Dense Subgraph Partition of Positive Hypergraphs

Hairong Liu, Longin Jan Latecki, *Senior Member, IEEE*, and Shuicheng Yan, *Senior Member, IEEE*

Abstract—In this paper, we present a novel partition framework, called *dense subgraph partition* (DSP), to automatically, precisely and efficiently decompose a *positive hypergraph* into dense subgraphs. A positive hypergraph is a graph or hypergraph whose edges, except self-loops, have positive weights. We first define the concepts of *core subgraph*, *conditional core subgraph*, and *disjoint partition* of a conditional core subgraph, then define DSP based on them. The result of DSP is an ordered list of dense subgraphs with decreasing densities, which uncovers all underlying clusters, as well as outliers. A divide-and-conquer algorithm, called *min-partition evolution*, is proposed to efficiently compute the partition. DSP has many appealing properties. First, it is a nonparametric partition and it reveals all meaningful clusters in a bottom-up way. Second, it has an exact and efficient solution, called *min-partition evolution algorithm*. The min-partition evolution algorithm is a divide-and-conquer algorithm, thus time-efficient and memory-friendly, and suitable for parallel processing. Third, it is a unified partition framework for a broad range of graphs and hypergraphs. We also establish its relationship with the densest k-subgraph problem (DkS), an NP-hard but fundamental problem in graph theory, and prove that DSP gives precise solutions to DkS for all k in a graph-dependent set, called *critical k-set*. To our best knowledge, this is a strong result which has not been reported before. Moreover, as our experimental results show, for sparse graphs, especially web graphs, the size of critical k-set is close to the number of vertices in the graph. We test the proposed partition framework on various tasks, and the experimental results clearly illustrate its advantages.

Index Terms—Graph partition, dense subgraph, densest *k*-subgraph, mode seeking, image matching

1 Introduction

Hypergraph partition (including graph partition) is a fundamental problem in many important disciplines [1], and it has numerous applications, such as partitioning VLSI design circuits [2], task scheduling in multi-processor systems [3], clustering and detection of communities in various networks [4], and image segmentation [5], to name just a few. There are many public softwares, such as METIS,[1] JOSTLE,[2] SCOTCH[3] and CHACO,[4] developed for such purposes.

Since the optimal partition of a hypergraph heavily depends on applications, a huge number of partition methods have been developed to fulfill the needs of various applications. However, to our best knowledge, there is no partition method satisfying the following requirement:

(**R1**) *Automatically, precisely and efficiently partition a hypergraph into dense subgraphs.*

Here *automatically* means that the number of dense subgraphs is a natural output of the partition method, and it solely depends on the structure of a hypergraph; *precisely* indicates that the partition method guarantees to find the optimal solution of the objective and there is no approximation; *efficiently* says that the partition method has low time and memory complexities, with the ability of partitioning large hypergraphs.

A partition method satisfying the requirement (R1) is extremely useful as our experimental results demonstrate. This is because a dense subgraph represents a potential cluster, thus, partitioning a hypergraph into dense subgraphs means that clusters underlying the hypergraph are enumerated. In fact, the problem of enumerating dense subgraphs has been intensively studied for a few decades [6], due to its importance.

1.1 Our Contributions

The main contributions of this paper are manyfold. First, we propose a novel partition framework satisfying the requirement (R1), called *dense subgraph partition* (DSP). Second, we propose an effective algorithm to compute DSP, called *min-partition evolution*. This algorithm works in a divide-and-conquer way, thus it is very efficient and scales well to large hypergraphs, such as web graphs. Third, we reveal the relationship between densest k-subgraph problem (DkS) [7] and DSP, and prove some important theoretic results. DkS is known to be a NP-hard problem. However, we found that for every hypergraph, there are many ks that DSP can give precise DkSs. Finally, we apply DSP to numerous tasks and our experiments show that DSP is a powerful tool to extract meaningful clusters even in the presence of a large number of outliers.

Fig. 1 illustrates the DSP of a weighted graph $\mathbf{G}$ and its relations to densest subgraph and densest k-subgraphs. First, $\mathbf{G}$ is partitioned into three dense subgraphs, namely, $\mathbf{G}_{V_1}$, $\mathbf{G}_{V_2}$

1. http://glaros.dtc.umn.edu/gkhome/views/metis/index.html
2. http://staffweb.cms.gre.ac.uk/~c.walshaw/jostle/
3. http://www.labri.u-bordeaux.fr/perso/pelegrin/scotch/
4. http://www.sandia.gov/~bahendr/chaco.html

- *H. Liu is with the Department of Mechanical Engineering, Purdue University, West Lafayette, IN 47907. E-mail: lhrbss@gmail.com.*
- *L.J. Latecki is with Department of Computer and Information Sciences, Temple University, Philadelphia, PA 19122. E-mail: latecki@temple.edu.*
- *S. Yan is with the Department of Electrical and Computer Engineering, National University of Singapore, 119077 Singapore. E-mail: eleyans@nus.edu.sg.*

Manuscript received 28 Apr. 2013; revised 30 July 2014; accepted 3 Aug. 2014. Date of publication 6 Aug. 2014; date of current version 13 Feb. 2015. Recommended for acceptance by H. Ishikawa.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPAMI.2014.2346173

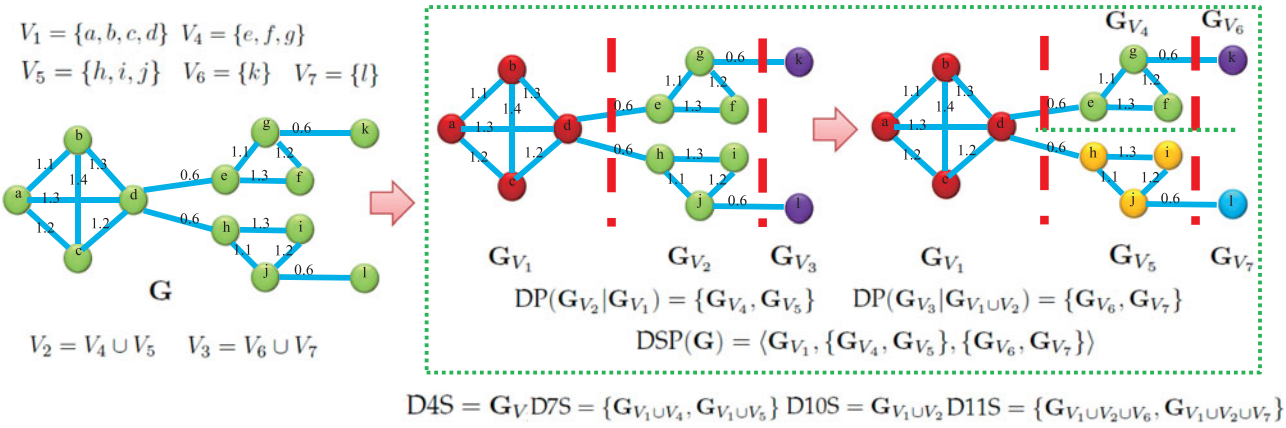

Fig. 1. Dense subgraph partition of a weighted graph $\mathbf{G}$ and its relation to densest k-subgraphs. DSP has two layers of partition. In the first layer, $\mathbf{G}$ is partitioned into three ordered subgraphs, namely, $\mathbf{G}_{V_1}$, $\mathbf{G}_{V_2}$ and $\mathbf{G}_{V_3}$, ordered by their densities, from large to small. In the second layer, $\mathbf{G}_{V_2}$ is partitioned into two pseudo-disjoint subgraphs, $\mathbf{G}_{V_4}$ and $\mathbf{G}_{V_5}$, and $\mathbf{G}_{V_3}$ is partitioned into two pseudo-disjoint subgraphs, $\mathbf{G}_{V_6}$ and $\mathbf{G}_{V_7}$. Thus, the graph $\mathbf{G}$ has been partition into 5 ordered dense subgraphs by DSP, where $\mathbf{G}_{V_4}$ and $\mathbf{G}_{V_5}$ are exchangeable, as well as $\mathbf{G}_{V_6}$ and $\mathbf{G}_{V_7}$. From DSP, we can easily get exact densest k-subgraphs for some ks, such as D4S, D7S, D10S and D11S for this graph.

and $\mathbf{G}_{V_3}$. Second, $\mathbf{G}_{V_2}$ is partitioned into two components, $\mathbf{G}_{V_4}$ and $\mathbf{G}_{V_5}$, and $\mathbf{G}_{V_3}$ is partitioned into another two components, $\mathbf{G}_{V_6}$ and $\mathbf{G}_{V_7}$. Thus, $\mathbf{G}$ is finally partitioned into five ordered dense subgraphs, namely, $\mathbf{G}_{V_1}$, $\mathbf{G}_{V_4}$, $\mathbf{G}_{V_5}$, $\mathbf{G}_{V_6}$, $\mathbf{G}_{V_7}$. The first dense subgraph $\mathbf{G}_{V_1}$ is the densest subgraph of $\mathbf{G}$. From this partition, it is easy to get precise DkSs for some ks by merging the front part of the ordered subgraphs. For example, we can merge $\mathbf{G}_{V_1}$ and $\mathbf{G}_{V_4}$ to form one D7S. Since $\mathbf{G}_{V_4}$ and $\mathbf{G}_{V_5}$ are similar, another D7S is the merge of $\mathbf{G}_{V_1}$ and $\mathbf{G}_{V_5}$.

2 Related Methods

Our method is closely related to two categories of methods. The first category is hypergraph (including graph) partition methods, especially these methods that can automatically determine the number of subgraphs. The second category is dense subgraph detection methods.

Hypergraph partition methods. The majority of hypergraph partition methods are to divide a hypergraph into a pre-specified number of parts. These methods generally optimize a global objective function. For example, Kernighan-Lin algorithm [8] attempts to partition a graph into two disjoint parts with equal size, such that the cut between these two parts is minimized; while the k-way Maximum Sum of Densities method [9] partitions a hypergraph into k parts, such that the sum of the densities of all k parts is maximized. For general graphs, two kinds of methods are popular, due to their good performance and solid theoretical foundation. The first kind is spectral partition [4], [5], [10], and the second kind is netflow based partition [11]. Spectral partition methods rely on eigendecomposition of a matrix constructed from hypergraphs, such as Laplacian matrix [5] and Modularity matrix [4]. Netflow based partition methods are rooted in the well-known network max-flow min-cut theorem [12]. For graphs of specific structures, some methods yielding better partitions have been proposed, such as the multicut for planar graph in [13], which gives globally optimal partitions.

For hypergraphs, a classic heuristic method is Fiduccia-Mattheyses algorithm [14], which partitions a hypergraph into two parts under certain area ratio such that the cut is minimized. The spectral partition methods are also generalized to hypergraphs [15], [16], resulting in various spectral hypergraph cuts. Using a peeling-off strategy, the k-way Maximum Sum of Densities method [9] iteratively finds maximum density subgraphs of hypergraphs, from which a linear order of vertices is obtained, then dynamic programming is applied on the linear order to split the hypergraph into k parts. As the size of a hypergraph grows, the computational cost of partition increases quickly. Multilevel partition methods [17] have been proposed to achieve a balance between partition quality and computational burden. These methods first simplify original graphs, then partition them, finally refine the partitions to achieve better results. By focuing on separating edges, Bansal et al. proposed the correlation clustering method [18], which automatically partitions a binary graph into a few parts. Emanuel and Fiat generalized it to arbitrary weighted graphs and pointed out its relation to multicut [19]. Kim et al. further generated this method to hypergraph and achieved good results for the task of image segmentation [20]. For image segmentation, Felzenszwalb and Huttenlocher proposed a classic method [21], where an image is represented as a graph and then partitioned into regions using a predicate.

Dense subgraph detection methods. Due to the importance of dense subgraphs, there are many dense subgraph detection methods [6], [22], [23], [24], [25], [26], [27]. In [27], a polynomial-time algorithm for finding the maximum density subgraph is introduced. In [26], the concept of clique is generalized to weighted graphs and an efficient method to detect dense subgraphs is proposed. This idea has also been generalized to hypergraphs [28]. Liu et al. generalized [26] and proposed a method to efficiently enumerate all dense subgraphs [23], [24]. Saha et al. [29] defined a generalization of the densest subgraph problem by an additional distance restriction to the nodes of the subgraph and showed its application in gene annotation graphs. Note that these dense subgraph detection methods can be easily generalized to hypergraphs.

Although sharing some similarities, DSP is quite different from these reviewed methods. Compared with other partition methods, DSP is the first method satisfying the requirement (R1). Compared with dense subgraph

detection methods, DSP efficiently enumerates all dense subgraphs in a precise and principled way.

3 Basic Definitions

Set. A *set* is a collection of distinct elements. There is no order between elements in a set. In this paper, we use the symbol $\{\ldots\}$ to represent a set.

Sequence. A *sequence* is an ordered list of elements and we use the symbol $\langle\ldots\rangle$ to represent a sequence.

Permutation. A *permutation* of a set of elements is an arrangement of those elements into a particular order.

Sub-Permutation. A *sub-permutation* of a permutation $\mathbf{P}$ is a contiguous subsequence of $\mathbf{P}$. For a sub-permutation $\mathbf{R}$, its first and last elements are denoted by $\mathbf{R}^F$ and $\mathbf{R}^L$, respectively. $[a,b]$, (a,b), $(a,b]$ and $[a,b)$ all represent sub-permutations of $\mathbf{P}$, from the element a to the element b, where square bracket and round bracket indicate the inclusion and non-inclusion of boundary elements, respectively. $\overrightarrow{\Upsilon}_{\mathbf{R}}$ represents the set of sub-permutations of $\mathbf{R}$ whose first element is $\mathbf{R}^F$; while $\overleftarrow{\Upsilon}_{\mathbf{R}}$ represents the set of sub-permutations of $\mathbf{R}$ whose last element is $\mathbf{R}^L$. For example, if $\mathbf{P} = \langle a,b,c,d,e,f\rangle$, then $\mathbf{P}^F = a$, $\mathbf{P}^L = f$, $[b,d] = \langle b,c,d\rangle$, $[b,d) = \langle b,c\rangle$, $\overrightarrow{\Upsilon}_{[b,d]} = \{\langle b\rangle, \langle b,c\rangle, \langle b,c,d\rangle\}$, $\overleftarrow{\Upsilon}_{(b,e]} = \{\langle c,d,e\rangle, \langle d,e\rangle, \langle e\rangle\}$.

Hypergraph. A hypergraph $\mathbf{G}$ is a triple $\mathbf{G} = (V, E, w)$, where V is a set of vertices, E is a set of hyperedges, and w is the set of weights of all hyperedges. A hyperedge $e \in E$ is a non-empty subset of V, and the size of this subset is called the degree of e, denoted by $d(e)$. For example, $d(e) = 2$ means that e is a pairwise edge, and $d(e) = 1$ can be interpreted as e is a self-loop. Each hyperedge $e \in E$ has a weight $w(e)$.

Positive hypergraph. A *positive hypergraph* is a hypergraph whose weights of edges, except for self-loops, are positive. In other words, only the weights of self-loops might be negative. The positive hypegraph is a very general definition. In fact, it includes most of commonly used graphs, e.g., pairwise graphs, hypergraphs and multipartite graphs. In this paper, we restrict our discussions to positive hypergraphs.

Subgraph. For a subset $U \subset V$, the subgraph induced by U is denoted by $\mathbf{G}_U = (U, E_U)$, where E_U is constituted by all hyperedges which are subsets of U. That is, $E_U = \{e \mid e \in E, e \subseteq U\}$. For a sub-permutation $\mathbf{R}$, the subgraph induced by $\mathbf{R}$, denoted by $\mathbf{G}_{\mathbf{R}}$, is the subgraph induced by the vertex set of $\mathbf{R}$.

Total weight of a hypergraph. The *total weight* of a hypergraph $\mathbf{G}$, denoted by $w(\mathbf{G})$, is defined to be the sum of weights of all hyperedges of $\mathbf{G}$, that is, $w(\mathbf{G}) = \sum_{e\in E} w_e$.

Density of a hypergraph.[5] The *density* of a hypegraph $\mathbf{G} = (V, E)$ is defined to be $\rho(\mathbf{G}) = \frac{w(\mathbf{G})}{|V|}$, where $|V|$ is the cardinality of V.

Densest subgraph. The *densest subgraph* of $\mathbf{G}$ is a subgraph of $\mathbf{G}$ with maximum density.

4 Definition of Dense Subgraph Partition

In this section, we will first define a core subgraph and a conditional core subgraph, then define DSP.

5. This is different from another popular definition of density: ratio between the sum of weights and the number of possible hyperedges.

4.1 Core Subgraph and Conditional Core Subgraph

A positive hypergraph $\mathbf{G}$ may have multiple densest subgraphs. However, only one of them has maximal number of vertices, as proven later. We define this one to be *core subgraph*, denoted by $\mathrm{CS}(\mathbf{G})$.

For two sets $U \subseteq V$ and $S \subseteq V$, the *conditional total weight* of a subgraph $\mathbf{G}_U$ conditioned on a subgraph $\mathbf{G}_S$ is defined as $w(\mathbf{G}_U \mid \mathbf{G}_S) = w(\mathbf{G}_{U\cup S}) - w(\mathbf{G}_S)$. If $U \cap S = \emptyset$, then $w(\mathbf{G}_U \mid \mathbf{G}_S) \geq w(\mathbf{G}_U)$, since $w(\mathbf{G}_{U\cup S}) \geq w(\mathbf{G}_U) + w(\mathbf{G}_S)$. For any $U, T,\ S \subseteq V$, it is easy to verify the following important relation:

$$w(\mathbf{G}_U \mid \mathbf{G}_S) + w(\mathbf{G}_T \mid \mathbf{G}_S) \leq w(\mathbf{G}_{U\cap T} \mid \mathbf{G}_S) + w(\mathbf{G}_{U\cup T} \mid \mathbf{G}_S).$$

The *conditional density* of $\mathbf{G}_U$ conditioned on $\mathbf{G}_S$ is defined to be $\rho(\mathbf{G}_U \mid \mathbf{G}_S) = \frac{w(\mathbf{G}_U \mid \mathbf{G}_S)}{|U|}$. When $U \cap S = \emptyset$, since $w(\mathbf{G}_U \mid \mathbf{G}_S) \geq w(\mathbf{G}_U)$, $\rho(\mathbf{G}_U \mid \mathbf{G}_S) \geq \rho(\mathbf{G}_U)$.

Conditioned on a subgraph $\mathbf{G}_S$, there might be multiple subgraphs whose conditional density reach maximum, such as $\mathbf{G}_{V_4}$ and $\mathbf{G}_{V_5}$ in Fig. 1 (conditioned on $\mathbf{G}_{V_1}$). Among these subgraphs, only one of them has maximal number of vertices, as proven later. Similar to the definition of core subgraph, we define this one to be *conditional core subgraph*, denoted by $\mathrm{CCS}(\mathbf{G} \mid \mathbf{G}_S)$. In Fig. 1, the conditional core subgraph conditioned on $\mathbf{G}_{V_1}$ is $\mathbf{G}_{V_2}$. Note that a core subgraph is a special CCS, that is, $\mathrm{CS}(\mathbf{G}) = \mathrm{CCS}(\mathbf{G} \mid \emptyset)$.

For CCSs, we have the following important theorem.

Theorem 1. *Conditioned on a subgraph $\mathbf{G}_S$, if the set of subgraphs whose conditional densities reach maximum is $\Pi = \{\mathbf{G}_{V_1}, \ldots, \mathbf{G}_{V_k}\}$ and the CCS is $\mathbf{G}_U$, then we have: 1) $\mathbf{G}_U \in \Pi$, and 2) $V_i \subseteq U$ for all $i = 1, \ldots, k$.*

Proof. Please see Supplement Material [which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPAMI.2014.2346173]. □

According to Theorem 1, it is clear that both core subgraph and CCS are unique.

4.2 Partition of a Conditional Core Subgraph

Theorem 1 also tells us that a CCS may have finer structure. In this section, we will define a partition to uncover the structure inside a CCS.

Definition 1. *Suppose $\mathbf{G}_U$ is the CCS conditioned on $\mathbf{G}_S$ and $\rho(\mathbf{G}_U \mid \mathbf{G}_S) = \rho^*$, a **disjoint partition** of $\mathbf{G}_U$ divides $\mathbf{G}_U$ into maximal number of subgraphs, denoted by $DP(\mathbf{G}_U \mid \mathbf{G}_S) = \{\mathbf{G}_{U_1}, \ldots, \mathbf{G}_{U_t}\}$, such that $\rho(\mathbf{G}_{U_i} \mid \mathbf{G}_S) = \rho^*$ for all $i = 1, \ldots, t$. This also introduces a partition of U, denoted by $\Gamma(U) = \{U_1, \ldots, U_t\}$.*

The disjoint partition partitions U into maximal number of subsets such that there is no such hyperedge e: $e \subseteq U \cup S$ and e has non-empty intersections with at least two subsets. If discarding all vertices not in U, all subgraphs are disjoint, this is why we say that these subgraphs are pseudo-disjoint. In Fig. 1, two examples of disjoint partition are $\mathbf{DP}(\mathbf{G}_{V_2} \mid \mathbf{G}_{V_1}) = \{\mathbf{G}_{V_4}, \mathbf{G}_{V_5}\}$ and $\mathbf{DP}(\mathbf{G}_{V_3} \mid \mathbf{G}_{V_1\cup V_2}) = \{\mathbf{G}_{V_6}, \mathbf{G}_{V_7}\}$. Note that there is no order between subgraphs in a disjoint partition, since these subgraphs have the same conditional density.

Theorem 2. *The disjoint partition $DP(\mathbf{G}_U \mid \mathbf{G}_S)$ is unique.*

Proof. Please see Supplement Material, available online. □

4.3 Dense Subgraph Partition

Definition 2. *The **dense subgraph partition** of a positive hypergraph* $\mathbf{G}$ *is defined as follows:*

$$DSP(\mathbf{G}) = \langle DP(\mathbf{G}_{V_1} \mid \emptyset), \ldots, DP(\mathbf{G}_{V_i} \mid \mathbf{G}_{\cup_{j=1}^{i-1} V_j}), \ldots, DP(\mathbf{G}_{V_m} \mid \mathbf{G}_{\cup_{j=1}^{m-1} V_j}) \rangle, \cup_{i=1}^{m} V_i = V, \tag{1}$$

with $\mathbf{G}_{V_1}$ *being the core subgraph of* $\mathbf{G}$ *and* $\mathbf{G}_{V_i}$ $(i > 1)$ *being the CCS conditioned on the subgraph* $\mathbf{G}_{\cup_{j=1}^{i-1} V_j}$.

That is, DSP includes two layers of partitions. First, $\mathbf{G}$ is sequentially partitioned into a sequence of conditional core subgraphs, $\langle \mathbf{G}_{V_1}, \ldots, \mathbf{G}_{V_m} \rangle$. This introduces a partition of V, denoted by $\Psi(V) = \langle V_1, \ldots, V_m \rangle$. Second, each $\mathbf{G}_{V_i}$ is partitioned into pseudo-disjoint subgraphs by the operation of disjoint partition.

Due to the uniqueness of CCS and its disjoint partition, DSP is unique. A notable characteristic of DSP is that there is no parameter and the number of subgraphs is automatically determined.

Theorem 3. *In* $DSP(\mathbf{G})$, $\rho(\mathbf{G}_{V_i} \mid \mathbf{G}_{\cup_{j=1}^{i-1} V_j})$ *strictly decreases as* i *increases from* 1 *to* m.

Proof. Please see Supplement Material, available online. □

The conditional densities define an order over subgraphs $\mathbf{G}_{V_i}$, from large to small. Recall that all subgraphs in $\mathrm{DP}(\mathbf{G}_{V_i} \mid \mathbf{G}_{\cup_{j=1}^{i-1} V_j})$ have the same conditional densities. Hence, the result of DSP is a non-increasing order of dense subgraphs, ordered by their conditional densities.

Since subgraphs with large densities are more likely to represent real clusters, and subgraphs with small densities are usually formed by outliers, DSP is a powerful tool to discover meaningful clusters in massive outliers. Intuitively speaking, it is similar to discover isles in a large ocean. More importantly, since there is a precise and efficient algorithm, these isles are *guaranteed* to be discovered, no matter how huge the ocean is. This is a big advantage over many previous methods.

According to Definition 2, an intuitive way to compute DSP is to iteratively compute and partition every CCS. However, iteratively computing CCSs is computationally expensive. First, the number of CCSs, m, is usually very large, especially for large graphs. Second, it is very time-consuming and memory-expensive to directly compute each CCS of a large graph. Fortunately, DSP can be computed in a divide-and-conquer way. In the next section, we will present such an algorithm, called *min-partition evolution*, which is very efficient. In fact, on a regular PC, it can precisely partition a positive hypergraph with millions of vertices and hyperedges in a few minutes.

5 Min-Partition Evolution Algorithm

In Definition 2, $\Psi(V)$ defines a partial order over V, since there is no order between two vertices in the same subset. Among all $|V|!$ permutations of V, there is a subset of permutations, denoted by $\Theta(\mathbf{G})$, satisfying all orders in $\Psi(V)$. That is, for each permutation in $\Theta(\mathbf{G})$, the vertices in V_1 are put front, then the vertices in V_2, ..., finally the vertices in V_m. Since the vertices in V_i have $|V_i|!$ permutations, $|\Theta(\mathbf{G})| = \Pi_{i=1}^{m} |V_i|!$.

Our algorithm is inspired by a simple observation: it is very easy to compute $\mathrm{DSP}(\mathbf{G})$ based on a permutation in $\Theta(\mathbf{G})$. Of course, we do not know such a permutation. An intuitive idea is to *start from an initial permutation, gradually modify it to approach a permutation in* $\Theta(\mathbf{G})$.

The min-partition evolution algorithm, which is summarized in Algorithm 1, is exactly an implementation of this idea.[6] It consists of four major procedures,

Algorithm 1. Min-Partition Evolution

1: **Input:** $\mathbf{G}$ and an initial permutation $\mathbf{P}$.
2: Apply *min-partition()* on $\mathbf{P}$ to get $\mathrm{MP}(\mathbf{P})$;
3: Set $\Omega = \mathrm{MP}(\mathbf{P})$, $\tilde{\Omega} = \emptyset$ and $\hat{\mathbf{P}} = \mathbf{P}$;
4: **repeat**
5: **for** each $\mathbf{R} \in \mathrm{MP}(\mathbf{P})$ **do**
6: **if** $\mathbf{R} \notin \tilde{\Omega}$ **then**
7: Apply *permutation-reorder()* on $\mathbf{R}$ to get $\hat{\mathbf{R}}$;
8: **if** $\hat{\mathbf{R}} \neq \mathbf{R}$ **then**
9: Replace $\mathbf{R}$ by $\hat{\mathbf{R}}$ in $\hat{\mathbf{P}}$;
10: Apply *min-partition()* on $\hat{\mathbf{R}}$ to get $\mathrm{MP}(\hat{\mathbf{R}})$ and replace $\mathbf{R}$ by $\mathrm{MP}(\hat{\mathbf{R}})$ in Ω;
11: **end if**
12: **end if**
13: **end for**
14: Apply *min-merge()* on Ω to get $\mathrm{MP}(\hat{\mathbf{P}})$;
15: Set $\tilde{\Omega} = \mathrm{MP}(\mathbf{P})$, $\mathbf{P} = \hat{\mathbf{P}}$ and $\mathrm{MP}(\mathbf{P}) = \mathrm{MP}(\hat{\mathbf{P}})$;
16: **until** $\mathbf{P}$ does not change
17: For each $\mathbf{R} \in \mathrm{MP}(\mathbf{P})$, apply *disjoint-partition()* on $\mathbf{G}_{\mathbf{R}}$.
18: **Output:** $\mathrm{DSP}(\mathbf{G})$.

1, min-partition(): a procedure to partition a permutation $\mathbf{P}$ into a sequence of sub-permutations. The result is called a *min-partition* of $\mathbf{P}$, denoted by $\mathrm{MP}(\mathbf{P})$. This also introduces a partition of $\mathbf{G}$, with every sub-permutation in $\mathrm{MP}(\mathbf{P})$ inducing a subgraph. Especially, when $\mathbf{P}$ belongs to $\Theta(\mathbf{G})$, the output of *min-partition()* is $\Psi(V)$, the first layer partition of DSP.

2, min-merge(): a fast variant of *min-partition()*. It operates on a partition of $\mathbf{P}$ to quickly get $\mathrm{MP}(\mathbf{P})$ by merging some consecutive sub-permutations.

3, permutation-reorder(): the procedure to find a better permutation. The input of *permutation-reorder()* is a sub-permutation $\mathbf{R} \in \mathrm{MP}(\mathbf{P})$ and it tries to find a better permutation $\hat{\mathbf{R}}$ to replace $\mathbf{R}$, which also changes $\mathbf{P}$.

4, disjoint-partition(): the procedure of computing the disjoint partition of a CCS.

The details of these procedures will be explained later.

In each iteration (step 5 to step 15), this algorithm finds a better $\hat{\mathbf{P}}$ to replace $\mathbf{P}$. The meaning of a better permutation is defined in Section 5.3. If there is no better permutation, $\mathbf{P} \in \Theta(\mathbf{G})$. Only the procedure *permutation-reorder()* modifies the order of vertices and a significant characteristic of Algorithm 1 is that it updates $\mathbf{P}$ by updating its sub-permutations independently. Note that if a sub-permutation $\mathbf{R} \in \mathrm{MP}(\mathbf{P})$

6. The source code is published in the following website: https://sites.google.com/site/lhrbss/

also belongs to the min-partition of the previous permutation, it means that there is no better alternative for this sub-permutation. Thus, we do not apply *permutation-reorder()* on $\mathbf{R}$ (step 6). Although the number of vertices in $\mathbf{P}$ may be very large, the number of vertices in each sub-permutation $\mathbf{R}$ is usually small. Thus, Algorithm 1 is very efficient. Moreover, it is very suitable for parallel processing.

5.1 *min-Partition()*: Min-Partition under a Permutation

In this section, we first define the concept of reward and mean reward, then define min-partition and present the algorithmic details of *min-partition()*.

5.1.1 Reward and Mean Reward

Under a permutation $\mathbf{P}$ of the vertex set V, the *reward* of a vertex v, denoted by $r_{\mathbf{P}}(v)$, is defined as follows:

$$r_{\mathbf{P}}(v) = \sum_{e\in E, v\in e, e\subseteq[\mathbf{P}^F, v]} w_e. \tag{2}$$

Here $e \subseteq \mathbf{R}$ means that e is a subset of the elements in $\mathbf{R}$. Intuitively speaking, among all elements in a hyperedge $e \in E$, if v is the one whose position in $\mathbf{P}$ is backmost, then the weight of e is added to the reward of v. The rewards of all vertices in a permutation $\mathbf{P}$ form a vector, denoted by $r_{\mathbf{P}}$.

According to the definition of reward, we have:

$$\sum_{v\in\mathbf{P}} r_{\mathbf{P}}(v) = \sum_{e\in E} w_e \tag{3a}$$

$$\sum_{v\in\mathbf{R}} r_{\mathbf{P}}(v) = w(\mathbf{G}_{\mathbf{R}} \,|\, \mathbf{G}_{[\mathbf{P}^F, \mathbf{R}^F)}). \tag{3b}$$

The first equation says that the sum of rewards of all vertices is a constant, which is the sum of weights of all hyperedges; while the second equation connects rewards to the conditional total weights.

The *mean reward* of $\mathbf{R}$ is defined to be $\overline{m}(\mathbf{R}) = \sum_{v\in\mathbf{R}} r_{\mathbf{P}}(v)/|\mathbf{R}|$. According to (3b), we have $\overline{m}(\mathbf{R}) = \rho(\mathbf{G}_{\mathbf{R}} \,|\, \mathbf{G}_{[\mathbf{P}^F, \mathbf{R}^F)})$. Therefore, mean reward corresponds to conditional density.

5.1.2 Min-Partition

A sub-permutation $\mathbf{R}$ is called a *min-sub-permutation* (MSP) if for any bi-partition $\mathbf{R} = \langle \mathbf{R}_1, \mathbf{R}_2 \rangle$, we have $\overline{m}(\mathbf{R}_1) \le \overline{m}(\mathbf{R}_2)$. That is, if $\mathbf{R}$ is a MSP, no matter how you divide it into two parts, the mean reward of the first part is always not larger than the mean reward of the second part. From this definition, we can immediately get the following result.

Proposition 1. *Suppose that* $\mathbf{R}_1$ *and* $\mathbf{R}_2$ *are two consecutive MSPs of* $\mathbf{P}$*, where* $\mathbf{R}_1$ *is before* $\mathbf{R}_2$*, if* $\overline{m}(\mathbf{R}_1) \le \overline{m}(\mathbf{R}_2)$*, then* $\langle \mathbf{R}_1, \mathbf{R}_2 \rangle$ *is also a MSP of* $\mathbf{P}$*.*

A MSP of $\mathbf{P}$ is called a *maximal min-sub-permutation* (MMSP) if it is not a sub-permutation of any other MSPs of $\mathbf{P}$. That is, a MMSP cannot be further extended.

Proposition 2. *Two MMSPs of a permutation cannot overlap.*

Based on these two propositions, we can give a definition of min-partition.

Definition 3. *A* ***min-partition*** *of* $\mathbf{P}$*, denoted by* $\mathrm{MP}(\mathbf{P})$*, is a partition of* $\mathbf{P}$ *into MMSPs. That is,* $MP(\mathbf{P}) = \langle \mathbf{P}_i \,|\, i = 1, \ldots, s \rangle$*, with each* $\mathbf{P}_i$ *being a MMSP of* $\mathbf{P}$*.*

Mathematically, $\mathbf{P}_1 = \arg\max_{\mathbf{R}\in\overrightarrow{\Upsilon}_{\mathbf{P}}} \overline{m}(\mathbf{R})$ and $\mathbf{P}_i = \arg\max_{\mathbf{R}\in\overrightarrow{\Upsilon}_{(\mathbf{P}^L_{i-1}, \mathbf{P}^L]}} \overline{m}(\mathbf{R})$ for all $i = 2, \ldots, s$. In both cases, if there are multiple sub-permutations whose mean rewards reach maximum, the longest one is the right one.

Proposition 3. *For a fixed* $\mathbf{G}$ *and* $\mathbf{P}$*, min-partition* $\mathrm{MP}(\mathbf{P})$ *is unique.*

Proposition 4. *If* $\mathrm{MP}(\mathbf{P}) = \langle \mathbf{P}_i \,|\, i = 1, \ldots, s \rangle$ *and* $s > 1$*, then* $\overline{m}(\mathbf{P}_1) > \ldots > \overline{m}(\mathbf{P}_s)$*.*

These two propositions are direct results of the Definition 3.

The procedure of *min-partition()* is summarized in Algorithm 2. It starts from the first vertex of $\mathbf{P}$ and iteratively searches for MMSPs. y is the integral histogram [30] of the reward vector $r_{\mathbf{P}}$. In each iteration, i and β store the first and last index of current MMSP, respectively, and α stores the maximal mean reward. It is very efficient, with time complexity being linear in $|\mathbf{P}|$.

Algorithm 2. *min-partition()*

1: **Input:** $\mathbf{G}$ and a permutation $\mathbf{P} = \langle a_1, \ldots, a_n \rangle$.
2: Compute the reward vector $r_{\mathbf{P}}$
3: Construct an integral histogram $\{y_i | i = 1, \ldots, n\}$ with $y_1 = r_{\mathbf{P}}(a_1)$ and $y_i = y_{i-1} + r_{\mathbf{P}}(a_i)$ for $i = 2, \ldots, n$;
4: Set $\mathrm{MP}(\mathbf{P}) = \emptyset$ and $i = 1$;
5: **repeat**
6: Set $\alpha = y_i$ and $\beta = i$;
7: **for** $j = i + 1, \ldots, n$ **do**
8: If $\frac{y_j}{j-i+1} \ge \alpha$, then set $\alpha = \frac{y_j}{j-i+1}$ and $\beta = j$;
9: **end for**
10: Add $\langle a_i, \ldots, a_\beta \rangle$ into $\mathrm{MP}(\mathbf{P})$ and set $i = \beta + 1$;
11: **for** $j = i, \ldots, n$ **do**
12: $y_j = y_j - y_\beta$.
13: **end for**
14: **until** $i > n$
15: **Output:** $\mathrm{MP}(\mathbf{P})$.

Fig. 2 illustrates the process of min-partition. First, according to $r_{\mathbf{P}}$, computing the integral histogram y. In the first round of iteration, i is fixed to 1 and β moves backward to find the position where the mean reward $\frac{y_j}{\beta-i+1}$ is maximal, which is 4. Thus, $\langle a, c, d, b \rangle$ is the first MMSP. In the second round, i is fixed to 5, β move backward to 11. In the third round, $i = \beta = 12$. Thus, the min-partition partitions $\mathbf{P}$ into three MMSPs.

5.2 *min-merge()*: Fast Min-Partition

In Algorithm 1, after reordering vertices in each MMSP, we get a new permutation $\hat{\mathbf{P}}$ and one of its partitions, Ω. Although we can directly compute $\mathrm{MP}(\hat{\mathbf{P}})$ using Algorithm 2; based on Ω, there is a more efficient algorithm.

Since every sub-permutation in Ω is a MSP, according to Proposition 1 and Proposition 2, we can get min-partition $\mathrm{MP}(\hat{\mathbf{P}})$ by iteratively merging consecutive MSPs in Ω. The

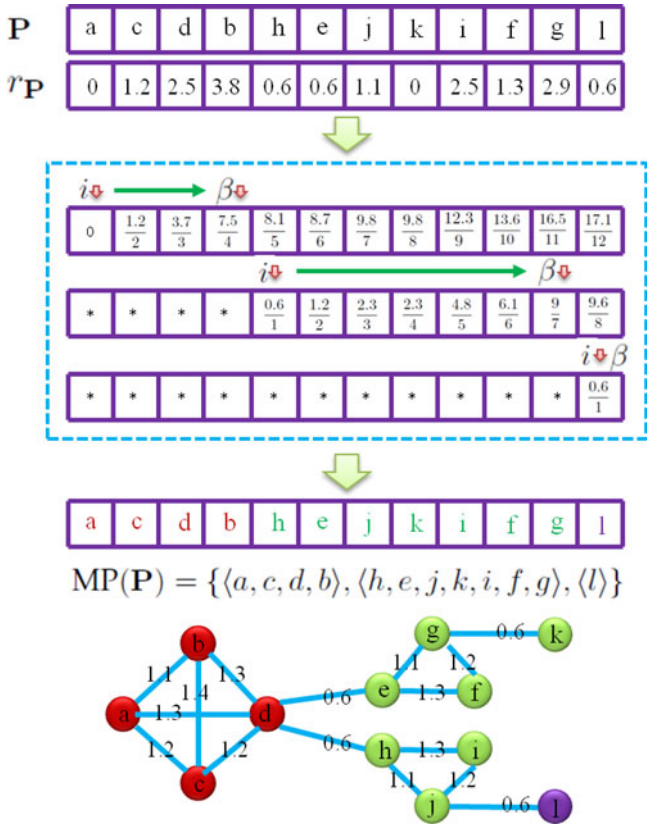

Fig. 2. The process of min-partition for the graph in Fig. 1 under the permutation $\mathbf{P}$. The inputs are the permutation $\mathbf{P}$ and corresponding reward vector $r_{\mathbf{P}}$, the output is the min-partition $\mathrm{MP}(\mathbf{P})$. Only one scan of the reward vector $r_{\mathbf{P}}$ is needed.

algorithm, called *min-merge* algorithm, denoted by $\mathrm{MM}(\Omega)$, is summarized in Algorithm 3.

Algorithm 3. *min-merge()*

1: **Input:** A partition Ω of $\mathbf{P}$ whose sub-permutations are all MSPs.
2: **repeat**
3: Scan Ω to find two consecutive sub-permutations, namely, $\mathbf{R}_1$ and $\mathbf{R}_2$, where $\mathbf{R}_1$ is before $\mathbf{R}_2$, such that $\overline{m}(\mathbf{R}_1) \leq \overline{m}(\mathbf{R}_2)$. If such a pair is found, merge them into one sub-permutation.
4: **until** Ω does not change
5: **Output:** $\mathrm{MM}(\Omega)$.

Theorem 4. *If Ω is a partition of $\mathbf{P}$ whose sub-permutations are all MSPs, the min-merge of Ω is the min-partition of $\mathbf{P}$, that is,* $\mathrm{MP}(\mathbf{P}) = \mathrm{MM}(\Omega)$.

Proof. Please see Supplement Material, available online. □

Since Algorithm 3 operates on a partition Ω of $\mathbf{P}$ and the number of elements in Ω is usually much smaller than $|\mathbf{P}|$, Algorithm 3 is much more efficient than Algorithm 2, especially on large positive hypergraphs.

5.3 *permutation-reorder()*: Reorder Vertices within a Maximal Min-Sub-Permutation

In Algorithm 1, the procedure *permutation-reorder()* is responsible for updating $\mathbf{P}$. To gradually approach a permutation in $\Theta(\mathbf{G})$, *permutation-reorder()* needs to replace $\mathbf{R}$ by a better sub-permutation, $\hat{\mathbf{R}}$. In this section, we will first define what the word "better" means in our context, then demonstrate how to find it. Note that reordering $\mathbf{R}$ does not affect the rewards of vertices not in $\mathbf{R}$.

5.3.1 Reordering by Division

Definition 4. *For a MMSP $\mathbf{R}$, if there is a new permutation $\hat{\mathbf{R}}$ of $\mathbf{R}$ such that $\hat{\mathbf{R}} = \langle \hat{\mathbf{R}}_1, \hat{\mathbf{R}}_2\rangle$ and $\overline{m}(\hat{\mathbf{R}}_1) > \overline{m}(\hat{\mathbf{R}}_2)$, $\mathbf{R}$ is said to be **divisible**; otherwise, it is said to be **indivisible**.*

In other words, for a MMSP, if no matter how to reorder them, it cannot be divided into two parts such that the mean reward of the first part is larger than the mean reward of the second part, it is indivisible; otherwise it is divisible.

permutation-reorder() updates a MMSP $\mathbf{R}$ in the following way: check whether $\mathbf{R}$ is divisible or not, if $\mathbf{R}$ is indivisible, output $\mathbf{R}$; otherwise, output a new permutation $\hat{\mathbf{R}}$ with $|\mathrm{MP}(\hat{\mathbf{R}})| > 1$.

The hyperedges which contribute to the rewards of vertices in $\mathbf{R}$ form a set, denoted by $E_{\mathbf{R}}$, that is, $E_{\mathbf{R}} = \{e \,|\, e \in E, e \subseteq [\mathbf{P}^F, \mathbf{R}^L], e \cap \mathbf{R} \neq \emptyset\}$. If $\mathbf{R}$ is divisible, that is, there is a permutation $\hat{\mathbf{R}} = \langle \hat{\mathbf{R}}_1, \hat{\mathbf{R}}_2\rangle$ of $\mathbf{R}$ such that $\overline{m}(\hat{\mathbf{R}}_1) > \overline{m}(\hat{\mathbf{R}}_2)$, we have $\overline{m}(\hat{\mathbf{R}}_1) > \overline{m}(\hat{\mathbf{R}}) = \overline{m}(\mathbf{R})$.

Suppose $\mathbf{R} = \langle r_1, \ldots, r_m\rangle$ and $\mathbf{x}$ is an $m \times 1$ indicator vector such that $x_i = \begin{cases} 1, & r_i \in \hat{\mathbf{R}}_1; \\ 0, & \text{otherwise.} \end{cases}$ Then $\overline{m}(\hat{\mathbf{R}}_1)$ can be expressed as:

$$\overline{m}(\hat{\mathbf{R}}_1) = \frac{\sum_{e \in E_{\mathbf{R}}} w_e \prod_{i=1}^{m} x_i^{\delta(e, r_i)}}{\sum_{i=1}^{m} x_i}, \tag{4}$$

where $\delta(e, r_i) = \begin{cases} 1, & r_i \in e; \\ 0, & \text{otherwise.} \end{cases}$ Note that here we require $0^0 = 1$.

Suppose $\overline{m}(\mathbf{R}) = \alpha$, $\mathbf{R}$ is divisible means that there is a $\hat{\mathbf{R}}_1$ satisfying $\overline{m}(\hat{\mathbf{R}}_1) > \overline{m}(\mathbf{R}) = \alpha$. Thus, we can judge whether $\mathbf{R}$ is divisible or not by solving the following pseudo-boolean optimization problem [31]:

$$\max_{\mathbf{x} \in \{0,1\}^m} f(\mathbf{x}) \equiv \sum_{e \in E_{\mathbf{R}}} w_e \prod_{i=1}^{m} x_i^{\delta(e, r_i)} - \alpha \sum_{i=1}^{m} x_i. \tag{5}$$

Proposition 5. *Suppose $\mathbf{x}^*$ is the solution of (5), then $\mathbf{R}$ is divisible if and only if $f(\mathbf{x}^*) > 0$.*

5.3.2 Division by QPBO

In general, the pseudo-Boolean optimization problem is NP-hard [31]; however, in our setting, $f(\mathbf{x})$ has a special characteristic: the coefficients of all its terms whose degrees are larger than 1 are positive. Due to this characteristic, the optimization problem (5) can be efficiently and precisely solved. This explains why we only allow the weights of self-loops to be negative in the definition of positive hypergraph.

For a high order term $ax_1 \ldots x_d, d > 2$, when $a > 0$, there is the following important equation[32]:

$$ax_1 \cdots x_d = \max_{w \in \{0,1\}} aw\left\{\sum_{i=1}^{d} x_i - (d-1)\right\}. \tag{6}$$

That is, by introducing another boolean variable w, we can express it by binary and unary terms.

Using the Equation (6), we can transform the optimization problem (5) into a quadratic pseduo-boolean optimization problem $\max F(\mathbf{z}), z_i \in \{0,1\}$, where $\mathbf{z}$ contains all variables of $\mathbf{x}$ and all introduced auxiliary variables. Since each high order term in $f(\mathbf{x})$ introduces an auxiliary variable, the number of auxiliary variables is equal to the number of high order terms in $f(\mathbf{x})$. More importantly, the coefficients of all binary terms in $F(\mathbf{z})$ are positive. Thus, $F(\mathbf{z})$ is a supermodular function and the optimization problem $\max F(\mathbf{z}), z_i \in \{0,1\}$ can be exactly solved [11]. In our implementation, we use the QPBO algorithm [33] to solve it.

The whole procedure is summarized in Algorithm 4. The output of Algorithm 4 is a new permutation $\hat{\mathbf{R}}$ and its min-partition, $\mathrm{MP}(\hat{\mathbf{R}})$. If $|\mathrm{MP}(\hat{\mathbf{R}})| > 1$, then $\mathbf{R}$ is divisible; otherwise $\mathbf{R}$ is indivisible.

Algorithm 4. Divide a MMSP $\mathbf{R}$ by QPBO

1: **Input:** $\mathbf{R} = \langle r_1, \ldots, r_m \rangle$ and $\mathbf{G}$.
2: Compute $E_{\mathbf{R}}$ and construct the function $f(\mathbf{x})$;
3: For all high order terms in $f(\mathbf{x})$, express them by binary and unary terms, thus obtain a quadratic function $F(\mathbf{z})$.
4: Solve the optimization problem $\max F(\mathbf{z}), z_i \in \{0,1\}$ by QPBO, obtain the optimal solution $\mathbf{z}^*$, thus also obtain the optimal solution x^* for the optimization problem (5).
5: $\forall i = 1, \ldots, m$, if $x_i^* = 1$, put r_i into $\hat{\mathbf{R}}_1$, otherwise, put r_i into $\hat{\mathbf{R}}_2$.
6: Obtain a new sub-permutation $\hat{\mathbf{R}} = \langle \hat{\mathbf{R}}_1, \hat{\mathbf{R}}_2 \rangle$ and compute $\mathrm{MP}(\hat{\mathbf{R}})$.
7: **Output:** $\hat{\mathbf{R}}$ and $\mathrm{MP}(\hat{\mathbf{R}})$

5.3.3 Speedup by Heuristics

When $\mathbf{R}$ is large, it is computationally expensive to divide $\mathbf{R}$ by QPBO. In contrast, a heuristic algorithm has a high probability to divide it, although without guarantee. Thus, we adopt the following strategy: first try to divide $\mathbf{R}$ by a fast heuristic algorithm; if it cannot divide $\mathbf{R}$, then divide $\mathbf{R}$ by Algorithm 4.

The heuristic algorithm should be both fast and effective. Note that (5) can be interpreted as selecting a subset of highly related vertices in $\mathbf{R}$, with only the relations expressed by hyperedges in $E_{\mathbf{R}}$ being considered. If a vertex connects to more hyperedges in $E_{\mathbf{R}}$, its probability to be selected should be higher. Using this heuristic, we propose to fast divide $\mathbf{R}$ by Algorithm 5, whose time complexity is linear in $|E_{\mathbf{R}}|$.

Algorithm 5. Divide a MMSP $\mathbf{R}$ by a simple heuristic

1: **Input:** $\mathbf{R}$ and $\mathbf{G}$.
2: Construct a zero array $y = \langle y_1, \ldots, y_{|\mathbf{R}|} \rangle$.
3: **for** each hyperedge $e \in E_{\mathbf{R}}$ **do**
4: For each vertex $v \in e$, if $v \in \mathbf{R}$, then set $y_i = y_i + w_e$, where i is the position of v in $\mathbf{R}$;
5: **end for**
6: Sort y in descending order and arrange $\mathbf{R}$ accordingly to form a new sub-permutation $\hat{\mathbf{R}}$;
7: Compute $\mathrm{MP}(\hat{\mathbf{R}})$.
8: **Output:** $\hat{\mathbf{R}}$ and $\mathrm{MP}(\hat{\mathbf{R}})$

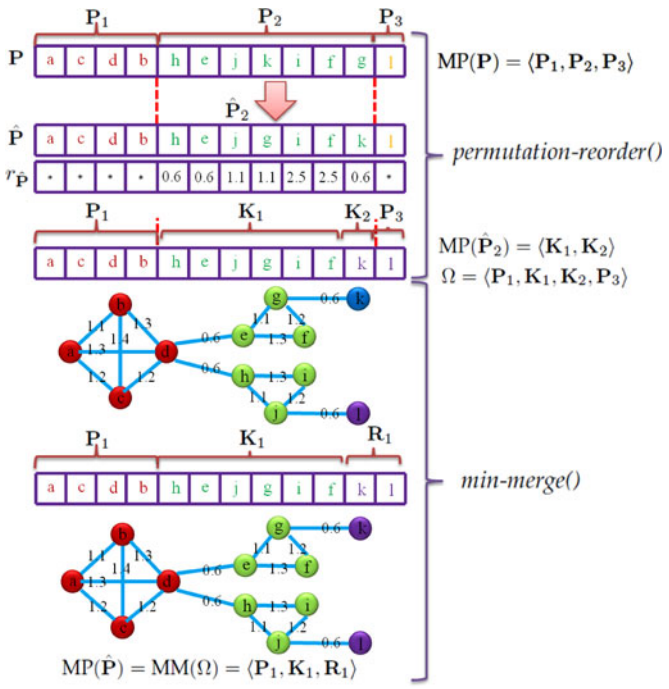

Fig. 3. Illustration of an iteration of Alg. 1 (step 5 to step 15). First, apply *permutation-reorder()* on $\mathbf{P}_2$ to find a better one, $\hat{\mathbf{P}}_2$, and apply *min-partition()* on $\hat{\mathbf{P}}_2$ to get $\mathrm{MP}(\hat{\mathbf{P}}_2)$. Replacing $\mathbf{P}_2$ by $\mathrm{MP}(\hat{\mathbf{P}}_2)$, we get a partition of $\mathbf{P}$, $\Omega = \langle \mathbf{P}_1, \mathbf{K}_1, \mathbf{K}_2, \mathbf{P}_3 \rangle$. Second, apply *min-merge()* on Ω to get $\mathrm{MP}(\hat{\mathbf{P}})$.

permutation-reorder() integrates both Algorithm 4 and Algorithm 5, which is summarized in Algorithm 6. It first tries to divide $\mathbf{R}$ by Algorithm 5 and only when Algrithm 5 cannot divide $\mathbf{R}$, Algorithm 4 is used. When $\mathbf{R}$ is large, Algorithm 5 usually divides it; thus, Algorithm 4 usually works on small $\mathbf{R}$s. Note that Algorithm 5 can be replaced by any other heuristic algorithms, and the correctness of Algorithm 6 is guaranteed by Algorithm 4.

Algorithm 6. *permutation-reorder()*

1: **Input:** $\mathbf{R}$ and $\mathbf{G}$.
2: Divide $\mathbf{R}$ by Algorithm 5 and get $\mathrm{MP}(\hat{\mathbf{R}})$;
3: If $|\mathrm{MP}(\hat{\mathbf{R}})| = 1$, divide $\mathbf{R}$ by Algorithm 4, get a new $\hat{\mathbf{R}}$ and $\mathrm{MP}(\hat{\mathbf{R}})$;
4: If $|\mathrm{MP}(\hat{\mathbf{R}})| = 1$, set $\hat{\mathbf{R}} = \mathbf{R}$ and $\mathrm{MP}(\hat{\mathbf{R}}) = \mathbf{R}$;
5: **Output:** $\hat{\mathbf{R}}$ and $\mathrm{MP}(\hat{\mathbf{R}})$

Fig. 3 illustrates both *permutation-reorder()* and *min-merge()*, two basic procedures in Algorithm 1. In this figure, $\mathrm{MP}(\mathbf{P}) = \langle \mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3 \rangle$. Only $\mathbf{P}_2$ is divisible and the *permutation-reorder()* procedure updates it to $\hat{\mathbf{P}}_2$. The min-partition of $\hat{\mathbf{P}}_2$ is $\mathrm{MP}(\hat{\mathbf{P}}_2) = \langle \mathbf{K}_1, \mathbf{K}_2 \rangle$, thus, we get $\Omega = \langle \mathbf{P}_1, \mathbf{K}_1, \mathbf{K}_2, \mathbf{P}_3 \rangle$. In the *min-merge()* procedure, $\mathbf{K}_2$ and $\mathbf{P}_3$ are merged to form a new MMSP $\mathbf{R}_1$, thus $\mathrm{MP}(\hat{\mathbf{P}}) = \mathrm{MM}(\Omega) = \langle \mathbf{P}_1, \mathbf{K}_1, \mathbf{R}_1 \rangle$.

5.4 *disjoint-partition()*: Disjoint Partition of a Conditional Core Subgraph

The algorithm to compute disjoint partition is straightforward, and it is given in Algoorithm 7. For a CCS $\mathbf{G}_{\mathbf{R}}$ conditioned on $\mathbf{G}_{\mathbf{S}}$, it first computes the set $E_{\mathbf{R}}$, which contains all hyperedges contributing to the reward of the vertices in $\mathbf{R}$,

then iterates according to the following principle: the vertices of $\mathbf{R}$ in the same hyperedge in $E_{\mathbf{R}}$ belong to the same subgraph. This procedure is very efficient, with time complexity being linear in $|E_{\mathbf{R}}|$.

Algorithm 7. *disjoint-partition()*

1: **Input:** $\mathbf{G_R}$ and $\mathbf{G_S}$.
2: Construct the edge set $E_{\mathbf{R}}$;
3: Set $\mathrm{DP}(\mathbf{G_R} \mid \mathbf{G_S}) = \beth = \emptyset$. Consider each vertex in $\mathbf{R}$ as a set and add it into $\beth$.
4: **for** each $e \in E_{\mathbf{R}}$ **do**
5: Merge all sets in $\beth$ which contain vertices in e into one set;
6: **end for**
7: For each vertex set $U \in \beth$, add the subgraph $\mathbf{G}_U$ into $\mathrm{DP}(\mathbf{G_R} \mid \mathbf{G_S})$.
8: **Output:** $\mathrm{DP}(\mathbf{G_R} \mid \mathbf{G_S})$.

5.5 Convergence Analysis

In this section, we prove that Algorithm 1 converges after finite iterations and the output is $\mathrm{DSP}(\mathbf{G})$.

First, we prove the relation between min-partition and DSP.

Theorem 5. *For every* $\mathbf{P} \in \Theta(\mathbf{G})$, *if* $\mathrm{MP}(\mathbf{P}) = \langle \mathbf{P}_1, \ldots, \mathbf{P}_m \rangle$, *then we have:*

$$\mathrm{DSP}(\mathbf{G}) = \langle \mathrm{DP}(\mathbf{G}_{\mathbf{P}_i} | \mathbf{G}_{[\mathbf{P}^F, \mathbf{P}_i^F)}) | i = 1, \ldots, m \rangle. \quad (7)$$

Proof. Please see Supplement Material, available online. □

Theorem 5 tells us that if a permutation $\mathbf{P} \in \Theta(\mathbf{G})$ is known, we can efficiently obtain $\mathrm{DSP}(\mathbf{G})$ by min-partition. Note that only one permutation in $\Theta(\mathbf{G})$ is needed, although $\Theta(\mathbf{G})$ contains a huge number of permutations.

Second, we define an order over min-partitions and prove that the min-partitions of all permutations in $\Theta(\mathbf{G})$ have maximum order.

From two permutations of V, $\mathbf{P}$ and $\mathbf{R}$, we have two min-partitions, $\mathrm{MP}(\mathbf{P}) = \langle \mathbf{P}_1, \ldots, \mathbf{P}_{m_1} \rangle$ and $\mathrm{MP}(\mathbf{R}) = \langle \mathbf{R}_1, \ldots, \mathbf{R}_{m_2} \rangle$. We define an order between them, with $\mathrm{MP}(\mathbf{P}) >: \mathrm{MP}(\mathbf{R})$, $\mathrm{MP}(\mathbf{P}) =: \mathrm{MP}(\mathbf{R})$ and $\mathrm{MP}(\mathbf{P}) <: \mathrm{MP}(\mathbf{R})$ represent the order of $\mathrm{MP}(\mathbf{P})$ is larger than, equal to, and smaller than the order of $\mathrm{MP}(\mathbf{R})$, respectively.

Let $m = \min\{m_1, m_2\}$. We compare $\mathbf{P}_i$ and $\mathbf{R}_i$ with i increasing from 1 to m to find the smallest i such that either 1) $\overline{m}(\mathbf{P}_i) \neq \overline{m}(\mathbf{R}_i)$ or 2) $|\mathbf{P}_i| \neq |\mathbf{R}_i|$. If such i exists, then we define

$$\begin{cases} \mathrm{MP}(\mathbf{P}) >: \mathrm{MP}(\mathbf{R}), & \overline{m}(\mathbf{P}_i) > \overline{m}(\mathbf{R}_i); \\ \mathrm{MP}(\mathbf{P}) <: \mathrm{MP}(\mathbf{R}), & \overline{m}(\mathbf{P}_i) < \overline{m}(\mathbf{R}_i); \\ \mathrm{MP}(\mathbf{P}) >: \mathrm{MP}(\mathbf{R}), & \overline{m}(\mathbf{P}_i) = \overline{m}(\mathbf{R}_i), |\mathbf{P}_i| > |\mathbf{R}_i|; \\ \mathrm{MP}(\mathbf{P}) <: \mathrm{MP}(\mathbf{R}), & \overline{m}(\mathbf{P}_i) = \overline{m}(\mathbf{R}_i), |\mathbf{P}_i| < |\mathbf{R}_i|. \end{cases}$$

If such i does not exist, then we define $\mathrm{MP}_{\mathbf{P}}(\mathbf{G}) =: \mathrm{MP}_{\mathbf{R}}(\mathbf{G})$.

It is easy to verify that when $\mathbf{P} \in \Theta(\mathbf{G})$ and $\mathbf{R} \in \Theta(\mathbf{G})$, $\mathrm{MP}(\mathbf{P}) =: \mathrm{MP}(\mathbf{R})$. Moreover, we have the following important Theorem.

Theorem 6. *For two permutations,* $\mathbf{P}$ *and* $\mathbf{R}$, *if* $\mathbf{P} \in \Theta(\mathbf{G})$ *and* $\mathbf{R} \notin \Theta(\mathbf{G})$, *then* $\mathrm{MP}(\mathbf{P}) >: \mathrm{MP}(\mathbf{R})$.

Proof. Please see Supplement Material, available online. □

Theorem 6 tells us that if and only if $\mathbf{P} \in \Theta(\mathbf{G})$, the order of $\mathrm{MP}(\mathbf{P})$ reaches maximum. Thus, a practical strategy to approach a permutation in $\Theta(\mathbf{G})$ is to iteratively modify the current permutation $\mathbf{P}$ to a new permutation $\hat{\mathbf{P}}$ such that $\mathrm{MP}(\hat{\mathbf{P}}) >: \mathrm{MP}(\mathbf{P})$, and this is exactly what Algorithm 1 does.

Third, we prove that each iteration (except the last iteration) of Algorithm 1 (from step 5 to step 15) increases the order of $\mathbf{P}$.

According to Algorithm 6, only when $\mathbf{R}$ is divisible, we replace $\mathbf{R}$ by $\hat{\mathbf{R}}$ in step 10; thus, in Algorithm 1, $\mathbf{P}$ and $\hat{\mathbf{P}}$ are different if and only if some MMSPs in $\mathrm{MM}(\mathbf{P})$ are divisible. When $\mathbf{R}$ is divisible and we replace it by $\hat{\mathbf{R}}$, we have the following important result.

Theorem 7. *Suppose* $\mathbf{R}$ *is a MMSP of* $\mathbf{P}$ *and we replace* $\mathbf{R}$ *by* $\hat{\mathbf{R}}$ *(thus change* $\mathbf{P}$ *to* $\hat{\mathbf{P}}$ *accordingly), if* $|\mathrm{MP}(\hat{\mathbf{R}})| > 1$, *then* $\mathrm{MP}(\hat{\mathbf{P}}) >: \mathrm{MP}(\mathbf{P})$.

Proof. Please see Supplement Material, available online. □

Changing multiple MMSPs in $\mathbf{P}$ simultaneously is equivalent to changing them one by one. Thus, if some MMSPs of $\mathbf{P}$ are divisible, the iteration in Algorithm 1 increases the order of $\mathbf{P}$.

Fourth, we show that all permutations in $\Theta(\mathbf{G})$ are indivisible; while all permutations not in $\Theta(\mathbf{G})$ are divisible.

Theorem 8. *For a permutation* $\mathbf{P}$, $\mathbf{P} \in \Theta(\mathbf{G})$ *if and only if all MMSPs in* $\mathrm{MP}(\mathbf{P})$ *are indivisible.*

Proof. Please see Supplement Material, available online. □

Theorem 8 tells us that if $\mathbf{P} \notin \Theta(\mathbf{G})$, then in the min-partition $\mathrm{MP}(\mathbf{P})$, at least one MMSP is divisible. Thus, by applying *permutation-reorder()* on all MMSPs of $\mathbf{P}$, we know whether $\mathbf{P} \in \Theta(\mathbf{G})$ or not.

Finally, we prove that Algorithm 1 converges in finite iterations and the output is $\mathrm{DSP}(\mathbf{G})$.

According to Theorem 8, only when a permutation $\mathbf{P} \in \Theta(\mathbf{G})$ is found, Algorithm 1 terminates. Since each iteration of Algorithm 1 increases the order of $\mathbf{P}$ and the total number of permutations is finite ($|V|!$), Algorithm 1 is guaranteed to terminate after finite iterations and to reach a $\mathbf{P} \in \Theta(\mathbf{G})$. According to Theorem 5, the output is $\mathrm{DSP}(\mathbf{G})$.

5.6 Complexity Analysis

In each iteration of Algorithm 1, there are two basic operations, *permutation-reorder()* and *min-merge()*. The time complexity of *min-merge()*, that is, Algorithm 3, is at most $O(|\Omega|)$. In Algorithm 6, Algorithm 5 is first called, whose time complexity is $O(d|E_{\mathbf{R}}| + |\mathbf{R}|\log(|\mathbf{R}|))$, where d is the average degree of the hyperedges in $E_{\mathbf{R}}$. If Algorithm 5 does not divide $\mathbf{R}$, then Algorithm 4 is called. In Algorithm 4, the main computational burden is to solve the optimization problem $\max F(\mathbf{z})$ using the QPBO algorithm. Since the number of variables in $\mathbf{z}$ is approximately $|\mathbf{R}| + |E_{\mathbf{R}}|$ and the number of quadratic terms in $F(\mathbf{z})$ is approximately $d|E_{\mathbf{R}}|$, the time complexity of Algorithm 4 is $O(d|E_{\mathbf{R}}|(|\mathbf{R}| + |E_{\mathbf{R}}|)^2))$. In both Algorithm 5 and Algorithm 4, Algorithm 2 is called, whose time complexity is $O(|\mathbf{P}|)$. The time complexity of Algorithm 7 is $O(|E_{\mathbf{R}}|)$, that is, linear in the number of

hyperedges in $E_{\mathbf{R}}$. Note that Algorithm 5 can divide $\mathbf{R}$ most of the time, especially when $\mathbf{R}$ is large. Thus, the overall time complexity of Algorithm 1 is approximately $O(\tau d n_e (n_v + n_e)^2)$, where τ is the number of iterations in Algorithm 1, and n_v and n_e are number of vertices and hyperedges, respectively, of the dense subgraph with largest size.

6 Relation to Densest k-Subgraph

For a graph G, the densest k-subgraph problem (DkS) is to find a subgraph with k vertices, whose total weight of edges is maximum among all subgraphs of G with k vertices. This is a fundamental but notoriously hard problem in graph theory, generally known as NP-hard [7]. However, we will show that for a large number of ks, DkS can be solved precisely and efficiently.

Based on $\mathrm{DSP}(\mathbf{G})$, we can define an integer set, called critical k-set.

Definition 5. *When $\Psi(V) = \langle V_1, \ldots, V_m \rangle$, the **critical k-set** is defined to be $\kappa(\mathbf{G}) = \{k | \exists i \in \{1, \ldots, m\}, \exists U \subseteq 2^{\Gamma(V_i)}, U \neq \emptyset, k = \sum_{j=1}^{i-1} |V_j| + \sum_{e \in U} |e|\}$. Here 2^S represents the power set of S, Ψ is the first layer partition of V in DSP and $\Gamma(V_i)$ is the disjoint partition of V_i.*

Since all components in $\Gamma(V_i)$ are exchangeable, for any $U \subseteq 2^{\Gamma(V_i)}$, we can rearrange $\Gamma(V_i)$ to put all components in U at the front, then k is the number of vertices in $\{V_1, \ldots, V_{i-1}, U\}$ and $\kappa(\mathbf{G})$ contains all such possible k.

For example, for the graph in Fig. 1, we have $\Gamma(V_1) = V_1$, $\Gamma(V_2) = \{V_4, V_5\}$ and $\Gamma(V_3) = \{V_6, V_7\}$. Thus, $2^{\Gamma(V_1)} = \{\emptyset, V_1\}$, $2^{\Gamma(V_2)} = \{\emptyset, V_4, V_5, \{V_4, V_5\}\}$ and $2^{\Gamma(V_3)} = \{\emptyset, V_6, V_7, \{V_6, V_7\}\}$. The critical k-set of $\mathbf{G}$ is $\kappa(\mathbf{G}) = \{4, 7, 10, 11, 12\}$. $7 \in \kappa(\mathbf{G})$ because when $i = 2$ and $U = V_4$ or V_5, $|V_1| + |V_4| = 7$ or $|V_1| + |V_5| = 7$.

The following theorem connects DkS and $\mathrm{DSP}(\mathbf{G})$.

Theorem 9. *For each $k \in \kappa(\mathbf{G})$, $\mathrm{DSP}(\mathbf{G})$ gives precise solution to DkS. More specifically, if $U \subseteq 2^{\Gamma(V_i)}, U \neq \emptyset$ and $k = \sum_{j=1}^{i-1} |V_j| + \sum_{e \in U} |e|$, then $\mathbf{G}_{\{V_1, \ldots, V_{i-1}, U\}}$ is a densest k-subgraph of $\mathbf{G}$.*

Proof. Please see Supplement Material, available online. □

This is a strong theoretic result on DkS. It tells us that the precise solutions of DkS for $k \in \kappa(\mathbf{G})$ can be obtained efficiently. Note that $\kappa(\mathbf{G})$ is only a subset of $\{1, \ldots, |V|\}$, thus, for k not in $\kappa(\mathbf{G})$, the exact DkS cannot be obtained by our algorithm. Also there is not a universal k for all hypergraphs such that their DkSs can be found. For the graph in Fig. 1, since $\kappa(\mathbf{G}) = \{4, 7, 10, 11, 12\}$, we can get: the densest four-subgraph of $\mathbf{G}$ is $\mathbf{G}_{V_1}$, the densest seven-subgraph of $\mathbf{G}$ are $\mathbf{G}_{V_1 \cup V_4}$ and $\mathbf{G}_{V_1 \cup V_5}$, the densest 10-subgraph of $\mathbf{G}$ is $\mathbf{G}_{V_1 \cup V_4 \cup V_5}$, and the densest 11-subgraph of $\mathbf{G}$ are $\mathbf{G}_{V_1 \cup V_4 \cup V_5 \cup V_6}$ and $\mathbf{G}_{V_1 \cup V_4 \cup V_5 \cup V_7}$. $k = 5$ does not belong to $\kappa(\mathbf{G})$; however, we may obtain a good approximation of the densest five-subgraph of $\mathbf{G}$ based on the densest four-subgraph and the densest seven-subgraph of $\mathbf{G}$.

Although DSP decomposes $\mathbf{G}$ into many subgraphs, its relation with DkS shows that these subgraphs can be pieced up to form large globally optimal clusters.

7 Objective, Strengths and Limitations

Unlike many other partition methods, DSP lacks a global objective function, which leads to some difficulties in understanding its overall picture.

According to Theorem 9, the objective of DSP can be described as follows: partition $\mathbf{G}$ into ordered subgraphs $\langle \mathbf{G}_{V_1}, \ldots, \mathbf{G}_{V_m} \rangle$ such that $\mathbf{G}_{V_1}$ is the densest $|V_1|$-subgraph, $\mathbf{G}_{V_1 \cup V_2}$ is the densest $|V_1 \cup V_2|$-subgraph, and so on. Of course, we cannot formulate DSP by this objective, since the values of $|V_1|, \ldots, |V_m|$ are not known before partition. However, this description gives us some insights into DSP. As mentioned before, the sequence $\langle V_1, \ldots, V_m \rangle$ defines a partial order over V, then we can also describe the objective inaccurately as follows: find a permutation of all vertices such that the connections among the front part of the permutation are as strong as possible. Different to cut-based partition methods, the connections between different subgraphs in DSP are not necessarily weak, since multiple subgraphs may belong to the same cluster and uncover the internal structure inside this cluster. Of course, there are some important relations. For example, the average connection within $V_1 \cup V_2$ and the average connection between V_2 and V_1, are weaker than the average connection within V_1.

A subgraph with strong connections among its vertices usually forms a meaningful cluster, thus, DSP can be considered as a process of detecting one meaningful cluster at multiple scales: first $\mathbf{G}_{V_1}$, then $\mathbf{G}_{V_1 \cup V_2}, \ldots$, finally the whole graph $\mathbf{G}$. This is closely related to the one-class problem [34]. Here the obtained meaningful cluster may in fact contain multiple real clusters. Since vertices not in the obtained meaningful cluster are considered to be outliers, DSP can also be regarded as a process of identifying outliers at multiple scales.

A significant strength of DSP is to detect clusters and identify outliers simultaneously, and at multiple scales. This is in sharp contrast to existing approaches, where clustering and outlier detection are usually separated. The strength of DSP makes it a powerful tool to detect meaningful clusters in a dataset with massive outliers. Besides, DSP is precise, thus has guaranteed performance, and it is also efficient, with the ability to partition very large hypergraphs.

The main limitation of DSP comes from its definition of density, which is the ratio between total weight and the number of vertices. However, the total weight depends on the number of hyperedges, which grows much faster than the number of vertices on dense hypergraphs. Thus, on dense hypergraphs, dense subgraphs tend to be very large and therefore cannot reveal the underlying cluster structure. Besides, DSP cannot partition a hypergraph into a specified number of subgraphs and this is not desirable in some applications.

8 Experiments

All the experiments are done on a regular PC with Intel Core 2 Quad CPU and 4GB memory. Since our implementation is single-threaded, only one CPU is used at a time. For the initial permutation $\mathbf{P}$, we compute it by applying Algorithm 5 on the whole graph (without executing Step 7), which is usually better than random permutations.

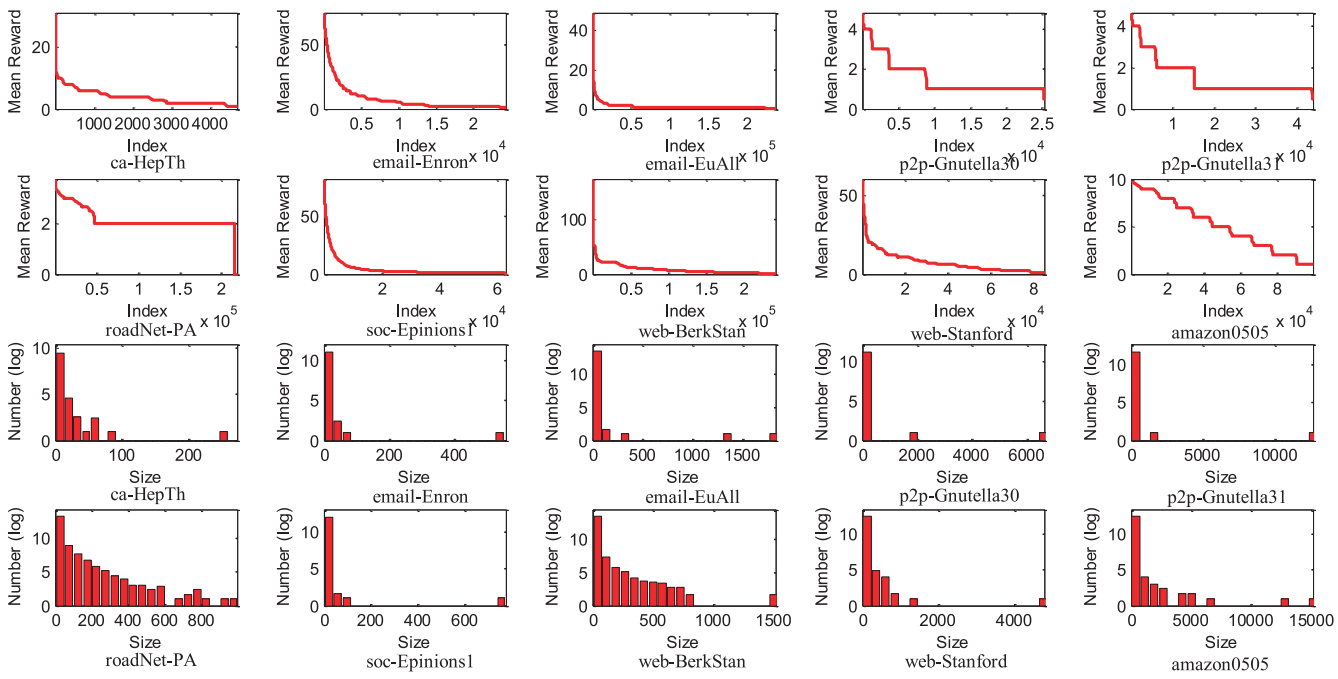

Fig. 4. Illustrations of the statistics of subgraphs obtained by DSP.

8.1 Partition of Networks

In this section, we do experiments on ten networks from Stanford Large Network Dataset Collection,[7] listed in Table 1 together with their statistics.

In the top two rows of Fig. 4, the mean rewards as functions of the index of subgraphs are illustrated. Clearly, the mean reward is non-increasing. In the bottom two rows, x-axis is the size of subgraphs, and y-axis is the number of subgraphs whose sizes are in a range centered at corresponding x.[8] This figure reveals some interesting phenomenon. First, the figures of similar networks are similar. For example, two Internet peer-to-peer networks (*p2p-Gnutella30* and *p2p-Gnutella31*), two communication networks (*email-Enron* and *email-EnAll*) and two Web graphs (*web-BerkStan* and *web-Stanford*), have very similar curves. Second, five graphs, namely, *email-Enron*, *email-EnAll*, *p2p-Gnutella30*, *p2p-Gnutella31* and *soc-Epinions1*, are composed by a few large dense subgraphs, together with many scattered nodes; the other five graphs are composed by dense subgraphs of various sizes.

As discussed in Section 6, DSP yields precise solution to DkS for all $k \in \kappa(\mathbf{G})$. Four statistics of DSP results on these ten networks are listed in Table 2, namely, the number of components $|\mathrm{DSP}(\mathbf{G})|$, the size of critical k-set $|\kappa(\mathbf{G})|$, the ratio $\frac{|\kappa(\mathbf{G})|}{|V|}$ and the time used for partition. Note that in the process of computing $\kappa(\mathbf{G})$, for each $\Gamma(V_i)$, we need to enumerate all possible sizes of the subsets of $2^{\Gamma(V_i)}$. When $|\Gamma(V_i)|$ is large, this is very time consuming. Therefore, when $|\Gamma(V_i)|$ is large, we only sample a few subsets of $2^{\Gamma(V_i)}$ to compute a subset of ks, thus the obtained $\kappa(\mathbf{G})$ is only a subset of real $\kappa(\mathbf{G})$. This is why in the two columns corresponding to $|\kappa(\mathbf{G})|$ and $\frac{|\kappa(\mathbf{G})|}{|V|}$, we add $\geq$ to all values.

From both Tables 1 and 2, we have the following observations. First, DSP is very efficient. For all graphs whose number of edges is below one million, the computing time is less than 10 seconds; for graphs with millions of edges, the time is only a few minutes. Second, DSP decomposes graphs into many small components. However, based on these small components, we can piece up large dense clusters, such as precise densest k-subgraph for large ks in the critical k-set. Third, the size of critical k-set is very large, compared with the number of vertices. In fact, critical k-set is a dense sampling of the set $\{1, \ldots, |V|\}$. On some graphs, such as *email-EuAll* and *soc-Epinions1*, the ratio $\frac{|\kappa(\mathbf{G})|}{|V|}$ is even larger than 90 percent. The large value of $\frac{|\kappa(\mathbf{G})|}{|V|}$ means that our result in Section 6 is really useful in practical applications: for a specified k, it has a large probability to belong to $\kappa(\mathbf{G})$.

We compare DSP with two other efficient methods, namely, Feige's method [7] and truncated power method (TP) [35]. The source codes of these two methods were obtained from web.[9] Both of them are heuristic-based methods, Feige's method relies on the degrees of vertices and truncated power method utilizes the power iteration. The truncated power method is the state-of-the-art method to solve DkS.

For each graph, we select ten ks in its $\kappa(\mathbf{G})$, and then compute DkSs by all three methods. The "goodness" of a subgraph is measured by its *total weight*, which is defined as the sum of the weights of all edges in this subgraph. Fig. 5 shows the total weight of detected dense subgraphs versus the cardinality k. Our approach consistently

7. http://snap.stanford.edu/data/

8. y-axis is in log space, and to show the value 1 whose logarithm is 0, after logarithm, we add 1 to all y-values.

9. https://sites.google.com/site/xtyuan1980

TABLE 1
The Statistics of 10 Networks Used in Our Experiments

Graph	Type	Vertices($\lvert V\rvert$)	Arcs($\lvert E\rvert$)
ca-HepTh	Undirected	9,877	51,971
email-Enron	Undirected	36,692	367,662
email-EuAll	Directed	265,214	420,045
p2p-Gnutella30	Directed	36,682	88,328
p2p-Gnutella31	Directed	62,586	147,892
roadNet-PA	Undirected	1,088,092	3,083,796
soc-Epinions1	Directed	75,879	508,837
web-BerkStan	Directed	685,230	7,600,595
web-Stanford	Directed	281,903	2,312,497
amazon0505	Directed	410,236	3,356,824

TABLE 2
Statistics of DSP on 10 Networks

Graph	$\lvert\mathrm{DSP}(\mathbf{G})\rvert$	$\lvert\kappa(\mathbf{G})\rvert$	$\frac{\lvert\kappa(\mathbf{G})\rvert}{\lvert V\rvert}$	Time(s)
ca-HepTh	4,671	$\geq$6,475	$\geq$65.6%	0.2267
email-Enron	24,366	$\geq$28,566	$\geq$77.9%	2.0768
email-EuAll	237,337	$\geq$239,642	$\geq$90.4%	4.0250
p2p-Gnutella30	25,320	$\geq$26,489	$\geq$72.2%	0.5408
p2p-Gnutella31	43,889	$\geq$45,940	$\geq$73.4%	1.6561
roadNet-PA	219,264	$\geq$279,685	$\geq$25.7%	48.6139
soc-Epinions1	63,021	$\geq$69,662	$\geq$91.8%	5.4625
web-BerkStan	241,972	$\geq$306,580	$\geq$44.7%	153.6173
web-Stanford	85,040	$\geq$108,710	$\geq$38.6%	52.1276
amazon0505	99,810	$\geq$122,226	$\geq$29.8%	73.4478

outperforms other two methods on all graphs, since our method gives precise DkS. Truncated power method performs better than Feige's method on most of graphs, except for *web-BerkStan*.

8.2 Cluster Enumeration on Affinity Graphs

In this section, we conduct experiment on the UCI Handwritten Digits Data Set. In this dataset, there are $5,620$ instances of 10 digits. Every instance is encoded in a 64-dimensional vector, with each dimension being the number of "on" pixels in a 4×4 patch. That is, the value of each dimension being an integer value in $\{0, \ldots, 16\}$. We randomly generate $4{,}380$ outliers, each dimension of which follows the same distribution as digits. Thus, we get a dataset with $10{,}000$ instances in total.

From this dataset, we construct an affinity graph $\mathbf{G}$ as follows: each instance forms a vertex, and the weight of the edge between the instance s_i and s_j is defined to be $w(s_i, s_j) = \exp(-\frac{d^2(s_i,s_j)}{20^2})$, where $d(s_i, s_j)$ is the Euclidean distance between s_i and s_j. Our goal is to automatically discover all significant dense subgraphs in $\mathbf{G}$, that is, all significant modes of this dataset [23], [24].

First, we consider all ten clusters as a large cluster, the "digit" cluster, and illustrate the performance of our method in separating inliers and outliers. Based on a permutation $\mathbf{P} \in \Theta(\mathbf{G})$, we plot a Precision-Recall curve, which is demonstrated in Fig. 6a, and also illustrate the label distribution along this permutation, which is demonstrated in Fig. 6b. From Fig. 6a, we found that DSP preforms excellently in separating inliers and outliers; while from Fig. 6b, we found that DSP clearly reveals all 10 meaningful clusters. Here we emphasize that these two tasks are done simultaneously.

Second, we compare with four methods, spectral clustering (SC) [36], power iteration clustering (PIC)[37], dominant set (DS) [26] and graph shift (GS) [23], [24]. SC and PIC are partition methods; while DS and GS are methods to detect clique-like clusters. Both SC and PIC require the number of clusters as input, and we use three values, 11, 20 and 40. For DS, as suggested in [26], we iteratively detect dense clusters. To measure the performance of a method, we utilized two novel measures, ξ_r-Precision and ξ_r-Recall, which are defined as follows: for each class, in the detected clusters, find the compositive cluster with highest F-measure and consisting of no more than r original clusters; the average precision and recall of such compositive clusters for all classes is the ξ_r-Precision and ξ_r-Recall, respectively. Clearly, ξ_1-Precision and ξ_1-Recall means that for each class, we only select one detected cluster. In the ideal case, this cluster should be identical to that class. However, some classes may have internal structure and thus been divided into

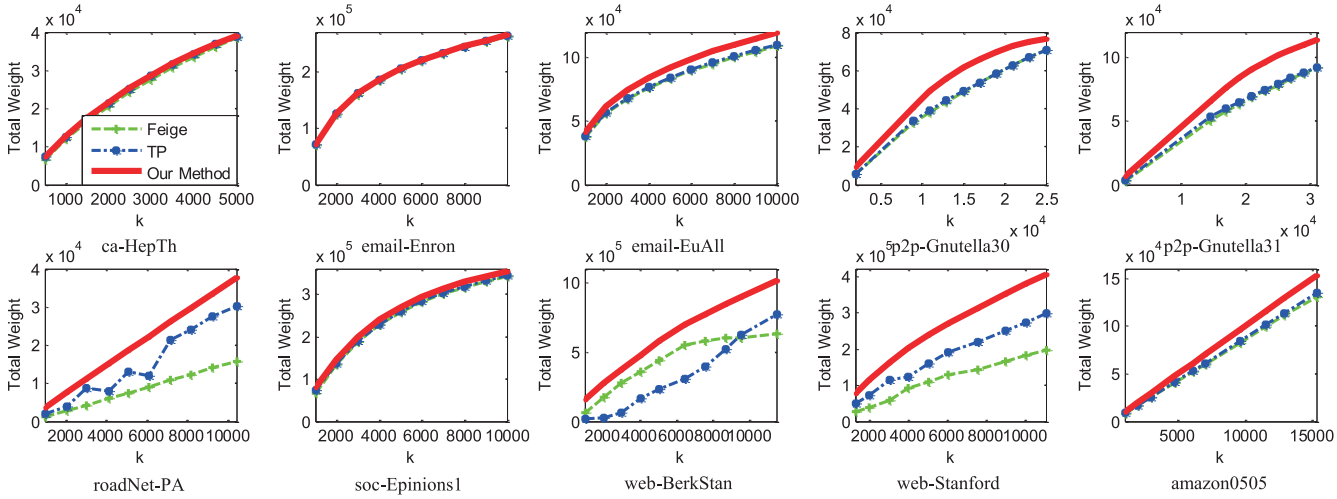

Fig. 5. The results of DkS on 10 webgraphs. Feige's method is shown in green dotted curve, the truncated power method is shown in blue dashdot curve, and our method is shown in red solid curve. This figure is best viewed in color.

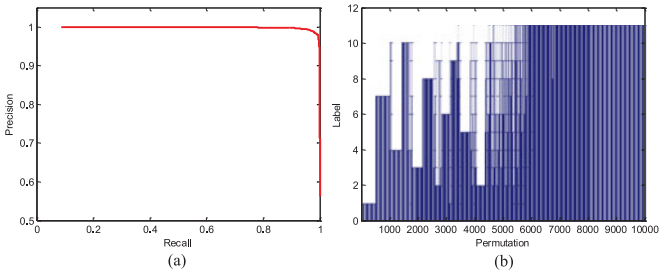

Fig. 6. (a) The Precision-Recall curve from a $\mathbf{P} \in \Theta(\mathbf{G})$, (b) the label distribution along $\mathbf{P}$, where the label 11 representing outliers.

multiple clusters, and these clusters can be easily merged into a large cluster by post-processing. In such case, the ξ_r-Precision and ξ_r-Recall with $r > 1$ may better measure the performances. Of course, r should not be too large since this adds difficulties in merging small clusters into large clusters. The results of all four methods are shown in Table 3, where the measures under $r = 1$ and $r = 10$ are reported. Our method successfully discovers all ten clusters, which can be seen from its high ξ_1-Precision and good ξ_1-Recall. Since both DS and GS detect clique-like clusters, they only extract a very small subset of each real cluster, thus have high ξ_1-Precisions but low ξ_1-Recalls. Their ξ_{10}-Precisions are still high and their ξ_{10}-Recalls are much better. This is a real cluster has been divided into several sub-clusters. Both SC and PIC divide the whole graph into the specified number of subgraphs. As expected, their ξ_1-Precisions improve as the number of classes increases, since more classes can be used to accommodate the outliers, but their ξ_1-Recalls go down. Note that their ξ_1-Precisions are very low when $k = 11$, which is the actual number of classes (ten true clusters plus outlier cluster). Strictly speaking, only our method has the ability to correctly detect all ten clusters. The other methods either divide a real cluster into too many sub-clusters (DS, GS) or inherently do not identify outliers (SC, PIC). As for the time complexity, PIC is the fastest, then our method, both of them are mush faster than the other three methods.

8.3 Image Matching via Hypergraphs

In recent years, hypergraph based matching methods become popular, due to their flexility and good performance [23], [28], [38], [39], [40]. However, constructing hypergraph is a severe computational burden, since the number of hyperedges is usually huge. In an image, an object only occupies a local region, thus, we can construct the hypergraph locally to greatly reduce the number of hyperedges.

In the first column of Fig. 7, there are two images with logos of multiple credit cards. Our task is to discover all possible matchings between them. By finding similar SIFT interest points in two images [41], 3,532 correspondences are detected, among them only 193 correspondences are correct. In this experiment, we only consider similarity transformations. Thus, the order of a hyperedge is 3, and the total number of hyperedges is then $\binom{3532}{3}$, a huge number. To reduce the number of hyperedges, we construct the hypergraph in the following way: for three correspondences (p_1, q_1), (p_2, q_2) and (p_3, q_3), where p_i is a point in the first image and q_i is its corresponding point in the second image, only when $d(p_i, p_j) < 40$ for all $i, j \in \{1, 2, 3\}$, we add a hyperedge formed by these three correspondences, where $d(p_i, p_j)$ is the Euclidean distance in pixels between the point p_i and p_j, and the weight of this hyperedge is computed using the method in [39]. The obtained hypergraph has only 17,544 hyperedges, which is very small compared to $\binom{3532}{3}$. Obviously, a

TABLE 3
Result of Cluster Detection on Handwritten Dataset

Method	SC			PIC			DS	GS	DSP
	11	20	40	11	20	40			
ξ_1-Precision (%)	52.89	75.18	90.15	74.56	83.37	82.06	100	100	94.38
ξ_1-Recall (%)	95.83	83.73	79.14	84.32	75.59	78.68	6.17	8.23	78.26
ξ_{10}-Precision (%)	52.88	72.82	88.30	71.75	81.10	81.15	99.9	99.91	92.77
ξ_{10}-Recall (%)	97.68	91.73	91.36	89.94	88.44	91.12	40.26	24.17	89.13
Time (s)	1500.9	1504.4	1495	0.8857	3.0570	1.4311	358.6	263.2	20.76

Fig. 7. The experimental results of image matching. The first column shows two images to be matched, there is a one-to-one matching (American Express), a one-to-two matching (MasterCard) and a two-to-one matching (Visa). The second, third, fourth, fifth and sixth column show the matching results of DSP, CG, TM, HGM and RRWHM, respectively. Green dots are interest points, lines represent correspondences. CG and our method can distinguish different matchings, therefore their correspondences in different matchings are shown in different colors; TM, HGM and RRWHM only detect correct correspondences, thus there correspondences are only shown in blue color.

TABLE 4
Performances in the Image Matching Experiments

Method	hMETIS				CG	DSP
	6	20	100	1000		
ξ_1-Precision(%)	6.81	22.93	51.54	95.56	100	100
ξ_1-Recall(%)	100	99.78	83.16	29.47	33.09	68.27
ξ_{10}-Precision(%)	6.81	22.93	48.72	93.84	100	100
ξ_{10}-Recall(%)	100	99.78	90.12	97.49	95.44	96.17
Time(s)	2.836	5.527	8.373	11.582	419.06	0.3367

correct matching should form a dense subgraph, and we can find all matchings by enumerating all dense subgraphs.

We compare our method with five other methods, namely, hMETIS [42], clustering game (CG) [28], tensor matching (TM) [39], hypergraph matching (HGM) [38] and re-weighted random walk hypergraph matching (RRWHM) [40]. hMETIS divides a hypergraph into a specified number of parts. CG is a generalization of the dominant set method to hypergraphs and it can only detect clique-like clusters. TM, HGM and RRWHM are matching methods, with the assumption that each point in the first image has only one correspondence in the second image. The results are shown in Fig. 7 and Table 4. Note that the shape of each real cluster is complex, since the hypergraph has been constructed locally. From Fig. 7, we find that our method correctly detect all matchings; while CG method detects many small clique-like clusters. TM performs well, however, its matchings of Visa and MasterCard only consists of a part of correct correspondences. Both HGM and RRWHM perform badly, especially RRWHM, which only finds one matching. For hMETIS, according to Table 4, when $k = 6$, the performance is very bad. This is because its goal is to minimize the cuts, which is dramatically affected by outliers. Only when k is very large, such as $1{,}000$, some clusters indicate real matchings, at the cost of a real matching is divided into multiple clusters.

9 Conclusion

In this paper, DSP is proposed, along with an efficient algorithm to compute it. DSP partitions a positive hypergraph into many dense subgraphs, thus reveals the cluster structure underlying the hypergraph in a bottom-up way, and at the same time, correctly identifies outliers. DSP is very useful, both in theory and in practical applications. Due to proposed efficient divide-and-conquer algorithm, DSP scales very well so that large hypergraphs can be precisely and quickly partitioned.

Acknowledgments

This work was in part supported by National Science Foundation (NSF) under Grants OIA-1027897 and IIS-1302164, and also partially supported by Singapore Ministry of Education under research Grant MOE2010-T2-1-087.

References

[1] P. Fjällström, "Algorithms for graph partitioning: A survey," *Comput. Inf. Sci.*, vol. 3, no. 10, pp. 143–179, 1998.
[2] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning: Application in vlsi domain," in *Proc. 34th Annu. Des. Autom. Conf.*, 1997, pp. 526–529.
[3] K. Andreev and H. Racke, "Balanced graph partitioning," *Theory Comput. Syst.*, vol. 39, no. 6, pp. 929–939, Aug. 2006.
[4] M. Newman, "Modularity and community structure in networks," *Proc. Nat. Acad. Sci.*, vol. 103, no. 23, pp. 8577–8582, 2006.
[5] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 8, pp. 888–905, Aug. 2000.
[6] S. Khuller and B. Saha, "On finding dense subgraphs," in *Proc. Automata Languages Program.*, 2009, pp. 597–608.
[7] U. Feige, G. Kortsarz, and D. Peleg, "The dense k-subgraph problem," *Algorithmica*, vol. 29, pp. 410–421, 2001.
[8] B. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell Syst. Tech. J.*, vol. 49, pp. 291–307, 1970.
[9] D.-H. Huang and A. B. Kahng, "When clusters meet partitions: New density-based methods for circuit decomposition," in *Proc. Eur. Conf. Des. Test*, 1995, pp. 60–64.
[10] I. Dhillon, Y. Guan, and B. Kulis, "Kernel k-means: spectral clustering and normalized cuts," in *Proc. ACM Int. Conf. Knowl. Discov. Data Min.*, 2004, pp. 551–556.
[11] V. Kolmogorov and R. Zabin, "What energy functions can be minimized via graph cuts?" *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 26, no. 2, pp. 147–159, Jun. 2004.
[12] C. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*. Mineola, NY, USA: Dover, 1998.
[13] J. H. Kappes, M. Speth, B. Andres, G. Reinelt, and C. Schn, "Globally optimal image partitioning by multicuts," in *Proc. Energy Minim. Methods Comput. Vis. Pattern Recognit.*, 2011, pp. 31–44.
[14] C. Fiduccia and R. Mattheyses, "A linear-time heuristic for improving network partitions," in *Proc. 19th Conf. Des. Autom.*, 1982, pp. 175–181.
[15] D. Zhou, J. Huang, and B. Schölkopf, "Learning with hypergraphs: Clustering, classification, and embedding," in *Proc. Adv. Neural Inf. Process. Syst.*, 2006, pp. 1601–1608.
[16] J. Rodríguez, "Laplacian eigenvalues and partition problems in hypergraphs," *Appl. Math. Letters*, vol. 22, no. 6, pp. 916–921, 2009.
[17] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM J. Sci. Comput.*, vol. 20, no. 1, pp. 359–392, 1998.
[18] N. Bansal, A. Blum, and S. Chawla, "Correlation clustering," *Mach. Learn.*, no. 1-3, vol. 56, pp. 89–113, 2004.
[19] D. Emanuel and A. Fiat, "Correlation clustering–minimizing disagreements on arbitrary weighted graphs," in *Proc. 11th Annu. Eur. Symp. Algorithms*, 2003, pp. 208–220.
[20] S. Kim, S. Nowozin, P. Kohli, and C. D. Yoo, "Higher-order correlation clustering for image segmentation," in *Proc. Adv. Neural Inf. Process. Syst.*, 2011, pp. 1530–1538.
[21] P. F. Felzenszwalb and D. P. Huttenlocher, "Efficient graph-based image segmentation," *Int. J. Comput. Vis.*, vol. 59, no. 2, pp. 167–181, 2004.
[22] D. Gibson, R. Kumar, and A. Tomkins, "Discovering large dense subgraphs in massive graphs," in *Proc. Int. Conf. Very Large Data Bases*, 2005, pp. 721–732.
[23] H. Liu and S. Yan, "Robust graph mode seeking by graph shift," in *Proc. Int. Conf. Mach. Learn.*, 2010, pp. 671–678.
[24] H. Liu, L. Latecki, and S. Yan, "Fast detection of dense subgraph with iterative shrinking and expansion," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 9, pp. 2131–2142, 2013.
[25] J. Chen and Y. Saad, "Dense subgraph extraction with application to community detection," *IEEE Trans. Knowl. Data Eng.*, vol. 24, no. 7, pp. 1216–1230, Jul. 2012.
[26] M. Pavan and M. Pelillo, "Dominant sets and pairwise clustering," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, no. 1, pp. 167–172, Jan. 2007.
[27] A. Goldberg, *Finding a Maximum Density Subgraph*, University of California Berkeley, CA, 1984.
[28] S. R. Bulö and M. Pelillo, "A game-theoretic approach to hypergraph clustering," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 6, pp. 1312–1327, Apr. 2013.
[29] B. Saha, A. Hoch, S. Khuller, L. Raschid, and X.-N. Zhang, "Dense subgraphs with restrictions and applications to gene annotation graphs," in *Proc. Res. Comput. Mol. Biol.*, 2010, pp. 456–472.
[30] F. Porikli, "Integral histogram: A fast way to extract histograms in cartesian spaces," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2005, vol. 1, pp. 829–836.

[31] P. Hammer, P. Hansen, and B. Simeone, "Roof duality, complementation and persistency in quadratic 0–1 optimization," *Math. Program.*, vol. 28, no. 2, pp. 121–155, 1984.
[32] H. Ishikawa, "Transformation of general binary mrf minimization to the first-order case," *IEEE Trans. Pattern Anal. Mach. Intel.*, vol. 33, no. 6, pp. 1234–1249, Apr. 2011.
[33] E. Boros, P. Hammer, and X. Sun, "Network flows and minimization of quadratic pseudo-boolean functions," Tech. Rep. RRR 17-1991, 1991.
[34] L. M. Manevitz and M. Yousef, "One-class svms for document classification," *J. Mach. Learn. Res.*, vol. 2, pp. 139–154, 2002.
[35] X. Yuan and T. Zhang, "Truncated power method for sparse eigenvalue problems," *J. Mach. Learn. Res.*, vol. 14, pp. 899–925, 2013.
[36] A. Y. Ng, M. I. Jordan, Y. Weiss, "On spectral clustering: Analysis and an algorithm," *Adv. Neural Inf. Process. Syst.*, vol. 2, pp. 849–856, 2002.
[37] F. Lin and W. W. Cohen, "Power iteration clustering," in *Proc. Int. Conf. Mach. Learn.*, vol. 10, 2010, pp. 655–662.
[38] R. Zass and A. Shashua, "Probabilistic graph and hypergraph matching," in *Proc. IEEE Conf. Comput. Vis. Pattern Recog.*, 2008, pp. 1–8.
[39] O. Duchenne, F. Bach, I.-S. Kweon, and J. Ponce, "A tensor-based algorithm for high-order graph matching," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 12, pp. 2383–2395, Dec. 2011.
[40] J. Lee, M. Cho, and K. M. Lee, "Hypergraph matching via reweighted random walks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recogn.*, 2011, pp. 1633–1640.
[41] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vis.*, vol. 60, no. 2, pp. 91–110, 2004.
[42] G. Karypis and V. Kumar, "hmetis: A hypergraph partitioning package, version 1.5. 3," 1998.

Hairong Liu is currently a postdoctoral research associate in Purdue University. His current research interests include computer vision and machine learning, focusing on matching and graph analysis. He received the Best Paper Award from The International Conference on Multimedia and Expo in 2010, and he is the reviewer of the *Computer Vision and Pattern Recognition*, *International Conference on Computer Vision*, *IEEE Transactions on Neural Networks and Learning Systems*, *IEEE Transactions on Image Processing*, *IEEE Transactions on Circuits and Systems for Video Technology* and *IEEE Transactions on Pattern Analysis and Machine Intelligence* journals.

Longin Jan Latecki is currently a professor at Temple University. His current research interests include computer vision and pattern recognition. He has published 200 research papers and books. He is an editorial board member of *Pattern Recognition* and *International Journal of Mathematical Imaging*. He received the annual Pattern Recognition Society Award together with Azriel Rosenfeld for the best article published in the journal *Pattern Recognition* in 1998.

Shuicheng Yan is currently an associate professor at National University of Singapore. His current research areas include computer vision, multimedia and machine learning, and he has authored or co-authored over 200 technical papers. He is an associate editor of the *IEEE Transactions on Circuits and Systems for Video Technology*. He received the Best Paper Awards from ACM Multimedia Conference in 2010 and the The International Congress on Mathematical Education in 2010, the prize winner of the classification task in the The PASCAL Visual Object Classes Conference (PASCAL VOC) in 2010, the honorable mention award of the detection task in PASCAL VOC'10, and the 2010 IEEE Transactions on Circuits and Systems for Video Technology Best Associate Editor Award.