2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)

Co-Clustering Signed 3-Partite Graphs

Sefa Şahin Koç
Aselsan Rehis
Ankara, Turkey
Email: sskoc@aselsan.com.tr

İsmail Hakkı Toroslu
Computer Engineering Department
Middle East Technical University, Ankara, Turkey
Email: toroslu@ceng.metu.edu.tr

Hasan Davulcu
Computer Science Department
Arizona State University, Tempe, AZ
Email: hdavulcu@asu.edu

***Abstract*—In this paper, we propose a new algorithm, called STRICLUSTER, to find tri-clusters from signed 3-partite graphs. The dataset contains three different types of nodes. Hyperedges connecting three nodes from three different partitions represent either positive or negative relations among those nodes. The aim of our algorithm is to find clusters with strong positive relations among its nodes. Moreover, negative relations up to a certain threshold is also allowed. Also, the clusters can have no overlapping hyperedges. We show the effectiveness of our algorithm via several experiments.**

I. INTRODUCTION

A hyperedge in a tri-partite graph represents the relationship among the three nodes it connects. For example, in a social tagging system, which contains three types of nodes (users, tags, resources), a hyperedge means that a user annotates a resource with a tag [1]. A tripartite cluster of these hyperedges may give many information such as users' attitudes to multiple resources or users with common interests. As another example, in a biological analysis system, a level of gene in a sample at a particular time can be represented as a tripartite hyperedge. By mining tripartite clusters, genes showing common characteristics in samples at common time slots could be extracted [2].

Finding biclusters with maximum size from a bipartite graph is proven to be NP-hard, as well as discovering tripartite clusters with maximum size [3]. Therefore, works in literature [2], [4], [1] apply heuristics to determine clusters. As a common strategy, tri-clusters are generated by first constructing biclusters between each pair of three partitions [4]. Then, each bicluster is matched with two others in order to construct tri-clusters. Since this approach is very costly, as an alternative, first, two partitions are selected, and, then, biclusters of these bipartite graphs are constructed. After that, by iterating each one of these biclusters on the third partition, tripartite clusters can be constructed [2]. However, since the first two partitions are fixed, this approach has bias against the third partition. As another approach, tripartite clusters focus on one-to-one correspondence among the nodes [5], [6]. However, the real-world data is usually more complex. For example, in a social tagging system, a group of users may tag multiple sources with the same set of tags, which corresponds to many-to-many relationship.

In this paper, we present an effective algorithm which generates tri-clusters from tripartite hyperedges with positive signs. Our method has the following properties: 1) A minimum threshold for positive signed hyperedge density ratio over all possible hyperedges among tri-partitions of the cluster is defined, and, it must be satisfied by clusters. 2) A simple greedy approach is used in order to trim the hyperedges from tri-clusters with negative signs to increase the positive density ratio of the cluster. 3) In order to prevent constructing very small clusters, both negative signed hyperedges and triples with no connections are also allowed as long as they satisfy user defined density threshold constraints. 4) Clusters are not allowed to have overlaps in terms of hyperedges. A simple heuristic is used to mark hyperedges in order to prevent hyperedge overlaps among clusters, and fast termination of the algorithm while searching potentially maximal clusters. 5) The effectiveness of our approach is shown using a coverage-based metric.

To the best of our knowledge this is the first work that attempts to find co-clusters with potentially overlapping nodes from signed tri-partite graphs. In our work, the clusters are constructed considering the hyperedges, and thus, it is possible to generate clusters with common nodes from the same dimension. A typical problems that motivates this work is finding co-clusters from sentiments of tweets on issues. The three dimensions of this problems are people who write tweets, the selected set of issues (or named entities) and the chosen sentiment words by the users on these issues. The sign of the sentiment words also sepresent the sign of the hyperedge between the three nodes of these dimenstions. It is very likely that same sentiment words are used by people with different clusters corresponding to different camps. Similarly more than one camp may have similar sentiments towards the same issue as well. Therefore, clusters generated on sentiment words and on issues which corresponds to positive feelings of different camps may have many common items. Even on people dimension, it is likely to generate clusters with several common people, which may be interpreted as these people being close to more than one different political camps.

The rest of the paper is organized as follows. Section II introduces STRICLUSTER algorithm. Section III presents experiments and section IV concludes the paper.

II. THE STRICLUSTER ALGORITHM

In this paper, we use the notations given in Table 1. STRICLUSTER algorithm takes a set of hyperedges, Γ as an input, such that each hyperedge h connects three nodes from three different types $U = (U_1, U_2, U_3)$. Figure 1 illustrates

IEEE/ACM ASONAM 2016, August 18-21
2016, San Francisco, CA, USA
978-1-5090-2846-7/16/$31.00 © 2016 IEEE

TABLE I
SYMBOL TABLE

Symbol	Meaning
Γ	set of hyperedges
$\Gamma_{v/iv}$	set of valid/invalid hyperedges
α	a tripartite cluster
ϵ_p	minimum ratio of h_+ in a cluster
ϵ_n	maximum ratio of h_- in a cluster
λ_i	minimum size for type i in a cluster
L_i	number of nodes for type i in a cluster (size of type i)
$\Re$	set of tripartite clusters
$h_{+/-}$	a hyperedge with positive/negative label
U_i	set of nodes for type i
E_{ir}	affectiveness value for node r of type i
U_{ir}	minimum E_{ir} in U_i, which belongs to node r
S_i	maximum number of h in which a node from type i can be

hyperedges given as 3D matrix. These hyperedges have either positive or negative labels which are also represented by green and red colors respectively in Figure 1. Remaining entries (white cells) corresponds to node triples without connecting hyperedges. In the example, nodes are {{A,B}, {a,b,c,d,e}, {1,2,3,4,5}} from types U_1, U_2, U_3 respectively.

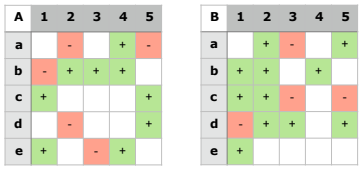


Fig. 1. Input Data

The aim of STRICLUSTER is to find tripartite clusters of hyperedges with highly positive labels. To be a valid tripartite cluster, it has to satisfy threshold values for both density and size. The density threshold values are ϵ_p and ϵ_n, such that $0 \leq \epsilon_p, \epsilon_n \leq 1$, $(\epsilon_p + \epsilon_n) \leq 1$. The former one represents the minimum ratio density of positive hyperedges (h_+) among all possible hyperedges (i.e., there may be $L_1 \times L_2 \times L_3$ number of possible hyperedges for a cluster with size (L_1, L_2, L_3), where L_i is number of nodes with U_i type in the cluster). If C_p is the number of h_+, then:

$$\epsilon_p \leq \frac{C_p}{L_1 \times L_2 \times L_3}, \tag{1}$$

If $\epsilon_p = 1$, generated tripartite clusters become tripartite cliques as well. ϵ_n is the value to control the density of negatively signed hyperedges (h_-). If C_n represnts the number of h_-, then:

$$\epsilon_n \geq \frac{C_n}{L_1 \times L_2 \times L_3}, \tag{2}$$

shows maximum allowed tolerance of h_- in a cluster if $\epsilon_n \neq 0$.

In order to prevent constructing very small clusters λ_i is defined, such that:

$$L_i \geq \lambda_i, \tag{3}$$

for $1 \leq i \leq 3$, and this constraint should also be satisfied by every cluster.

Algorithm 1 STriCluster Algorithm

```
1:  procedure STRICLUSTER(Γ, ε_p, ε_n, λ_1, λ_2, λ_3)
2:      loop
3:          generate α from Γ
4:          β = CLEANINVALIDS(Γ, Γ_iv, α, λ_1, λ_2, λ_3)
5:          if not β then
6:              return ℜ
7:          end if
8:          DENSITYCHECKING(α, ε_p, ε_n, λ_1, λ_2, λ_3)
9:          if α ⇁ formula (3) then          ▷ if α satisfies
10:             ℜ ← ℜ ⊕ α                    ▷ ⊕ means appending
11:         end if
12:         for each h in α do               ▷ h is a hyperedge in α
13:             Γ_iv ← Γ_iv ⊕ h
14:         end for
15:     end loop
16: end procedure
```

STRICLUSTER algorithm (Algorithm 1) starts by generating a potential cluster α which contains all hyperedges in Γ. For the example input data in Figure 1, α initially is equal to the whole graph. If there are invalid hyperedges (used to prevent hyperedge overlaps), they will be removed from α (Section 2.B). After invalid hyperedges are removed, if α does not satisfy the condition (3), (i.e., β is *FALSE*), the algorithm terminates.

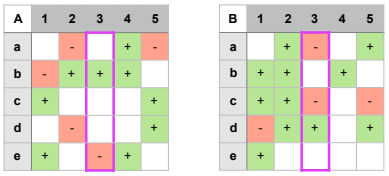
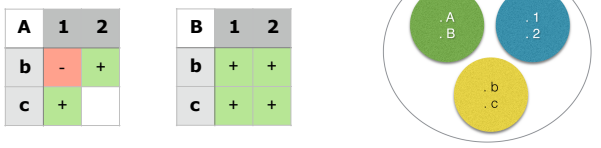


Fig. 2. Removing Node (3) From a Potential Cluster

After a potential cluster α is generated, density check operation is applied on α (Section 2.A). This operation aims to get α to satisfy conditions (1), (2), and (3). There are three possible cases that can happen: In the first case (case I), if conditions (1) or (2) are not satisfied, the least useful node is removed from α (Figure 2) iteratively, until both constraints are satisfied. For the example in Figure 1, this step will remove nodes {{a,d,e}, {3,4,5}} from types U_2, U_3 respectively from α. As the second case (case II), if the removal of a node from α violates the constraint (3), the process stops. In this case, one h_- in α is labeled as invalid. This prevents the construction of exactly the same potential tripartite cluster again, because of CLEANINVALIDS operation (Line 4) of the algorithm. As the third case (case III), α satisfies conditions (1), (2), and (3). Then, DENSITYCHECKING operation returns

α (as a reference parameter). In the example, returned cluster α contains nodes {{A,B}, {b,c}, {1,2}} shown in Figure 3. In this case, α is added to the cluster list $\Re$ (Line 7). After that, all hyperedges in α are labeled as invalid and added into Γ_{iv} which is the list of invalid hyperedges (Line 13). The following sub-section describes the details of density checking procedure which handles these operations.



(a) Matrix Representation (b) Cluster Representation

Fig. 3. A Tripartite Cluster Mined in Given Input

A. Density Checking

If given cluster α does not satisfy conditions (1) and (2), the density checking algorithm searches for nodes to exclude until α satisfies these constraints. If a node is connected by high number of h_+, it should be less likely to be removed. We define hyperedge's usefullness as follows:

$$val(h) = \begin{cases} 2 & \text{if } h \text{ has positive label} \\ -1 & \text{if } h \text{ has negative label.} \end{cases} \quad (4)$$

Then, the effectiveness of a node (r) is determined with the following formula where S_i represents the maximum number of hyperedges which contains that node in U_i:

$$E_{ir} = \frac{\sum_{h \in \alpha} \begin{cases} val(h) & \text{if } r \in h \\ 0 & \text{otherwise} \end{cases}}{S_i}. \quad (5)$$

Then, for each partition, all nodes are checked to find a node with minimum effectiveness value:

$$U_{ir} = min(\sum_{r \in U_i} E_{ir}). \quad (6)$$

At the end of this stage, there will be three U_{ir} values which are U_{1a}, U_{2b}, U_{3c} corresponding to partitions U_1, U_2, U_3 respectively. Minimum of them will be the effectiveness value E_{ix} of node x. This node will be the one to be removed from type i in α in this iteration.

TABLE II
NUMBER OF POSSIBLE HYPEREDGES FOR EACH NODE TYPE

Type	Value
S_1	$L_2 \times L_3$
S_2	$L_1 \times L_3$
S_3	$L_1 \times L_2$

For the input in Figure 1, when DENSITYCHECKING operation applied on α, node 3 will be removed in the first iteration (Figure 2). Node 3 has the lowest effectiveness value, E_{33}, compared to others. E_{33} is obtained as $\frac{2+2-1-1-1}{2\times}5 = 0.1$ using formula (5). In following iterations, nodes $\{a, d, e, 3, 4, 5\}$ will be removed from α. Then, DENSITYCHECKING will reach to case III satisfying all three conditions (1), (2), and (3) and, then returns α which contains nodes $\{A, B, b, c, 1, 2\}$.

If α does not satisfy conditions (1) and (2) and if node removal results the violation of condition (3), it means that DENSITYCHECKING is in case II. In this case, the procedure marks the first h_- as invalid.

B. Clean Invalids

Invalid hyperedges include the hyperedges of all tripartite clusters previously generated as well as all edges marked as invalid by DENSITYCHECKING. In order to remove a hyperedge one of the nodes from this hyperedge should be removed. To do this, CLEANINVALIDS procedure picks a node to remove and it repeats the same action until no invalid hyperedge is left in α. While selecting a node, it uses a heuristic that reduces the cluster size as minimum as possible.

In order to do this, we first determine the number of invalid hyperedges connected to each node. If the ratio of this number to S_i is high, that node is more likely to be removed. This ratio, called as θ_{ir} for node r from type i. Then, we calculate the effectiveness for all the valid nodes of α, which is called as E^v_{ir}. These two values values are combined with the following formula:

$$\gamma_{ir} = \frac{\theta_{ir}}{0.9 + E^v_{ir}}. \quad (7)$$

Among all nodes, the one which has highest γ value is the one to be removed.

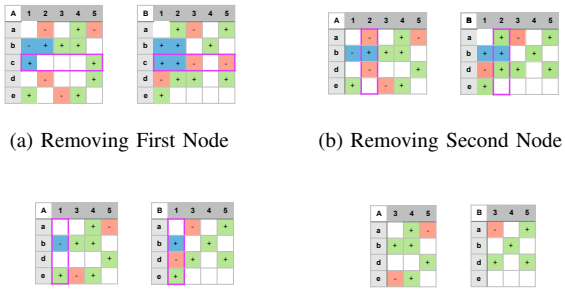
As a constraint, if removing node x will result violation of condition (3), CLEANINVALIDS procedure returns **FALSE**. If there is no invalid hyperedge left in α, CLEANINVALIDS returns **TRUE**.

For the input data in Figure 1, STRICLUSTER algorithm finds the cluster in Figure 3 in the first iteration. Then, hyperedges of this newly generated cluster are labeled as invalid. In the next iteration, new potential cluster α (Figure 4-a) is generated from Γ. But α contains some invalid hyperedges (colored with blue in Figure 4-a). Therefore, α is passed to CLEANINVALIDS procedure to be cleaned from invalid hyperedges. First, node c is removed since γ_{2c} is $3 \div 0.9 = 3.33$, is the maximum among γ values. Then, nodes 2 and 1 are selected and removed respectively (Figure 4-b, 4-c). This will result a clean α (Figure 4-d) and the procedure terminates.

III. EXPERIMENTS

In order to evaluate our algorithm, we have generated data sets with varying sizes. We have done all the experiments on MacBook Pro Mid 2015 (Intel i7 2,5 GHz, 16GB memory).

In the first set of experiments, we have fixed h_+ and h_- density ratios while changing input sizes. Other parameters are also fixed as ϵ_p = 0.75, ϵ_n = 0.10, λ_i = (2,2,2). In this test, we have generated 6 sample datasets. Each one contains



(a) Removing First Node (b) Removing Second Node

(c) Removing Third Node (d) A Clean Tripartite Cluster

Fig. 4. A Scenario of CLEANINVALIDS Procedure

positive hyperedges with 60%, negative hyperedges with 20%, and 20% is empty. $(L_1 \times L_2 \times L_3)$ values for these samples are (31.25K, 62.5K, 125K, 250K, 500K, 1M) respectively. Figure 5 presents the results.

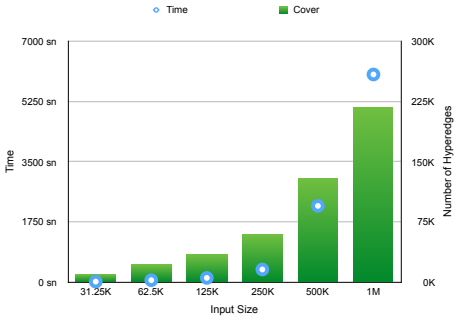


Fig. 5. Test Scenario Depending on Input Size

In the second test, we have generated 5 datasets. In this test, we have fixed the size as $(L_1 \times L_2 \times L_3) = 125K$ and we have varying density ratios for (h_+, h_-) pairs as $\{$(0.2,0.4), (0.2,0.2), (0.4,0.2), (0.4,0.4), (0.6,0.2)$\}$. The results are shown in Figure 6.

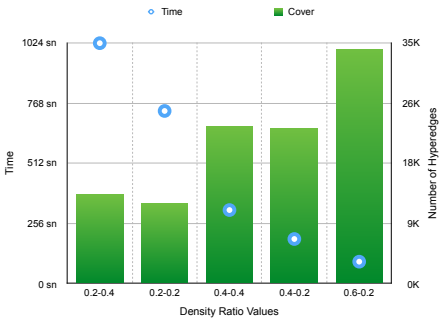


Fig. 6. Test Scenario Depending on Density Ratios in Input Data

The figures show both execution times and the number of hyperedges included in the constructed clusters. We prefer most (positive) hyperedges to be included in clusters while clusters being non-trivial. The results show that we have achieved very high coverage in that sense, since constructed clusters include almost as many hyperedges as the half of the number of positively signed hyperedges.

We have also tested our approach using real data set, which corresponds to the tweets of users on selected issues. These tweets are processed in order to determine the sentiments of users towards these issues. From these tweets, a three dimensional data set is generated. These dimensions are users, issues, and sentiment words chosen by the users on these issues, which also represent the sign of the hyperedge connecting these three items. We have large datasets with 10K users, 45K sentiment words and 20 different issues. This 3-dimensional data is very sparse with only 280K non-empty entries. So far, we have applied our algorithm to a fraction of this dataset which corresponds to randomly select few percentages of it. We have obtained fairly large and overlapping clusters in all three dimensions. Some largest clusters have as many items as the 10% of the nodes of its corresponding dimensions, even for user or sentiment words dimensions.

IV. CONCLUSION

In this paper, we have proposed a new method, called STRICLUSTER, to mine tripartite clusters of positively labeled hyperedges. The input data is composed of three dimensions. Each hyperedge connects three nodes from each dimension. Clusters are generated depending on density ratio of positively (minimum) and negatively (maximum) labeled hyperedges.

We have showed the effectiveness of our approach using both syntetic and real data sets.

ACKNOWLEDGMENT

This research was supported partially by USAF Grant FA9550-15-1-0004.

REFERENCES

[1] C. Lu, X. Chen, and E. Park, "Exploit the tripartite network of social tagging for web clustering," in *Proceedings of the 18th ACM conference on Information and knowledge management*. ACM, 2009, pp. 1545–1548.

[2] L. Zhao and M. J. Zaki, "Tricluster: an effective algorithm for mining coherent clusters in 3d microarray data," in *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*. ACM, 2005, pp. 694–705.

[3] M. Dawande, P. Keskinocak, J. M. Swaminathan, and S. Tayur, "On bipartite and multipartite clique problems," *Journal of Algorithms*, vol. 41, no. 2, pp. 388–403, 2001.

[4] L. Zhu, A. Galstyan, J. Cheng, and K. Lerman, "Tripartite graph clustering for dynamic sentiment analysis on social media," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM, 2014, pp. 1531–1542.

[5] X. Liu and T. Murata, "Detecting communities in tripartite hypergraphs," *arXiv preprint arXiv:1011.1043*, 2010.

[6] Y.-R. Lin, J. Sun, P. Castro, R. Konuru, H. Sundaram, and A. Kelliher, "Metafac: community discovery via relational hypergraph factorization," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2009, pp. 527–536.