Scalable Affinity Propagation for Massive Datasets

Hiroaki Shiokawa
Center for Computational Sciences, University of Tsukuba, Japan
shiokawa@cs.tsukuba.ac.jp

Abstract

Affinity Propagation (AP) is a fundamental algorithm to identify clusters included in data objects. Given a similarities among objects, it iteratively performs message updates between all data object pairs until convergence. Although AP yields a higher clustering quality compared with other methods, it is computationally expensive. Hence, it has difficulty handling massive datasets that include numerous data objects. This is because the message updates require a quadratic cost of the number of data objects. Here, we propose a novel fast algorithm, *ScaleAP*, which outputs the same clusters as AP but within a shorter computation time. ScaleAP dynamically excludes unnecessary message updates without sacrificing its clustering accuracy. Our extensive evaluations demonstrate that ScaleAP outperforms existing AP algorithms in terms of running time by up to two orders of magnitude.

1 Introduction

Affinity Propagation (AP) (Frey and Dueck 2007) is one of the most successful clustering methods to overview complicated data objects in an unsupervised way. AP detects clusters and their corresponding representative data objects called exemplars by message-passing processes between all pairs of data objects. AP can (1) have lower clustering errors compared to other methods and (2) support any similarity that does not satisfy the triangle inequality (Sun and Guo 2014; Fujiwara et al. 2015). Hence, AP has been employed in many applications (Dueck and Frey 2007; Ambrogi et al. 2008; Kazantseva and Szpakowicz 2011).

Although AP is useful in many applications, it has a quadratic cost to identify clusters since the message-passing iteratively updates real-valued messages for all object pairs. If N is the number of data objects and T is the number of iterations, AP needs $O(N^2T)$ time. In the late-2000s, AP was applied to only small datasets composed of a few thousand data objects (*e.g.,* social networks and location datasets). However, recent AI-powered applications handle massive datasets with numerous data objects because dataset resources are not only becoming larger but also more prevalent (Shiokawa, Fujiwara, and Onizuka 2013, 2015). Hence, current AP algorithms require several days to obtain clusters from massive datasets.

1.1 Existing Approaches and Challenges

Many studies have strived to overcome the expensive cost. One major approach is *message sampling* (Jia et al. 2008; Sun et al. 2017). AP updates the real-valued messages for all object pairs, but it is more reasonable to update only essential pairs that are potentially included in the same cluster. To specify such pairs, the message sampling drops off unpromising message updates before the message-passing. For instance, *FSAP* (Jia et al. 2008) constructs a k-nearest neighbor (kNN) graph from object similarities to prune unnecessary object pairs. Similarly, *FastAP* (Sun et al. 2017) extracts important messages by previously finding micro-clusters. Although these methods are faster than AP, they have two drawbacks. First, they still have high costs for sampling messages, *e.g.,* $O(N^2)$ time. Second, they sacrifice clustering accuracy, which makes it difficult to realize truly effective applications.

Instead of sampling methods, Fujiwara *et al.* proposed *message bounding techniques* (Fujiwara, Irie, and Kitahara 2011). They found that the real-valued messages converge as increasing the number of iterations, and they theoretically derived the upper and lower bounds of the messages. Based on these bounds, Fujiwara *et al.* designed incremental message-pruning algorithms, including *GraphAP* (Fujiwara, Irie, and Kitahara 2011) and *F-AP* (Fujiwara et al. 2015). GraphAP and F-AP guarantee that their pruning methods do not sacrifice clustering accuracy. Unlike other AP algorithms, GraphAP and F-AP can reduce the running time while maintaining the same clustering results as AP.

Although the bounding methods have improved the efficiency, they still suffer from large computational costs to handle massive datasets (Matsushita, Shiokawa, and Kitagawa 2018). In the bounding methods, the upper and lower bounds estimation requires $O(N^2)$ time. That is, they have a total time complexity of $O(N^2+MT)$ time, where M is the number of non-pruned pairs of data objects. Furthermore, the bounding methods fail to prune unnecessary messages if the datasets include many large clusters since the bounds becomse loose in such cases. This incurs $O(M) \approx O(N^2)$ time in the worst case. Consequently, the bounding methods still require almost the same cost as AP, *e.g.,* $O(N^2T)$ time. Hence, it is a challenging task to develop a scalable AP algorithm that guarantees the same results as AP.

1.2 Our Approaches and Contributions

Our goal is to speed up AP without sacrificing its clustering accuracy. Here, we present a novel efficient algorithm, *ScaleAP*, which is designed to reduce the number of message updates with a clustering accuracy assurance. The basic idea underlying ScaleAP is to remove redundant message updates computed in the message-passing. To identify which updates to exclude, ScaleAP focuses on the deterministic property of the message updates. In this paper, we theoretically clarified that most message updates are uniquely determined by corresponding similarities between data objects (Lemmas 2 and 4). That is, messages do not need to be computed repeatedly for similar object pairs.

Based on this property, ScaleAP employs the following two approaches to improve efficiency: (1) it theoretically derives *prunable pairs* that do not require repeated computations during the message-passing (Section 3.2), and (2) it introduces *aggregated message updates* to efficiently skip computations for prunable pairs based on the deterministic property (Section 3.3). As a result, ScaleAP has the following attractive characteristics:

- **Efficiency:** ScaleAP achieves faster clustering than the state-of-the-art AP algorithms proposed in the last few years (Section 4.1). We proved that ScaleAP has a better time complexity than the AP algorithms (Theorem 1).
- **Exactness:** We theoretically and experimentally confirmed that ScaleAP always outputs the same clustering results as AP, although ScaleAP dynamically removes message updates (Theorem 2 and Section 4.4).
- **Scalability:** ScaleAP is more scalable than the other AP algorithms (Section 4.3). It shows a nearly linear scalability against the number of data objects (Section 3.4).
- **Easy to deploy:** Unlike existing fast algorithms for AP, ScaleAP does not require user-specified parameters (Algorithm 1). ScaleAP provides users with a simple solution for applications using AP.

ScaleAP is the first solution to achieve a high efficiency while ensuring the same clustering results as AP on massive datasets. ScaleAP outperforms state-of-the-art algorithms by up to two orders of magnitude in terms of clustering time. For example, for 120,000 data objects, ScaleAP finds clusters within 30 minutes, whereas AP consumes more than 24 hours. AP is a fundamental tool to enhance application quality, but it is difficult to apply to massive datasets. However, ScaleAP, which is well suited to massive datasets, should improve the quality in diverse AI-powered applications.

2 Preliminary

We formally define the notations and briefly review AP (Frey and Dueck 2007). Let $\mathcal{X}$ be a set of data objects $\mathcal{X} = \{x_1, x_2, \ldots, x_N\}$, $s(i,j)$ be a real-valued similarity of a data object pair (x_i, x_j), and $e(i)$ be the exemplar of data object x_i. AP extracts clusters and the corresponding exemplars from pairwise similarities $\mathcal{S} = \{s(i,j) \mid x_i, x_j \in \mathcal{X}\}$. It places each data object x_i into the same cluster as exemplar $e(i)$ to maximize $\sum_{i=1}^{N} s(i, e(i))$. If $i = j$, $s(i,j)$ is referred to as the *preference*, which is a special class of similarities to control the number of clusters (Frey and Dueck 2007). The preference is typically set to the minimum or median value of the similarities.

To identify exemplars from datasets, AP performs a message-passing process, which is derived from the belief propagation on factor graphs (Kschischang, Frey, and Loeliger 2001; Yedidia, Freeman, and Weiss 2005). AP iteratively exchanges two types of real-valued messages among data objects: *responsibility* and *availability*. In the t-th iteration, AP sends a responsibility $r_t(i,j)$ from x_i to x_j, which indicates how strongly x_i wants to choose x_j as its exemplar. It also sends an availability $a_t(i,j)$ from x_j to x_i, which reflects the accumulated evidence for how well suited it would be for x_i to choose x_j as its exemplar. To compute the converged values, AP lets $\lambda \in (0,1)$ be a dumping factor and iteratively updates all messages, which are given as

$$r_t(i,j) = (1-\lambda)\rho_t(i,j) + \lambda r_{t-1}(i,j), \quad (1)$$

$$a_t(i,j) = (1-\lambda)\alpha_t(i,j) + \lambda a_{t-1}(i,j), \quad (2)$$

where $\rho_t(i,j)$ and $\alpha_t(i,j)$ are computed as

$$\rho_t(i,j) = \begin{cases} s(i,j) - \max_{k \neq j}\{a_{t-1}(i,k) + s(i,k)\} & (i \neq j) \\ s(i,j) - \max_{k \neq j}\{s(i,k)\} & (i = j) \end{cases}, \quad (3)$$

$$\alpha_t(i,j) = \begin{cases} \min\left\{0, r_{t-1}(j,j) + \sum_{k \neq i,j} \max\{0, r_{t-1}(k,j)\}\right\} & (i \neq j) \\ \sum_{k \neq i} \max\{0, r_{t-1}(k,j)\} & (i = j) \end{cases}. \quad (4)$$

If $t = 0$, the messages are initially set to

$$r_0(i,j) = s(i,j) - \max_{k \neq j}\{s(i,k)\}, \text{ and } a_0 = 0. \quad (5)$$

After message-passing termination, AP determines an exemplar $e(i)$ for each data object x_i as

$$e(i) = \arg\max_j \{r_t(i,j) + a_t(i,j)\}. \quad (6)$$

AP requires $O(N^2 T)$ time, where N and T are the number of data objects and iterations, respectively. That is, if a given dataset is too large, AP incurs an excessive running time. Hence, AP fails to extract clusters and corresponding exemplars in massive datasets.

3 Proposed Method: ScaleAP

We present ScaleAP to efficiently detect the same clustering results as AP. We first overview main ideas and then provide detailed descriptions of ScaleAP.

3.1 Ideas

Our goal is to efficiently find clusters and the corresponding exemplars without sacrificing clustering accuracy. AP iteratively computes real-valued messages, responsibility and availability, for all pairs of data objects until the messages are no longer updated. By contrast, ScaleAP skips unnecessary message updates by introducing two approaches. First, we theoretically identify *prunable pairs* of data objects for responsibility and availability. The message updates for prunable pairs are deterministically computed from the corresponding similarities without message-passing processes. Second, we employ an *aggregated message update* to remove unnecessary message updates for prunable pairs while maintaining the same clustering accuracy as AP. Unlike message sampling and bounding methods, which invoke exhaustive message computations, ScaleAP computes only the essential object pairs to efficiently find clusters and the corresponding exemplars.

Our ideas have two main advantages. (1) ScaleAP finds all clusters and exemplars on real-world datasets with a short running time. Our ideas successfully handle the data manifolds, which are well-known intrinsic structures embedded in real-world data (Roweis and Saul 2000; Belkin and Niyogi 2003). This is because these manifolds involve many similar object pairs that are removed by the adaptive message aggregation. This is why ScaleAP can increase its performance on real-world datasets. (2) ScaleAP always outputs the same clusters and exemplars as those of AP (Frey and Dueck 2007). We theoretically demonstrate that ScaleAP does not miss opportunities to improve clustering accuracy, even though it dynamically removes prunable pairs from the message-passing processes. Thus, ScaleAP does not sacrifice clustering quality compared to AP.

3.2 Prunable Pairs

We introduce *prunable pairs* of data objects whose messages are uniquely computed from the corresponding similarity.

Prunable responsibility: We first define prunable pairs for responsibility computations.

Definition 1 (Prunable pairs for responsitiblity)**.** *In the t-th iteration, let $\hat{\mathcal{R}}_t(i)$ be the prunable pairs of responsibility computations for data object x_i. Then $\hat{\mathcal{R}}_t(i)$ is given as*

$$\hat{\mathcal{R}}_t(i) = \{(x_i, x_j) \in \mathcal{X}^2 \mid i \neq j, j \neq \eta_{t-1}(i)\}, \qquad (7)$$

where $\eta_{t-1}(i) = \arg\max_k\{a_{t-1}(i,k) + s(i,k)\}$.

Note that $\hat{\mathcal{R}}_t(i)$ includes all object pairs between x_i and each of $\mathcal{X}$ except for (x_i, x_i) and $(x_i, x_{\eta_{t-1}(i)})$. From Definition 1, we can derive the following property:

Lemma 1. *Given two object pairs $(x_i, x_j), (x_i, x_k) \in \hat{\mathcal{R}}_t(i)$, $\rho_t(i,j) = \rho_t(i,k)$ always holds iff $s(i,j) = s(i,k)$.*

Proof. We first prove $\rho_t(i,j) = \rho_t(i,k) \Rightarrow s(i,j) = s(i,k)$. From Definition 1, $x_i \neq x_j$ and $x_i \neq x_j$ since $(x_i, x_j), (x_i, x_k) \in \hat{\mathcal{R}}_t$. Hence, as shown in Eq. (3), we have

$$\begin{aligned} s(i,j) &= \rho_t(i,j) + \max_{l \neq j}\{a_{t-1}(i,l) + s(i,l)\} \\ &= \rho_t(i,k) + \max_{l \neq j}\{a_{t-1}(i,l) + s(i,l)\} = s(i,k). \end{aligned} \qquad (8)$$

Thus, if $\rho_t(i,j) = \rho_t(i,k)$, $s(i,j) = s(i,k)$ holds.

Similarly, we can prove $s(i,j) = s(i,k) \Rightarrow \rho_t(i,j) = \rho_t(i,k)$. Therefore, Lemma 1 holds. □

Lemma 1 implies that object pairs in $\hat{\mathcal{R}}_t(i)$, (x_i, x_j) and (x_i, x_k), always have $\rho_t(i,j) = \rho_t(i,k)$ if their similarities are equivalent, *i.e.*, $s(i,j) = s(i,k)$, and vice versa. By Lemma 1, we derive the following property, which plays a key role in Section 3.3.

Lemma 2. *If $(x_i, x_j) \in \hat{\mathcal{R}}_t(i)$, $r_t(i,j)$ is uniquely determined by the corresponding similarity $s(i,j)$.*

Proof. We prove by contradiction. Given $(x_i, x_j), (x_i, x_k) \in \hat{\mathcal{R}}_t(i)$, we assume $s(i,j) = s(i,k) \Rightarrow r_t(i,j) \neq r_t(i,k)$. From Eq. (1), we have the following

$$r_t(i,j) = (1-\lambda)\textstyle\sum_{l=1}^{t} \lambda^{t-l}\rho_l(i,j) + \lambda^t r_0(i,j). \qquad (9)$$

From Lemma 1, $\rho_l(i,j) = \rho_l(i,k)$ for $l = 1, \ldots, t$. Additionally, from Eq. (5), $r_0(i,j) = r_0(i,k)$ since $s(i,j) = s(i,k)$. Thus, we can derive the following equation.

$$r_t(i,j) = (1-\lambda)\textstyle\sum_{l=1}^{t} \lambda^{t-l}\rho_l(i,k) + \lambda^t r_0(i,k) = r_t(i,k). \quad (10)$$

Since Eq. (10) contradicts the assumption, $s(i,j) = s(i,k) \Rightarrow r_t(i,j) = r_t(i,k)$ holds. This completes the proof of Lemma 2. □

Prunable availability: Next, we define prunable pairs for availability computations.

Definition 2 (Prunable pairs for availability)**.** *In the t-th iteration, the prunable pairs of availability computations for data object x_j, which are denoted by $\hat{\mathcal{A}}_t(j)$, are defined as*

$$\hat{\mathcal{A}}_t(j) = \{(x_i, x_j) \in \mathcal{X}^2 \mid i \neq j, \max_k\{r_{t-1}(k,j)\} \leq 0\}. \quad (11)$$

The prunable pairs $\hat{\mathcal{A}}_t(j)$ contain all object pairs $(x_i, x_j) \in \mathcal{X}^2$ except for (x_j, x_j) only if x_j does not have any positive responsibilities in the (t-1)-th iteration. Otherwise, $\hat{\mathcal{A}}_t(j) = \emptyset$. From Definition 2, we have the following property:

Lemma 3. *Given two object pairs $(x_i,x_j), (x_k,x_j) \in \hat{\mathcal{A}}_t(j)$, $\alpha_t(i,j) = \alpha_t(k,j)$ always holds.*

Proof. As shown in Definition 2, $\max_l\{r_{t-1}(l,j)\} \leq 0$ holds since $(x_i, x_j), (x_k, x_j) \in \hat{\mathcal{A}}_t(j)$. Thus, we can derive

$$\textstyle\sum_{l \neq i,j} \max_l\{0, r_{t-1}(l,j)\} = 0. \qquad (12)$$

From Eqs. (4) and (12), we can compute $\alpha_t(i,j)$ as

$$\alpha_t(i,j) = \min\{0, r_{t-1}(j,j)\} = \alpha_t(k,j), \qquad (13)$$

which completes the proof of Lemma 3. □

The object pairs, (x_i, x_j) and (x_k, x_j), included in $\hat{\mathcal{A}}_t(j)$ always hold $\alpha_t(i,j) = \alpha_t(k,j)$, regardless of their similarities. From Lemma 3, we can derive the following key property, which is an essential in Section 3.3.

Lemma 4. *If $(x_i, x_j) \in \hat{\mathcal{A}}_t(j)$, $a_t(i,j)$ is uniquely determined by the corresponding initial responsibility $r_0(j,j)$.*

Proof. From Eq. (2) and Lemma 3, we can derive $a_t(i,j)$ as

$$a_t(i,j) = (1-\lambda)\textstyle\sum_{l=1}^{t} \min\{0, r_{l-1}(j,j)\}. \qquad (14)$$

That is, $a_t(i,j)$ is determined by $r_{l-1}(j,j)$ in each iteration. From Eq. (1), we can also derive $r_{l-1}(j,j)$ as

$$r_{t-1}(j,j) = (1-\lambda)\textstyle\sum_{l=1}^{t-1} \lambda^{t-1-l}\rho_l(j,j) + \lambda^{t-1} r_0(j,j). \quad (15)$$

Recall that, as shown in Eqs. (1) and (5), $\rho_{t-}(j,j) = \rho_{t-2}(j,j) = \ldots = \rho_1(j,j) = r_0(j,j)$ clearly holds. Thus, from Eq. (15), the following equation holds

$$r_{t-1}(j,j) = r_0(j,j)\{(1-\lambda)\textstyle\sum_{l=1}^{t-1}\lambda^{t-1-l} + \lambda^{t-1}\} = r_0(j,j). \quad (16)$$

Thus, $a_t(i,j) = t(1-\lambda)r_0(j,j)$ if $r_0(j,j) > 0$. Otherwise, $a_t(i,j) = 0$. Hence, from Eq. (14), Lemma 4 holds. □

Complexity analysis: Finally, we theoretically assess the time complexity to compute the prunable pairs as follows:

Lemma 5. *In the t-th iteration, if $\hat{A}_t(1), \hat{A}_t(2), \ldots, \hat{A}_t(N)$ are computed, $\hat{R}_{t+1}(i)$ can be obtained in $O(1)$ time.*

Proof. As shown in Definition 1, $\hat{R}_{t+1}(i)$ includes all object pairs (x_i, x_j) for $i \neq j$ and $j \neq \arg\max_l\{s(i,l) + a_t(i,l)\}$. Since $a_t(i,1), a_t(i,2), \ldots, a_t(i,N)$ are included in $\hat{A}_t(1), \hat{A}_t(2), \ldots, \hat{A}_t(N)$, respectively, $\arg\max_l\{s(i,l) + a_t(i,l)\}$ is known after computing $\hat{A}_t(1), \hat{A}_t(2), \ldots, \hat{A}_t(N)$. By letting (x_i, x_l) be an object pair maximizing $s(i,l) + a_t(i,l)$, we can obtain $\hat{R}_{t+1}(i)$ in $O(1)$ time by removing (x_i, x_l) from $\{(x_i, x_j) \mid i \neq j\}$. □

Lemma 6. *In the t-the iteration, if $\hat{R}_t(1)$, $\hat{R}_t(2)$, $\ldots$, $\hat{R}_t(N)$ are computed, $\hat{A}_{t+1}(i)$ can be obtained in $O(1)$ time.*

Proof. From Definition 2, we need to find $\max_k\{r_t(k,i)\}$ to obtain $\hat{A}_{t+1}(i)$. Since $r_t(1)$, $r_t(2)$, $\ldots$, $r_t(N)$ are respectively included in $\hat{R}_t(1)$, $\hat{R}_t(2)$, $\ldots$, $\hat{R}_t(N)$, we can find $\max_k\{r_t(k,i)\}$ when we compute $\hat{R}_t(1)$, $\hat{R}_t(2)$, $\ldots$, $\hat{R}_t(N)$. Thus, $\hat{A}_{t+1}(i)$ can be obtained in $O(1)$ time by checking whether $\max_k\{r_t(k,i)\} \leq 0$ holds or not. □

3.3 Aggregated Message Update

We introduce *aggregated message update* to remove unnecessary message-passing processes in AP algorithms. If data objects are included in the prunable pairs, responsibilities and availabilities are uniquely determined by the corresponding similarities as shown in Lemmas 2 and 4. Thus, once the prunable pairs are obtained, messages of the prunable pairs can be updated from the corresponding similarities without performing iterative message-passing processes.

To leverage the above properties, ScaleAP replaces the responsibility and availability computations (Eqs. (1) and (2)) with aggregated responsibility and aggregated availability, respectively. These are given as:

Definition 3 (Aggregated responsibility)**.** *Given $(x_i, x_j) \in \mathcal{X}^2$ and $\hat{\mathcal{R}}_t(i)$ in the t-th iteration, the aggregated responsibility $\acute{r}_t(i,j)$ for (x_i, x_j) is defined as*

$$\acute{r}_t(i,j) = \begin{cases} (1-\lambda)\rho_t(i,j) + \lambda\acute{r}_{t-1}(i,j) & ((x_i,x_j) \in R_t(i)) \\ \acute{r}_t(i,k) + \triangle s_i(j,k) & (\textit{Otherwise}) \end{cases}, \quad (17)$$

where $R_t(i) = \{(x_i, x_j) \mid (x_i, x_j) \notin \hat{\mathcal{R}}_t(i)\} \cup \{(x_i, x_k)\}$ such that $k = \arg\max_l\{s(i,l) \mid (x_i, x_j) \in \hat{\mathcal{R}}_t\}$, and $\triangle s_i(j,k) = s(i,j) - s(i,k)$.

Definition 4 (Aggregated availability)**.** *Given $(x_i, x_j) \in \mathcal{X}^2$ and $\hat{\mathcal{A}}_t(j)$ in the t-th iteration, the aggregated availability $\acute{r}_t(i,j)$ for (x_i, x_j) is defined as*

$$\acute{a}_t(i,j) = \begin{cases} (1-\lambda)\alpha_t(i,j) + \lambda\acute{a}_{t-1}(i,j) & ((x_i,x_j) \notin \hat{\mathcal{A}}_t(j)) \\ t(1-\lambda)\min\{0, r_0(j,j)\} & (\textit{Otherwise}) \end{cases}. \quad (18)$$

As shown in Definition 3, ScaleAP computes responsibility by following the iterative message-passing processes in Eq. (1) only if the object pair $(x_i, x_j) \in \mathcal{R}_t(i)$. Otherwise, it skips the message-passing and reconstructs responsibilities from the corresponding similarities. Additionally, in the aggregated availability shown in Definition4, ScaleAP directly obtains the availability from $r_0(j,j)$ without using message-passing if object pairs $(x_i, x_j) \in \hat{\mathcal{A}}_t(j)$.

To discuss the theoretical aspects of Definitions 3 and 4, we derive the following two properties:

Lemma 7. *In the t-th iteration, $r_t(i,j) = \acute{r}_t(i,j)$ holds.*

Proof. If $(x_i, x_j) \in R_t(i)$, $r_t(i,j) = \acute{r}_t(i,j)$ clearly holds. Hence, we prove $r_t(i,j) = r_t(i,k) + \triangle s_i(j,k)$ such that $k = \arg\max_l\{s(i,l) \mid (x_i, x_l) \in \hat{\mathcal{R}}_t(i)\}$ using mathematical induction.

First, suppose that $t = 1$. From Eq. (1), $r_1(i,j) = (1-\lambda)\rho_1(i,j) + \lambda r_0(i,j)$. As shown in Definition 1, $j, k \neq \max_l\{s(i,l)\}$ since $(x_i, x_j), (x_i, x_k) \in \hat{\mathcal{R}}_1(i)$. Thus, using $s(i,j) = s(i,k) + \triangle s_i(j,k)$, $\rho_1(i,k)$ and $r_0(i,k)$ should be computed as $\rho_1(i,j) = \rho_1(i,k) + \triangle s_i(j,k)$ and $r_0(i,j) = r_0(i,k) + \triangle s_i(j,k)$, respectively. That is, from the above equations, Lemma 7 clearly holds for $t = 1$.

Next, we prove Lemma 7 for $t = n$ if $r_{n-1}(i,j) = r_{n-1}(i,k) + \triangle s_i(j,k)$ holds. From Eq. (1), we have

$$r_n(i,j) = (1-\lambda)\rho_n(i,j) + \lambda r_{n-1}(i,j). \quad (19)$$

From Lemma 1, $\rho_n(i,j)$ is uniquely determined by the corresponding similarity $s(i,j) = s(i,k) + \triangle s_i(j,k)$. Thus, from Eq. (3), we can derive $\rho_n(i,j)$ as

$$\begin{aligned} \rho_n(i,j) &= s(i,k) + \triangle s_i(j,k) - \max_{k\neq l}\{a_{t-1}(i,l) + s(i,l)\} \\ &= \rho_n(i,k) + \triangle s_i(j,k). \end{aligned} \quad (20)$$

Recall that we have $r_{n-1}(i,j) = r_{n-1}(i,k) + \triangle s_i(j,k)$. Hence, from Eq. (19), Lemma 7 holds for $t = n$, which completes the proof of Lemma 7. □

Lemma 8. *In the t-th iteration, $a_t(i,j) = \acute{a}_t(i,j)$ holds.*

Proof. If $(x_i, x_j) \notin \hat{A}_t(j)$, then $a_t(i,j) = \acute{a}_t(i,j)$ holds from Eq. (2) and Definition 4. Here, we prove Lemma 8 for object pairs included in $\hat{A}_t(j)$. From Lemma 4, if $(x_i, x_j) \in \hat{A}_t(j)$, $a_t(i,j)$ is uniquely determined by $r_0(j,j)$. That is,

$$a_t(i,j) = (1-\lambda)\textstyle\sum_{l=1}^{t}\min\{0, r_0(j,j)\} = \acute{a}_t(i,j), \quad (21)$$

which completes the proof of Lemma 8. □

Lemmas 7 and 8 indicate that ScaleAP can obtain the same responsibilities and availabilities, even though it skips the message-passing by the aggregated message update.

3.4 Algorithm

Algorithm 1 gives a full description of ScaleAP. ScaleAP iteratively performs message updates using a given set of similarities, $\mathcal{S}$, until all messages converge (lines 3-14).

In each iteration, ScaleAP first computes responsibilities (lines 4–8). At the beginning of the responsibility computation, it constructs $R_t(i)$ for each data object $x_i \in \mathcal{X}$ (lines 5–6). From Lemma 5, we can obtain $\hat{R}_t(i)$ in $O(1)$. Thus, ScaleAP can constructs $R_t(i)$ in $O(1)$. Afterward, ScaleAP computes the responsibilities only for the object pairs included in $R_t(i)$ by following Definition 3 (lines 7-8). As

Algorithm 1 Proposed method: ScaleAP

```
Input: A set of similarities $\mathcal{S}$;
Output: A set of exemplars $\mathcal{E}$;
 1: $\mathcal{E} \leftarrow \emptyset$;
 2: Obtain $\hat{\mathcal{R}}_1(i)$ and $\hat{\mathcal{A}}_1(i)$ for $i = 1, \ldots, N$;
 3: for $t = 1$ to $T$ do
 4:     for $i = 1$ to $N$ do
 5:         $k \leftarrow \arg\max_l \{(x_i, x_l) \mid (x_i, x_l) \in \hat{\mathcal{R}}_t(i)\}$;
 6:         $\mathcal{R}_t(i) \leftarrow \{(x_i, x_j) \mid (x_i, x_j) \notin \hat{\mathcal{R}}_t(i)\} \cup \{(x_i, x_k)\}$;
 7:         for each $(x_i, x_j) \in \mathcal{R}_t(i)$ do
 8:             Compute $\acute{r}_t(i, j)$ by Definition 3;
 9:     for $j = 1$ to $N$ do
10:         $\mathcal{A}_t(j) \leftarrow \{(x_i, x_j) \mid (x_i, x_j) \notin \hat{\mathcal{A}}_t(j)\}$;
11:         for each $(x_i, x_j) \in \mathcal{A}_t(j)$ do
12:             Compute $\acute{a}_t(i, j)$ by Definition 4;
13:     for $i = 1$ to $N$ do
14:         Obtain $\hat{\mathcal{R}}_{t+1}(i)$ and $\hat{\mathcal{A}}_{t+1}(i)$ by Definition 1 and 2;
15: for $i = 1$ to $N$ do
16:     $e(i) \leftarrow \arg\max_j \{r_t(i, j) + a_t(i, j)\}$;
17:     $\mathcal{E} \leftarrow \mathcal{E} \cup \{e(i)\}$;
18: return $\mathcal{E}$;
```

we proved in Lemma 7, we can obtain exactly the same responsibilities for $(x_i, x_j) \notin \mathcal{R}_t(i)$ in $O(1)$ if object pairs in $\mathcal{R}_t(i)$ are computed. Hence, in Algorithm 1, ScaleAP skips to compute responsibilities for $(x_i, x_j) \notin \mathcal{R}_t(i)$. Then it computes the availabilities (lines 9–12). Similar to the above responsibility computations, ScaleAP constructs $\mathcal{A}_t(j)$ from $\hat{\mathcal{A}}_t(j)$ in $O(1)$ (line 10). Afterward, it computes the availabilities for object pairs included in $\mathcal{A}_t(j)$ using Definition 4. After updating the messages by following Lemmas 5 and 6, ScaleAP constructs $\hat{R}_{t+1}(i)$ and $\hat{A}_{t+1}(i)$ in $O(1)$ for each object $x_i \in \mathcal{X}$ (lines 13-14). ScaleAP iterates the above message updates until those messages are no longer updated. Finally, it outputs exemplar $\mathcal{E}$ using Eq. (6) (lines 15-18).

Algorithm 1 shows that ScaleAP does not require any pre-computations or user-specified parameters. This sharply contrasts existing AP algorithms. Consequently, ScaleAP provides a simple solution for users.

Finally, we discuss the theoretical aspects of ScaleAP. Let N and T be the number of data objects and iterations, respectively. ScaleAP has the following properties:

Theorem 1. *ScaleAP incurs $O((1+\epsilon)NT)$ time, where ϵ is the average number of exemplar candidates.*

Proof. From Lemmas 5 and 6, ScaleAP can obtain $\hat{\mathcal{R}}_t(i)$ and $\hat{\mathcal{A}}_t(i)$ in $O(1)$ for each iteration. Hence, ScaleAP incurs $O(NT)$ time to find prunable pairs. In each iteration, ScaleAP computes responsibilities for object pairs included in $\mathcal{R}_t(i)$ (lines 4-8 in Algorithm 1). From Definition 3, $\mathcal{R}_t(i)$ contains three object pairs at most (*i.e.,* (x_i, x_i), (x_i, x_j), and (x_i, x_k), where $j = \arg\max_l\{a_{t-1}(i,l) + s(i,l)\}$, and $k = \arg\max_l\{s(i,l)\}$). Thus, ScaleAP incurs $O(3NT) = O(NT)$ time for responsibility computations. Additionally, in the availability computations (lines 9-12), ScaleAP computes the availability only if a data object x_j has $\max\{r_{t-1}(i,j)\} > 0$. Recall from Eqs. (1) and (5), $r_{t-1}(i,j)$ should be a positive value only if $s(x_i, x_j)$ is the largest value among all possible $x_j \in \mathcal{X}$. $r_{t-1}(i,j)$ is the message sent from x_i to x_j to represent how strongly x_i wants to choose x_j as its exemplar. That is, a data object x_j is a candidate of exemplars in the $(t-1)$-th iteration, x_j has $\max\{r_{t-1}(i,j)\} > 0$, and ScaleAP computes availabilities for x_j. Thus, ScaleAP incurs $O(\epsilon NT)$ time for the availability computations. Finally, it obtains exemplars in $O(N)$ time. Therefore, ScaleAP requires $O(NT + NT + \epsilon NT + N) = O((1+\epsilon)NT)$ time to obtain exemplars. □

As shown in Section 1, existing AP algorithms require $O(N^2T)$ time. Consequently, Theorem 1 indicates that ScaleAP is dramatically faster than AP and other state-of-the-art AP algorithms. In practice, ϵ should be a small constant in real-world datasets (*i.e.,* $\epsilon \ll N$) since real-world datasets generally have a small number of clusters (Tibshirani, Walther, and Hastie 2001; Shiokawa, Amagasa, and Kitagawa 2019). Specifically, as shown in Table 1, the real-world datasets examined in the next section have at most 31 clusters, which are significantly smaller than the dataset sizes. Thus, ScaleAP has a nearly linear scalability against the number of data objects. In other words, $O((1+\epsilon)NT) \approx O(NT)$ on the real-world datasets. In Section 4, we experimentally verify the efficiency of ScaleAP.

Theorem 2. *ScaleAP always outputs the same exemplars as those of AP (Frey and Dueck 2007).*

Proof. As we proved in Lemmas 7 and 8, $r_t(i,j) = \acute{r}_t(i,j)$ and $a_t(i,j) = \acute{a}_t(i,j)$ always hold even if an object pair (x_i, x_j) is not computed by Algorithm 1. In each iteration, ScaleAP can perform the same message updates as those of AP. From Eq. (6), the exemplars are uniquely determined by the values of responsibility and availability. Therefore, ScaleAP always outputs the same exemplars as AP. □

4 Experimental Evaluation

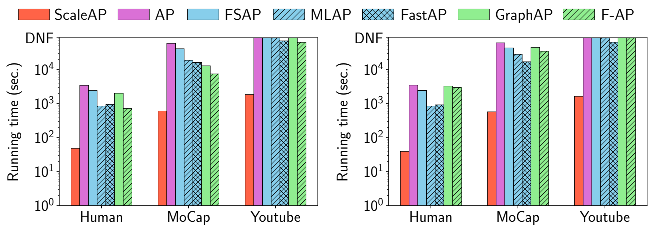
We experimentally evaluated the effectiveness of ScaleAP by comparing it with the following AP algorithms.

- **AP:** The original AP algorithm (Frey and Dueck 2007).
- **FSAP:** The most popular message sampling approach (Jia et al. 2008). It samples object pairs by constructing a kNN graph and performs message-passing on the graph until convergence. For the graph construction, we set $k = 15$.
- **FastAP:** The state-of-the-art message sampling approach (Sun et al. 2017). FastAP constructs a sparse factor graph by finding micro-cluster structures from given similarities. It performs the message-passing on the graph.
- **GraphAP:** The message bounding algorithm, which prunes unpromising object pairs while keeping its clustering quality (Fujiwara, Irie, and Kitahara 2011).
- **F-AP:** The state-of-the-art bounding algorithm (Fujiwara et al. 2015). It improves the efficiency of GraphAP by using dynamic message pruning methods.
- **MLAP:** The multi-level coarsening algorithm proposed by (Shang et al. 2012). It recursively coarsens object pairs by using the eigenvalue decomposition.

All experiments were conducted on a server with Intel Xeon CPU 2.60 GHz and 768 GiB RAM.

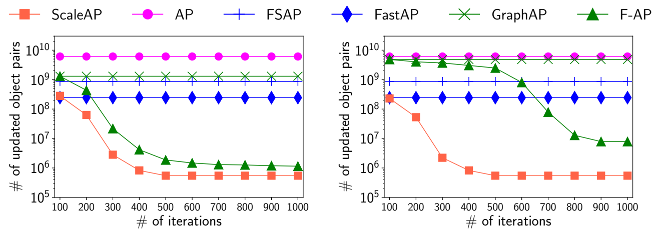
Name	N	# of attributes	# of clusters	Data source
Human	10,299	561	6	(Anguita et al. 2013)
MoCap	78,095	38	5	(Gardner et al. 2014)
Youtube	120,000	647	31	(Madani, Georg, and Ross 2013)

Table 1: Statistics of real-world datasets.



(a) Median (b) Minimum

Figure 1: Running time



(a) Median (b) Minimum

Figure 2: # of message updates on MoCap

Datasets: We used three real-world datasets summarized in Table 1. All the datasets are published by UCI Machine Learning Repository (Dua and Graff 2017). Note that in the Youtube dataset, we chose the 647-dimensional HOG features provided by (Madani, Georg, and Ross 2013).

Experimental settings: The experiments used the negative Euclidean distance as the similarity between data objects. In accordance with (Frey and Dueck 2007), we set the preference to both the median and minimum of the input similarities, $\lambda = 0.5$, and the maximum number of iterations to $T = 1,000$. The results were the average of over 10 independent runs. We also reported the standard deviation values of accuracy, whereas they were omitted from runtime evaluations since they were smaller than 0.1 seconds.

4.1 Efficiency

Figure 1 shows the running time on the real-world datasets, where DNF indicates that the runtime exceeded 24 hours. Overall, ScaleAP achieves the highest performance. On average, ScaleAP is 58.7 times faster than the other approaches. Although ScaleAP guarantees the same results as AP, it is up to 106.6, 83.4, and 75.9 times faster than AP, GraphAP, and F-AP, respectively. As we proved in Theorem 1, ScaleAP shows a better time complexity compared to other competitive algorithms. This feature is responsible for the superior running time compared to other methods.

ScaleAP is still efficient even if the preference is set to the minimum. This differs from the bounding algorithms, which significantly degrade their running time for such preference setting. The bounding algorithms prune messages by updating the bounds of the messages. However, if one preference is smaller than the others, the bounds become loose. This is why bounding algorithms fail to prune message updates for the minimum preference. By contrast, ScaleAP finds the prunable pairs (Section 3.2) regardless of the preference value. Thus, ScaleAP can efficiently compute massive datasets compared to the other AP algorithms.

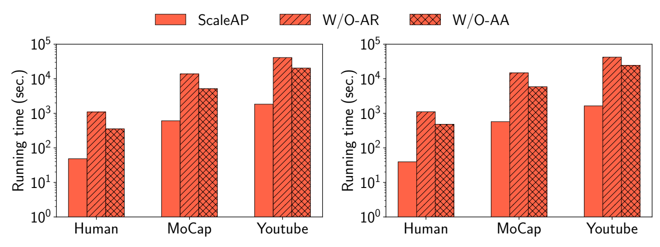
4.2 Effectiveness

To verify how ScaleAP effectively reduces message updates during the message-passing, Figure 2 plots the number of computed pairs in each iteration. The results of MLAP are excluded because MLAP does not employ the message-passing. Compared to AP, ScaleAP shows a 95.4% reduction in the updates during iterative computations. Moreover, the other algorithms computed many more messages than ScaleAP. Although FSAP and FastAP also reduced the number of message updates, they require a longer running time than ScaleAP (Figure 1). To preserve the clustering quality, these algorithms have sampling overheads prior to message-passing processes. By contrast, as we proved in Lemmas 5 and 6, ScaleAP can find each prunable pair in $O(1)$ without pre-computations and overheads. Thus, ScaleAP can efficiently reduce the number of message updates.

Next, we evaluated the effectiveness of our approaches by comparing the runtime of ScaleAP with variants that exclude either the aggregated responsibility or the aggregated availability. Figure 3 shows the runtime of each algorithm, where *W/O-AR* and *W/O-AA* represent ScaleAP without the aggregated responsibility and the aggregated availability, respectively. On average, ScaleAP is 24.5 and 10.7 times faster than W/O-AR and W/O-AA, respectively. These results indicate that the aggregated responsibility leads to an enhanced efficiency improvement. As discussed in Section 3.4, $\mathcal{R}_t(i)$ includes at most three object pairs regardless of the similarities. Consequently, the cost for the responsibility computations approaches an almost linear running time.

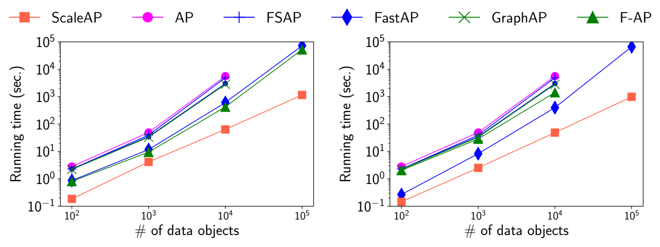
4.3 Scalability

We assessed the scalability of ScaleAP. Figure 4 shows the running time on Youtube as a function of the number of data objects. We randomly sampled 10^2, 10^3, 10^4, and 10^5 objects from Youtube and evaluated the runtime on each sampled dataset. If the method did not finish within 24 hours, the results are excluded from Figure 4. ScaleAP is more



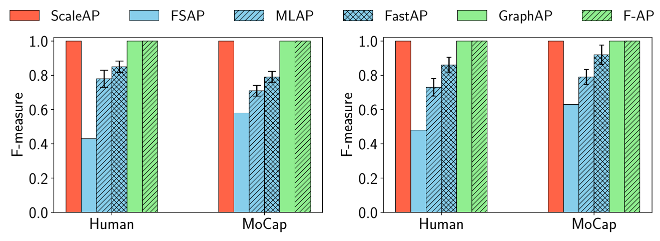
(a) Preference (median) (b) Preference (minimum)

Figure 3: Effectiveness



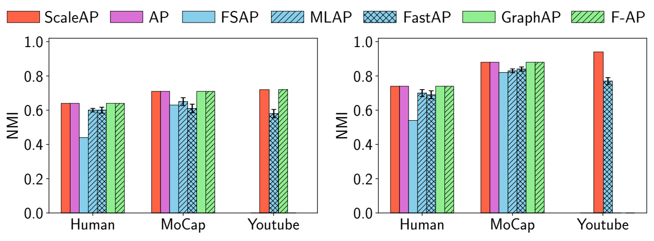
(a) Preference (median) (b) Preference (minimum)

Figure 4: Scalability



(a) Preference (median) (b) Preference (minimum)

Figure 5: F-measure (Error bars show standard deviation)



(a) Preference (median) (b) Preference (minimum)

Figure 6: NMI (Error bars show standard deviation)

scalable than the other AP algorithms. It shows a nearly linear scalability against the number of data objects. As described in Section 1, AP and its variants require $O(N^2T)$ time, whereas ScaleAP requires $O((1+\epsilon)NT)$ time, as discussed in Theorem 1. Furthermore, the real-world datasets have very small numbers of clusters (Table 1). This implies that the number of exemplars, ϵ, should be a small constant in the datasets. Consequently, the practical cost of ScaleAP is $O((1+\epsilon)NT) \approx O(NT)$. Thus, ScaleAP achieves a nearly linear scalability.

4.4 Exactness

One advantage of ScaleAP is that it outputs the same exemplars as those of AP, while dynamically excluding unnecessary message updates. To verify this advantage, we used the following metrics to evaluate AP algorithms:

- **F-measure:** We measured F-measure (Manning, Raghavan, and Schütze 2008) of the obtained exemplars against those of AP. F-measure is 1 if the exemplars exactly match those of AP. Otherwise, it approaches 0.
- **Normalized mutual information (NMI):** We measured NMI (Cilibrasi and Vitányi 2005) to evaluate the clustering accuracy of each algorithm against the ground-truth clusters of each dataset. NMI is 1 if the obtained clusters are the same as the ground truth.

To compare the exactness of ScaleAP, Figure 5 shows the F-measure. ScaleAP detects the same exemplars as those of AP. On the other hand, the other sampling algorithms (FSAP, MLAP, and FastAP) fail to reproduce the exemplars of AP. ScaleAP theoretically guarantees the same responsibilities and availabilities as AP by Lemmas 7 and 8, while removing message updates during the message-passing processes. That is, ScaleAP performs the message-passing processes equivalent to those processes of AP. Hence, ScaleAP finds the same exemplars as AP by Theorem 2.

Figure 6 compares NMI scores to analyze the clustering accuracy. ScaleAP has higher NMI scores than FSAP, MLAP, and FastAP. In addition, ScaleAP yields the same NMI scores as AP. As described in Section 1.1, FSAP, MLAP, and FastAP samples, exemplars, or object pairs aim to reduce the high computational cost. However, these sampling approaches fail to reproduce the same exemplars as AP (Figure 5). By contrast, as we proved in Theorem 2, ScaleAP is theoretically designed to find the same exemplars as AP. Therefore, ScaleAP can successfully inherit the high clustering quality from AP.

5 Conclusion

ScaleAP is an efficient AP algorithm that produces the same clustering results as AP but with a faster computation time. ScaleAP excludes unnecessary message updates by finding prunable pairs during message passing. In an experiment, ScaleAP offers an improved efficiency on massive datasets compared to other AP algorithms without sacrificing the clustering quality. Consequently, employing ScaleAP for massive datasets should enhance the effectiveness of AI-powered applications.

Acknowledgments

This work was partially supported by JST ACT-I and JST PRESTO JPMJPR2033, Japan. I thank to Hiroyuki Kitagawa, Ken-ichi Kawarabayashi, and Tomohiro Matsushita for their helps and useful discussions.

References

Ambrogi, F.; Boracchi, P.; Biganzoli, E.; Raimondi, E.; and Soria, D. 2008. Cancer Profiles by Affinity Propagation. In *Proceedings of the 7th IEEE International Conference on Machine Learning and Applications (ICMLA 2008)*, 650–655.

Anguita, D.; Ghio, A.; Oneto, L.; Parra, X.; and Reyes-Ortiz, J. L. 2013. A Public Domain Dataset for Human Activity Recognition Using Smartphones. In *Proceedings of the 21st European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN 2013)*, 437–442. URL https://archive.ics.uci.edu/ml/datasets/Human+Activity+Recognition+Using+Smartphones.

Belkin, M.; and Niyogi, P. 2003. Laplacian Eigenmaps for Dimensionality Reduction and Data Representation. *Neural Compututation* 15(6): 1373–1396. ISSN 0899-7667.

Cilibrasi, R.; and Vitányi, P. M. 2005. Clustering by Compression. *IEEE Transactions on Information Theory* 51(4): 1523–1545.

Dua, D.; and Graff, C. 2017. UCI Machine Learning Repository. URL http://archive.ics.uci.edu/ml. Accessed: September 1st, 2020.

Dueck, D.; and Frey, B. J. 2007. Non-metric Affinity Propagation for Unsupervised Image Categorization. In *Proceedings of the 11th IEEE International Conference on Computer Vision (ICCV 2007)*, 1–8.

Frey, B. J.; and Dueck, D. 2007. Clustering by Passing Messages Between Data Points. *Science* 315(5814): 972–976.

Fujiwara, Y.; Irie, G.; and Kitahara, T. 2011. Fast Algorithm for Affinity Propagation. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)*, 2238–2243.

Fujiwara, Y.; Nakatsuji, M.; Shiokawa, H.; Ida, Y.; and Toyoda, M. 2015. Adaptive Message Update for Fast Affinity Propagation. In *Proceedings of the 21st ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD 2015)*, 309–318.

Gardner, A.; Kanno, J.; Duncan, C. A.; and Selmic, R. 2014. Measuring Distance between Unordered Sets of Different Sizes. In *Proceedings of the 27th IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2014)*, 137–143. URL https://archive.ics.uci.edu/ml/datasets/MoCap+Hand+Postures.

Jia, Y.; Wang, J.; Zhang, C.; and Hua, X.-S. 2008. Finding Image Exemplars Using Fast Sparse Affinity Propagation. In *Proceedings of the 16th ACM International Conference on Multimedia (MM 2008)*, 639–642.

Kazantseva, A.; and Szpakowicz, S. 2011. Linear Text Segmentation Using Affinity Propagation. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing (EMNLP 2011)*, 284–293.

Kschischang, F. R.; Frey, B. J.; and Loeliger, H. 2001. Factor Graphs and the Sum-Product Algorithm. *IEEE Transactions on Information Theory* 47(2): 498–519.

Madani, O.; Georg, M.; and Ross, D. A. 2013. On Using Nearly-Independent Feature Families for High Precision and Confidence. *Machine Learning* 92: 457–477. URL https://archive.ics.uci.edu/ml/datasets/YouTube+Multiview+Video+Games+Dataset.

Manning, C. D.; Raghavan, P.; and Schütze, H. 2008. *Introduction to Information Retrieval*. New York, NY, USA: Cambridge University Press. ISBN 0521865719, 9780521865715.

Matsushita, T.; Shiokawa, H.; and Kitagawa, H. 2018. C-AP: Cell-based Algorithm for Efficient Affinity Propagation. In *Proceedings of the 20th International Conference on Information Integration and Web-based Applications & Services (iiWAS 2018)*, 156–163.

Roweis, S. T.; and Saul, L. K. 2000. Nonlinear Dimensionality Reduction by Locally Linear Embedding. *Science* 290(5500): 2323–2326.

Shang, F.; Jiao, L.; Shi, J.; Wang, F.; and Gong, M. 2012. Fast Affinity Propagation Clustering: A Multilevel Approach. *Pattern Recognition* 45(1): 474–486. ISSN 0031-3203.

Shiokawa, H.; Amagasa, T.; and Kitagawa, H. 2019. Scaling Fine-grained Modularity Clustering for Massive Graphs. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI 2019)*, 4597–4604.

Shiokawa, H.; Fujiwara, Y.; and Onizuka, M. 2013. Fast Algorithm for Modularity-based Graph Clustering. In *Proc. AAAI 2013*, 1170–1176.

Shiokawa, H.; Fujiwara, Y.; and Onizuka, M. 2015. SCAN++: Efficient Algorithm for Finding Clusters, Hubs and Outliers on Large-scale Graphs. *Proceedings of the Very Large Data Bases Endowment (PVLDB)* 8(11): 1178–1189.

Sun, L.; and Guo, C. 2014. Incremental Affinity Propagation Clustering Based on Message Passing. *IEEE Transactions on Knowledge and Data Engineering* 26(11): 2731–2744.

Sun, L.; Guo, C.; Liu, C.; and Xiong, H. 2017. Fast Affinity Propagation Clustering Based on Incomplete Similarity Matrix. *Knowledge and Information Systems* 51: 941–963.

Tibshirani, R.; Walther, G.; and Hastie, T. 2001. Estimating the Number of Clusters in a Data Set via the Gap Statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 63(2): 411–423.

Yedidia, J. S.; Freeman, W. T.; and Weiss, Y. 2005. Constructing Free-Energy Approximations and Generalized Belief Propagation Algorithms. *IEEE Transactions on Information Theory* 51(7): 2282–2312.