# ALID: Scalable Dominant Cluster Detection

Lingyang Chu
Key Lab of Intell. Info. Process., ICT, CAS, Beijing, China
lingyang.chu@vipl.ict.ac.cn

Shuhui Wang
Key Lab of Intell. Info. Process., ICT, CAS, Beijing, China
wangshuhui@ict.ac.cn

Siyuan Liu
Heinz College, Carnegie Mellon University, Pittsburgh, USA
siyuan@cmu.edu

Qingming Huang
University of Chinese Academy of Sciences, Beijing, China
qmhuang@jdl.ac.cn

Jian Pei
School of Computing Science, Simon Fraser University, Vancouver, Canada
jpei@cs.sfu.ca

## ABSTRACT

Detecting dominant clusters is important in many analytic applications. The state-of-the-art methods find dense subgraphs on the affinity graph as dominant clusters. However, the time and space complexities of those methods are dominated by the construction of affinity graph, which is quadratic with respect to the number of data points, and thus are impractical on large data sets. To tackle the challenge, in this paper, we apply *Evolutionary Game Theory* (EGT) and develop a scalable algorithm, Approximate Localized Infection Immunization Dynamics (ALID). The major idea is to perform Localized Infection Immunization Dynamics (LID) to find dense subgraphs within local ranges of the affinity graph. LID is further scaled up with guaranteed high efficiency and detection quality by an estimated Region of Interest (ROI) and a Candidate Infective Vertex Search method (CIVS). ALID only constructs small local affinity graphs and has time complexity $\mathcal{O}(C(a^* + \delta)n)$ and space complexity $\mathcal{O}(a^*(a^* + \delta))$, where $a^*$ is the size of the largest dominant cluster, and $C \ll n$ and $\delta \ll n$ are small constants. We demonstrate by extensive experiments on both synthetic data and real world data that ALID achieves the state-of-the-art detection quality with much lower time and space cost on single machine. We also demonstrate the encouraging parallelization performance of ALID by implementing the Parallel ALID (PALID) on *Apache Spark*. PALID processes 50 million SIFT data points in 2.29 hours, achieving a speedup ratio of 7.51 with 8 executors.

## 1. INTRODUCTION

A dominant cluster is a group of highly similar objects that possesses maximal inner group coherence [19, 29]. On a massive data set, more often than not the dominant clusters carry useful information and convey important knowledge. For example, in a big collection of news data, such as official news, RSS-feeds and tweet-streams, the dominant clusters may indicate potential real world hot events [4, 32]. In a large repository of interpersonal communication data, such as emails and social networks, the dominant clusters may reveal stable social hubs [39]. Therefore, efficiently and effectively detecting dominant clusters from massive data sets has become an important task in data analytics.

In real applications, dominant cluster detection often faces two challenges. First, the number of dominant clusters is often unknown. Second, large data sets are often noisy. An unknown number of dominant clusters are often hidden deeply in an overwhelming amount of background noise [32, 39]. For instance, numerous news items about almost every aspect of our daily life are added to the Web everyday. Most of the news items are interesting to small groups of people, and hardly attract sufficient social attention or become a dominant cluster of a hot event. As another example, billions of spam messages are sent everyday. Finding meaningful email threads and activities is a typical application of dominant cluster detection, and is challenging mainly due to the large amount of spam messages as noise [19].

Traditional partitioning-based clustering methods like $k$-means [5, 22, 33] and spectral clustering [16, 18, 25, 37] are often used in cluster detection. However, such methods are not robust in processing noisy data with an unknown number of dominant clusters [29]. First, those methods typically require a pre-defined number of (dominant) clusters. Without prior knowledge on the true number of (dominant) clusters, setting an improper number of clusters may lead to low detection accuracy. Second, each data item, including both members of dominant clusters and noise data items, is forced to be assigned to a certain cluster, which inevitably leads to degenerated detection accuracy and subtracted cluster coherence under high background noise.

The affinity-based methods [29, 31], which detect dominant clusters by finding dense subgraphs on an affinity graph, are effective in detecting an unknown number of dominant clusters from noisy background. Since the data objects in a dominant cluster are very similar to each other, they naturally form a highly cohesive dense subgraph on the affinity graph. Such high cohesiveness is proven to be a stable characteristic to accurately identify dominant


*Proceedings of the VLDB Endowment,* Vol. 8, No. 8

clusters from noisy background without knowing the exact number of clusters [19, 31]. Nevertheless, efficiency and scalability are the bottlenecks of the affinity-based methods, since the complexity of constructing the affinity matrix from $n$ data items is $\mathcal{O}(n^2)$ in both time and space. Even though the computational cost can be saved by forcing the affinity graph sparse [9], the enforced sparsity breaks the intrinsic cohesiveness of dense subgraphs and consequently affects the detection quality of dominant clusters.

To tackle the challenge, in this paper, we propose Approximate Localized Infection Immunization Dynamics (ALID), a dominant cluster detection approach that achieves high scalability and retains high detection quality. The key idea is to avoid constructing the global complete affinity graph. Instead, ALID finds a dense subgraph within an accurately estimated local Region of Interest (ROI). The ROI is guaranteed by the *law of triangle inequality* to completely cover a dominant cluster, and thus fully preserves the high intrinsic cohesiveness of the corresponding dense subgraph and ensures the detection quality. Moreover, since dominant clusters generally exist in small local ranges, ALID only searches a small local affinity subgraph within the ROI. Therefore, ALID only constructs small local affinity graphs and largely avoids the expensive construction of the global affinity graph. Consequently, the original $\mathcal{O}(n^2)$ time and space complexity of affinity graph construction is significantly reduced to $\mathcal{O}(C(a^* + \delta)n)$ in time and $\mathcal{O}(a^*(a^* + \delta))$ in space, where $a^*$ is the size of the largest dominant cluster, and $C \ll n$ and $\delta \ll n$ are small constants.

We make the following major contributions. First, we propose LID to detect dense subgraphs on a local affinity graph within a ROI. LID localizes the Infection Immunization Dynamics [31] to efficiently seek dense subgraphs on a small local affinity graph. It only computes a few columns of a local affinity matrix to detect a dense subgraph without sacrificing detection quality. This significantly reduces the time and space complexity.

Second, we estimate a Region of Interest (ROI) and propose a Candidate Infective Vertex Search (CIVS) method to significantly improve the scalability of LID and ensure high detection quality. The ROI is guaranteed to accurately identify the local range of the "*true*" dense subgraph, which ensures the detection quality of ALID. The CIVS method is proposed to quickly retrieve the data items within the ROI, where all data items are efficiently indexed by Locality Sensitive Hashing (LSH) [13]. Demonstrated by extensive experiments on synthetic data and real world data, ALID achieves substantially better scalability than the other affinity-based methods without sacrificing the detection quality.

Third, we carefully design a parallelized solution on top of the MapReduce framework [14] to further improve the scalability of ALID. The promising parallelization performance of Parallel ALID (PALID) is demonstrated by the experiments on *Apache Spark* (http://spark.apache.org/). PALID can efficiently process 50 million SIFT data [23] in 2.29 hours and achieve a speedup ratio of 7.51 with 8 executors.

The rest of the paper is organized as follows. We review related work in Section 2, and revisit the problem of dense subgraph finding in Section 3. We present the ALID method in Section 4. Section 5 reports the experimental results. Section 6 concludes the paper. Limited by space, the mathematical proofs are omitted in this paper. The full version of this work can be found at [11].

## 2. RELATED WORK

For dominant cluster detection, the affinity-based methods [19, 28, 29, 31] that find dense subgraphs on affinity graphs are more resistant against background noise than the canonical partitioning-based methods like $k$-means [5, 22] and spectral clustering [16, 18, 37]. The dense subgraph seeking problem is well investigated in literature [3, 4, 36]. Motzkin *et al.* [24] proved that seeking dense subgraphs on an un-weighted graph can be formulated as a quadratic optimization problem on the simplex. Their method was extended to weighted graphs by the dominant set method (DS) [29], which solves a standard quadratic optimization problem (StQP) by replicator dynamics (RD) [38].

Bulò *et al.* [31] and Pavan *et al.* [28] showed that, given the full affinity matrix of an affinity graph with $n$ vertices, the time complexity for each RD iteration is $\mathcal{O}(n^2)$, which hinders its application on large data sets. Thus, they proposed the infection immunization dynamics (IID) to solve the StQP problem in $\mathcal{O}(n)$ time and space. However, the overall time and space complexity of IID is still $\mathcal{O}(n^2)$, since each iteration of IID needs the full affinity matrix, which costs quadratic time and space to compute and store. Pavan *et al.* [28] reduced the computational overheads by sampling graph vertices from the original affinity graph with an efficient out-of-sample method. As the tradeoff, the cluster detection accuracy is affected by the sampling rate.

Since most dense subgraphs exist in local ranges of an affinity graph, running RD on the entire graph is inefficient [19, 21]. Therefore, Liu *et al.* [19] proposed the shrinking and expansion algorithm (SEA) to effectively prevent unnecessary time and space cost by restricting all RD iterations on small subgraphs. Both the time and space complexities of SEA are linear with respect to the number of graph edges [19]. The scalability of SEA is sensitive to the sparse degree of the affinity graph. Liu *et al.* [20] and Bulò *et al.* [7] also extended dominant cluster detection from pairwise affinity graphs to hypergraphs. Their methods [20, 7] focus on capturing high-order coherence of vertices with hypergraph, which leads to high computational complexity and limited scalability.

Affinity propagation (AP) [17] is another noise resistant solution to detect an unknown number of dominant clusters. It finds the dominant clusters by passing real valued messages along graph edges, which is very time consuming when there are many vertices and edges. aiNet [26] is an evolutionary artificial immune network designed for affinity based clustering. However, the scalability of aiNet is limited due to its high computational complexity.

Mean shift [12] differs from the affinity-based methods by directly seeking clusters in the feature space. It assumes that the discrete data items are sampled from a pre-defined density distribution, and detects clusters by iteratively seeking the maxima of the density distribution. However, the detection quality of mean shift is sensitive to the type and bandwidth of the pre-defined density distribution.

The above affinity-based methods are able to achieve high detection quality when the affinity matrix is already materialized. However, the scalability of those methods is limited by the $\mathcal{O}(n^2)$ time and space complexities of the affinity matrix computation. To the best of our knowledge, ALID developed in this paper is the first attempt to achieve high scalability and retain good clustering accuracy.

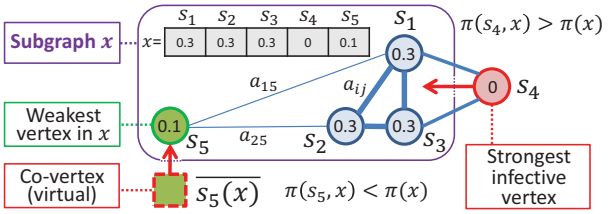



**Figure 1: An affinity graph $G$ with 5 vertices $\{s_1, s_2, s_3, s_4, s_5\}$. The thicker edges indicate larger affinity. The rounded rectangle highlights a subgraph $x$ of $G$. The subgraph $x$ is composed of vertices $\{s_1, s_2, s_3, s_5\}$ and is represented by $x = [0.3, 0.3, 0.3, 0, 0.1]^\top$, $x \in \triangle^5$. $s_4$ is an infective vertex against $x$, since $\pi(s_4, x) > \pi(x)$. $s_5$ is a weak vertex in $x$, since $\pi(s_5, x) < \pi(x)$. The co-vertex $\overline{s_5(x)}$ represents an infective subgraph composed of $\{s_1, s_2, s_3\}$.**

## 3. DENSE SUBGRAPH FINDING REVISIT

In this section, we revisit the dense subgraph finding problem from the perspective of Evolutionary Game Theory (EGT) [38] and discuss the common scalability bottleneck of the infection immunization dynamics (IID) [31] and other affinity-based methods.

Consider a global affinity graph, denoted by $G = (V, I, A)$, where $V = \{v_i \in R^d \mid i \in I = [1, n]\}$ is the set of vertices, each vertex $v_i$ uniquely corresponds to a $d$-dimensional data point in $R^d$ space, $R$ is the set of real numbers, and $I = [1, n]$ is the range of indices of all vertices. Let $A$ be the affinity matrix, where each entry $a_{ij}$ of $A$ represents the affinity between $v_i$ and $v_j$, that is,

$$a_{ij} = \begin{cases} e^{-k||v_i - v_j||_p} & i \neq j \\ 0 & i = j \end{cases} \tag{1}$$

where $||\cdot||_p$ represents the $L_p$-norm ($p \geq 1$) and $k > 0$ is the scaling factor of the Laplacian Kernel.

Given an affinity graph $G$ of $n$ vertices, each graph vertex $v_i$ can be further referenced by an $n$-dimensional index vector $s_i = [\underbrace{0 \cdots 0}_{i-1} \; 1 \; \underbrace{0 \cdots 0}_{n-i}]^\top$. In other words, both $v_i$ and $s_i$ refer to the same $i$-th vertex in the graph.

A subgraph can be modeled by a subset of vertices as well as their probabilistic memberships. Take Figure 1 as an example. We assign $L_1$ normalized non-negative weights to a subgraph $x$ containing vertices $\{s_1, s_2, s_3, s_5\}$ and represent the subgraph by $x = 0.3 \cdot s_1 + 0.3 \cdot s_2 + 0.3 \cdot s_3 + 0 \cdot s_4 + 0.1 \cdot s_5$. Alternatively, $x$ can be regarded as an $n$-dimensional vector storing all the vertex weights, where the $i$-th dimension $x_i$ is the weight of vertex $s_i$. Intuitively, $x_i$ embodies the probability that vertex $s_i$ belongs to subgraph $x$. Thus $x_i = 0$ indicates that $s_i$ does not belong to $x$. For example, vertex $s_4$ in Figure 1 does not belong to $x$. In general, a subgraph can be represented by an $n$-dimensional vector $x \in \triangle^n$ in the standard simplex $\triangle^n = \{x \in R^n \mid \sum_{i \in I} x_i = 1, x_i \geq 0\}$.

The average affinity between two subgraphs $x, y \in \triangle^n$ is measured by the weighted affinity sum between all their member vertices. That is,

$$\pi(y, x) = y^\top A x = \sum_i \sum_j x_i y_j a_{ij} \tag{2}$$

As a special case, when $y = s_i$, that is, $y$ is a subgraph of a single vertex, $\pi(s_i, x) = (s_i)^\top A x$ represents the average affinity between vertex $s_i$ and subgraph $x$. Moreover, the average affinity between subgraph $x$ and itself is $\pi(x) = \pi(x, x) = x^\top A x$, which measures the internal connection strength between all vertices of subgraph $x$. Liu and Yan [21] indicated that such internal connection strength is a robust measurement of the intrinsic cohesiveness of $x$. Thus, $\pi(x)$ is also called the **density** of subgraph $x$.

Bulò *et al.* [31] indicated that a dense subgraph is a subgraph of local maximum density $\pi(x)$ [21, 29], and every local maximum argument $x^*$ of $\pi(x)$ uniquely corresponds to a dense subgraph. Thus, the dense subgraph seeking problem can be reduced to the following standard quadratic optimization problem (StQP):

$$\begin{array}{ll} \text{Maximize} & \pi(x) = x^\top A x = \sum_i x_i \pi(s_i, x) \\ \text{s.t.} & x \in \triangle^n \end{array} \tag{3}$$

which can be solved by the Infection Immunization Dynamics (IID) [31]. IID finds the dense subgraph $x^*$ by iteratively increasing the graph density $\pi(x)$ in the following two steps.

**Infection** : for a vertex $s_i$ whose average affinity $\pi(s_i, x)$ is larger than the graph density $\pi(x)$, such as $\pi(s_4, x) > \pi(x)$ in Figure 1, increase the its weight $x_i$.

**Immunization** : for a vertex $s_i$ whose average affinity $\pi(s_i, x)$ is smaller than the graph density $\pi(x)$, such as $\pi(s_5, x) < \pi(x)$ in Figure 1, decrease its weight $x_i$.

For the sake of clarity, we write $\pi(y - x, x) = \pi(y, x) - \pi(x)$ and define the relationship between subgraphs $x$ and $y$ as follow. If $\pi(y - x, x) > 0$, then $y$ is said to be *infective* against $x$. Otherwise, $x$ is said to be *immune* against $y$. The set of infective subgraphs against $x$ is defined as

$$\gamma(x) = \{y \in \triangle^n \mid \pi(y - x, x) > 0\} \tag{4}$$

IID performs **infection** and **immunization** using the following **invasion model**.

$$z = (1 - \varepsilon)x + \varepsilon y \tag{5}$$

The new subgraph $z \in \triangle^n$ is obtained by invading $x \in \triangle^n$ with a subgraph $\varepsilon y$, where $\varepsilon \in [0, 1]$ and $y \in \triangle^n$. This transfers an amount of $\varepsilon$ weight from the vertices in subgraph $x$ to the vertices in subgraph $y$, while the vertex weights in $z$ still sum to 1. In other words, the weights of vertices in subgraph $x$ are decreased and the weights of vertices in subgraph $y$ are increased.

For each iteration, IID selects the optimal graph vertex $s_i$ that maximizes the absolute value of $|\pi(s_i - x, x)|$ by function $s_i = M(x)$, where

$$\begin{array}{l} M(x) = \arg \max_{s_i \in (\mathcal{C}_1 \cup \mathcal{C}_2)} |\pi(s_i - x, x)| \\ \mathcal{C}_1 = \{s_i \mid \pi(s_i - x, x) > 0)\} \\ \mathcal{C}_2 = \{s_i \mid \pi(s_i - x, x) < 0, x_i > 0\} \end{array} \tag{6}$$

If $s_i = M(x) \in \mathcal{C}_1$, then $\pi(s_i, x) > \pi(x)$ and $s_i$ is the strongest infective vertex. In this case, an **infection** is performed by the invasion model (Equation 5) with $y = s_i$. This increases $\pi(x)$ by increasing the weight of infective vertex $s_i$. For example, in Figure 1, invading $x$ with a weight of $\varepsilon$ of the strongest infective vertex $s_4$ transfers a weight of $\varepsilon$ from $\{s_1, s_2, s_3, s_5\}$ to $s_4$ and increases $\pi(x)$.

If $s_i = M(x) \in \mathcal{C}_2$, then $\pi(s_i, x) < \pi(x)$. Thus $x$ is immune against $s_i$ and $s_i$ is the weakest vertex in subgraph

$x$. In such a case, an **immunization** is performed by the invasion model (Equation 5) with $y = \overline{s_i(x)}$:

$$\overline{s_i(x)} = \frac{x_i}{x_i - 1}(s_i - x) + x \tag{7}$$

Here, $\overline{s_i(x)}$ is called the *co-vertex* of $s_i$, and represents an infective subgraph composed of all the vertices in subgraph $x$ except $s_i$. Thus, invading $x$ with $y = \overline{s_i(x)}$ by the invasion model (Equation 5) reduces the weight of vertex $s_i$ and increases the weights of the other vertices in subgraph $x$. For example, in Figure 1, the co-vertex $\overline{s_5(x)}$ represents a subgraph composed of $\{s_1, s_2, s_3\}$. Invading subgraph $x$ with a weight $\varepsilon$ of $\overline{s_5(x)}$ transfers a weight of $\varepsilon$ from $s_5$ to $\{s_1, s_2, s_3\}$ and increases $\pi(x)$.

Formally, the infective vertex (or co-vertex) $y$ of the invasion model (Equation 5) can be selected by $y = S(x)$, where

$$S(x) = \begin{cases} s_i & \text{if } s_i = M(x) \in \mathcal{C}_1 \\ \overline{s_i(x)} & \text{if } s_i = M(x) \in \mathcal{C}_2 \end{cases} \tag{8}$$

Bulò *et al.* [31] showed the following.

THEOREM 1 ([31]). *The following three statements are equivalent for $x \in \triangle^n$. 1) $x$ is immune against all vertices $s_i \in \triangle^n$; 2) $\gamma(x) = \emptyset$; and 3) $x$ is a dense subgraph with local maximum $\pi(x)$.*

According to Theorem 1, IID searches for the global dense subgraph $x^*$ by iteratively shrinking $\gamma(x)$ until $\gamma(x) = \emptyset$.

Moreover, Bulò *et al.* [30] showed the following.

THEOREM 2. *Let $y \in \gamma(x)$ and $z = (1-\varepsilon)x + \varepsilon y$, where $\varepsilon = \varepsilon_y(x)$ is defined as follows.*

$$\varepsilon_y(x) = \begin{cases} \min\left[-\frac{\pi(y-x,x)}{\pi(y-x)}, 1\right] & \text{if } \pi(y-x) < 0 \\ 1 & \text{otherwise} \end{cases} \tag{9}$$

*Then, $y \notin \gamma(z)$ and $\pi(z) > \pi(x)$.*

According to Theorem 2, any infective subgraph $y \in \gamma(x)$ can be excluded from $\gamma(z)$ by invading $x$ with weight $\varepsilon = \varepsilon_y(x)$ of $y$. This monotonously reduces the volume of $\gamma(z)$ and guarantees the convergence of IID.

IID needs the full affinity matrix to find dense subgraphs by solving the StQP problem (Equation 3). Therefore, its scalability is heavily affected due to the $\mathcal{O}(n^2)$ time and space complexities in computing the complete affinity matrix $A$. Although forcing $A$ sparse can reduce such expensive time and space cost to some extent [9], the enforced sparsity breaks the intrinsic cohesiveness of dense subgraphs, thus inevitably impairs the detection quality. The scalability of both DS [29] and SEA [19] is limited due to the same reason, since they need the complete affinity matrix $A$ as well.

# 4. THE ALID APPROACH

In this section, we introduce our ALID method. The major idea of ALID is to confine the computation of infection and immunization in small local ranges within the Region of Interest (ROI), so that only small submatrices of the affinity matrix need to be computed. As a result, ALID largely avoids the affinity matrix computation and significantly reduces both the time and space complexities.

The framework of ALID is an iterative execution of the following three steps, as summarized in Figure 2.

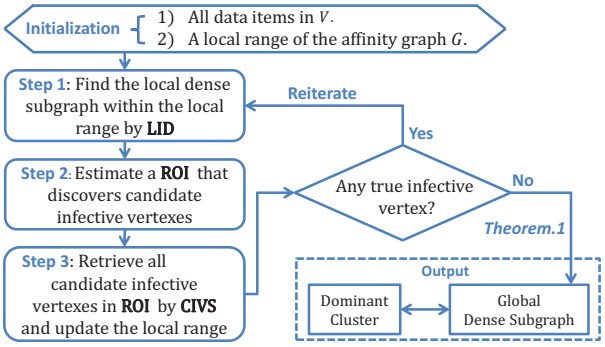



**Figure 2: An overview of the ALID framework.**

**Step 1** *Finding local dense subgraph by Localized Infection Immunization Dynamics (LID).* LID confines all infection immunization iterations in a small local range to find the local dense subgraph. It avoids computing the full affinity matrix by selectively computing a few columns of the local affinity matrix.

**Step 2** *Estimating a Region of Interest (ROI).* The local dense subgraph found in Step 1 may or may not be a global dense subgraph, since there may still be global infective vertices that are not covered by the current local range, as indicated by Theorem 1. Therefore, we estimate a ROI to identify the candidate infective vertices in the global range.

**Step 3** *Candidate Infective Vertex Search (CIVS).* CIVS efficiently retrieves the candidate infective vertices within a ROI and uses them to update the local range towards the global dense subgraph.

The iterative process of ALID terminates when the local dense subgraph found in the last round of iteration is immune against all vertices in the global range. According to Theorem 1, such a subgraph is a global dense subgraph that identifies a true dominant cluster. We explain the details of the three steps in the first three subsections as follow.

## 4.1 Localized Infection Immunization Dynamics (Step 1)

The key idea of Localized Infection Immunization Dynamics (LID) is that a dense subgraph on a small local range of the affinity graph can be detected by selectively computing only a few columns of the corresponding local affinity submatrix. Denote by $\beta \subset I$ the local range of the affinity graph, which is the index set of a local group of graph vertices. LID finds the local dense subgraph $\hat{x}$ that maximizes $\pi(x)$ in the local range $\beta$ by localizing the infection immunization process on the selectively computed submatrix $A_{\beta\alpha}$ (see Figure 3). Here, $\alpha \triangleq \{i \in \beta \mid x_i > 0\}$ is the *support* of the subgraph $x \in \triangle_\beta^n$, where $\triangle_\beta^n = \{x \in \triangle^n \mid \sum_{i \in \beta} x_i = 1\}$ represents the set of all possible subgraphs within local range $\beta$. In this way, the computation of the full matrix $A_{\beta\beta}$ can be effectively avoided. Consequently, LID is more efficient than directly running IID in the local range.

Algorithm 1 shows the steps in a LID iteration, which takes $\left[x^{(t)}, (A_{\beta\alpha}x_\alpha)^{(t)}\right]$ as input and obtains the output $\left[x^{(t+1)}, (A_{\beta\alpha}x_\alpha)^{(t+1)}\right]$ in linear time and space with respect

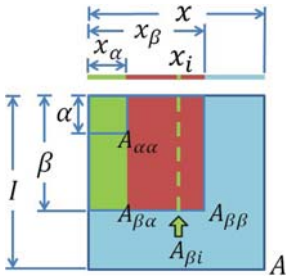



**Figure 3: The affinity matrix with ordered data item indexes. $A$ is the global affinity matrix, $A_{\beta\beta}$ is the affinity matrix in the local range $\beta$, and $A_{\beta\alpha}$ is the group of columns in $A_{\beta\beta}$ corresponding to the support $\alpha$. The dashed line denotes the new column $A_{\beta i}$ : $i \in (\beta - \alpha)$. Only the green parts ($A_{\beta\alpha}$ and $A_{\beta i}$) are involved in the LID iteration.**

to the size of $\beta$. The superscripts $(t)$ and $(t+1)$ indicate the number of iterations. For the interest of simplicity, we omit $(t)$ and illustrate the details of Algorithm 1 as follows.

First, we use the function $S(x)$ in Equation 8 to select the infective vertex or subgraph (i.e., *co-vertex* in Equation 7) in the local range $\beta$. Since $x \in \triangle_\beta^n$, the component $\pi(s_i - x, x)$ can be computed as

$$\begin{aligned}\pi(s_i - x, x) &= (s_i - x_\beta)^\top A_{\beta\beta} x_\beta \\ &= (A_{\beta\alpha}x_\alpha)_i - \sum_{i\in\alpha}(A_{\beta\alpha}x_\alpha)_i x_i\end{aligned} \tag{10}$$

where only the graph vertices $s_i \in \triangle_\beta^n$ are considered.

Second, we compute the invasion share $\varepsilon_y(x)$ by Equation 9, whose value depends on $y = S(x)$ in two cases:

**Case 1** : *Infection* $(y = S(x) = s_i)$. The key components $\pi(y - x, x)$ and $\pi(y - x)$ in Equation 9 are computed by Equations 10 and 11, respectively.

$$\begin{aligned}\pi(s_i - x) &= (s_i - x_\beta)^\top A_{\beta\beta}(s_i - x_\beta) \\ &= -2(A_{\beta\alpha}x_\alpha)_i + \sum_{i\in\alpha}(A_{\beta\alpha}x_\alpha)_i x_i\end{aligned} \tag{11}$$

**Case 2** : *Immunization* $(y = S(x) = \overline{s_i(x)})$. $\varepsilon_y(x)$ is computed by plugging Equation 12 into Equation 9.

$$\begin{aligned}\pi(\overline{s_i(x)} - x, x) &= \tfrac{x_i}{x_i - 1}\pi(s_i - x, x) \\ \pi(\overline{s_i(x)} - x) &= (\tfrac{x_i}{x_i - 1})^2\pi(s_i - x)\end{aligned} \tag{12}$$

Last, the new subgraph $x^{(t+1)}$ is obtained by Equation 13 and $(A_{\beta\alpha}x_\alpha)^{(t+1)}$ is computed by Equation 14 in linear time and space for the next iteration.

$$x^{(t+1)} = (1 - \varepsilon_y(x))x + \varepsilon_y(x)S(x) \tag{13}$$

$$(A_{\beta\alpha}x_\alpha)^{(t+1)} = (Ax^{(t+1)})_\beta =$$
$$\begin{cases} \varepsilon_y(x)[A_{\beta i} - A_{\beta\alpha}x_\alpha] + A_{\beta\alpha}x_\alpha & \text{if } y{=}s_i \\ (\tfrac{x_i}{x_i - 1})\varepsilon_y(x)[A_{\beta i} - A_{\beta\alpha}x_\alpha] + A_{\beta\alpha}x_\alpha & \text{if } y{=}\overline{s_i(x)} \end{cases} \tag{14}$$

Each LID iteration in Algorithm 1 is guaranteed by Theorem 2 to shrink the size of the local infective subgraph set $\gamma_\beta(x) = \{y \in \triangle_\beta^n \mid \pi(y - x, x) > 0\}$. Thus, we obtain the local dense subgraph $\hat{x} \in \triangle_\beta^n$ in the local range $\beta$ by repeating Algorithm 1 to shrink $\gamma_\beta(x)$ until $\gamma_\beta(x) = \emptyset$. According to Theorem 1, $\gamma_\beta(\hat{x}) = \emptyset$ indicates that $\hat{x}$ is immune against all vertices $s_i \in \triangle_\beta^n$. Thus, $\hat{x}$ is a local dense subgraph with local maximum $\pi(\hat{x})$ in the local range $\beta$. In practice, we stop the LID iteration when $\pi(x)$ is stable or the total number of iterations exceeds an upper limit $T$. Let $b = |\beta|$ be the size of $\beta$. The time and space complexities of LID method are $\mathcal{O}(Tb)$ and $\mathcal{O}(b)$, respectively. We will discuss the cost of initializing $[x, A_{\beta\alpha}x_\alpha]$ in Section 4.3.

**Algorithm 1:** A single period of LID iteration

**Input**: $x^{(t)}$, $(A_{\beta\alpha}x_\alpha)^{(t)}$
**Output**: $x^{(t+1)}$, $(A_{\beta\alpha}x_\alpha)^{(t+1)}$

1: Select the infective vertex $y = S(x)$ by Equation 8
2: Calculate the invasion share $\varepsilon_y(x)$ by Equation 9
3: Update $x^{(t+1)}$ by the invasion model of Equation 13
4: Update $(A_{\beta\alpha}x_\alpha)^{(t+1)}$ by Equation 14
5: **return** $x^{(t+1)}$, $(A_{\beta\alpha}x_\alpha)^{(t+1)}$

All ALID iterations are restricted on the dynamically computed submatrix $A_{\beta\alpha}$, where the new matrix column $A_{\beta i}$ only needs to be computed and stored when $i \in (\beta - \alpha)$ (see Figure 3). We will discuss how LID effectively reduces the original $\mathcal{O}(n^2)$ time and space complexity of affinity matrix computation in Section 4.5.

## 4.2 Estimating ROI (Step 2)

A local dense subgraph $\hat{x} \in \triangle_\beta^n$ may not be a global dense subgraph in $\triangle^n$, since $\beta$ may not fully cover the true dense subgraph in the global range $I$. Therefore, there may still be graph vertices in the complementary range $U = I - \beta$, which are infective against the local dense subgraph $\hat{x}$. Thus, the current local range $\beta$ should be updated to include such infective vertices in $U$, so that the true dense subgraph with maximum graph density can be detected by LID.

A natural way to find the global dense subgraph $x^*$ is to keep invading $\hat{x}$ using the infective vertices in $U$ until no infective vertex exists in the global range $I$. However, fully scanning $U$ for infective vertices leads to an overall time complexity of $\mathcal{O}(n^2)$ in detecting all dominant clusters, since $U$ contains all the remaining vertices in $I$ with an overwhelming proportion of irrelevant vertices. To tackle this problem, we estimate a Region of Interest (ROI) from $\hat{x}$ to include all the infective vertices and exclude most of the irrelevant ones. Only a limited amount of vertices inside the ROI are used to update $\beta$. This strategy can largely reduce the amount of vertices to be considered and effectively reduces the time and space complexities of ALID.

Before estimating the ROI, we first construct a **double-deck hyperball** $H(D, R_{in}, R_{out})$ from $\hat{x}$, where $D \in R^d$ is the ball center and $R_{in}, R_{out}$ are the radiuses of the inner and outer balls, respectively, which are defined as follows.

$$\begin{aligned} &D = \sum_{i\in\alpha} v_i\hat{x}_i, \ \textit{where } v_i \in V \textit{ are the data items.} \\ &R_{in} = \tfrac{1}{k}\ln(\tfrac{\lambda_{in}}{\pi(\hat{x})}), \ \textit{where } \lambda_{in} = \sum_{i\in\alpha}\hat{x}_i e^{-k||v_i - D||_p} \\ &R_{out} = \tfrac{1}{k}\ln(\tfrac{\lambda_{out}}{\pi(\hat{x})}), \ \textit{where } \lambda_{out} = \sum_{i\in\alpha}\hat{x}_i e^{k||v_i - D||_p} \end{aligned} \tag{15}$$

where $k$ is the positive scaling factor in Equation 1. Proposition 1 discloses two important properties of the double-deck hyperball $H(D, R_{in}, R_{out})$.

PROPOSITION 1. *Given a local dense subgraph $\hat{x}$, the double-deck hyperball $H(D, R_{in}, R_{out})$ has the following properties:*

1. $\forall j \in I$ *and* $||v_j - D||_p < R_{in}$, $\pi(s_j - \hat{x}, \hat{x}) > 0$; *and*
2. $\forall j \in I$ *and* $||v_j - D||_p > R_{out}$, $\pi(s_j - \hat{x}, \hat{x}) < 0$.

Again, all mathematical proofs can be found in Appendix of the full version [11].

According to the two properties in Proposition 1, the surfaces of the inner ball $H(D, R_{in})$ and the outer ball $H(D, R_{out})$ are two boundaries, which guarantee that every data entry inside the inner ball corresponds to an infective vertex and the ones outside the outer ball are non-infective.

The **ROI** is defined as a growing hyperball $H_c(D, R)$, whose surface starts from the inner ball and gradually approaches the outer ball as the ALID iteration continues. The radius $R$ is defined as

$$R = R_{in} + \theta(c)(R_{out} - R_{in}) \tag{16}$$

where $\theta(c) = \frac{1}{1+e^{(4-c/2)}}$ is a shifted logistic function to control the growing speed and $c$ is the number of the current ALID iteration. When $c$ grows large, we have $\theta(c) \approx 1$ and $R \approx R_{out}$. Thus, the ROI is guaranteed to coincide with the outer ball as $c$ grows. Since the outer ball $H(D, R_{out})$ is guaranteed by Proposition 1 to contain all infective vertices in the global range $I$, the finally found local dense subgraph $\hat{x}$ within the ROI is guaranteed to be immune against all vertices in $I$, thus $\hat{x}$ is a global dense subgraph according to Theorem 1. Moreover, starting the ROI from the small inner ball can effectively reduce the number of vertices to be scanned in the first several ALID iterations.

## 4.3 Candidate Infective Vertex Search (Step 3)

The hyperball of ROI identifies a small local region in the $d$-dimensional space $R^d$. The data items $v_i \in V$ inside the ROI correspond to the candidate graph vertices, which are possibly infective against the current local dense subgraph $\hat{x}$ and may further increase the graph density $\pi(\hat{x})$. Therefore, we carefully design the Candidate Infective Vertex Searching (CIVS) method to efficiently retrieve such data items inside the ROI and use them to update the local range $\beta^{(c)}$. The variable $c$ is the current number of ALID iteration.

Retrieving the data items inside a ROI $H_c(D, R)$ is equivalent to a *fixed-radius near neighbor problem* in the $d$-dimensional space $R^d$. For the interest of scalability, we adopt the Locality Sensitive Hashing (LSH) method [13] to solve this problem. However, LSH can only retrieve data items within a small Locality Sensitive Region (LSR) of a query (Figure 4). Thus, when using the hyperball center $D$ as a single LSH query (Figure 4(a)), the corresponding LSR may fail to find all data items within the ROI, which would prevent ALID from converging to the optimal result.

To solve this problem, we propose CIVS. As shown in Figure 4(b), CIVS applies multiple LSH queries using all the supporting data items of $\hat{x}^{(c)}$ (i.e., $\{v_i \in V \mid \hat{x}_i^{(c)} > 0\}$). The benefit of CIVS is that multiple LSRs effectively cover most of a ROI, thus most of the data items within the ROI can be retrieved. Specifically, CIVS first collects all the new data items that is retrieved by any of the supporting data items of $\hat{x}^{(c)}$ using LSH. Then, we retrieve at most $\delta$ new data items within the ROI that are the nearest to the ball center

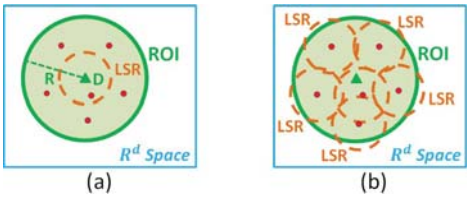



**Figure 4: Each dashed circle is a locality sensitive region (LSR). The red points show the supporting data items of the current local dense subgraph $\hat{x}^{(c)}$. (a) A single LSR cannot cover a ROI. (b) CIVS covers most of a ROI by multiple LSRs.**

$D$. The index set of the retrieved data items is denoted by $\psi = \{i \mid i \in (I - \alpha^{(c)}), v_i \in H_c(D, R)\}, |\psi| \leq \delta$.

The retrieved data $\{v_i \mid i \in \psi\}$ are used to perform update: $\left[\hat{x}^{(c)}, (A_{\beta\alpha}\hat{x}_\alpha)^{(c)}\right] \rightarrow \left[x^{(c+1)}, (A_{\beta\alpha}x_\alpha)^{(c+1)}\right]$ as follow.

$$\begin{aligned} x^{(c+1)} &= \hat{x}^{(c)} \\ (A_{\beta\alpha}x_\alpha)^{(c+1)} &= \begin{bmatrix} (A_{\alpha\alpha}\hat{x}_\alpha)^{(c)} \\ (A_{\psi\alpha}\hat{x}_\alpha)^{(c)} \end{bmatrix} \end{aligned} \tag{17}$$

Equation 17 updates the local range by $\beta^{(c+1)}{=}\alpha^{(c)} \cup \psi$, where $\psi$ involves new infective vertices against $x^{(c+1)}$. Then, we can re-run LID (i.e., Step 1) with the initialization of $[x^{(c+1)}, (A_{\beta\alpha}x_\alpha)^{(c+1)}]$ to find the local dense subgraph $\hat{x}^{(c+1)}$ in the new range $\beta^{(c+1)}$. Since $\hat{x}^{(c+1)}$ is guaranteed by Theorem 1 to be immune against all vertices in $\psi \subset U$, the number of infective vertices in $I$ is further reduced.

Theoretically, CIVS does not cover a ROI completely. However, by iteratively using every support data item as a LSH query, ALID can retrieve new candidate infective vertices progressively. Since the number of infective vertices is finite and will be monotonically reduced to zero through a series of iterations, the final result of ALID will be optimal. The convergence of ALID is proved in Appendix in the full version of this paper [11].

The time and space complexities for building the hash tables are linear with respect to $n$, the size of the data set. Specifically, the time complexity to build $l$ hash tables by $\mu$ hash functions is $\mathcal{O}(ndl\mu)$. The space complexity consists of $\mathcal{O}(nd)$ space for all the $d$ dimensional data items, $\mathcal{O}(nl)$ space for an inverted list that maps each data item to their buckets and $\mathcal{O}(nl)$ space for $l$ hash tables [13]. Since all possible LSH queries are built into the hash tables, we check the inverted list to retrieve neighbor data items and do not store the hash keys.

## 4.4 Summary of ALID

The entire iteration of ALID is summarized in Algorithm 2. The LID in Step 1 makes the local dense subgraph immune against all vertices within a local range of the ROI. The ROI and CIVS in Steps 2 and 3 update the local range by the new infective vertices retrieved from global range. In this way, the number of infective vertices in global range is guaranteed to be iteratively reduced to zero. Then, according to Theorem 1, the last found local dense subgraph is a global one that identifies a dominant cluster [38]. Algorithm 2 stops when a global dense subgraph is found or the total number of iterations exceeds an upper limit $C$. Since Algorithm 2 is initialized with $A_{\beta\alpha}x_\alpha = 0$, which

**Algorithm 2:** The entire ALID iteration

**Input**: An initial vertex index $i \in I$
**Output**: A global dense subgraph $x^*$ in global range $I$

1: **Set** $\alpha = \beta = i$, $x = s_i$, $A_{\beta\alpha}x_\alpha = a_{ii} = 0$, $c = 1$
2: **repeat**
3: **Step 1**: $\left[x^{(c)}, (A_{\beta\alpha}x_\alpha)^{(c)}\right] \rightarrow \left[\hat{x}^{(c)}, (A_{\beta\alpha}\hat{x}_\alpha)^{(c)}\right]$
Find the local dense subgraph $\hat{x}^{(c)}$ by LID in Step 1
4: **Step 2**: $\hat{x}^{(c)} \rightarrow H_c(D, R)$
Estimate ROI from the local dense subgraph $\hat{x}^{(c)}$
5: **Step 3**: $\left[\hat{x}^{(c)}, (A_{\beta\alpha}\hat{x}_\alpha)^{(c)}\right] \rightarrow \left[x^{(c+1)}, (A_{\beta\alpha}x_\alpha)^{(c+1)}\right]$
Apply CIVS to retrieve candidate vertices within the ROI and update the local range $\beta^{(c)}$ by Equation 17 for the next iteration
6: Index update: $c \leftarrow c + 1$
7: **until** $\hat{x}^{(c)}$ is a global dense subgraph, or $c > C$
8: **return** $x^* = \hat{x}^{(c)}$

cannot be used to compute the radius of ROI $H_{c=1}(D, R)$ (Equations 15 and 16), we empirically set $R = 0.4$ for the first iteration $c = 1$. Limited by space, we omit a detailed analysis of the effect of $R$ on the performance.

In order to fairly compare with the other affinity-based methods, ALID adopts the same *peeling method* as what DS [29] and IID [31] do to detect all dominant clusters. The *peeling method* peels off the detected cluster and reiterates on the remaining data items to find another cluster until all data items are peeled off. Then, the clusters with large values of $\pi(x)$ (e.g., $\pi(x) \geq 0.75$) can be selected as the final result.

## 4.5 Complexity Analysis

The time and space complexities of ALID mainly consist of three parts:

- The time and space complexities of LID in Step 1 are $\mathcal{O}(Tb)$ and $\mathcal{O}(b)$, respectively, where $b = |\beta| < n$ is the size of the local range $\beta$ and $T$ is a constant limit of the number of LID iterations.
- The time and space complexities for the hash tables of CIVS are $\mathcal{O}(ndl\mu)$ and $\mathcal{O}(n(2l + d))$, respectively, where $d, l, \mu$ are constant LSH parameters.
- The time and space complexities for the affinity matrix $A$ are $\mathcal{O}(C(a^* + \delta)n)$ and $\mathcal{O}(a^*(a^* + \delta))$, respectively, which are analyzed in detail as follows.

Since all ALID iterations are restricted by $A_{\beta\alpha}$, the time and space complexities for the affinity matrix are determined by the size of $A_{\beta\alpha}$. Let $(A_{\beta\alpha})_i^c$ be the submatrix computed in the $c$-th iteration in Algorithm 2 when detecting the $i$-th cluster. Denote by $a_i^c$ and $b_i^c$, respectively, the column and row sizes of $(A_{\beta\alpha})_i^c$. Since the maximum number of iterations of Algorithm 2 is $C$, the overall time cost for detecting the $i$-th cluster is $Time(i) < \sum_{c=1}^{C} a_i^c b_i^c$, which is a *loose upper bound*, since many matrix entries of different $(A_{\beta\alpha})_i^c$ are duplicate and are only computed once. Then, we can derive

$$Time(i) < Ca_i^C(a_i^C + \delta) \tag{18}$$

from the following observations. First, $a_i^c \leq a_i^C$, since more and more matrix columns (i.e., $A_{\beta i} : i \in (\beta - \alpha)$ in Figure 3) are computed. Second, $b_i^c \leq (a_i^c + \delta) \leq (a_i^C + \delta)$, since the size of $\beta$ is strictly limited by the ROI, where at most $\delta$ data items can be retrieved by CIVS.

**Table 1: The complexity of the affinity matrix**

| Typical Cases | Time Complexity | Space Complexity |
|---|---|---|
| $a^* = \omega n$ $(\omega \leq 1)$ | $\mathcal{O}(C(\omega n^2 + \delta n))$ | $\mathcal{O}(\omega^2 n^2 + \delta\omega n)$ |
| $a^* = n^\eta$ $(\eta < 1)$ | $\mathcal{O}(C(n^{1+\eta} + \delta n))$ | $\mathcal{O}(n^{2\eta} + \delta n^\eta)$ |
| $a^* \leq P$ | $\mathcal{O}(C((P + \delta)n))$ | $\mathcal{O}(P^2 + \delta P)$ |

Since $Time(i)$ is the time cost of affinity matrix computation for detecting the $i$-th dominant cluster, the overall cost in time of detecting all dominant clusters is $\sum_i Time(i)$. We can derive the following from Equation 18.

$$\sum_i Time(i) < \sum_i Ca_i^C(a_i^C + \delta) \tag{19}$$

Recall that ALID adopts the *peeling method* (see Section 4.4), which peels off one detected cluster and reiterates on the remaining data items to find another cluster until all the $n$ data items are peeled off. We have

$$\sum_i a_i^C = n \tag{20}$$

from the fact that $a_i^C$ is the size of the $i$-th detected cluster.

Define $a^* = \max_i\{a_i^C\}$ and $b^* = \max_i\{b_i^C\}$. Due to the restriction of ROI, $b^* \leq (a^* + \delta)$. Then, we can derive from Equations 19 and 20 that the overall cost in time of computing the affinity matrix is

$$\sum_i Time(i) < \sum_i Ca_i^C(a^* + \delta) = C(a^* + \delta)n \tag{21}$$

The maximum cost in space is $a^*b^* \leq a^*(a^* + \delta)$, since all submatrices $(A_{\beta\alpha})_i^c$ are released when the $i$-th cluster is peeled off. As a result, the time and space complexities for the affinity matrix of ALID are $\mathcal{O}(C(a^*+\delta)n)$ and $\mathcal{O}(a^*(a^*+\delta))$, respectively.

Recall that $a^* = \max_i\{a_i^C\}$ is the size of the largest (single) dominant cluster. We summarize in Table 1 the three typical cases how $a^*$ affects the time and space complexities of the affinity matrix. The data items belonging to the largest dominant cluster of size $a^*$ are referred to as "positive data", data items that do not belong to any dominant cluster are regarded as "noise data" and the size of the entire data set is denoted by $n$.

First, for data sets where the amount of positive data is proportional to the entire data set, that is, $a^* = \omega n$ and $\omega \leq 1$ is the constant proportion, ALID reduces the original $\mathcal{O}(n^2)$ time and space complexities of the affinity matrix to $\mathcal{O}(C(\omega n^2 + \delta n))$ and $\mathcal{O}(\omega^2 n^2 + \delta\omega n)$, respectively.

Second, for noisy data sets (e.g., tweet-streams and user comments) that generate noise data faster than positive data, the growth rate of $a^*$ is slower than $n$, that is, $a^* = n^\eta$ $(\eta < 1)$. ALID reduces the $\mathcal{O}(n^2)$ time and space complexities to $\mathcal{O}(C(n^{1+\eta} + \delta n))$ and $\mathcal{O}(n^{2\eta} + \delta n^\eta)$, respectively.

Third, for noisy data sets containing dominant clusters of limited-sizes, there is a constant upper bound $P$ for $a^*$ (i.e., $a^* \leq P$). For example, in phone books and email contacts, the largest number of people in a stable social group (i.e., dominant cluster) is limited by the *Dunbar's number* [15]. In such a case, we have $a^* \leq P$. The time

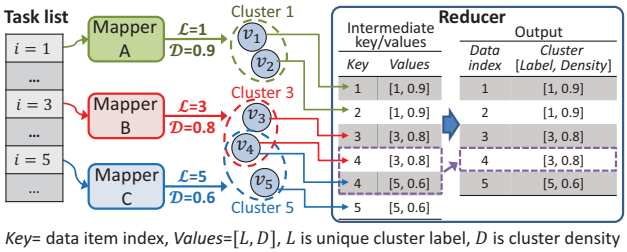



**Figure 5: An illustrative example of Algorithm 3. Each mapper runs Algorithm 2 independently with a different initial vertex.**

and space complexities of the affinity matrix are reduced to $\mathcal{O}(C(P+\delta)n)$ and $\mathcal{O}(P^2+\delta P)$, respectively. Note that, since the infection immunization process converges quickly in finite steps [30], a small value of $C$, such as 10, is adequate for the convergence of ALID.

## 4.6 Parallel ALID

ALID is suitable for parallelization in the MapReduce framework [14], since multiple tasks of ALID can be concurrently run in independent local ranges of the affinity graph. We introduce the parallel ALID (PALID) in Algorithm 3 and provide an illustrative example in Figure 5.

As shown in Figure 5, three initial graph vertex indexes $i = \{1, 3, 5\}$ are assigned to three Mappers (A,B,C). Each Mapper runs Algorithm 2 to detect a cluster independently. Once a Mapper detects a cluster, it produces a list of intermediate key/value pairs $(Key, Values{=}[\mathcal{L}, \mathcal{D}])$, where $Key$ is the index of one data item belonging to the cluster, $\mathcal{L}$ is the unique cluster label of the detected cluster and $\mathcal{D}$ is the density of the cluster. In the case of overlapping clusters, such as clusters 3 and 5 in the figure, we simply assign the overlapped data item $v_4$ to cluster 3 of maximum density, which can be easily handled by a reducer.

Since data items belonging to the same dominant cluster are highly similar to each other, such data items are likely to be mapped to the same set of LSH buckets. Therefore, buckets of large size reveal the potential data items of dominant clusters. In practice, PALID uniformly samples the initial graph vertices from every LSH hash bucket that contains more than 5 data items. The sample rate is set to 20% in our experiments.

In summary, ALID is highly parallelizeable in the MapReduce framework, which further improves its capability in handling massive data in real world applications.

**Algorithm 3:** The parallel ALID (PALID)

**Input**: $V = \{v_i \mid i \in I = [1, n]\}$
**Output**: Cluster labels and cluster densities of each data item in the detected clusters
**Tasklist**: A list of initial graph vertex indexes
**Map**($Key$, $Value$)
\\ $Key$: An initial vertex index $i$ for Algorithm 2
\\ $Value$: A unique cluster label $\mathcal{L}$ for the detected cluster
1: Call Algorithm 2 to find a global dense subgraph $x^*$, which identifies dominant cluster $\mathcal{L}$ with density $\mathcal{D} = \pi(x^*)$
2: **for all** data item index $h$ in $I = [1, n]$ **do**
3: **if** $x^*_h > 0$ **then**
4: Emit($h$, $[\mathcal{L}, \mathcal{D}]$) \\ $v_h$ belongs to cluster $\mathcal{L}$
5: **end if**
6: **end for**
**Reduce**($Key$, $Values$)
\\ $Key$: The index $h$ of a single data item $v_h \in V$
\\ $Values$: A list of $[\mathcal{L}, \mathcal{D}]$ w.r.t. the clusters containing $v_h$
1: Find $[\mathcal{L}^*, \mathcal{D}^*]$ in $Values$ with maximum density $\mathcal{D}^*$
2: Emit($h$, $[\mathcal{L}^*, \mathcal{D}^*]$) \\ Assign $v_h$ to cluster $\mathcal{L}^*$

# 5. EXPERIMENTAL RESULTS

We empirically examined and analyzed the performance of ALID and PALID, and compared with the following state-of-the-art affinity-based methods: 1) Affinity Propagation (AP) [17]; 2) the Shrinking Expansion Algorithm (SEA) [19]; and 3) Infection Immunization Dynamics (IID) [31]. We used the published source codes of AP [1] and SEA [2]. Since the code for IID is unavailable, we implemented it in *MATLAB*. All compared methods are carefully tuned. We report their best performance in this section. The parameter $\delta$ in Step 3 of ALID (PALID) was set to 800.

The detection quality was evaluated by the Average $F_1$ score (AVG-F), which is the same criterion as what Chen *et al.* [8] used. AVG-F was obtained by averaging the $F_1$ scores on all the true dominant clusters. Besides, as Chen *et al.* [8] showed, since the data items are partially clustered in this task, traditional evaluation criteria, such as entropy and normalized mutual information, are not appropriate in evaluating the detection quality.

The detection efficiency was measured from two perspectives, the runtime of each method including the time to compute the affinity matrix and the memory overhead including the memory storing the affinity matrix.

We used a PC computer with a Core i-5 CPU, 12 GB main memory and a 7200 RPM hard drive, running Windows 7 operating system. All experiments using single computer were conducted using *MATLAB*, since the compared methods were implemented on the same platform. We will explicitly illustrate the parallel experiments of PALID in Section 5.3.

The following data sets were used: 1) the news articles data set (NART); 2) the near duplicate image data set (NDI); 3) three synthetic data sets; 4) the SIFT-50M data set that consists of 50 million SIFT features [23]. For all data sets, the pairwise distance and affinity were calculated using Euclidean distance and Equation 1 ($p = 2$), respectively.

*The news articles data set* (NART) was built by crawling 5,301 news articles from `news.sina.com.cn`. It contains 13 real world "hot" events happened from May to June 2012, each of which corresponds to a dominant cluster of news articles. All 734 news articles of the 13 dominant clusters were manually labeled as ground truth by 3 volunteers without professional background. The remaining 4,567 articles are daily news that do not form any dominant cluster. Each article is represented by a normalized 350-dimensional vector generated by standard Latent Dirichlet Allocation (LDA) [6].

*The near duplicate image data set* (NDI) contains 109,815 images crawled from `images.google.com.hk`. It includes a labeled set of ground truth of 57 dominant clusters and

11,951 near duplicate images, where images with similar contents are grouped as one dominant cluster. The remaining 97,864 images with diverse contents are regarded as background noise data. Each image is represented by a 256-dimensional GIST feature [27] that describes the global texture of the image content.

Details of the three synthetic data sets and the SIFT-50M data set will be described in Sections 5.2 and 5.3, respectively.

## 5.1 Influence of Sparsity

As mentioned in Section 2, the scalability of canonical affinity-based methods (i.e., IID, SEA, AP) is constrained by the $\mathcal{O}(n^2)$ time and space complexity in fully computing and storing the affinity matrix. Although the computational efficiency can be improved by sparsifying the affinity matrix [9], the enforced sparsity breaks the high cohesiveness of dense subgraphs, which inevitably weakens the noise resistance capability and impairs the detection quality.

In this section, we specifically analyze how the sparse degree of sparsified affinity matrix affects the detection quality and runtime of all compared methods. The sparse degree is defined as the ratio of the number of entries in the matrix taking value 0 over the total number of entries in the matrix.

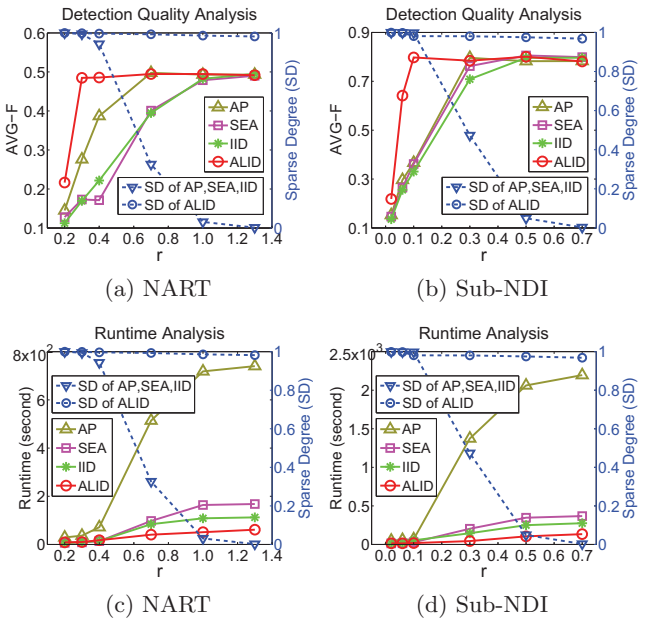
All experiment results were obtained on the NART and Sub-NDI data sets. Sub-NDI is a subset of the NDI data set containing 6 clusters of 1420 ground truth images and 8520 noise images. We used Sub-NDI instead of NDI, since AP cannot deal with the entire NDI data set with 12GB RAM.

Chen *et al.* [9] provided two approaches to sparsify the affinity matrix: the exact nearest neighbors (ENN) method and the approximate nearest neighbors (ANN) method. The ENN method is expensive on large data sets, while the ANN method is efficient by employing LSH [13]. Thus, we sparsify the affinity matrix by LSH due to its efficiency.

For AP, IID and SEA, we directly applied LSH to sparsify the affinity matrix, where only the affinities between the nearest neighbors were computed and stored. The same LSH module is utilized by CIVS in ALID. To remove possible uncertainty caused by the LSH approximation, the parameter settings of LSH were kept exactly the same for all the compared methods, including ALID.

Standard LSH projects each data item onto an equally segmented real line. The line segment length $r$ controls the recall of LSH, and thus affects the sparse degree of the affinity matrix. Figure 6 shows that the sparse degree of all affinity-based methods, including ALID, decreases when $r$ increases. However, the sparse degree of ALID remains high, since ALID only computes small submatrices corresponding to the vertices within the ROI.

Figures 6(a) and 6(c) show the experiment results on data set NART. The AVG-F of all methods increases to a stable level as the sparse degree decreases. This is because the cohesiveness of dense subgraphs are better retained as the sparse degree decreases. For AP, SEA and IID, when the sparse degree approaches zero, the original cohesiveness of subgraphs is maximally preserved by a full affinity matrix, and thus the methods all approach their best performance. Since most dense subgraphs exist in small local ranges, the relative local submatrices are good enough to retain their cohesiveness. ALID largely preserves such cohesiveness by accurately estimating the local range of true dense subgraphs and fully computing the relative local affinity matrices.




(a) NART (b) Sub-NDI

(c) NART (d) Sub-NDI

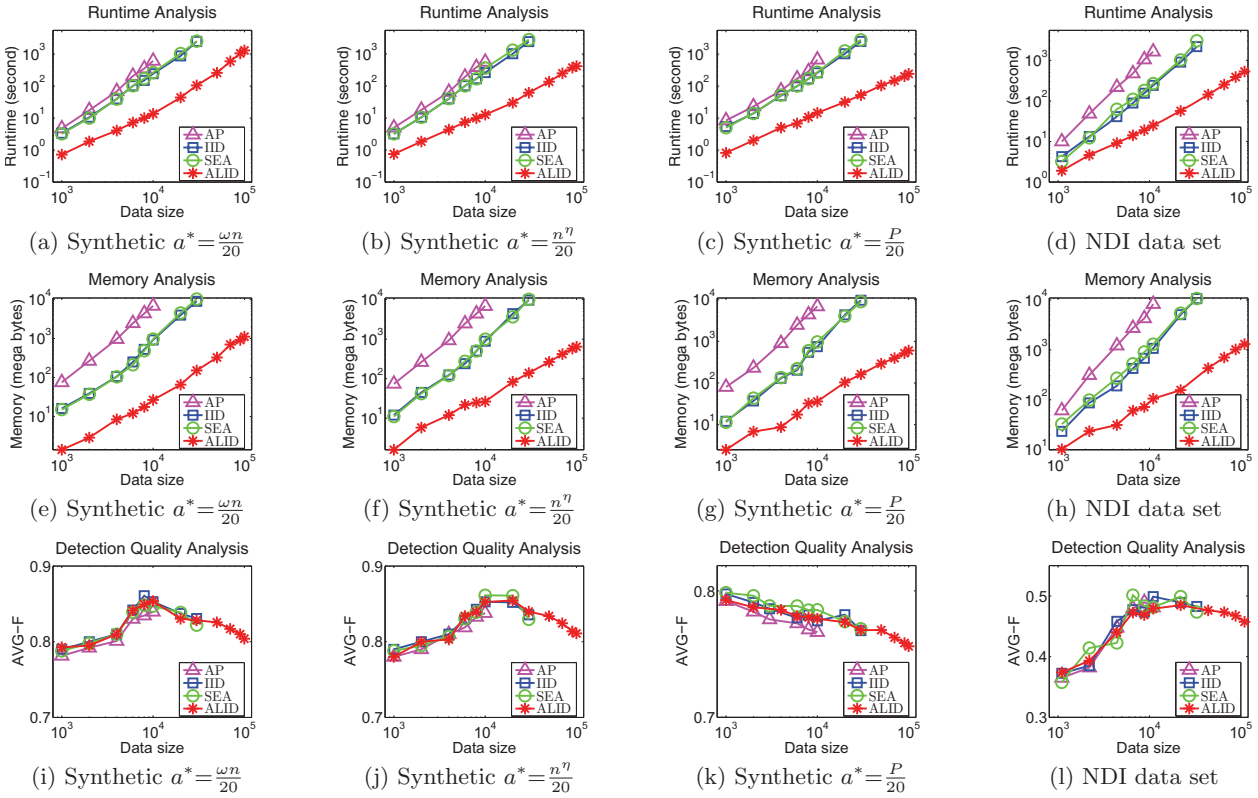
**Figure 6: The results on NART and Sub-NDI. (a)-(b) show how sparse degree affects AVG-F. (c)-(d) show how sparse degree affects runtime. For LSH, we used 40 projections per hash value and 50 hash tables. $r$ is the length of the equally divided segments of LSH.**

Consequently, ALID achieves a good AVG-F performance under an extremely high sparse degree of 0.998 ($r = 0.3$), which indicates that the computation and storage of 99.8% of the matrix entries are effectively pruned. Such a situation is understandable, since the useful matrix entries that correspond to the 13 true clusters of 734 data items in data set NART only counts for $734^2/(13 \times 5301^2) = 0.147\%$ of the entire affinity matrix.

The results in Figure 6(c) demonstrate that sparsifying the affinity matrix reduces the runtime of the affinity-based methods. The runtime of all methods are comparably low when the sparse degree is high. However, when the sparse degree decreases, the differences in runtime among the methods become significant. When $r = 1.3$, AP is significantly slower than the other methods due to its expensive message passing cost. Moreover, SEA is much slower than IID due to the time consuming replicator dynamics [38]. ALID is the fastest, since it effectively prunes the computation of 99.8% of the affinity matrix entries. Similar results were also observed on data set Sub-NDI (Figures 6(b) and 6(d)).

## 5.2 Scalability

In this section, we analyze the scalability of the affinity-based methods on data set NDI and three synthetic data sets. The *synthetic data sets* were made up by sampling $n$ 100-dimensional data items from 20 different multivariate Gaussian distributions as the dominant clusters and one uniform distribution as the background noise. To better simulate typical properties of real world data, we made some Gaussian distributions partially overlapped by setting their mean vectors close to each other and variating the




(a) Synthetic $a^*=\frac{\omega n}{20}$ (b) Synthetic $a^*=\frac{n^\eta}{20}$ (c) Synthetic $a^*=\frac{P}{20}$ (d) NDI data set

(e) Synthetic $a^*=\frac{\omega n}{20}$ (f) Synthetic $a^*=\frac{n^\eta}{20}$ (g) Synthetic $a^*=\frac{P}{20}$ (h) NDI data set

(i) Synthetic $a^*=\frac{\omega n}{20}$ (j) Synthetic $a^*=\frac{n^\eta}{20}$ (k) Synthetic $a^*=\frac{P}{20}$ (l) NDI data set

**Figure 7: Scalability on synthetic data sets with 3 types of dominant clusters and the real world data set NDI. Parameters for the 3 synthetic data sets are: $\omega = 1.0$, $\eta = 0.9$ and $P = 1000$**

shapes of the Gaussian distributions by different diagonal covariance matrices with elements ranging in $[0, 10]$. Then, we sampled $a^*$ data items from each Gaussian distribution as the ground truth and $(n - 20a^*)$ data items from the surrounding uniform distribution as noise. Since all the 20 clusters were sampled in equal size, $a^*$ is the largest size of dominant clusters, which is consistent with the definition of $a^*$ in Section 4.5. The synthetic data sets were mainly used to test the efficiency and scalability of ALID. Thus, we simulated the three typical cases of $a^*$ analyzed in Table 1 by controlling the amount of sampled data items with $a^*=\frac{\omega n}{20}$, $a^*=\frac{n^\eta}{20}$ and $a^*=\frac{P}{20}$, respectively, where the constant denominator 20 does not affect the complexity of ALID. We used different values of $n \in [1\times10^3, 1\times10^5]$ to generate data sets of different sizes. For the experiments on the NDI data set, we generated subsets of different sizes by randomly sampling the original NDI data set. All the performance curves of runtime and memory overheads in this section are plotted in double logarithmic scale.

In Figures 7(a)-(c), the growths of runtime on the synthetic data sets are are consistent with the orders of time complexities (Table 1) analyzed in Section 4.5. The results on the real world data set NDI (Figure 7(d)) also demonstrate that the growth of runtime of ALID is substantially lower than the other affinity-based methods.

In Figures 7(e)-(g), the growth of memory usage of ALID on the three synthetic data sets are consistent with the space complexities (see Table 1) analyzed in Section 4.5. The results on NDI (Figure 7(h)) further demonstrate the superior memory performance of ALID, which consumes about 90% less memory to process 3.3 times larger data size with a more moderate increase of memory usage than the other affinity-based methods.

Figures 7(i)-(l) show that ALID achieves comparable AVG-F performance with the other affinity-based methods on all the four data sets. The AVG-F curves in Figures 7(i), (j), and (l) have similar trends as the data set size increases: the AVG-F grows first due to the increasing dense subgraph cohesiveness caused by the growing amount of data in dominant clusters (i.e., $a^*=\frac{\omega n}{20}$ and $a^*=\frac{n^\eta}{20}$). Then, the AVG-F decreases due to the growing amount of data in overlapping dominant clusters and the increasing influence of noise. The AVG-F curves in Figure 7(k) monotonously decreases, since the dense subgraph cohesiveness does not increase when the amount of data in clusters is fixed to $a^*=\frac{P}{20}$.

In summary, ALID achieves remarkably better scalability than the other affinity-based methods, and, at the same time, retains high detection quality.

## 5.3 Parallel Experiments of PALID

PALID is the parallel implementation of ALID. It was implemented in Java on the parallel platform of *Apache Spark* on Ubuntu 13.10. In this section, we report the evaluation results of the parallel performance of PALID on a cluster of 5 PC computers each with an i-7 CPU, 32 GB RAM and a 7200 RPM hard drive. We set 1 machine as the master and the other 4 machines as workers. Each worker was assigned

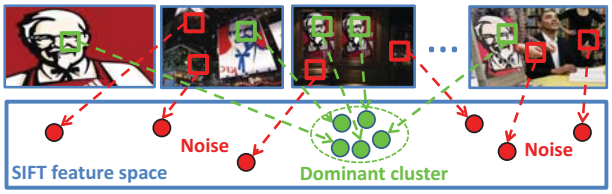



**Figure 8: Illustration of SIFT dominant cluster.**

**Table 2: Performance of PALID on SIFT-50M**

| Methods | Executors | Runtime | Speadup Ratio |
|---|---|---|---|
| PALID-1Exec | 1 | 17.2 hours | 1 |
| PALID-2Exec | 2 | 8.96 hours | 1.92 |
| PALID-4Exec | 4 | 4.48 hours | 3.84 |
| PALID-8Exec | 8 | 2.29 hours | 7.51 |

with 2 executor processes, where each executor took a single CPU core. The hash tables and the data items were stored in a MongoDB (`http://www.mongodb.org/`) server on the master. All machines were connected by a 1000 Mbit/s ethernet. The performance of PALID was evaluated on the SIFT-50M data set, which is an unlabeled data set containing 50 million SIFT features [23]. The SIFTs were extracted from the IPDID/1000k image data set [10] using the VLFeat toolbox [35].

Scale Invariant Feature Transform (SIFT) [23] is a $L_2$ normalized 128-dimensional vector describing the texture of a small image region. In the field of computer vision, it is a standard procedure to represent similar image regions by highly cohesive SIFT dominant clusters named "*visual words*" [34]. Figure 8 demonstrates that, since partial duplicate images always share a common image content (e.g., KFC grandpa), the SIFTs [23] extracted from similar image regions are highly similar to each other and naturally form a dominant cluster (i.e., visual word). However, the number of visual words is unknown and there are also a large proportion of noisy SIFTs extracted from the random non-duplicate regions (i.e., red points), leading to a high degree of background noise. As a result, the scalability and the strong capability of being resistant against noise in PALID are very suitable for visual word generation.

Table 2 shows the performance of PALID. PALID was able to process 50 million SIFT features in 2.29 hours, achieving a speedup ratio of 7.51 with 8 executors. This demonstrates the promising parallel performance of PALID.

Since there is no parallel solution for the other affinity-based methods, such as IID [31], SEA [19] and AP [17], we fairly compared them with ALID on the SIFT-50M data set using the same single-machine experimental settings as what Section 5.2 used. Figure 9 shows the memory and runtime performance of the affinity-based methods on the uniformly sampled subsets of SIFT-50M. The increases of ALID in both runtime and memory usage are significantly lower than the other methods. Especially, ALID consumes 10 GB memory to process 1.29 million SIFTs in 4.4 hours on a standard PC. In contrast, such a large amount of data is far beyond the capability of the other affinity-based methods, which can at most deal with 0.04 million SIFTs on the same platform.

Although SIFT-50M is too large to be manually labeled, the dominant cluster detection quality can still be qualitatively assessed by the results in Figure 10. In Figures 10(b)-10(f), the affinity-based methods effectively detect most of the SIFT features (i.e., green points) extracted from the similar image regions of "KFC grandpa", since such SIFTs can naturally form dense subgraphs with high cohesiveness. At the same time, the large proportion of noisy SIFTs (i.e., red points) extracted from the random non-duplicate background image regions are filtered out. The results demonstrate the effectiveness of the affinity-based methods in resisting overwhelming amount of noisy SIFTs. Additionally, we also evaluated the AVG-F performance of PALID on the labeled data sets of NART and NDI. The resulting AVG-F performances are consistent with ALID. Limited by space, we omit the details here.

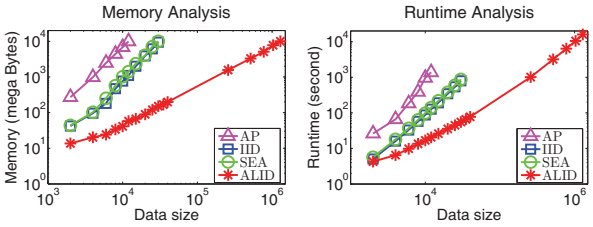
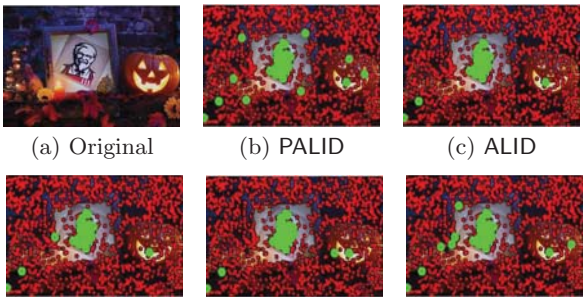



**Figure 9: Scalability on the SIFT-50M subset.**



(a) Original (b) PALID (c) ALID

(d) IID (e) SEA (f) AP

**Figure 10: Detection quality on (a) "KFC grandpa" of the IPDID/1000k data set. The green points are SIFTs from dominant clusters with high densities ($\pi(x) > 0.75$). The red points in (b)-(f) are noise SIFTs filtered out by the affinity-based methods.**

## 6. CONCLUSIONS

In this paper, we proposed ALID, a scalable and effective dominant cluster detection approach against high background noise. ALID demonstrates remarkable noise resistance capability, achieves significant scalability improvement over the other affinity-based methods, and is highly parallelizable in MapReduce framework. As future work, we will further extend ALID towards the online version to efficiently process streaming data.

## Acknowledgements

The authors are grateful to the anonymous reviewers for their constructive comments and suggestions. This work was supported in part by National Basic Research Program of China (973 Program): 2012CB316400 and 2015CB352400, National Natural Science Foundation of China: 61025011,

61332016, 61390511 and 61303160, 863 Program of China: 2014AA015202, Postdoctoral Science Foundation of China: 2014T70126, Basic Research Program of Shenzhen: JCYJ20140610152828686, National Research Foundation under its IRC@Singapore Funding Initiative, CMU Technologies for Safe and Efficient Transportation (T-SET UTC) sponsored by USDOT, an NSERC Discovery grant and a BCIC NRAS Team Project. S. Liu also acknowledges the Google Faculty Research Award. All opinions, findings, conclusions and recommendations in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

## 7. REFERENCES

[1] `www.psi.toronto.edu/index.php?q=affinity%20propagation`.

[2] `https://sites.google.com/site/lhrbss/`.

[3] P. Anderson, A. Thor, J. Benik, L. Raschid, and M. E. Vidal. Pang: Finding patterns in annotation graphs. In *SIGMOD*, pages 677–680, 2012.

[4] A. Angel, N. Sarkas, N. Koudas, and D. Srivastava. Dense subgraph maintenance under streaming edge weight updates for real-time story identification. In *PVLDB*, volume 5, pages 574–585, 2012.

[5] B. Bahmani, B. Moseley, A. Vattani, R. Kumar, and S. Vassilvitskii. Scalable k-means++. In *PVLDB*, volume 5, pages 622–633, 2012.

[6] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *JMLR*, 3:993–1022, 2003.

[7] S. R. Bulò and M. Pelillo. A game-theoretic approach to hypergraph clustering. In *NIPS*, pages 1571–1579, 2009.

[8] J. Chen and Y. Saad. Dense subgraph extraction with application to community detection. *TKDE*, 24(7):1216–1230, 2012.

[9] W. Chen, Y. Song, H. Bai, C. Lin, and E. Y. Chang. Parallel spectral clustering in distributed systems. *TPAMI*, 33(3):568–586, 2011.

[10] L. Chu, S. Jiang, S. Wang, Y. Zhang, and Q. Huang. Robust spatial consistency graph model for partial duplicate image retrieval. *TMM*, 15(8):1–15, 2013.

[11] L. Chu, S. Wang, S. Liu, Q. Huang, and J. Pei. ALID: Scalable dominant cluster detection. in arxiv.org, 2014.

[12] D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. *TPAMI*, 24(5):603–619, 2002.

[13] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *SoCG*, pages 253–262, 2004.

[14] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *OSDI*, page 1, 2004.

[15] R. I. Dunbar. Neocortex size as a constraint on group size in primates. *Journal of Human Evolution*, 22(6):469–493, 1992.

[16] C. Fowlkes, S. Belongie, F. Chung, and J. Malik. Spectral grouping using the nystrom method. *TPAMI*, 26(2):214–225, 2004.

[17] B. J. Frey and D. Dueck. Clustering by passing messages between data points. *Science*, 315(5814):972–976, 2007.

[18] M. Li and J. T. Kwok. Making large-scale nystrom approximation possible. In *ICML*, pages 631–638, 2010.

[19] H. Liu, L. Latecki, and S. Yan. Fast detection of dense subgraphs with iterative shrinking and expansion. *TPAMI*, 35(9):2131–2142, 2013.

[20] H. Liu, L. J. Latecki, and S. Yan. Dense subgraph partition of positive hypergraph. *TPAMI*, 37(3):541–554, 2014.

[21] H. Liu and S. Yan. Robust graph mode seeking by graph shift. In *ICML*, 2010.

[22] S. Lloyd. Least squares quantization in pcm. *Information Theory, IEEE Transactions on*, 28(2):129–137, 1982.

[23] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60(2):91–110, 2004.

[24] T. S. Motzkin and E. G. Straus. Maxima for graphs and a new proof of a theorem of turán. *Canadian Journal of Mathematics*, 17(4):533–540, 1965.

[25] A. Y. Ng, M. I. Jordan, Y. Weiss, et al. On spectral clustering: Analysis and an algorithm. In *NIPS*, pages 849–856, 2002.

[26] L. Nunes de Casto and F. J. Von Zuben. An evolutionary immune network for data clustering. In *Neural Networks*, pages 84–89. IEEE, 2000.

[27] A. Oliva and A. Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *IJCV*, 42(3):145–175, 2001.

[28] M. Pavan and M. Pelillo. Efficient out-of-sample extension of dominant-set clusters. In *NIPS*, pages 1057–1064, 2004.

[29] M. Pavan and M. Pelillo. Dominant sets and pairwise clustering. *TPAMI*, 29(1):167–172, 2007.

[30] S. Rota Bulò and I. M. Bomze. Infection and immunization: a new class of evolutionary game dynamics. *Games and Economic Behavior*, 71(1):193–211, 2011.

[31] S. Rota Bulò, M. Pelillo, and I. M. Bomze. Graph-based quadratic optimization: A fast evolutionary approach. *CVIU*, 115(7):984–995, 2011.

[32] J. Sankaranarayanan, H. Samet, B. E. Teitler, M. D. Lieberman, and J. Sperling. Twitterstand: News in tweets. In *SIGSPATIAL*, pages 42–51, 2009.

[33] M. Shindler, A. Meyerson, and A. Wong. Fast and accurate k-means for large datasets. In *NIPS*, 2011.

[34] J. Sivic and A. Zisserman. Video google: A text retrieval approach to object matching in videos. In *ICCV*, pages 1470–1477, 2003.

[35] A. Vedaldi and B. Fulkerson. VLFeat: An open and portable library of computer vision algorithms, 2008.

[36] N. Wang, S. Parthasarathy, K.-L. Tan, and A. K. H. Tung. Csv: Visualizing and mining cohesive subgraphs. In *SIGMOD*, pages 445–458, 2008.

[37] F. L. Wauthier, N. Jojic, and M. I. Jordan. Active spectral clustering via iterative uncertainty reduction. In *SIGKDD*, pages 1339–1347, 2012.

[38] J. W. Weibull. *Evolutionary game theory*. The MIT Press, 1997.

[39] X. Xu, N. Yuruk, Z. Feng, and T. A. J. Schweiger. Scan: A structural clustering algorithm for networks. In *SIGKDD*, pages 824–833, 2007.