K-AP: Generating Specified K Clusters by Efficient Affinity Propagation

Xiangliang Zhang*, Wei Wang†, Kjetil Nørvåg ‡, and Michèle Sebag §
* *Division of Mathematical and Computer Sciences and Engineering,*
King Abdullah University of Science and Technology (KAUST), Saudi Arabia
Email: xiangliang.zhang@kaust.edu.sa
† *Interdisciplinary Centre for Security, Reliability and Trust (SnT Centre), University of Luxembourg, Luxembourg*
Email: wwangemail@gmail.com
‡ *Department of Computer and Information Science, Norwegian University of Science and Technology (NTNU), Norway*
Email: kjetil.norvag@idi.ntnu.no
§ *TAO - LRI, CNRS, INRIA, Université Paris-Sud 11, France*
Email: michele.sebag@lri.fr

***Abstract*—The Affinity Propagation (AP) clustering algorithm proposed by Frey and Dueck (2007) provides an understandable, nearly optimal summary of a data set. However, it suffers two major shortcomings: i) the number of clusters is vague with the user-defined parameter called self-confidence, and ii) the quadratic computational complexity. When aiming at a given number of clusters due to prior knowledge, AP has to be launched many times until an appropriate setting of self-confidence is found. The re-launched AP increases the computational cost by 1 order of magnitude. In this paper, we propose an algorithm, called K-AP, to exploit the immediate results of K clusters by introducing a constraint in the process of message passing. Through theoretical analysis and experimental validation, K-AP was shown to be able to directly generate K clusters as user defined, with a negligible increase of computational cost compared to AP. In the meanwhile, K-AP preserves the clustering quality as AP in terms of the distortion. K-AP is more effective than k-medoids w.r.t. the distortion minimization and higher clustering purity.**

***Keywords*-clustering; affinity propagation; k-medoids;**

I. Introduction

The Affinity Propagation (AP) [5] is a clustering algorithm proposed to find out the most representative actual items of a data set, referred to as *exemplars*. In many application fields, finding out the *exemplars* is more interesting and informative than dividing items into clusters [10]. For example, the *exemplars* identified from the sentences of a document can be used for document summarization or abstraction. The quality of the set of exemplars is measured by *distortion*, which is the sum of squared distance between the data items and their associated exemplars.

As a traditional exemplar-based clustering algorithm, k-medoids also aims at finding out *exemplars*, known as *medoids*. It defines a combinatory optimization problem. Several algorithms have been proposed for the realization of k-medoids providing different trade-off between optimality and tractability, e.g., Partitioning Around Medoids (PAM) [6], CLARA [7], CLARANS [11] and EM algorithm based k-medoids which is a variant of k-means clustering algorithm [1, 5]. Differing from k-means that represents a cluster by an averaged artifact, k-medoids represents a cluster by a real item [6].

As shown in [5], AP provides optimality guarantee about minimizing the clustering *distortion*, compared to k-medoids. In counterpart for this guarantee, AP is limited by its quadratic computational complexity, and by the fact that it does not allow directly specifying the number of clusters. Instead, the number of clusters produced by AP is implicitly controlled by a user-defined parameter, which is the self-confidence for each item to be an exemplar. However, k-medoids has an advantage in directly generating a specified number of clusters.

In many application domains, prior knowledge indicates the number of clusters in the data. Clustering algorithms are thus required to generate a desired number of clusters. For example, clustering is applied for grouping documents into a given number of categories [12], and for recognizing the given number of social communities [13].

This paper aims at modifying AP to directly provide a given number of clusters while remaining all its advantages in clustering, e.g., the minimum *distortion*. Frey and Dueck [5] have suggested to re-launch AP many times with different parameters setting searched by bisection method until the desired number of clusters are found. AP suffers from the quadratic computational complexity. Re-launching AP to obtain a given number of clusters, therefore, increases the computational complexity by one order of magnitude. In order to generate specified K clusters, we introduce a constraint of limiting the number of clusters to be K for automatically adapting the message passing. The proposed K-AP method offers the same guarantee of optimality of AP and generates user-specified number K of clusters for negligible computational cost overhead compared to AP.

The rest of this paper is organized as follows. Section II describes our proposed clustering algorithm K-AP. Section

*This work was completed when the author was an ERCIM Postdoctoral fellow at Norwegian University of Science and Technology, Norway.

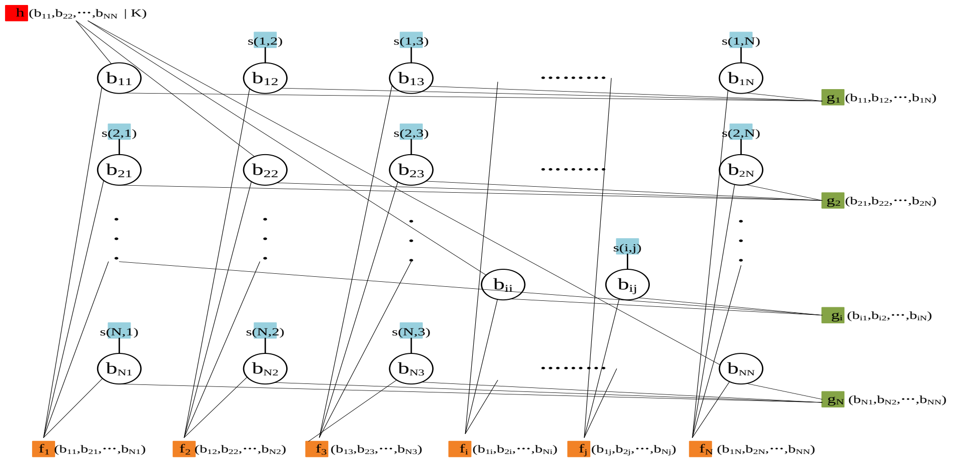


Figure 1. The grid-topology factor graph with N^2 binary variable nodes $\{b_{ij}\}$ (circles), $N(N-1)$ similarity function nodes $\{s(i,j)\}$ (blue square) and 3 types of constraint function nodes g_i (green square), f_i (yellow square) and h (red square)

III shows the validation of K-AP on several data sets from different application domains. Finally, Section IV concludes and gives our perspectives in further research.

II. Generating K clusters by message passing

A. Problem definition

We formalize the problem of extracting K exemplars (clusters) as follows. Given the data set $\mathcal{X} = \{x_1, ..., x_N\}$ and the similarities $S = \{s(x_i, x_j)\}$ between all pairs of x_i and x_j, the goal is to find out K exemplars $\mathcal{E} = \{e_1, ..., e_k\}$ which is a subset of $\mathcal{X}$ so that $\mathcal{E}$ maximizes:

$$E(\mathcal{E}) = \sum_{j=1}^{K} \sum_{x_i:\mathbf{c}(x_i)=e_j} s(x_i, e_j) \tag{1}$$

where $\mathbf{c}$ is a mapping between x_i and its closest exemplar $\mathbf{c}(x_i)$. A cluster with exemplar e_j can be defined as c_j which includes all x_i choosing e_j as its exemplar.

The problem of finding K exemplars can be re-formulated by 0-1 integer programming. Let us introduce binary variables $\{b_{ij} \in \{0,1\}, i,j = 1, ..., N\}$ indicating the exemplar (cluster) assignments: $b_{ij} = 1, i \neq j$ if x_i selects x_j as its exemplar, and $b_{ii} = 1$ if x_i itself is an exemplar. Based on these indicator variables, it naturally comes that maximization of function (1) is equal to *maximize*:

$$E(\{b_{ij}\}) = \sum_{i=1}^{N} \sum_{j=1}^{N} b_{ij} s(x_i, x_j) \tag{2}$$

subject to

$$\sum_{j=1}^{N} b_{ij} = 1, \quad b_{ii} = 1 \text{ if } \exists b_{ji} = 1, \quad \sum_{i=1}^{N} b_{ii} = K \tag{3}$$

The constraint functions in (3) respectively enforce that i) each item x_i can only have one exemplar; ii) if there is one item x_j selecting x_i as its exemplar, x_i must be an exemplar; iii) the number of exemplars must be K.

The programming problem of maximizing function (2) subject to constraints (3) can be expressed by a factor graph [3, 9]. A factor graph is a graphical model used for representing global functions which can be factored into simpler local functions. As shown in Figure 1, there exist N^2 binary variable nodes $\{b_{ij} \in \{0,1\}\}$ indicating the assignment of exemplars, and $N(N-1)$ function nodes $\{s(i,j)\}$ corresponding to the similarities of all pairs of items x_i and x_j. In addition, three types of constraint function nodes, $\{g_i\}$, $\{f_i\}$ and h, encode the constraints (3), respectively. The constraint functions are defined as follows.

Functions $\{g_i(b_{i1}, b_{i2}, ..., b_{iN}), i = 1, ..., N\}$ make sure that the clusters are disjoint and each item can only select one exemplar:

$$g_i(b_{i1}, b_{i2}, ..., b_{iN}) = \begin{cases} 0, & \text{if } \sum_{j=1}^{N} b_{ij} \neq 1; \\ 1, & \text{otherwise.} \end{cases} \tag{4}$$

Functions $\{f_i(b_{1i}, b_{2i}, ..., b_{Ni}), i = 1, ..., N\}$ enforce that one item must be an exemplar if any other items select it as its exemplar:

$$f_i(b_{1i}, b_{2i}, ..., b_{Ni}) = \begin{cases} 0, & \text{if } b_{ii} \neq 1 \textit{ but } \exists j : b_{ji} = 1; \\ 1, & \text{otherwise.} \end{cases} \tag{5}$$

Function $h(b_{11}, b_{22}, ..., b_{NN}|K)$ limits the number of exemplars (clusters) to be the given K:

$$h(b_{11}, b_{22}, ..., b_{NN}|K) = \begin{cases} 0, & \text{if } \sum_{i=1}^{N} b_{ii} \neq K; \\ 1, & \text{otherwise.} \end{cases} \tag{6}$$

The problem of finding out K exemplars is then transformed into searching over configurations of variable b_{ij} in the factor graph to maximize the global function

$$F(\mathbf{b};\mathbf{s};\mathbf{K}) = \prod_{i=1}^{N} (e^{b_{ii}} \prod_{j=1, j\neq i}^{N} e^{b_{ij} s(i,j)}) h(b_{11}, ..., b_{NN}|K) \prod_{j=1}^{N} f_j(b_{1j}, ..., b_{Nj}) \prod_{i=1}^{N} g_i(b_{i1}, ..., b_{iN}) \tag{7}$$

B. Belief Propagation for integer programming

Belief Propagation (BP) [14], a message passing algorithm for performing inference on graphical models, e.g., factor graph, has been shown to be an efficient way to solve the linear programming problem (linear relaxation of integer programming) [2]. It finds out the best configuration of variables for maximizing the global function (7) [2]. AP is a successful example of using BP to achieve the optimal clustering results. In a factor graph, the real valued messages are sent from a variable node to a factor node, and vice-versa. Figure 2 shows the messages passed between variable nodes and function nodes in the factor graph shown in Figure 1. After iteratively updating the passed messages until they have converged, the optimal result of b_{ij} can be decided by collecting all the received messages and by calculating the *beliefs* related to each individual variable node b_{ij} [2].

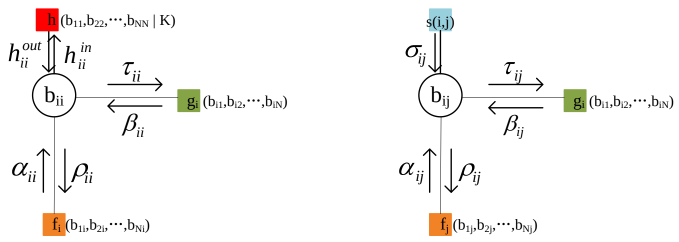


Figure 2. The messages passed between variable nodes and function nodes in factor graph

According to the BP algorithm, the message outgoing from one variable node to a function node is the element-wise product of all other incoming messages from the neighboring function nodes [14]. As shown in Figure 2, the messages sent out from variable node b_{ii} and b_{ij} to the other function nodes are:

$$\begin{array}{lll} \tau_{ii} = h_{ii}^{out}\cdot\alpha_{ii} & \rho_{ii} = h_{ii}^{out}\cdot\beta_{ii} & h_{ii}^{in} = \alpha_{ii}\cdot\beta_{ii} \\ \tau_{ij} = \sigma_{ij}\cdot\alpha_{ij} & \rho_{ij} = \sigma_{ij}\cdot\beta_{ij} & \end{array} \qquad (8)$$

Regarding the computing of messages sent from a function node to a variable node, there are two main BP algorithms for operating on factor graphes: max-product and sum-product [8]. Sum-product defines this outgoing message from a function node as the *sum of the product* of the function with messages from all the other neighboring variable nodes *over* all possible status of neighboring variable nodes. Max-product algorithm uses the *maximization* instead of the summary operator on the *product*. In the factor graph shown in Figure 1, each function node f_i or g_i is connected with N variables, and its outgoing messages are the sum or maximum of possible N elements. In this work, we use max-product to operate on factor graphes for the sake of numerical precision issues.

The messages outgoing from g_i-constraint nodes for max-product message-passing are:

$$\begin{aligned} \beta_{ij}(b_{ij}) &= \underset{t_1}{max}...\underset{t_{j-1}}{max}\,\underset{t_{j+1}}{max}...\underset{t_N}{max} \\ &\qquad [\,g_i(t_1,...,t_{j-1},b_{ij},t_{j+1},...,t_N)\textstyle\prod_{j':j'\neq j}\tau_{ij'}(t_{j'})] \\ &= \begin{cases} \prod_{j':j'\neq j}\tau_{ij'}(0), & \text{for } b_{ij}=1; \\ \underset{j':j'\neq j}{max}\left[\tau_{ij'}(1)\cdot\prod_{j'':j''\notin\{j,j'\}}\tau_{ij''}(0)\right], & \text{for } b_{ij}=0. \end{cases} \end{aligned}$$

where $\{t_{j'}=\{0,1\}\}$ are the possible states of all neighboring variable nodes. If $b_{ij}=1$, all of $\{t_{j'},j'\neq j\}$ are 0 to let g_i =1. Otherwise, only one of $\{t_{j'},j'\neq j\}$ can be 1 and all the others are 0.

$\beta_{ij}(b_{ij})$ and τ_{ij} are binary messages. They can be normalized by a scalar ratio [3]: $\tau_{ij}(0)=1$ and $\tau_{ij}(1)=\tau_{ij}$,

$$\beta_{ij} = \frac{\beta_{ij}(1)}{\beta_{ij}(0)} = \frac{1}{\underset{j':j'\neq j}{max}\,\tau_{ij'}} = \begin{cases} \frac{1}{max\{\tau_{ii},\ \underset{j':j'\notin\{i,j\}}{max}\tau_{ij'}\}} & \text{if } i\neq j; \\ \frac{1}{\underset{j':j'\neq i}{max}\tau_{ij'}}, & \text{if } i=j. \end{cases}$$

After normalizing the binary messages of $\beta_{ij}(b_{ij})$ and τ_{ij}, and defining the message sent from similarity function to a variable node as $\sigma_{ij}=e^{s(i,j)}$, from equation (8) we have the messages ρ_{ij} sent from a variable node to function node f_j computed by:

$$\rho_{ij} = \begin{cases} \frac{e^{s(i,j)}}{max\{h_{ii}^{out}\alpha_{ii},\ \underset{j':j'\notin\{i,j\}}{max}e^{s(i,j')}\alpha_{ij'}\}} & \text{if } i\neq j; \\ \frac{h_{ii}^{out}}{\underset{j':j'\neq i}{max}e^{s(i,j')}\alpha_{ij'}} & \text{if } i=j. \end{cases} \qquad (9)$$

The messages α_{ij} outgoing from constraints f_i to variable b_{ij} are computed by the max-product algorithm based on messages ρ_{ij} and function f_i, as:

$$\begin{aligned} \alpha_{ij}(b_{ij}) &= \underset{t_1}{max}...\underset{t_{i-1}}{max}\,\underset{t_{i+1}}{max}...\underset{t_N}{max} \\ &\qquad [\,f_j(t_1,...,t_{i-1},b_{ij},t_{i+1},...,t_N)\textstyle\prod_{i':i'\neq i}\rho_{i'j}(t_{i'})] \end{aligned}$$

After normalizing $\rho_{ij}(0)=1$, $\rho_{ij}(1)=\rho_{ij}$ and $\alpha_{ij}=\frac{\alpha_{ij}(1)}{\alpha_{ij}(0)}$, it comes

$$\alpha_{ij} = \begin{cases} \prod_{i':i'\neq i} max\{1,\rho_{i'j}\} & \text{if } i=j; \\ min\{1,\rho_{jj}\prod_{i':i'\notin\{i,j\}} max\{1,\rho_{i'j}\}\} & \text{if } i\neq j. \end{cases} \qquad (10)$$

As defined earlier, the outgoing messages of the constraint function $h(b_{11},...,b_{NN}|K)$ enforce the number of exemplars (clusters) to be a given value K. Let $U^K(\mathcal{Q})$ denote the subset of $\mathcal{Q}$ with K values, and $\mathcal{R}^t(\mathcal{Q})$ be the t-th largest value in data set $\mathcal{Q}$, the messages h_{ii}^{out} can be computed by the max-product algorithm considering the constraint function $h(b_{11},...,b_{NN}|K)$ and other $(N-1)$ variables $b_{jj}, j\neq i$ as

$$\begin{aligned} h_{ii}^{out}(b_{ii}) &= \underset{t_1}{max}...\underset{t_{i-1}}{max}\,\underset{t_{i+1}}{max}...\underset{t_N}{max} \\ &\qquad [h(t_1,...,t_{i-1},b_{ii},t_{i+1},...,t_N|K)\cdot\textstyle\prod_{i':i'\neq i}h_{i'i'}^{in}(t_{i'})] \end{aligned}$$

After normalizing, we then have

$$h_{ii}^{out} = \frac{\prod_{t=1}^{K-1}\mathcal{R}^t(\{h_{jj}^{in},j\neq i\})}{\prod_{t=1}^{K}\mathcal{R}^t(\{h_{jj}^{in},j\neq i\})} = \frac{1}{\mathcal{R}^K(\{h_{jj}^{in},j\neq i\})} \qquad (11)$$

where $\mathcal{R}^K(\{h_{jj}^{in}, j \neq i\})$ is the K-th largest value of h_{jj}^{in}, and $j \neq i$.

C. K-AP algorithm and discussion w.r.t. AP

Inspired from AP clustering algorithm [3, 5], we define the *responsibilities* as $r(i,j) = \log\rho_{ij}$, the *availabilities* as $a(i,j) = \log\alpha_{ij}$, the new messages *confidences* (η^{out} and η^{in}) as $\eta^{out}(i) = \log h_{ii}^{out}$ and $\eta^{in}(i) = \log h_{ii}^{in}$. From equation (8), (9), (10), and (11), we have the computation of *responsibilities, availabilities*, and *confidences* as shown in K-AP Algorithm 1.

Algorithm 1 K-AP Algorithm

```
Input: Similarities $\{s(i,j)\}_{i,j \in \{1,...,N\}\, i \neq j}$, K
Initialize:
    availabilities: $\forall i, j:\ a(i,j) = 0$
    confidence: $\forall i:\ \eta^{out}(i) = min(s)$
Repeat:
    update responsibilities, $\forall i, j$:
    $r(i,j) = s(i,j) - max\{\eta^{out}(i) + a(i,i), \max_{j':j' \notin \{i,j\}} \{s(i,j') + a(i,j')\}\}$
    $r(i,i) = \eta^{out}(i) - \max_{j':j' \neq i} \{s(i,j') + a(i,j')\}$
    update availabilities, $\forall i, j$:
    $a(i,j) = min\left\{0, r(j,j) + \sum_{i':i' \notin \{i,j\}} max\{0, r(i',j)\}\right\}$
    $a(j,j) = \sum_{i':i' \neq j} max\{0, r(i',j)\}$
    update confidences, $\forall i$:
    $\eta^{in}(i) = a(i,i) - \max_{j':j' \neq i} \{s(i,j') + a(i,j')\}$
    $\eta^{out}(i) = -\mathcal{R}^K(\{\eta^{in}(j), j \neq i\})$
    until converge
Output:
    clustering assignments: $\mathbf{c} = \{c_1, ..., c_N\}$
    $c_i = \underset{j}{argmax}\{a(i,j) + r(i,j)\}$
```

Contrasting to the original AP algorithm, the updates of *availabilities* in K-AP are the same. The update of *responsibilities* $r(i,j)$ and $r(i,i)$ differ as K-AP uses $\eta^{out}(i)$ instead of $s(i,i)$. Note that message $\eta^{out}(i)$ (like $s(i,i)$) outgoing from constraint function $h(b_{11}, ..., b_{NN}|K)$ indicates the confidence of an item to be an exemplar. While $s(i,i)$ is a preference defined by the users in AP, $\eta^{out}(i)$ is self-adapted by $\eta^{in}(i)$ according to the given K and constraints (3).

The computational complexity of K-AP is analyzed as follows. K-AP uses $\eta^{in}(i)$ and $\eta^{out}(i)$ to control the number of exemplars (clusters). Computing these two messages only increases $\mathcal{O}(2 * N)$ computational complexity. The update of *responsibilities* and of *availabilities* has quadratic complexity. The computational complexity of K-AP is as the same as that of AP, $\mathcal{O}(N^2)$. It is clear that K-AP is much more efficient than re-launching AP in order to generate a specified K clusters as suggested by Frey and Dueck [5]. The main contribution of K-AP is to achieve the desired number of clusters directly through message passing while keeping the same computational complexity.

III. Experimental validation of K-AP

In this section, we report the experimental validation of K-AP, after describing the goal of validation and the experimental settings.

A. The goal of validation and the experimental settings

The goal of the validation is to assess K-AP in terms of distortion and computational complexity for generating a given number of clusters, comparing to the re-launched AP and k-medoids. For the sake of fair comparison, k-medoids is independently run 2000 times with different random initialization to have the similar computation cost as K-AP, and the best running results w.r.t. the distortion is reported.

The clustering quality is measured by the *distortion*, which is the sum of the squared distance between each item and its exemplar, $D(\mathbf{c}) = \sum_{i=1}^{N} d^2(x_i, \mathbf{c}(x_i))$. The high quality clustering result has low distortion. If no special statement, the distances between the items are all measured by Euclidean metric, and the similarity of two items is the negative squared distance between them. All reported computational times were measured on a computer with Intel 2.66GHz Dual-Core and 2 GB memory in Matlab code[1].

If the data are labeled, the clustering quality can also be measured in the way of supervised learning, by comparing the label of an item with the label of the cluster it is associated (the label of its exemplar) with. Purity1 $= 100\% \times \frac{\sum_{i=1}^{K} |C_i^d|}{N}$ considers the total number of items belonging to the majority class in each cluster, where N is the number of items, $|C_i^d|$ is the number of items belonging to the majority class in cluster i. Purity2 $= 100\% \times (\sum_{i=1}^{K} \frac{|C_i^d|}{|C_i|})/K$ is based on the average over K clusters, where $|C_i|$ is the size of cluster i.

B. Data sets

In order to facilitate the comparison with the original AP, we validate K-AP firstly on two data sets which were used for AP in [5]. The first data set contains 25 data items with 2-dimensionality for visual validation while the second contains 900 face images of 10 persons in 10 different facial details, each of which is rotated and scaled into 9 images.

We also used another 5 data sets from UCI Machine Learning Repository [4], including IRIS plant, Breast Cancer Wisconsin (Original, Diagnostic and Prognostic), and Character Trajectories Data Set.

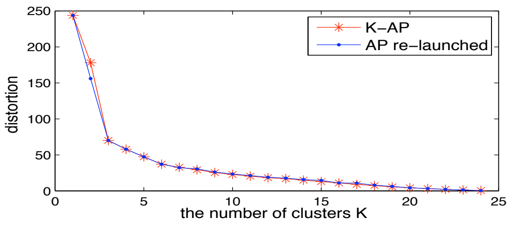
The IRIS data set contains 3 classes with 50 instances each. The Breast Cancer data set includes instances with two classes: benign or malignant. For both IRIS and Breast

[1]The Matlab code of K-AP is available.

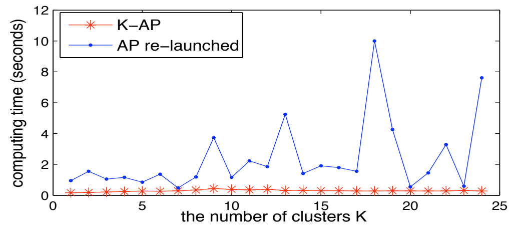
Cancer data sets, negative Manhattan distance is used to compute the similarity between two instances. Character Trajectories data set consists of 2858 character samples of 20 types of letters. Each character sample is a 3-dimensional pen tip velocity trajectory, described by a matrix with 3 rows and T columns, where T is the length of the character sample. We compute the similarity between two character w^1 and w^2 as $-\sum_{d=1}^{3}\sum_{t=1}^{T}|w_{dt}^1 - w_{dt}^2|$.

C. Experimental results

On the first data set with 25 items, we compare the clustering distortion and computational time of K-AP with that of re-launched AP approach given the number of clusters K ranging from 1 to 24 as shown in Figure 3.



(a)

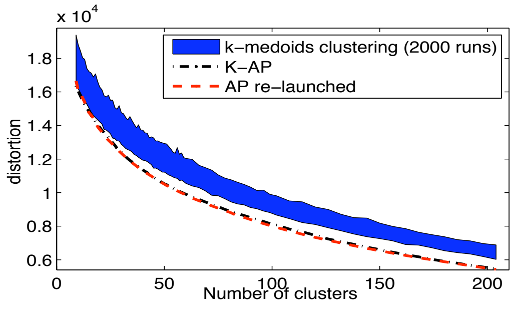


(b)

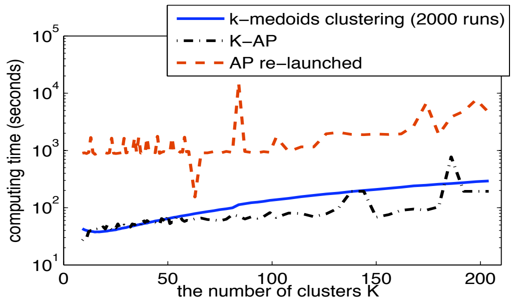
Figure 3. The distortion (a) and computational time (b) of K-AP and re-launched AP on a 2-dimensional data set.

It is seen that K-AP has the same clustering distortion as the re-launched AP approach except when $K = 2$. Meanwhile, K-AP only needs less than 1 second for the clustering, which is much less than the computational time of re-launched AP. Since the number of data items is very small (25 items), the efficiency of K-AP is not so obvious as clustering the face images data set.

On clustering a larger data set of face images, Figure 4 shows the clustering distortion and computational time of K-AP, re-launched AP approach and k-medoids method given the number of clusters K ranging from 9 to 204. The band in Figure 4 (a) shows all the distortion obtained by k-medoids in different runs. It is observed that K-AP has a lower distortion than the best case of k-medoids. While K-AP and re-launched AP approach have the similar performance on the distortion, K-AP improves by a factor 20 in term of the computation cost compared to re-launched AP, as shown in Figure 4 (b). Given a specified number of clusters K, K-AP needs much less computational time than re-launched AP



(a)



(b)

Figure 4. The distortion (a) and computational time (b) of K-AP, re-launched AP and k-medoids method on the data set of face images.

approach and achieves almost the same clustering results. The running time of K-AP has picks higher than the running time of k-medoids when $K = 138$ and $K = 186$ because of the large number of iterations required before converge.

As mentioned earlier, the face images were generated by scaling and rotating from 10 persons with 10 different facial details according to the prior knowledge of the data set. Each person has 90 images approximately and these images thus can be summarized into 10 exemplars (clusters). The 10 exemplars are expected to represent the 10 original different facial details of one person. Figure 5 left part shows the 90 images of a person, and each row has 9 images rotated and scaled from the same image. The right part of Fig 5 shows the 10 exemplars extracted from these 90 images by K-AP. These exemplars are different from each other, and correspond to the 10 different facial details. It is observed that the 10 examples represent very well the 10 different facial details.

Figure 6 shows several clusters of the face images and the exemplars obtained by K-AP on the whole data set with K=100. We show the clusters and the exemplars to check whether the images in one cluster are originally from the same person with the same facial detail, and whether the exemplar in this cluster is representative. In Figure 6, 3 clusters are shown as 3 rows of images, and the exemplars are marked by a light square. The first cluster contains the images of the same person who is speaking. The images in the second cluster are a man in smiling. The third cluster

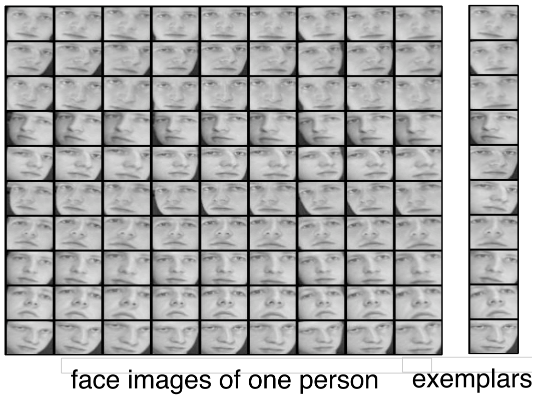


Figure 5. The face images of one person with 10 different facial details rotated and scaled (left part), together with the 10 exemplars obtained by K-AP (right part)

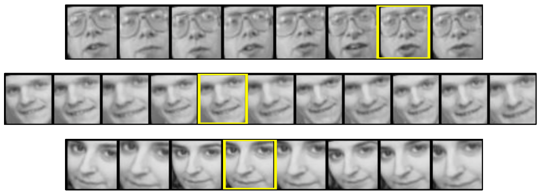


Figure 6. Several clusters of face images and the exemplars (marked by light square) obtained by K-AP when K=100

contains the images from the same girl who is smiling and looking at the right. The exemplar in each cluster is quite representative and can be used to represent the cluster.

The clustering results on the 5 UCI data sets are shown in Table I in which K-AP is compared to k-medoids method w.r.t. the clustering purity defined in section III-A. It is seen that K-AP can be applied for clustering data sets from different application fields and in most cases achieves higher clustering purity than k-medoids.

Table I
CLUSTERING RESULTS OF K-AP AND k-MEDOIDS ON 5 UCI DATA SETS

data set	N	D	K	method	purity1	purity2
IRIS	150	4	3	K-AP	**91.33%**	**91.42%**
				k-medoids	88.00%	88.89%
Breast Cancer (Original)	699	10	2	K-AP	**95.31%**	**95.49%**
				k-medoids	**95.31%**	**95.49%**
Breast Cancer (Diagnostic)	569	30	2	K-AP	**91.92%**	92.75%
				k-medoids	90.16%	**93.22%**
Breast Cancer (Prognostic)	198	33	2	K-AP	**76.26%**	**78.08%**
				k-medoids	**76.26%**	76.04%
Character Trajectories	2858	$3 \times T$	20	K-AP	**87.30%**	**88.61%**
				k-medoids	76.80%	85.29%

Through the validation results of K-AP, we see that K-AP method directly achieves a given number of clusters (exemplars) by a very low computational cost compared to re-launch AP and achieves optimal clustering distortion compared to k-medoids.

IV. CONCLUSION AND PERSPECTIVES

In this paper, we proposed a clustering algorithm, called K-AP, to enable generating a given number of optimal set of exemplars through affinity propagation. Through adding a constraint function to limit the number of clusters to be a requested one, the confidence of one data item to be an exemplar is automatically self-adapted in K-AP, while the confidence is a parameter specified by users in AP. From theoretical analysis and experimental validation results, K-AP has been shown to achieve the goal with negligible computational cost increment compared to original AP which solves the problem by increasing the computational cost by 1 order of magnitude. In the meanwhile, K-AP preserves the clustering quality in terms of distortion. In summary, K-AP is more efficient than AP w.r.t. the computational cost in generating specified K clusters, and more effective than k-medoids w.r.t. the distortion minimization and higher clustering purity.

Our further work is to combine K-AP with the Divide-and-Conquer strategy to achieve a quasi-linear complexity.

REFERENCES

[1] C. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[2] Y. Weiss C. Yanover, T. Meltzer. Linear programming relaxations and belief propagation – an empirical study. *Journal of Machine Learning Research*, 7:1887–1907, 2006.

[3] D. Dueck. *Affinity Propagation: Clustering Data by Passing Messages*. PhD thesis, University of Toronto, 2009.

[4] A. Frank and A. Asuncion. UCI machine learning repository, 2010.

[5] B. Frey and D. Dueck. Clustering by passing messages between data points. *Science*, 315:972–976, 2007.

[6] L. Kaufman and P. Rousseeuw. Clustering by means of medoids. In *Statistical Data Analysis Based on the L1 Norm and Related Methods*, pages 405–416. 1987.

[7] L. Kaufman and P.J. Rousseeuw. *Finding Groups in Data: an introduction to cluster analysis*. Wiley, 1990.

[8] F. Kschischang, B. Frey, and H. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47:498–519, 2001.

[9] H. Loeliger. An introduction to factor graphs. *IEEE Signal Processing Magazine*, pages 28–41, 2004.

[10] M. Mezard. Computer science: Where are the exemplars? *Science*, 315:949–951, 2007.

[11] R. Ng and J. Han. Efficient and effective clustering methods for spatial data mining. In *VLDB*, pages 144–155, 1994.

[12] M. Rege, M. Dong, and F. Fotouhi. Co-clustering documents and words using bipartite isoperimetric graph partitioning. In *ICDM*, pages 532–541, 2006.

[13] T. Yang, R. Jin, Y. Chi, and S. Zhu. Combining link and content for community detection: a discriminative approach. In *SIGKDD*, pages 927–936, 2009.

[14] J. Yedidia, W. Freeman, and Y. Weiss. Understanding belief propagation and its generalizations. *Exploring artificial intelligence in the new millennium*, pages 239–269, 2003.