a}[i,k] + s[i,k]\}$.*

Proof. From Lemma 1, $r[i,j] = \rho[i,j]$ holds after convergence. If $i \neq j$, $r[i,j] = s[i,j] - \max_{k\neq j}\{a[i,k] + s[i,k]\}$ holds from Equation (2) after convergence. Since $\max_{k\neq j}\{a[i,k]+s[i,k]\} \geq \max_{k\neq j}\{\underline{a}[i,k]+s[i,k]\}$ holds, if $\max_{k\neq j}\{\underline{a}[i,k] + s[i,k]\} > \overline{a}[i,j] + s[i,j]$ holds for node pair i and j, then $\max_{k\neq j}\{a[i,k] + s[i,k]\} > \overline{a}[i,j] + s[i,j]$ holds for the node pair. Therefore, the availability values of the pair does not affect the values of responsibilities. If $i = j$, responsibilities can be computed just from pair-wise similarities (see Equation (2)). □

From Lemma 3 and 4, we introduce the following lemma:

Lemma 5 (Convergence in the sparse graph structure) *For any node pair i and j, convergence values of messages can be exactly obtained by computing convergence values of edge messages in sparse graph G.*

Proof. This is obvious given Lemma 1, 3, and 4. □

Lemma 5 provides our algorithm with the property of clustering results which is the same as the original Affinity Propagation algorithm.

Clustering algorithm

Algorithm 1 is the full description of our algorithm. We first compute the upper/lower values of convergence messages by Definition 1 (lines 1-3), and then construct the graph structure by Definition 2 (lines 4-8). Since we can compute the convergence message values of all data point pairs from those of graph edges (Lemma 5), we iteratively update the message values of graph edges to compute convergence values (lines 9-13). Then we compute convergence message values of unlinked data point pairs after the iterations (lines 14-16). These values can be computed by propagating messages after the iterations (Lemma 1). We finally compute exemplars for all data points (lines 17-19).

Note that, our algorithm itself does not require any user-defined inner-parameters. Thus it provides to the user with a simple solution to Affinity Propagation with enhanced clustering speed.

3.3 Theoretical analyses

We provide theoretical analyses with regard to the clustering results and computation cost of our algorithm. Let M be the number of edges in the graph. That is $M = |E|$.

Theorem 1 (Clustering results) *After convergence, the clustering results of our algorithm are same as those of the original Affinity Propagation.*

Proof. It is obvious from Lemma 5. □

Theorem 2 (Computation cost) *Our algorithm requires $O(N^2 + MT)$ time to obtain exemplars for all data points.*

Proof. In the clustering process of our algorithm, we first compute the upper/lower values of convergence messages of all data point pairs, and construct the graph structure. These procedures require $O(N^2)$ time since the upper/lower values can be computed just from pair-wise similarities. Then, we iteratively compute the convergence message values for all edges in the graph, which needs $O(MT)$ time. We finally compute convergence message values of unlinked edges and exemplars for all data points which require $O(N^2 - M)$ and $O(N^2)$ time, respectively. Therefore, the computation cost of our algorithm is $O(N^2 + MT)$. □

4 Experimental evaluation

We conducted evaluation experiments to confirm the effectiveness of our algorithm. In these evaluations, we compare our algorithm to the original Affinity Propagation algorithm [Frey and Dueck, 2007] and FSAP [Jia *et al.*, 2008].

Our experiments will demonstrate that:

- Efficiency: Our algorithm outperforms the original algorithm and FSAP in terms of computation time (Section 4.1).
- Exactness: Unlike FSAP, our algorithm yields the same clustering result as the original algorithm (Section 4.2).
- Applicability: Case-studies on real world datasets shows that our algorithm and FSAP output different clustering results, and that the results of our algorithm are reasonable (Section 4.3). That is, our algorithm can be a better alternative for the original algorithm.

In this section, *Proposed* and *Original* represent the results of our algorithm and the original algorithm, respectively. FSAP is tested under several different values of K. *FSAP(5%)*, *FSAP(10%)*, and *FSAP(20%)* refer to the results of FSAP gained with K values of $0.05N$, $0.1N$, and $0.2N$, respectively. We iteratively computed responsibilities and availabilities a thousand times to obtain convergence values. We set the damping parameter, λ, to 0.5.

In the experiments, we used the following three datasets of different application domains that have different characteristics each other:

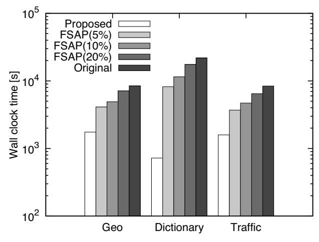


Figure 1: Clustering time.

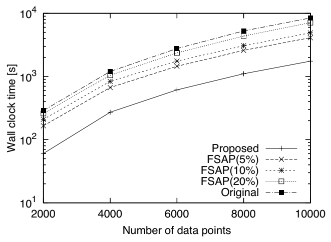
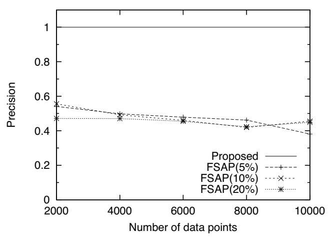
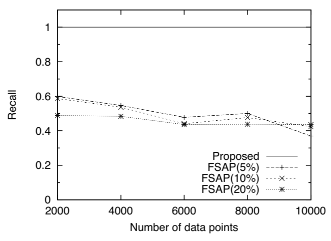


Figure 2: Scalability.



(1) Precision



(2) Recall

Figure 3: Accuracy of our algorithm and FSAP.

- *Geo*: This dataset was collected by downloading geo-tagged photographs from Flickr [2] using the site's public API [3]. The crawled data consists of $10,000$ photographs and their associated geotag data. All photos were taken in the central area of New York City, the United States between January 1st, 2006 and June 31st, 2009. Geo location of each photo is represented as its longitude and latitude (x, y). Similarity between photo i and j is the negative Euclidean distance.
- *Dictionary* [4]: This dataset was taken from word network in FOLDOC [5]. FOLDOC is a famous on-line, searchable, encyclopedic dictionary of computing subjects. Similarity between term i and j is computed by random walk with restart [Tong *et al.*, 2008] where relevance from term i to j is the frequency with which term j is used to describe the meaning of term i. The number of terms is $13,356$.
- *Traffic* [6]: This dataset contains loop sensor measurements of the Freeway Performance Measurement System. This loop sensor dataset was collected in Los Angeles from April 10th, 2005 to October 1st, 2005 (5 minute count aggregates). We extracted $10,000$ sequences from the sensor measurements. Similarity between sequence i and j is taken as the negative Euclidean distance.

All experiments were conducted on a Linux quad 3.33 GHz Intel Xeon server with 32GB of main memory. We implemented all algorithms using GCC.

4.1 Efficiency

We evaluated the clustering performance of each algorithm through wall clock time. Figure 1 shows the results. We also show the scalability of each algorithm in Figure 2; this figure shows the wall clock time as a function of the number of data points. We show only the result of *Geo* in Figure 2 due to space limitations.

These figures show that our algorithm is significantly faster than the original algorithm under all conditions examined. Specifically, our algorithm is more than 30 times faster than the original algorithm. As described in Section 2, the original algorithm requires $O(N^2T)$ while our algorithm requires $O(MT)$ time (see Theorem 2) in the iterations. And, as

Table 2: Number of edges in each algorithm.

Algorithm	Data set		
	Geo	*Dictionary*	*Traffic*
Proposed	10,009,002	2,799,319	10,034,228
Original	100,000,000	178,382,736	100,000,000

shown in Table 2, there are very few edges in our sparse graph (i.e., $M \ll N^2$) [7]. This is because a significant number of messages are pruned by our upper and lower bounding strategies. Therefore, our algorithm is an extremely efficient variant of the original algorithm.

And our algorithm is much faster than FSAP whose efficiency depends on the value of inner-parameter K as shown in the results. FSAP is a two-stage approach and it needs iterations in each stage (see Section 2.2). Therefore, our algorithm is faster than FSAP.

4.2 Exactness

One major advantage of our algorithm is that it guarantees the same clustering result as that of the original algorithm after convergence. However, this raises the following simple question: how successful is FSAP in providing the same clustering results?

To answer this question, we compared the exemplars extracted by our algorithm and FSAP to those extracted by the original algorithm. As the measures of accuracy, we used precision and recall. Precision is the fraction of exemplars by our algorithm that are match those of the original algorithm. And recall is the fraction of exemplars extracted by the original algorithm that were successfully extracted by our algorithm. We used *Geo* as the dataset in these experiments.

Figure 3-(1) and 3-(2) show the precision and recall, respectively. As we can see from Figure 3-(1) and 3-(2), the precision and recall of our algorithm are always 1. Because our algorithm is theoretically designed to compute convergence message values exactly, our algorithm does not sacrifice clustering accuracy. FSAP, on the other hand, has lower precision and recall. This is because FSAP determines edges based on heuristic ideas (see Section 2.2). The results shown in Figure 1, 2, and 3 confirm that our algorithm is superior to the original algorithm in terms of speed, and to FSAP in both of accuracy and speed.

[2] http://www.flickr.com/

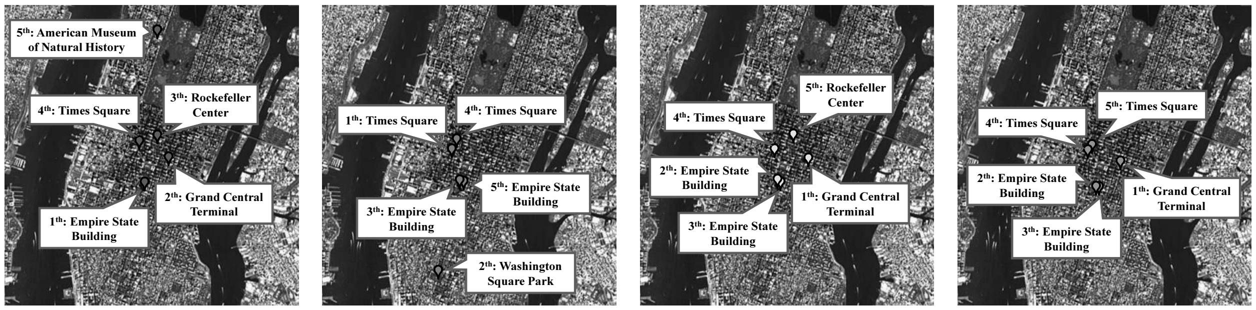
[3] http://maps.google.com/

[4] http://vlado.fmf.uni-lj.si/pub/networks/data/dic/foldoc/foldoc.zip

[5] http://foldoc.org/

[6] http://archive.ics.uci.edu/ml/datasets/Dodgers+Loop+Sensor

[7] Our additional experiments confirmed that a data pair which has higher similarity value has a higher chance to have an edge.


5th: American Museum of Natural History
4th: Times Square
3th: Rockefeller Center
2th: Grand Central Terminal
1th: Empire State Building
1th: Times Square
4th: Times Square
5th: Empire State Building
3th: Empire State Building
2th: Washington Square Park
5th: Rockefeller Center
4th: Times Square
2th: Empire State Building
1th: Grand Central Terminal
3th: Empire State Building
5th: Times Square
4th: Times Square
2th: Empire State Building
1th: Grand Central Terminal
3th: Empire State Building

(1) Proposed (2) FSAP(5%) (3) FSAP(10%) (4) FSAP(20%)

Figure 4: Top five popular landmarks to take photos in New York City detected by each algorithm.

4.3 Application example

To demonstrate the impact of our algorithm on real world applications, we give some examples of its clustering performance on *Geo*. The clustering of a geo-tagged photo dataset is regarded as the detection of popular landmarks that are frequently taken into photos, and this is recognized as a key process in many geo-location oriented applications such as geo-referenced image browsing [Crandall *et al.*, 2009] and landmark image search [Kennedy and Naaman, 2008].

Figure 4 shows exemplars of the top 5 (largest) clusters extracted by our algorithm and FSAP. To find a place name for a given exemplar (longitude/latitude), we used a public reverse geocoding tool based on georeferenced Wikipedia articles[8]. All algorithms detected famous landmarks such as the Empire State Building, Times Square, and Grand Central Terminal. However, FSAP detected the same landmark more than once. For instance, FSAP(5%) redundantly detected the Empire State Building (3rd and 5th) and Times Square (1st and 4th). On the other hand, all popular landmarks extracted by our algorithm were different from each other. The result of our algorithm is reasonable, and is consistent with our intuition. Note that the result of our algorithm equals that of the original algorithm. These results imply that our algorithm can be another option for the research community for clustering large datasets.

5 Conclusions

This paper proposed an efficient clustering algorithm for Affinity Propagation that gives the same clustering result as the original algorithm after convergence. Our algorithm is based on two ideas: (1) it prunes unnecessary message exchanges in the iterations, and (2) it computes the convergence values of pruned messages from those of unpruned messages. Experiments show that our algorithm can achieve efficient clustering without sacrificing clustering accuracy. Affinity Propagation is fundamental for many applications in various disciplines. The proposed solution allows many applications to be processed more efficiently, and helps to improve the effectiveness of future applications.

References

[Chang and Zheng, 2009] Xueping Chang and Zhonglong Zheng. Laplacian discriminant projection based on affinity propagation. In *Int. Conf. Artificial Intelligence and Computational Intelligence (AICI)*, pages 313–321, 2009.

[Crandall *et al.*, 2009] David J. Crandall, Lars Backstrom, Daniel Huttenlocher, and Jon Kleinberg. Mapping the world's photos. In *Int. World Wide Web Conf. (WWW)*, pages 761–770, 2009.

[Dueck and Frey, 2007] Delbert Dueck and Brendan J. Frey. Non-metric affinity propagation for unsupervised image categorization. In *IEEE Int. Conf. Computer Vision (ICCV)*, pages 1–8, 2007.

[Frey and Dueck, 2007] Brendan J. Frey and Delbert Dueck. Clustering by passing messages between data points. *Science*, 315:2007, 2007.

[Hu *et al.*, 2010] Rong Hu, Brian Mac Namee, and Sarah Jane Delany. Off to a good start: Using clustering to select the initial training set in active learning. In *Florida Artificial Intelligence Research Society Conf. (FLAIRS)*, 2010.

[Jain *et al.*, 1999] Anil K. Jain, M. Narasimha Murty, and Patrick J. Flynn. Data clustering: A review. *ACM Comput. Surv.*, 31(3):264–323, 1999.

[Jia *et al.*, 2008] Yangqing Jia, Jingdong Wang, Changshui Zhang, and Xian-Sheng Hua. Finding image exemplars using fast sparse affinity propagation. In *ACM Int. Conf. Multimedia (ACM MM)*, pages 639–642, 2008.

[Kennedy and Naaman, 2008] Lyndon S. Kennedy and Mor Naaman. Generating diverse and representative image search results for landmarks. In *Int. World Wide Web Conf. (WWW)*, pages 297–306, 2008.

[Liu *et al.*, 2009] Zhiyuan Liu, Peng Li, Yabin Zheng, and Maosong Sun. Clustering to find exemplar terms for keyphrase extraction. In *Conf. Empirical Methods in Natural Language Processing (EMNLP)*, pages 257–266, 2009.

[Tong *et al.*, 2008] Hanghang Tong, Christos Faloutsos, and Jia-Yu Pan. Random walk with restart: fast solutions and applications. *Knowl. Inf. Syst.*, 14(3):327–346, 2008.

[Zha *et al.*, 2009] Zheng-Jun Zha, Linjun Yang, Tao Mei, Meng Wang, and Zengfu Wang. Visual query suggestion. In *ACM Int. Conf. Multimedia (ACM MM)*, pages 15–24, 2009.

[8]http://www.geonames.org/export/wikipedia-webservice.html