Journal of Data Intelligence, Vol. 1, No. 1 (2020) 055–074
© Rinton Press

FAST AFFINITY PROPAGATION BY CELL-BASED INDEXING

HIROAKI SHIOKAWA
Center for Computational Sciences, University of Tsukuba
1-1-1 Tennodai, Tsukuba, Ibaraki 305–8573, Japan
shiokawa@cs.tsukuba.ac.jp

TOMOHIRO MATSUSHITA
Graduate School of SIE, University of Tsukuba
1-1-1 Tennodai, Tsukuba, Ibaraki 305–8573, Japan
matsushita@kde.cs.tsukuba.ac.jp

HIROYUKI KITAGAWA
Center for Computational Sciences, University of Tsukuba
1-1-1 Tennodai, Tsukuba, Ibaraki 305–8573, Japan
kitagawa@cs.tsukuba.ac.jp

Affinity Propagation is one of the fundamental clustering algorithms used in various Web-based systems and applications. Although Affinity Propagation finds highly accurate clusters, it is computationally expensive to apply Affinity Propagation to a large dataset since it requires iterative computations for all possible pairs of data objects in the dataset. To address the aforementioned issue, this paper presents efficient Affinity Propagation algorithms, namely *C-AP*. In order to increase the clustering speed, C-AP employs *cell-based index* to reduce the number of the computed data object pairs in the clustering procedure. By using the cell-based index, C-AP efficiently detects unnecessary pairs, which do not contribute to its clustering result. For further reducing the computation time, we also present an extension of our algorithm named *Parallel C-AP* that utilizes thread-parallelization techniques. As a result, C-AP and Parallel C-AP detects the same clusters as those of Affinity Propagation with much shorter computation time. Extensive evaluations demonstrate the performance superiority of our proposed algorithms over the state-of-the-art algorithms.

Keywords: Clustering, Affinity Propagation, Efficiency

1. Introduction

Recent advances in Web services have shown that large-scale data analysis is becoming increasingly important to understand or predict complicated phenomena included in the services [1]. The *clustering* is one of the essential data-mining techniques to understand such complicated and large-scale datasets in various research areas such as Web-based applications and social sciences. Since clustering can extract groups of data objects (*a.k.a.* cluster) or identify representative examples in an unsupervised way, it can reveal overview relationships among the data objects and find hidden patterns of them. From a practical perspective, it provides us useful insights into Web-based applications. For example, web pages that share similar topics tend to be in the same cluster [2]. Thus, the detection of clusters in web pages is useful in stripping spam pages from web pages [3, 4]. As well as the web page analysis, clustering plays important roles in various Web-based applications such as event detection [5] and marketing [6]. That is why extracting clusters from the large-scale dataset on Web-based applications have become an interesting and essential problem.

The problem of finding clusters in a set of data objects has studied for some decades in many fields [7, 8, 9]. *Affinity Propagation* [10], proposed by Frey and Dueck in 2007, is one of the most successful methods among recent works for the cluster finding problem. Affinity Propagation finds clusters and their corresponding representative data objects called *exemplar* from all data objects. By letting $X = \{x_1, x_2, \cdots, x_n\}$ be a set of given data objects, $s(x_i, x_j)$ be a similarity value (*e.g.,* Euclidean distance) between x_i and x_j, Affinity Propagation attempts to detect an exemplar $e(x_i)$ for each $x_i \in X$ so as to maximize an objective measure $\sum_{i=1}^{n} s(x_i, e(x_i))$. After finding the exemplars for all data objects in X, Affinity Propagation constructs clusters by assigning each data object x_i into the same cluster with $e(x_i)$. Different from the traditional clustering algorithms (*e.g.,* k-means [11], k-medoids [12], and so on), Affinity Propagation does not require the number of exemplars before the cluster computation; Affinity Propagation automatically determines the number of clusters from given data objects X. As a result, Affinity Propagation shows better clustering accuracy compared with the traditional algorithms on real-world datasets [10]. Therefore, Affinity Propagation has widely used in many applications such as community detection [13], and representative image extraction [14, 15].

Although Affinity Propagation is effective in detecting clusters, it has, unfortunately, a serious weakness; it requires high computational costs to find exemplars that maximize $\sum_{i=1}^{n} s(x_i, e(x_i))$ [10]. This is because Affinity Propagation explores the exemplars from all data objects by message passing among all data objects. Affinity Propagation uses $s(x_i, x_j)$ for all data object pairs $(x_i, x_j) \in X \times X$, and it then iteratively exchanges *messages* between the data objects until a set of exemplars is specified. In each iteration, each message reflects the affinity that a data object has for another data object considered as its exemplar. As a result, this message passing procedure entails $O(n^2T)$ times, where n is the number of data objects and T is the number of iterations for the message updates in the worst case [16].

1.1. *Existing Approaches and Challenges*

To address the expensive computational cost of Affinity Propagation, many efforts have been made for the recent few years, especially in the data mining community. One of the major approaches is message pruning technique: FSAP [17], Graph-AP [18], and F-AP [16] are the most representative methods. These methods prune data object pairs that do not contribute to explore the exemplars to avoid unnecessary message updates during the iterative computation. Although these methods certainly succeeded in reducing the runtime of Affinity Propagation for real-world datasets, the computation time is still expensive even if we compute small datasets (*e.g.,* $n \approx 10^3$.) This is because that the methods still spend $O(n^2)$ times for detecting the data object pairs that can be pruned in each iteration since they need to traverse all pairs of the data objects in X. Thus, it is a challenging task to improve the computational efficiency for Affinity Propagation.

1.2. *Our Approach and Contributions*

We focus on the problem of speeding up Affinity Propagation for large datasets. In this paper, we first present a novel algorithm, *C-AP* (<u>C</u>ell-based <u>A</u>ffinity <u>P</u>ropagation), that enables us to reduce the computation time of Affinity Propagation. The basic idea underlying C-AP is to reduce the cost to detect the data object pairs that should be pruned in the existing approaches [16, 18]. As we described in Section 1.1, the existing approaches require $O(n^2)$ times to prune unnecessary data object pairs in each iteration. To avoid such expensive costs, we employ *cell-based indexing* into Affinity Propagation algorithms. By providing the cell-based index, we partition the given data object set X into several cells that contain at least one data object. Then, C-AP seeks unnecessary data object pairs by traversing all pairs of the cells. Since the number of the cells should be smaller than n^2, C-AP successfully reduces the computational cost of Affinity Propagation compared with the existing approaches [18, 16].

In addition to the cell-based indexing, we discuss an extended algorithm of C-AP for further improving the clustering efficiency for large-scale datasets. This paper presents a novel algorithm named *Parallel C-AP*.[a] Parallel C-AP employs thread-based parallelization techniques to reduce the computation time of C-AP since our previous algorithm C-AP consists of several independent loop-blocks (Algorithm 2). By applying loop-level parallelization for the blocks, Parallel C-AP significantly reduces the running time for large-scale clustering.

As a result, our proposed algorithms have the following attractive characteristics:

- **Efficient:** Compared with the existing approaches [10, 18, 16], C-AP and Parallel C-AP achieve high-speed clustering by using the cell-based indexing and thread-based parallelization, respectively (Section 4.3); C-AP can avoid computing unnecessary data object pairs for the whole dataset (Section 4.4).

- **Scalable:** Our proposed algorithms show better scalability than the original Affinity Propagation algorithm [10] in terms of the number of data objects (Section 4.5). Parallel C-AP also shows near-linear scalability when we increase the number of threads.

- **Exact:** While our proposed algorithms achieve efficient and scalable computations for Affinity Propagation, we theoretically proved that C-AP and Parallel C-AP does not sacrifice the clustering accuracy (Theorem 1). That is, C-AP and Parallel C-AP always return the same clustering result as those of the original Affinity Propagation [10] (Section 4.6).

Our extensive experiments showed that C-AP is up to $\times 6.85$ and $\times 3.77$ faster than the original Affinity Propagation algorithm [10] and the state-of-the-art algorithms [18, 16], respectively while C-AP does not sacrifice the clustering quality. Also, our parallel algorithm Parallel C-AP further reduces the running time of C-AP; Parallel C-AP runs $\times 15.97$, $\times 23.77$, and $\times 52.92$ faster than C-AP, the state-of-the-art methods, and the original Affinity Propagation, respectively. Even though Affinity Propagation is effective in enhancing various applications, it has been difficult to apply Affinity Propagation to large datasets due to its performance limitations. However, by providing our efficient and scalable approaches, our proposed algorithms will help to improve the effectiveness of a broader range of applications.

The rest of this paper is organized as follows: Section 2 briefly describes the backgrounds of this paper. Section 3 introduces our proposed methods C-AP and Parallel C-AP, and we report the experimental results on public datasets in Section 4. In Section 5, we briefly review the related work, and we finally conclude this paper in Section 6.

[a]This work is an extension of [19].

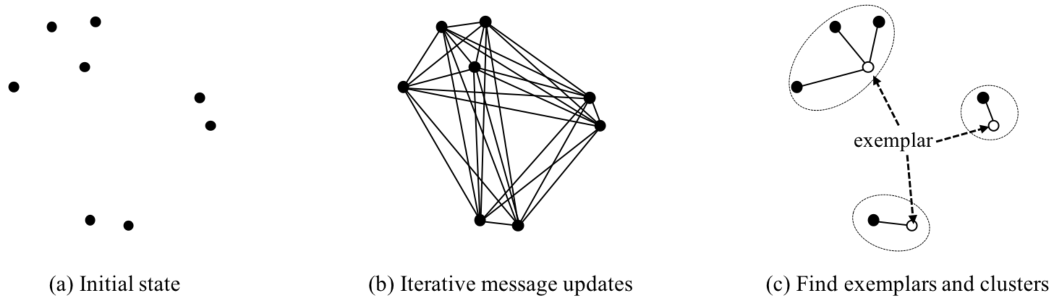
exemplar
(a) Initial state
(b) Iterative message updates
(c) Find exemplars and clusters

Fig. 1. Clustering proceedure of Affinity Propagation: (a) It first takes n data objects and a set of similarities between all pairs of the data objects; (b) Then, it iteratively updates the responsibility and the availability based on Definition 1 for all of the pairs; (c) Finally, it finds an exemplar based on Definition 2 for each data objects, and constructs clusters.

2. Preliminary

In this section, we formally define the notations and introduces the background of this paper. Given a set of d-dimentional data objects $X = \{x_1, x_2, \ldots, x_i, \ldots, x_n\}$ and a set of similarities $S = \{s(x_i, x_j)|x_i, x_j \in \mathbb{X}\}$, Affinity Propagation [10] is the problem to find clusters and corresponding exemplars that maximize $\sum_{i=1}^{n} s(x_i, e(x_i))$, where $e(x_i)$ is an exemplar of x_i. Note that if $x_i = x_j$, $s(x_i, x_j)$ is called as *preference*, and it is typically set to the madian or the minimum of S. Differ from the traditional clustering algorithms (e.g., k-means [11]), Affinity Propagation does not require the number of clusters, and thus it can automatically determine an appropriate number of clusters from given X and S.

Since it is NP-hard problem to find exemplars so as to maximize $\sum_{i=1}^{n} s(x_i, e(x_i))$, Affinity Propagation thus performs an approximated algorithm by using an iterative computation method. Specifically, it iteratively updates two types of messages, *responsibility* and *availability*, between all pairs of the data objects. The responsibility, $r(x_i, x_j)$, is a message that is sent from x_i to x_j to denote how well x_j is appropriate to be the exemplar of x_i. The availability, $a(x_i, x_j)$, is also a message that is sent from x_j to x_i that reflects how appropriate it would be for x_i to choose x_j as its exemplar. Formally, the responsibility $r(x_i, x_j)$ and the availability $a(x_i, x_j)$ are defined as follows:

Definition 1 (Responsibility and Availability) *Let λ be a damping factor ($0 < \lambda < 1$), responsibility $r(x_i, x_j)$ and availability $a(x_i, x_j)$ are real numbers between x_i and x_j that are obtained by*

$$r(x_i, x_j) = (1-\lambda)\rho(x_i, x_j) + \lambda r(x_i, x_j), \quad (1)$$

$$a(x_i, x_j) = (1-\lambda)\alpha(x_i, x_j) + \lambda a(x_i, x_j), \quad (2)$$

where $\rho(x_i, x_j)$ and $\alpha(x_i, x_j)$ are propagating responsibility and propagating availability [18], respectively; $\rho(x_i, x_j)$ and $\alpha(x_i, x_j)$ are formally defined as follows:

$$\rho(x_i, x_j) = \begin{cases} s(x_i, x_j) - \max_{x_k \neq x_j}\{a(x_i, x_k) + s(x_i, x_k)\} & (x_i \neq x_j), \\ s(x_i, x_j) - \max_{x_k \neq x_j}\{s(x_i, x_k)\} & (x_i = x_j). \end{cases} \quad (3)$$

Algorithm 1 The original Affinity Propagation [10]

```
Input: $X$, $S$
Output: exemplars for each data object in $X$
1: repeat
2:     for each $(x_i, x_j) \in X^2$ do
3:         compute $r(x_i, x_j)$ by Definition 1;
4:     for each $(x_i, x_j) \in X^2$ do
5:         compute $a(x_i, x_j)$ by Definition 1;
6: until all $r(x_i, x_j)$ and $a(x_i, x_j)$ are not updated
7: for each $x_i \in X$ do
8:     get an exemplar $e(x_i)$ by Definition 2;
```

$$\alpha(x_i, x_j) = \begin{cases} \min\{0, r(x_j, x_j) + \sum_{x_k \neq x_i, x_j} \max\{0, r(x_k, x_j)\}\} & (x_i \neq x_j), \\ \sum_{x_k \neq x_i} \max\{0, r(x_k, x_j)\} & (x_i = x_j). \end{cases} \tag{4}$$

Note that λ is typically set to 0.5 in the literature [10, 18, 16], and initial values of the responsibility and the availability are set as $r_0(x_i, x_j) = s(x_i, x_j) - \max_{x_k \neq x_j} \{s(x_i, x_k)\}$ and $a_0(x_i, x_j) = 0$, respectively.

As we can see from Definition 1, responsibility $r(x_i, x_j)$ and availability $a(x_i, x_j)$ are mutual recursion with each other; $r(x_i, x_j)$ is a polynominal function of $a(x_i, x_j)$ in $\rho(x_i, x_j)$, and visa versa. In order to compute such mutual recursion forms, Affinity Propagation iterates alternate updates of $r(x_i, x_j)$ and $a(x_i, x_j)$ for all data object pairs until $r(x_i, x_j)$ and $a(x_i, x_j)$ converge.

After the convergence of the responsibility and the availability on all data object pairs, the exemplar of each data object x_i, $e(x_i)$, is determined as follows:

Definition 2 (Exemplar) *Let x_i be an data object in X, the exemplar of x_i is defined as*

$$e(x_i) = \arg\max_{x_j} \{r(x_i, x_j) + a(x_i, x_j)\}. \tag{5}$$

Finally, Affinity Propagation constructs clusters by assigning non-exemplar data objects into the same cluster as an exemplar that has the largest similarity.

Figure 1 shows an overview of the original Affinity Propagation, and the pseudocode of the algorithm is shown in Algorithm 1. As we can see from Algorithm 1, by letting n and T be the number of data objects and iterations, respectively, Affinity Propagation takes $O(n^2T)$ computation time to obtain the converged responsibilities and availabilities (lines 1-6).

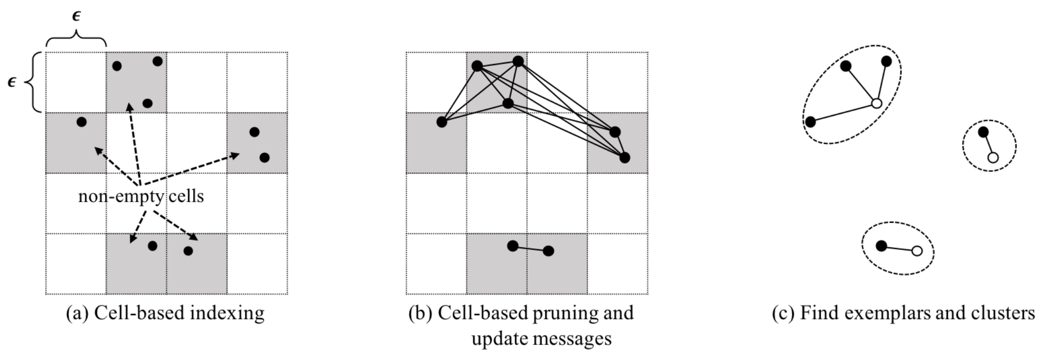
ϵ
ϵ
non-empty cells
(a) Cell-based indexing
(b) Cell-based pruning and update messages
(c) Find exemplars and clusters

Fig. 2. Clustering proceedure of C-AP: (a) First, C-AP constructs a cell-based index for the given set of data objects X; (b) Then, it performs cell-based pruning to reduce the number of data object pairs to be computed, and iteratively updates messages for the remaining pairs; (c) Finally, it finds an exemplar based on Definition 2 for each data objects, and constructs clusters.

3. Proposed method: C-AP

Our goal is to find the same clustering results as those of the original Affinity Propagation algorithm shown in Section 2 within short runtimes. In this section, we present the details of our proposal, C-AP. We first overview the main ideas underlying C-AP, and then give a full description of our proposed approaches.

3.1. *Main Ideas*

The basic idea underlying C-AP is to reduce the computational costs for obtaining the exemplars for all data objects. From Definition 2, we need to compute the converged responsibility and availability for all data object pairs before determining exemplars. As discussed in Section 2, this computation incurs a high clustering cost since it iteratively updates the responsibility and availability for all data object pairs until convergence. Thus, it is essential to reduce the number of data object pairs whose responsibility and availability should be updated in each iteration.

To reduce the number of data object pairs to be updated, C-AP employs the following two approaches:

(1) Cell-based Indexing: C-AP first constructs *cell-based index*; it enables us to exclude unnecessary data object pairs from updating their responsibilities and availabilities. C-AP divides d-dimensional data space, where given data objects X are located, into d-dimensional hypercubes whose side length equals to ϵ; we call each d-dimensional hypercube as *cell*. By providing the cell-based index, C-AP partitions the given data object set X into several cells that contain at least one data object.

(2) Cell-based Message Pruning: C-AP, then, performs *cell-based message pruning* to prune pairs of cells that contain unnecessary data object pairs from the iterative message updates. Recall Definition 2, Affinity Propagation selects a data object x_j as an exemplar of x_i only if x_j maximizes $r(x_i, x_j) + a(x_i, x_j)$ in X. This means that we do not need to compute a data object pair (x_i, x_j) whose value of $r(x_i, x_j) + a(x_i, x_j)$ is certainly smaller than another data objects in X after the convergence. Thus, C-AP excludes pairs of cells that only have unnecessary data object pairs by computing upper and lower estimates of the $r(x_i, x_j)$ and $a(x_i, x_j)$ for each cell pair before performing each iteration.

These simple ideas have two main advantages. First, we can extract clusters with smaller computational cost than the original algorithm [10] and the state-of-the-art algorithms [18, 16]. Our ideas successfully reduce the number of computed data object pairs by introducing the cell-based indexing and the cell-based message pruning. As discussed in Section 2, Affinity Propagation entails $O(n^2)$ times complexity in each iteration. In contrast, our proposal requires at most $O(c^2)$ time complexity in each iteration, where c is the number of cells (*s.t.* $c \ll n$), each of which contains at least one data object. In Section 4.5, we experimentally confirmed how C-AP successfully reduces the number of data object pairs to be computed in the clustering procedure. Second, our algorithm produces the same clustering results as those of the original algorithm [10] even though we prune data object pairs by using cell-based message pruning. Our pruning techniques guarantee the pruned data object pairs would never be the exemplars even if the message updates have not reached convergence. In Section 3.4, we theoretically confirm the exactness of our proposed pruning techniques.

3.2. *Cell-based Indexing*

In this section, we introduce the cell-based index. As we described in Section 2, a set of d-dimensional data objects $X = \{x_1, x_2, \ldots, x_n\}$ and a set of similarities $S = \{s(x_i, x_j) | x_i, x_j \in X\}$ are given. From X and S, C-AP constructs an arbitrary grid as an index on the d-dimensional data space where X locates; each cell of the grid is d-dimensional hypercube whose side length equals to ϵ. Figure 2 (a) shows an indexing example when $d = 2$. A cell C of the index is *non-empty* if it contains at least one data object of X; otherwise, C is *empty*; we denote that c as the number of *non-empty* cells. c is clearly less than n non-empty cells on the index, and typically $c \ll n$.

Based on the cell-based index, we define a cell-based similarity function $cs(C_i, C_j)$ that evaluates the similarity between two non-empty cells. The definition of $cs(C_i, C_j)$ is as follows:

Definition 3 (Cell-based similarity) *Let $cs(C_i, C_j)$ be the cell-based similarity function that evaluates the similarity between C_i and C_j. $cs(C_i, C_j)$ is defined as*

$$cs(C_i, C_j) = \begin{cases} \max\{s(x_a, x_b) | x_a \in C_i \text{ and } x_b \in C_j\} & (C_i \neq C_j), \\ \textit{preference} & (C_i = C_j). \end{cases} \tag{6}$$

3.3. *Estimating Upper/Lower Bounds of Messages*

To prune unnecessary data object pairs on the cell-based index, we employ the well-known upper and lower bounds estimation techniques of the responsibility and the availability [18, 20]. In this section, we briefly introduce the definitions.

3.3.1. *Upper and Lower bounds of responsibility*

Definition 4 (Upper and Lower Estimates) *Let $\overline{r}(x_i, x_j)$ and $\underline{r}(x_i, x_j)$ be the upper and lower bounds of the responsibility $r(x_i, x_j)$ between two data objects x_i and x_j. $\overline{r}(x_i, x_j)$ and $\underline{r}(x_i, x_j)$ are formally defined as follows:*

$$\overline{r}(x_i, x_j) = \begin{cases} \lambda r_0(x_i, x_j) + \overline{\rho}(x_i, x_j) & (r_0(x_i, x_j) > 0, \overline{\rho}(x_i, x_j) > 0), \\ \lambda r_0(x_i, x_j) + (1-\lambda)\overline{\rho}(x_i, x_j) & (r_0(x_i, x_j) > 0, \overline{\rho}(x_i, x_j) \leq 0), \\ \overline{\rho}(x_i, x_j) & (r_0(x_i, x_j) \leq 0, \overline{\rho}(x_i, x_j) > 0), \\ (1-\lambda)\overline{\rho}(x_i, x_j) & (r_0(x_i, x_j) \leq 0, \overline{\rho}(x_i, x_j) \leq 0). \end{cases} \quad (7)$$

$$\underline{r}(x_i, x_j) = \begin{cases} (1-\lambda)\underline{\rho}(x_i, x_j) & (r_0(x_i, x_j) > 0, \underline{\rho}(x_i, x_j) > 0), \\ \underline{\rho}(x_i, x_j) & (r_0(x_i, x_j) > 0, \underline{\rho}(x_i, x_j) \leq 0), \\ \lambda r_0(x_i, x_j) + (1-\lambda)\underline{\rho}(x_i, x_j) & (r_0(x_i, x_j) \leq 0, \underline{\rho}(x_i, x_j) > 0), \\ \lambda r_0(x_i, x_j) + \underline{\rho}(x_i, x_j) & (r_0(x_i, x_j) \leq 0, \underline{\rho}(x_i, x_j) \leq 0), \end{cases} \quad (8)$$

where $\overline{\rho}(x_i, x_j)$ and $\underline{\rho}(x_i, x_j)$ are upper and lower bounds of the propagating responsibility $\rho(x_i, x_j)$, respectively. They are formally given as follows:

$$\overline{\rho}(x_i, x_j) = \begin{cases} s(x_i, x_j) - s(x_i, x_i) & (x_i \neq x_j), \\ s(x_i, x_j) - \max\limits_{x_k \neq x_j} \{s(x_i, x_k)\} & (x_i = x_j). \end{cases} \quad (9)$$

$$\underline{\rho}(x_i, x_j) = \begin{cases} s(x_i, x_j) - \max\limits_{x_k \neq x_j} \{\overline{a}(x_i, x_k) + s(x_i, x_k)\} & (x_i \neq x_j), \\ s(x_i, x_j) - \max\limits_{x_k \neq x_j} \{s(x_i, x_k)\} & (x_i = x_j). \end{cases} \quad (10)$$

As can be seen from the above equations, we can obtain the upper and lower bounds of the responsibility by just using the similarities. Hence, we can estimate both bounds before iteratively exchanging the messages.

3.3.2. *Upper and lower bounds of availability*

Definition 5 (Upper and Lower Estimates) *Upper and lower bounds of availability, $\overline{a}(x_i, x_j)$ and $\underline{a}(x_i, x_j)$ are computed as follows:*

$$\overline{a}(x_i, x_j) = \begin{cases} \overline{\alpha}(x_i, x_j) & (\overline{\alpha}(x_i, x_j) > 0), \\ (1-\lambda)\overline{\alpha}(x_i, x_j) & (\overline{\alpha}(x_i, x_j) \leq 0). \end{cases} \quad (11)$$

$$\underline{a}(x_i, x_j) = \begin{cases} (1-\lambda)\underline{\alpha}(x_i, x_j) & (\underline{\alpha}(x_i, x_j) > 0), \\ \underline{\alpha}(x_i, x_j) & (\underline{\alpha}(x_i, x_j) \leq 0), \end{cases} \quad (12)$$

where $\overline{\alpha}(x_i, x_j)$ and $\underline{\alpha}(x_i, x_j)$ are upper and lower bounds of the propagating availability $\alpha(x_i, x_j)$, respectively. They are formally defined as follows:

$$\overline{\alpha}(x_i, x_j) = \begin{cases} \min\{0, \overline{r}(x_j, x_j) + \sum_{x_k \neq x_i, x_j} \max\{0, \overline{r}(x_k, x_j)\}\} & (x_i \neq x_j), \\ \sum_{x_k \neq x_i} \max\{0, \overline{r}(x_k, x_j)\} & (x_i = x_j). \end{cases} \quad (13)$$

$$\underline{\alpha}(x_i, x_j) = \begin{cases} \min\{0, s(x_i, x_j) - \max_{x_k \neq x_j}\{s(x_i, x_k)\}\} & (x_i \neq x_j). \\ 0 & (x_i = x_j). \end{cases} \tag{14}$$

As well as the case of the responsibility shown in Section 3.3.1, we can the upper and lower bounds of availability from the similarities S. That is, we can also estimate both bounds for the availability before performing the iterative computations.

3.4. *Cell-based Message Pruning*

By using the cell-based index (Section 3.2) and the upper/lower bounds estimates (Section 3.3), C-AP prunes the unnecessary data object pairs whose responsibilities and availabilities do not contribute to finding exemplars in each iteration. In order to reduce the number of computed data object pairs, C-AP detects unnecessary cell pairs that contain only unnecessary data object pairs by using the upper and lower bounds defined in Section 3.3. Here, we show how to find the unnecessary cell pairs to limit message updates before entering the iterations.

First, we define a prunable responsibility set P_r that is a set of data object pairs not to update the responsibilities; C-AP skips to compute the responsibilities for the data object pairs in P_r.

Definition 6 (Prunable Responsibility Set) *Let C_i and C_j be a pair of cells, and P_r be a set of data object pairs whose responsibility computations are pruned. P_r is defined as follows:*

$$P_r = \bigcup_{(C_i, C_j) \in R_p} \{(x_i, x_j) | x_i \in C_i \text{ and } x_j \in C_j\}, \tag{15}$$

where $R_p = \{(C_i, C_j) | C_i \neq C_j \text{ and } cs(C_i, C_j) - cs(C_i, C_i) \leq 0\}$.

As shown in Definition 6, we need to verify any pairs of cells, C_i and C_j, that satisfy both $C_i \neq C_j$ and $cs(C_i, C_j) - cs(C_i, C_i) \neq 0$ to make the prunable responsibility set P_r. The prunable responsibility set P_r has the following property:

Lemma 1 *If the responsibilities have been obtained for pairs in $R = \{X \times X\} \backslash P_r$, the availabilities of any pairs of data objects can be computed in each iteration.*

Proof: Here, we prove that $(x_i, x_j) \in P_r$ never contributes to updating the availabilities for any pairs of data objects. From Definition 6, $(x_i, x_j) \in P_r$ satisfies $C_i \neq C_j$ and $cs(C_i, C_j) - cs(C_i, C_i) < 0$. As shown in Definition 3, we have

$$cs(C_i, C_j) = \max\{s(x_a, x_b) | x_a \in C_i \text{ and } x_b \in C_j\}, \tag{16}$$

$$cs(C_i, C_i) = s(x_i, x_i). \tag{17}$$

The above equations indicate $s(x_i, x_j) < cs(C_i, C_j)$. Hence, from Definition 1, $\overline{\rho}(x_i, x_j) =$

$s(x_i, x_j) - s(x_i, x_i) < cs(C_i, C_j) - cs(C_i, C_i) < 0$. Recall that $r_0(x_i, x_j)$ is clearly less than 0 since $r_0(x_i, x_j) = s(x_i, x_j) - \max_{x_k \neq x_j}\{s(x_i, x_k)\} < 0$. Thus, if $C_i \neq C_j$ and $cs(C_i, C_j) - cs(C_i, C_i) < 0$, we hold $r_0(x_i, x_j) < 0$ and $\overline{\rho}(x_i, x_j) < 0$.

From Definition 4, $\overline{r}(x_i, x_j) = (1-\lambda)\overline{\rho}(x_i, x_j) < 0$ since $r_0(x_i, x_j) < 0$ and $\overline{\rho}(x_i, x_j) < 0$. Hence, as shown in Definition 1, the responsibility of (x_i, x_j) does not affect to $\alpha(x_i, x_j)$. This is because that $\alpha(x_i, x_j)$ takes $\max\{0, r(x_i, x_j)\} = 0$ for $x_i \neq x_j$ since we already have $r(x_i, x_j) < \overline{r}(x_i, x_j) < 0$. Therefore, $(x_i, x_j) \in P_r$ never contributes to updating the availabilities for any pairs of data objects, and finally, Lemma 1 holds. □

Next, we define a prunable availability set P_a that is a set of data object pairs not to update the availabilities.

Definition 7 (Prunable Availability Set) *Let C_i and C_j be a pair of cells, and P_a be a set of data object pairs whose responsibility computations are pruned. P_a is defined as follows:*

$$P_a = \bigcup_{(C_i, C_j) \in A_p} \{(x_i, x_j) | x_i \in C_i \textit{ and } x_j \in C_j\}, \tag{18}$$

where $A_p = \{(C_i, C_j) | \overline{ca}(C_i, C_j) - \underline{ca}(C_i) < 0\}$ such that

$$\overline{ca}(C_i, C_j) = \max\{\overline{a}(x_a, x_b) + s(x_a, x_b) | x_a \in C_i \textit{ and } x_b \in C_j\}, \tag{19}$$

$$\underline{ca}(C_i) = \min\{\underline{a}(x_a, x'') + s(x_a, x'') | x_a \in C_i \textit{ and } x'' \in X\}. \tag{20}$$

Note that x'' gives the second largest value of $\underline{a}(x_a, x'') + s(x_a, x'')$.

From Definition 7, we need to verify any pairs of cells, C_i and C_j, that satisfy $\overline{ca}(C_i, C_j) - \underline{ca}(C_i) < 0$ to make the prunable availability set P_a. The prunable availability set P_a has the following property:

Lemma 2 *In the iteration, the responsibilities of any pairs of data objects can be computed if the availabilities of all pairs in $A = \{X \times X\} \backslash P_a$ have been found.*

Proof: First, we prove that a pair (x_i, x_j) does not affect to $\rho(x_i, x_j)$ if (x_i, x_j) satisfies the following condition:

$$\overline{a}(x_i, x_j) + s(x_i, x_j) < \underline{a}(x_i, x'') + s(x_i, x''), \tag{21}$$

where x'' gives the second largest value of $\underline{a}(x_i, x'') + s(x_i, x'')$. For the pair (x_i, x_j), $\underline{a}(x_i, x'') + s(x_i, x'') \leq \max_{x_l \neq x_j}\{a(x_i, x_l) + s(x_i, x_l)\}$ holds. That is, we have

$$\overline{a}(x_i, x_j) + s(x_i, x_j) < \max_{x_l \neq x_j}\{a(x_i, x_l) + s(x_i, x_l)\}. \tag{22}$$

Thus, from Definition 1, the availability of the pair is not need to compute $\rho(x_i, x_j)$ if $x_i \neq x_j$. Clearly, in the case of $x_i = x_j$, we can obtain $\rho(x_i, x_j)$ without using the availability of the pair. Therefore, if $\overline{a}(x_i, x_j) + s(x_i, x_j) < \underline{a}(x_i, x'') + s(x_i, x'')$ holds, the pair (x_i, x_j) doesnot affect to $\rho(x_i, x_j)$.

Then, we prove that $(x_i, x_j) \in P_a$ does not contribute to updating the responsibility for any pairs of data objects. Since $x_i \in C_i$, $x_j \in C_j$, and $x'' \in X$ from Definition 7, we have $\overline{a}(x_i, x_j) + s(x_i, x_j) \leq \overline{ca}(C_i, C_j)$ and $\underline{ca}(C_i) \leq \underline{a}(x_i, x'') + s(x_i, x'')$. Hence, from Inequation 21, we always hold $\overline{a}(x_i, x_j) + s(x_i, x_j) < \underline{a}(x_i, x'') + s(x_i, x'')$ if $\overline{ca}(C_i, C_j) - \underline{ca}(C_i) < 0$ *s.t.* $x_i \in C_i$, $x_j \in C_j$, and $x'' \in X$. Thus, if $(x_i, x_j) \in P_a$, the pair (x_i, x_j) doesnot affect to $\rho(x_i, x_j)$. □

By following Lemma 1 and 2, C-AP can prune unnecessary pairs of data objects by finding the prunable responsibility set P_r (Definition 6) and the prunable availability set P_a (Definition 7). As shown in Definition 6 and 7, C-AP can find P_r and P_a by only checking the pairs of cells are included in R_p and A_p, respectively. The computational costs for obtaining R_p and A_p are $O(c^2)$, where c is the number of non-empty cells; thus, C-AP can run relatively faster than the competitive algorithms that require $O(n^2)$ times.

Thanks to Lemma 1 and 2, C-AP has the following theorem:

Theorem 1 *C-AP always returns the same exemplars as those of the original Affinity Propagation [10].*

Proof: As shown in Definition 2, in order to specify the exemplar for a data object x_i, we need to find the data object x_j that maximizes $r(x_i, x_j) + a(x_i, x_j)$ by using the converged $r(x_i, x_j)$ and $a(x_i, x_j)$. From Lemma 1 and 2, we theoretically proved that our cell-based pruning techniques do not affect to find the exact converged values of $r(x_i, x_j)$ and $a(x_i, x_j)$. Therefore, C-AP finds exactly same exemplars as those of the original Affinity Propagation algorithm since C-AP can obtain exact values of $r(x_i, x_j) + a(x_i, x_j)$. $\square$

As shown in Section 2, the clustering results are uniquely determined by the exemplars. Thus, from Theorem 1, C-AP finds exactly same clustering results as those of the original algorithm [10].

3.5. *Algorithm*

We can efficiently extract the same clustering results as those of the original algorithm [10] by using the cell-based index. Figure 2 illustrates the workflow of C-AP, and its pseudo code is shown in Algorithm 2. Algorithm 2 is designed to take three inputs and return exemplars; the inputs are a set of data objects X, a set of similarities S, the size of each cell ϵ. The main part of Algorithm 2 consists of three steps: cell-based message pruning (lines 1-5), message exchanging (lines 6-15), and exemplar detection (lines 16-21).

First, C-AP runs the cell-based message pruning (lines 1-5). C-AP constructs the cell-based index by using the user-specified parameter ϵ (line 1), then it obtains the upper and lower bounds of the messages (lines 2-3) to prepare the cell-based message pruning. After that C-AP prunes unnecessary pairs of data objects by comparing the cells based on the Definition 6 and 7 (lines 4-5).

Next, C-AP moves to the message exchanging (lines 6-15). C-AP iteratively exchanges the responsibility and the availability until convergence; C-AP successfully reduces the number of data object pairs by excluding the pairs included in P_r and P_a before entering the iterations. Similar to the state-of-the-art method [16], C-AP dynamically skips to compute the pairs (x_i, x_j) whose responsibility (or availability) is not updated (lines 8-9 and lines 13-14).

Last, C-AP detects exemplars (lines 16-21). At first, C-AP assigns the responsibility, and the availabilities for the pruned data object pairs, *i.e.* P_r and P_a, since these pairs do not have any values (lines 16-19). Finally, based on Definition 2, C-AP detects and exemplar $e(x_i)$ for each data object $x_i \in X$ by using the responsibility and the availability (lines 20-21).

Algorithm 2 C-AP

```
Input: $X$, $S$, $\epsilon \in \mathbb{R}$
Output: exemplars for each data object in $X$
 1: construct cell-based index by using $\epsilon$;
 2: for each $x_i \in X$ do
 3:     get $\overline{r}$, $\underline{r}$, $\overline{a}$, and $\underline{a}$ by Definition 4 and 5;
 4: obtain $R = \{X \times X\} \backslash P_r$ based on Definition 6;
 5: obtain $A = \{X \times X\} \backslash P_a$ based on Definition 7;
 6: repeat
 7:     for each $(x_i, x_j) \in R$ do
 8:         compute $r(x_i, x_j)$ by Definition 1;
 9:         if $r(x_i, x_j)$ is not updated then
10:             $R = R \backslash \{(x_i, x_j)\}$;
11:     for each $(x_i, x_j) \in A$ do
12:         compute $a(x_i, x_j)$ by Definition 1;
13:         if $a(x_i, x_j)$ is not updated then
14:             $A = A \backslash \{(x_i, x_j)\}$;
15: until $R, A = \emptyset$
16: for each $(x_i, x_j) \in P_r$ do
17:     get $r(x_i, x_j) = \rho(x_i, x_j)$ by Definition 4;
18: for each $(x_i, x_j) \in P_a$ do
19:     get $a(x_i, x_j) = \alpha(x_i, x_j)$ by Definition 5;
20: for each $x_i \in X$ do
21:     get an exemplar $e(x_i)$ by Definition 2;
```

3.6. *Parallel extension of C-AP*

For further improving the clustering speed of C-AP, we here introduce a simple extension of C-AP that employs thread parallelization for computations of data object pairs. As we can see from Algorithm 2, our proposed algorithm C-AP has several independent loop-blocks, *i.e.,* lines 2-3, lines 7-10, lines 11-14, lines 16-17, lines 18-19, and lines 20-21 in Algorithm 2. Since the loop-blocks compute large numbers of data object pairs, the blocks still require expensive computation time to handle large-scale datasets. Thus, in this paper, we apply loop-level parallelizations to the aforementioned independent loop-blocks in C-AP to reduce the expensive computation costs.

Algorithm 3 shows the pseudo code of our parallelized algorithm, namely *Parallel C-AP*. As shown in Algorithm 3, Parallel C-AP replaces the loop-blocks in Algorithm 2 with thread-parallel blocks. In each thread-parallelization, Parallel C-AP assigns a set of data object pairs for each thread in order to balance task granularities among threads. Specifically, Parallel C-AP divides the data object pairs, which is inputted to each loop-block, into equal subsets since most loop-blocks need to compute all of the given data object pairs. By balancing sizes of the subsets, Parallel C-AP attempts to avoid waiting for time-consuming threads.

Our parallelization approach does not sacrifice the exactness of C-AP that we proved in Theorem 1. That is, Parallel C-AP also outputs the same exemplars as those of the original Affinity Propagation algorithm [10]. This is because, as described earlier, we apply thread-

Algorithm 3 Parallel C-AP

```
Input: $X$, $S$, $\epsilon \in \mathbb{R}$
Output: exemplars for each data object in $X$
 1: construct cell-based index by using $\epsilon$;
 2: for each $x_i \in X$ do in thread-parallel
 3:     get $\overline{r}$, $\underline{r}$, $\overline{a}$, and $\underline{a}$ by Definition 4 and 5;
 4: obtain $R = \{X \times X\} \backslash P_r$ based on Definition 6;
 5: obtain $A = \{X \times X\} \backslash P_a$ based on Definition 7;
 6: repeat
 7:     for each $(x_i, x_j) \in R$ do in thread-parallel
 8:         compute $r(x_i, x_j)$ by Definition 1;
 9:         if $r(x_i, x_j)$ is not updated then
10:             $R = R \backslash \{(x_i, x_j)\}$;
11:     for each $(x_i, x_j) \in A$ do in thread-parallel
12:         compute $a(x_i, x_j)$ by Definition 1;
13:         if $a(x_i, x_j)$ is not updated then
14:             $A = A \backslash \{(x_i, x_j)\}$;
15: until $R, A = \emptyset$
16: for each $(x_i, x_j) \in P_r$ do in thread-parallel
17:     get $r(x_i, x_j) = \rho(x_i, x_j)$ by Definition 4;
18: for each $(x_i, x_j) \in P_a$ do in thread-parallel
19:     get $a(x_i, x_j) = \alpha(x_i, x_j)$ by Definition 5;
20: for each $x_i \in X$ do in thread-parallel
21:     get an exemplar $e(x_i)$ by Definition 2;
```

based parallelization only for the independent loop-blocks, each of which does not contain order dependent operations within the blocks. Therefore, our parallelization algorithm keeps the exactness of C-AP even though Parallel C-AP drastically reduces the running time for large datasets. In the next section, we experimentally verify the efficiency and the exactness of Parallel C-AP by using several real-world datasets.

4. Experimental Analysis

We conducted extensive experiments to evaluate the effectiveness of our proposed approaches. We designed our experiments to demonstrate that:

- **Efficiency:** C-AP and Parallel C-AP achieve faster clustering time than the original and the state-of-the-art algorithms (Section 4.3); our proposed methods effectively avoid computing unnecessary data object pairs for the whole dataset (Section 4.4).
- **Scalability:** C-AP and Parallel C-AP show better scalability than the original and state-of-the-art algorithms in terms of the number of data objects (Section 4.5). Parallel C-AP also shows near-linear scalability as increasing the number of threads.
- **Exactness:** While C-AP and Parallel C-AP employ efficient pruning techniques, it does not sacrifice the clustering quality; our approaches always output the same clustering results as those of the original algorithm [10] (Section 4.6).

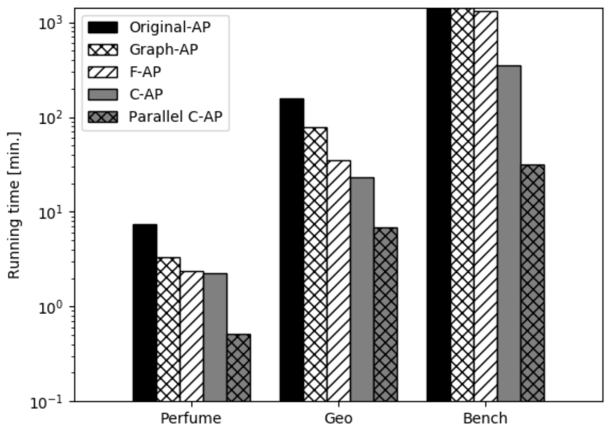

Fig. 3. Running time.

4.1. *Experimental Setup*

We compared our proposed algorithms, C-AP and Parallel C-AP, with the original algorithm, denoted by Original-AP [10] and the state-of-the-art algorithms, Graph-AP [18] and F-AP [16]. All algorithms were implemented in g++ using `-O2` option, and we conducted our experiments on a CentOS server that equips an Intel Xeon E5-2690 2.60 GHz CPU with 14 physical cores and 128 GB RAM. We set the dumping factor $\lambda = 0.5$ as recommended in the existing algorithms [10, 18, 16]. Unless otherwise stated, we set the default number of threads as 14 for Parallel C-AP.

4.2. *Datasets*

We evaluated the algorithms on three public datasets that are listed as follows:

- **Perfume:** This dataset consists of odors of 20 different perfumes, which were obtained by using an OMX-GR sensor .[b] The number of data objects is 560, ans we set $\epsilon = 2,500$.
- **Geo:** This is a building location dataset taken from all public facilities in City of Kitakyushu, Japan .[c] The number of data objects is 1,309, and we set $\epsilon = 0.05$.
- **Bench:** This is a public synthetic dataset, called A-sets, published in Clustering basic benchmark .[d] The number of data objects is 3,000, and we set $\epsilon = 2,500$.

We used negative Euclidean distance as the similarity S for the above datasets.

[b] `https://archive.ics.uci.edu/ml/index.php`
[c] `https://ckan.open-governmentdata.org/dataset/401005_shinoshisetsu`
[d] `http://cs.uef.fi/sipu/datasets/`

4.3. *Efficiency*

We compared the running time of our proposed algorithms, C-AP and Parallel C-AP, with the other competitive algorithms by using the three datasets described in Section 4.2. Figure 3 shows the results of each algorithm on Perfume, Geo, and Bench. Note that Original-AP and Graph-AP did not finish their clustering on Bench dataset within 24 hours. As shown in Figure 3, C-AP achieved $\times 6.85$ and $\times 3.77$ faster than runtimes of the original algorithm Original-AP and the state-of-the-art algorithms, respectively. Specifically, C-AP shows good performances on a large dataset (*i.e.,* Bench); it reduces by approximately 16 hours compared with the clustering time of the state-of-the-art algorithm F-AP. Moreover, Figure 3 also shows that our parallel algorithm successfully reduces the computation time of C-AP under all settings we examined. Specifically, Parallel C-AP achieved $\times 15.97$, $\times 23.77$, $\times 33.19$, and $\times 52.92$ faster than C-AP, F-AP, Graph-AP, and Original AP on average, respectively. These results imply that our Parallel C-AP is potentially applicable for large-scale datasets since it can significantly reduce the expensive computation time of Affinity Propagation by using thread-parallel techniques.

4.4. *Effectiveness of cell-based indexing*

In this section, we experimentally discuss the effectiveness of our cell-based indexing approaches.

In Figure 4, we first compared the number of data object pairs computed in each iteration to evaluate the effectiveness of our cell-based pruning approach. As we can see from Figure 4, C-AP successfully excludes unnecessary pairs of data objects compared with Original-AP. Specifically, C-AP computes up to only 52% of pairs of data objects what are computed in the original algorithm. That is, our cell-based indexing successfully reduces unnecessary computations of data object pairs during the clustering procedure of Affinity Propagation.

We then evaluated the effect of the user-specified parameter ϵ in C-AP. Figure 5 shows runtimes of C-AP when we varied ϵ from 1500 to 4000 on Perfume. As shown in Figure 5, if we set $\epsilon = 2500$, C-AP runs faster than the other ϵ settings. This is because it is difficult for C-AP to find cell pairs to be pruned when we set large ϵ values since most of prunable data object pairs might be in the same cell. Similarly, if we set small ϵ values, each cell contains a few data objects, and the number of non-empty cells approaches to the number of data objects n. Thus, the small ϵ settings also involve large computation costs.

4.5. *Scalability*

We assessed the scalability test of C-AP and Parallel in Figure 6 by varying the number of data objects. To evaluate the scalability, we additionally generated five datasets from Bench by randomly sampling 500, 1000, 1500, 2000, and 2500 data objects included in Bench; In Figure 6, we measured the running time of C-AP, Parallel AP, and Original-AP by using the five datasets.

As we can see from Figure 6, the runtimes of C-AP and Parallel C-AP showed better scalability than those of Original-AP in terms of the number of data objects. This is because, as we experimentally confirmed in Section 4.3, C-AP successfully reduces the number of computed data object pairs by employing the cell-based approaches. Thus, these results verify

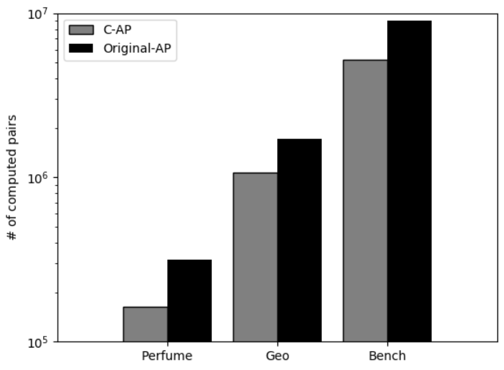

Fig. 4. # of computed pairs.

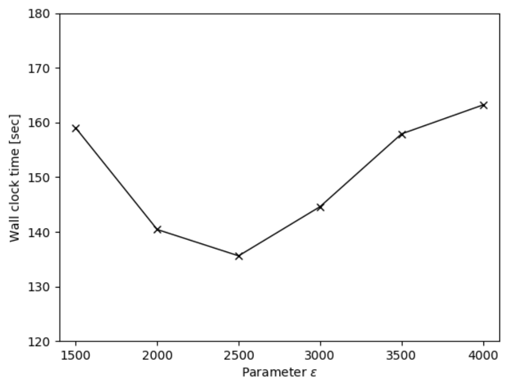

Fig. 5. Runtimes by varying ϵ.

that C-AP has better scalability than other competitive algorithms. Furthermore, as well as we showed in Figure 3, Parallel C-AP is more scalable than C-AP since our parallel algorithm can reduce the computational cost of C-AP significantly by using thread-based parallelization for the independent loop-blocks. That is, our proposed method, Parallel C-AP, is applicable to large datasets than those of C-AP and Original-AP.

Also, we evaluate the scalability of Parallel C-AP by varying the number of threads utilized in our proposed approach. In Figure 7, we measured the running time of Parallel C-AP on Bench dataset shown in Section 4.2 for the different settings of the number of threads; we varied the number of threads as 1, 2, 4, 8, 16, and 32 in this evaluation. From Figure 7, we can observe that Parallel C-AP shows almost linear scalability as increasing the thread sizes. This is because our approach employs parallelization only for the independent loop-blocks, which are major time-consuming parts of C-AP as we described in Section 3.6. Parallel C-AP successfully reduces the computation time by thread-based parallelization since most time-consuming parts are independent.

4.6. *Exactness*

Finally, we experimentally confirm the exactness of the clustering results produced by our proposed algorithms. In order to measure the exactness, we employed the well-known criteria, *precision* and *recall* [21]. In this evaluation, we compared the precision and the recall of exemplars produced by Original-AP and the others; if C-AP, Parallel C-AP, F-AP, and Graph-AP return the same set of exemplars as those of Original-AP, the precision, and the recall are 1. Table 1 shows the results on Perfume and Geo datasets; we omitted the evaluations on Bench since Original-AP and Graph-AP could not return any results. As we theoretically verified in Theorem 1, C-AP returns 1 in all settings that we examined. Furthermore, Table 1 shows that Parallel C-AP also outputs the same exemplars as those of Original-AP even though our parallel algorithm drastically reduces the computation time as shown in Figure 3. As

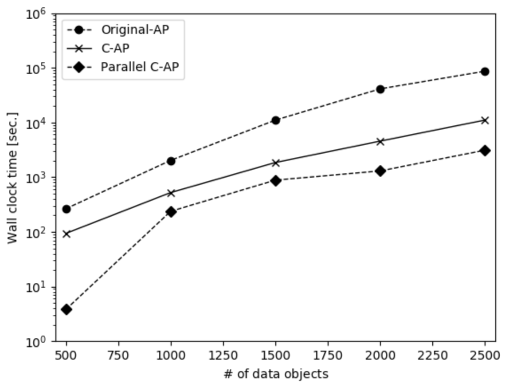

Fig. 6. Scalability by varying data size $|X|$.

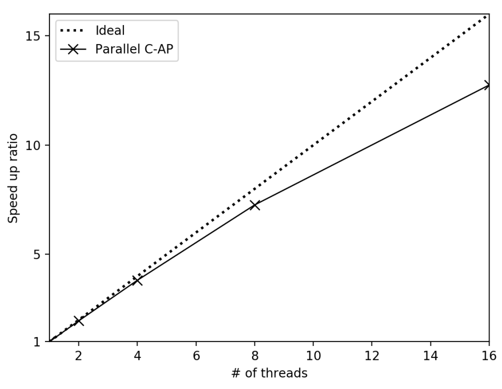

Fig. 7. Scalability by varying the number of threads.

Table 1. Accuracy

	Graph-AP	F-AP	C-AP	Parallel C-AP
Perfume (Recall)	1.00	1.00	1.00	1.00
Perfume (Precision)	1.00	1.00	1.00	1.00
Geo (Recall)	1.00	1.00	1.00	1.00
Geo (Precision)	1.00	1.00	1.00	1.00

we described in Section 3.6, we apply thread-based parallelization only for the independent loop-blocks. Hence, Parallel C-AP does not sacrifice the clustering quality of C-AP. From these results, we experimentally confirmed that C-AP and Parallel C-AP always produce the exact set of exemplars as those of Original-AP.

5. Related work

The clustering is one of the most fundamental data mining tools for finding hidden patterns in a given dataset. Thus, the problem of finding clusters has been studied for some decades [22]. k-means [23] and k-medoids [24] are natural choices for this problem. Since cluster structures are highly complex, Affinity Propagation [10] has been recently introduced. Here, we review some of the successful Affinity Propagation algorithms.

Affinity Propagation [10], proposed by Frey and Dueck in 2007, is one of the most successful methods among recent works for the cluster finding problem. Affinity Propagation finds clusters, and their corresponding representative data objects called exemplar from all data objects. Since Affinity Propagation automatically determines the number of clusters from given data objects, it achieves better clustering results than those of the traditional approaches [23, 24]; and as a result, Affinity Propagation widely used in various applications, *e.g.,* text classification [25] and flight analysis [18].

Although Affinity Propagation is effective in detecting clusters, it has, unfortunately, a serious weakness; its computational cost is quadratic in the number of data objects. As we discussed in Section 2, its complexity is $O(n^2T)$, where n and T are the number of data objects and the iterations, respectively. This high complexity has led to the introduction of pruning-based algorithms. Jia *et al.* proposed FSAP [17] that employs two-stage clustering approach. In FSAP, it first constructs a k-nearest neighbor graph from a give data objects; it then performs iterative updates of the responsibility and the availability on the graph. By introducing the k-nearest neighbor graph, FSAP certainly succeeded to reduce the number of computed data object pairs, it, however, suffers from the clustering accuracy. This is because that, different from our proposed method C-AP, FSAP prunes the pairs of data objects without any verifications.

To improve the efficiency, Fujiwara *et al.* proposed two state-of-the-art algorithms, named Graph-AP [18] and F-AP [16]. In Graph-AP, it employs the upper and lower bounds estimations, which we also used in Section 3.3, for pruning unnecessary data object pairs. Additionally, in F-AP, they extended Graph-AP by introducing an adaptive pruning technique that excludes pairs of data objects whose messages are not updated during the iterations. By introducing these pruning approaches, Graph-AP and F-AP achieved faster clustering than the original algorithm [10] and FSAP [17]. Furthermore, their clustering results are guaranteed to be the same as those of the original Affinity Propagation [10]. However, as we experimentally confirmed that their clustering speeds are still expensive for large datasets (*e.g.,* $n \geq 10^3$). As shown in Section 4, our C-AP is much faster than these algorithms even though C-AP is guaranteed to output exact clustering results; C-AP successfully reduces approximately 16 hours for the largest dataset that we examined. Furthermore, in this paper, we extended C-AP so as to reduce the computation time for clustering by using thread-based parallelization. As a result, our proposed algorithm Parallel C-AP shows near-linear scalability as increasing the number of data objects and threads as shown in Section 4.5.

6. Conclusion

In this paper, we tackled the problem of increasing efficiency of Affinity Propagation, and we proposed two cell-based algorithms, C-AP and Parallel C-AP. C-AP is a sequential algorithm that employs cell-based indexing on large datasets. By using the cell-based index, C-AP prunes the computations for unnecessary data object pairs from a given dataset. Meanwhile, Parallel C-AP is an extension of C-AP. We focused on the independent loop-blocks included in C-AP, and we applied thread-based parallelization for the blocks to increase the clustering efficiency on manycore processors. Our experimental results showed that C-AP outperforms the state-of-the-art algorithms [18, 16] even though C-AP does not sacrifices its clustering qualities compared with those of the original algorithm [10]. Moreover, Parallel C-AP shows near-linear scalability as increasing the size of datasets and the number of threads, while Parallel C-AP outputs the same clustering results as C-AP. Affinity Propagation is now a fundamental data mining tool to current and prospective Web-based systems and applications in various disciplines. By providing our fast algorithm, it will help to improve the effectiveness of future applications.

Acknowledgements

This work was partially supported by JST ACT-I, JSPS KEKENHI Early-Carrer Scientists

Grant Number JP18K18057, and Interdisciplinary Computational Science Program in CCS, University of Tsukuba.

References

1. T. Sato, H. Shiokawa, Y. Yamaguchi, and H. Kitagawa, "FORank: Fast ObjectRank for Large Heterogeneous Graphs," in *Companion Proceedings of the The Web Conference 2018*, pp. 103–104, 2018.
2. J. M. Kleinberg, "Authoritative Sources in a Hyperlinked Environment," *Journal of the ACM*, vol. 46, pp. 604–632, 9 1999.
3. B. Wu and B. D. Davison, "Identifying Link Farm Spam Pages," in *Proc. WWW*, pp. 820–829, 2005.
4. H. Shiokawa, Y. Fujiwara, and M. Onizuka, "SCAN++: Efficient Algorithm for Finding Clusters, Hubs and Outliers on Large-scale Graphs," *Proceedings of Very Large Data Bases Endowment*, vol. 8, no. 11, pp. 1178–1189, 2015.
5. T. C. Zhou, H. Ma, M. R. Lyu, and I. King, "UserRec: A User Recommendation Framework in Social Tagging Systems," in *Proc. AAAI*, pp. 1486–1491, 2010.
6. P. Domingos and M. Richardson, "Mining the Network Value of Customers," in *Proc. KDD*, pp. 57–66, 2001.
7. H. Shiokawa, Y. Fujiwara, and M. Onizuka, "Fast Algorithm for Modularity-based Graph Clustering," in *Proceedings of the 27th AAAI Conference on Artificial Intelligence*, pp. 1170–1176, 2013.
8. T. Takahashi, H. Shiokawa, and H. Kitagawa, "SCAN-XP: Parallel Structural Graph Clustering Algorithm on Intel Xeon Phi Coprocessors," in *Proceedings of the 2nd International Workshop on Network Data Analytics*, NDA, (New York, NY, USA), pp. 6:1–6:7, 2017.
9. H. Shiokawa, T. Takahashi, and H. Kitagawa, "ScaleSCAN: Scalable Density-based Graph Clustering," in *Proceedings of the 29th International Conference on Database and Expert Systems Applications*, DEXA, pp. 18–34, 2018.
10. B. J. Frey and D. Dueck, "Clustering by Passing Messages Between Data Points," *Science*, vol. 315, no. 5814, pp. 972–976, 2007.
11. J. B. MacQueen, "Some Methods for Classification and Analysis of Multivariate Observations," in *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, vol. 1, pp. 281–297, University of California Press, 1967.
12. L. Kaufman and P. J. Rousseeuw, "Clustering by Means of Medoids," 1987.
13. Z. Liu, P. Li, Y. Zheng, and M. Sun, "Community Detection by Affinity Propagation," tech. rep., Technical Report, 2008.
14. L. Gang, G. Lei, and L. Tianming, "Grouping of Brain MR Images via Affinity Propagation," in *Proceedings of IEEE International Symposium on Circuits and Systems*, pp. 2425–2428, 2009.
15. Y. Qian, F. Yao, and S. Jia, "Band Selection for Hyperspectral Imagery Using Affinity Propagation," *IET Computer Vision*, vol. 3, no. 4, pp. 213–222, 2009.
16. Y. Fujiwara, M. Nakatsuji, H. Shiokawa, Y. Ida, and M. Toyoda, "Adaptive Message Update for Fast Affinity Propagation," in *Proc. KDD*, pp. 309–318, 2015.
17. Y. Jia, J. Wang, C. Zhang, and X.-S. Hua, "Finding Image Exemplars Using Fast Sparse Affinity Propagation," in *Proceedings of the 16th ACM international conference on Multimedia*, pp. 639–642, 2008.
18. Y. Fujiwara, G. Irie, and T. Kitahara, "Fast Algorithm for Affinity Propagation," in *Proc. IJCAI*, pp. 2238–2243, 2011.
19. T. Matsushita, H. Shiokawa, and H. Kitagawa, "C-AP: Cell-based Algorithm for Efficient Affinity Propagation," in *Proceedings of the 20th International Conference on Information Integration and Web-based Applications & Services*, iiWAS2018, pp. 156–163, 2018.
20. Y. Fujiwara, Y. Ida, H. Shiokawa, and S. Iwamura, "Fast Lasso Algorithm via Selective Coordinate Descent," in *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, pp. 1561–1567,

2016.
21. C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval.* New York, NY, USA: Cambridge University Press, 2008.
22. M. Onizuka, T. Fujimori, and H. Shiokawa, "Graph Partitioning for Distributed Graph Processing," *Data Science and Engineering*, vol. 2, pp. 94–105, Mar 2017.
23. A. K. Jain, "Data Clustering: 50 Years Beyond K-means," *Pattern recognition letters*, vol. 31, no. 8, pp. 651–666, 2010.
24. L. Kaufman and P. J. Rousseeuw, "Partitioning Around Medoids (Program PAM)," *Finding groups in data: an introduction to cluster analysis*, pp. 68–125, 1990.
25. R. Guan, X. Shi, M. Marchese, C. Yang, and Y. Liang, "Text Clustering with Seeds Affinity Propagation," *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, no. 4, pp. 627–637, 2011.