Accelerating Hypergraph Motif Counting Based on Hyperedge Relations

Yuhang Su[1](✉), Yu Gu[1], Yang Song[2], and Ge Yu[1]

[1] College of Computer Science and Engineering, Northeastern University, Shenyang 110819, Liaoning, China
suyuhang_neu@163.com, {yuge,guyu}@mail.neu.edu.cn

[2] College of Information Science and Engineering, Northeastern University, Shenyang 110819, Liaoning, China

Abstract. Hypergraphs can naturally represent inter-group relations that are prevalent in many application domains by hyperedges. Hypergraph motifs can be described as the structural patterns of three connected hyperedges. As an effective tool, it plays an important role in the local structure analysis of hypergraphs. In this paper, we study exact hypergraph motif counting which is a fundamental problem of hypergraph motif research. Existing algorithms don't adequately consider hyperedge relations in real-world hypergraphs, which lead to a large number of redundant computations. This motivates us to improve performance by exploiting hyperedge relations. In our work, we classify hypergraph motifs with different hyperedge relations. For different types of motifs, we use set theory to demonstrate and propose different optimization methods to reduce the computation of excessive intersections. We also further reduce the cost of the proposed method by preserving hyperedge intersections when constructing the hyperdege projected graph. Extensive experiments on real datasets validate the superiority of our algorithm compared to existing methods.

Keywords: Hypergraph · Hypergraph motif · Hypergraph motif counting · Hyperedge relation

1 Introduction

A hypergraph consists of vertices and hyperedges that can connect multiple vertices, and can be seen as a general form of ordinary graphs. Since hypergraphs can effectively simulate complex intergroup relationships between entities, they have a wide range of applications such as bioinformatics [5] and social network analysis [9] . Specifically, complex analyses over hypergraphs have also been extensively explored for hypergraph motifs [6], classification [4] and hyperedge prediction [10]. Network motifs have achieved great success in exploring and discovering local structural features of real-world graphs [7]. However, due to the different structures of ordinary graphs between hypergraphs, it is difficult to directly apply related techniques to hypergraphs. In order to better explore the local structural patterns of real-world hypergraphs, Lee et al. [6] successfully define hypergraph motifs for the first time. Existing methods demonstrate the importance of hypergraph motifs in revealing hypergraph local structural patterns. However, existing algorithms do not effectively explore hyperedge relations to improve the computational efficiency. This motivates us to fully explore hyperedge relations (intersection and containment) to achieve the acceleration of hypergraph motifs counting. The major contributions are concluded as follows.

- We explore the widely existing hyperedge relations in real-world hypergraphs and classify hypergraph motifs according to specific relations. For different types of motifs, by using set theory, we study and demonstrate different optimal calculation methods to reduce the cost of excessive intersections.
- For the remaining hypergraph motifs that cannot be optimized, we further reduce the cost of the algorithm by preserving the hyperedge intersection when constructing the hyperdege projected graph.
- We conduct extensive experiments to verify that our algorithm outperforms existing algorithms. In total processing time, our algorithm is more than two times faster than existing algorithms.

2 Related Work

We examine existing related work on network motif counting for ordinary graph. Most of them apply the following three techniques to speed up motif counting: 1) Combinatorics: In order to speed up exact network motif counting, the existing work [8] adopt combinatorial relations computation methods. 2) MCMC sampling: Most approximate network motif counting algorithms estimate the number of motif instances by sampling [2,3]. 3) Color coding: The approximate network motif counting algorithm [1] uses color coding to randomly color each vertex and use this coloring information to randomly sample. However, due to the different structures of ordinary graphs and hypergraphs, it is difficult to directly apply related techniques to hypergraphs. We also review existing related work on hypergraph motifs. Hypergraph motifs are the basic building blocks of hypergraphs as defined by [6]. Unlike network motifs, it is formed by three connected hyperedges with 26 different connection patterns. Hypergraph motifs differ from network motifs in that they do not limit the number of vertices. Extensive experiments verify that hypergraph motifs play an important role in revealing local structural patterns of real-world hypergraphs. The only existing exact hypergraph motif counting algorithm is proposed by [6]. Although the algorithm efficiently implements hypergraph motif counting, it performs a lot of redundant intersection computations.

3 Hypergraph Motif Classification and Computation Acceleration Based on Hyperedge Relations

3.1 Basic Definition

Definition 1 (Hypergraph). *A hypergraph is represented by* $G = (V, E)$*, where* V *is a finite set of vertices,* $E = \bigcup_{i=1}^{|E|} e_i$ *is a finite set of hyperedges. Each hyperedge* $e_i \in E$ *is a non-empty subset of* V*.*

Definition 2 (Hyperdege Projected Graph). *A hyperdege projected graph of* $G = (V, E)$ *is an ordinary graph* $PG = (E, H)$*, where* $H = \{(e_i, e_j) \mid e_i \cap e_j \neq \varnothing\}$*. We use* $\overline{H}_{ij}$ *to denote the intersection of* e_i *and* e_j*, that is,* $\overline{H}_{ij} = \{v_i \in V \mid v_i \in e_i \cap e_j\}$*.*

Definition 3 (Hypergraph Motif). *Given three connected hyperedges* $\{e_i, e_j, e_k\}$*, hypergraph motifs are used to describe the connectivity patterns of the three connected hyperedges. Formally, a hypergraph motif is a binary vector of size 7 whose elements represent the emptiness of the following seven sets:* (1) $e_i \setminus e_j \setminus e_k$, (2) $e_j \setminus e_k \setminus e_i$, (3) $e_k \setminus e_i \setminus e_j$, (4) $e_i \cap e_j \setminus e_k$, (5) $e_j \cap e_k \setminus e_i$, (6) $e_k \cap e_i \setminus e_j$ *and* (7) $e_i \cap e_j \cap e_k$.

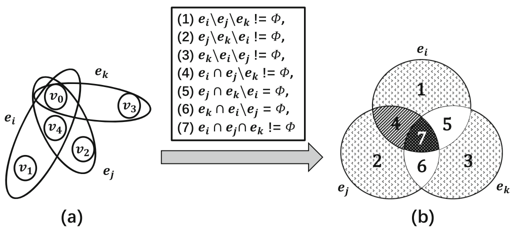

(1) $e_i \backslash e_j \backslash e_k$!= Φ,
(2) $e_j \backslash e_k \backslash e_i$!= Φ,
(3) $e_k \backslash e_i \backslash e_j$!= Φ,
(4) $e_i \cap e_j \backslash e_k$!= Φ,
(5) $e_j \cap e_k \backslash e_i$ = Φ,
(6) $e_k \cap e_i \backslash e_j$ = Φ,
(7) $e_i \cap e_j \cap e_k$!= Φ
e_k
e_i
e_j
v_0
v_1
v_2
v_3
v_4
(a)
e_i
1
4
7
5
2
6
3
e_j
e_k
(b)

Fig. 1. Hypergraph motif and hypergraph motif instance

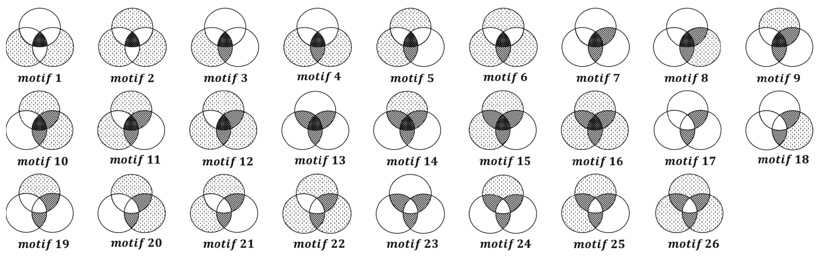


Fig. 2. The 26 hypergraph motifs

Example 1. As shown in Fig. 1(b), hypergraph motifs can be naturally represented in the Venn diagram. The three circles represent hyperedges e_i, e_j and e_k, respectively. The three circles are superimposed and divided into seven parts representing seven different sets. We usually use patterned parts to represent non-empty and white to represent empty. In fact, excluding symmetries and duplicated hyperedges, we can describe the pattern of all connected three hyperedges by means of 26 hypergraph motifs in Fig. 2. If the connectivity pattern of the three hyperedges corresponds to a particular hypergraph motif, we consider the three connected hyperedges as an instance of this hypergraph motif. As shown in Fig. 1, (a) is an instance of the hypergraph *motif* 6. It is worth noting that *motif* 17–22 are open motifs in Fig. 2. More intuitively, the open motif is the one that has two hyperedges which are not connected. Obviously, given three hyperedges e_i, e_j and e_k, if their connection pattern (motif) is a open motif, then $|e_i \cap e_j \cap e_k| = 0$.

Definition 4 (Hypergraph Motif Counting). *Hypergraph motif counting is to calculate the number of instances corresponding to 26 hypergraph motifs on a hypergraph.*

3.2 Double-Single-Inclusion Motifs

Definition 5 (Double-Single-Inclusion Motifs). *Given three hyperedges e_i, e_j and e_k, if their connection pattern (motif) satisfies any of the following three conditions* (1) $|e_i \cap e_j| = |e_j \cap e_k| = |e_j|$; (2) $|e_j \cap e_k| = |e_i \cap e_k| = |e_k|$; (3) $|e_i \cap e_j| = |e_i \cap e_k| = |e_i|$, *we call it a Double-Single-Inclusion Motif* (*DSI motif for short*).

Example 2. As shown in Fig. 2, *motif* 1 and *motif* 4 are DSI motifs. More intuitively, the DSI motif is the one that has one hyperedge contained by the other two hyperedges.

Theorem 1. *Given three hyperedges e_i, e_j and e_k, if their connection pattern (motif) is a DSI motif, there exist the following conclusions :* (1) *if* $|e_i \cap e_j| = |e_j \cap e_k| = |e_j|$ *then* $|e_i \cap e_j \cap e_k| = |e_j|$; (2) *if* $|e_j \cap e_k| = |e_i \cap e_k| = |e_k|$ *then* $|e_i \cap e_j \cap e_k| = |e_k|$; (3) *if* $|e_i \cap e_j| = |e_i \cap e_k| = |e_i|$ *then* $|e_i \cap e_j \cap e_k| = |e_i|$.

Proof. We first prove the conclusion (1). Given three hyperedges e_i, e_j and e_k, if $|e_i \cap e_j| = |e_j \cap e_k| = |e_j|$, then e_i contains e_j and e_k also contains e_j. Therefore, there is $|e_i \cap e_j \cap e_k| = |e_j \cap e_k| = |e_j|$. Similarly, conclusions (2) and (3) can be proved. Theorem 1 is proved.

3.3 Single-Double-Inclusion Motifs

Definition 6 (Single-Double-Inclusion Motifs). *Given three hyperedges e_i, e_j and e_k, if their connection pattern (motif) satisfies any of the following three conditions* (1) $|e_i \cap e_j| = |e_j|$ *and* $|e_i \cap e_k| = |e_k|$; (2) $|e_j \cap e_k| = |e_k|$ *and* $|e_i \cap e_j| = |e_i|$; (3) $|e_j \cap e_k| = |e_j|$ *and* $|e_i \cap e_k| = |e_i|$, *we call it a Single-Double-Inclusion Motif* (*SDI motif for short*).

Example 3. As shown in Fig. 2, *motif* 3, *motif* 7 and *motif* 8 are SDI motifs. More intuitively, the SDI motif is the one that has one hyperedge containing the other two hyperedges.

Theorem 2. *Given three hyperedges e_i, e_j and e_k, if their connection pattern (motif) is a DSI motif, there exist the following conclusions :* (1) *if* $|e_i \cap e_j| = |e_j|$ *and* $|e_i \cap e_k| = |e_k|$ *then* $|e_i \cap e_j \cap e_k| = |e_j \cap e_k|$*;* (2) *if* $|e_j \cap e_k| = |e_k|$ *and* $|e_i \cap e_j| = |e_i|$ *then* $|e_i \cap e_j \cap e_k| = |e_j \cap e_k|$*;* (3) *if* $|e_j \cap e_k| = |e_j|$ *and* $|e_i \cap e_k| = |e_i|$ *then* $|e_i \cap e_j \cap e_k| = |e_i \cap e_j|$.

Proof. We first prove the conclusion (1). Given three hyperedges e_i, e_j and e_k, if $|e_i \cap e_j| = |e_j|$ and $|e_i \cap e_k| = |e_k|$, then e_i contains e_j and e_k. Therefore, there is $|e_i \cap e_j \cap e_k| = |(e_i \cap e_j) \cap (e_i \cap e_k)| = |e_j \cap e_k|$. Similarly, conclusions (2) and (3) can be proved. Theorem 2 is proved.

3.4 Single-Single-Inclusion Motifs

Definition 7 (Single-Single-Inclusion Motifs). *Given three hyperedges e_i, e_j and e_k, if their connection pattern (motif) satisfies any of the following three conditions* (1) $|e_i \cap e_j| = |e_j|$ *and* $|e_i \cap e_k| \neq |e_k|$*;* (2) $|e_j \cap e_k| = |e_k|$ *and* $|e_i \cap e_j| \neq |e_i|$*;* (3) $|e_j \cap e_k| = |e_j|$ *and* $|e_i \cap e_k| \neq |e_i|$*, we call it a Single-Single-Inclusion Motif* (*SSI motif for short*).

Example 4. As shown in Fig. 2, *motif* 5, *motif* 9 and *motif* 10 are SSI motifs. More intuitively, the SSI motif is the one that has one hyperedge containing only one of other two hyperedges.

Theorem 3. *Given three hyperedges e_i, e_j and e_k, if their connection pattern (motif) is a SSI motif, there exist the following conclusions :* (1) *if* $|e_i \cap e_j| = |e_j|$ *and* $|e_i \cap e_k| \neq |e_k|$ *then* $|e_i \cap e_j \cap e_k| = |e_j \cap e_k|$*;* (2) *if* $|e_j \cap e_k| = |e_k|$ *and* $|e_i \cap e_j| \neq |e_i|$ *then* $|e_i \cap e_j \cap e_k| = |e_j \cap e_k|$*;* (3) *if* $|e_j \cap e_k| = |e_j|$ *and* $|e_i \cap e_k| \neq |e_i|$ *then* $|e_i \cap e_j \cap e_k| = |e_i \cap e_j|$.

Proof. We first prove the conclusion (1). Given three hyperedges e_i, e_j and e_k, if $|e_i \cap e_j| = |e_j|$ and $|e_i \cap e_k| \neq |e_k|$, then e_i contains e_j. Therefore, there is $|e_i \cap e_j \cap e_k| = |(e_i \cap e_j) \cap e_k| = |e_j \cap e_k|$. Similarly, conclusions (2) and (3) can be proved. Theorem 3 is proved.

As described in Subsect. 3.2–3.4, we propose 3 different special motifs through set theory. We also exploit set theory to give and prove their respective special properties. By determining the type of motifs, we can speed up the computation for the corresponding motifs through Theorems 1–3.

4 Hypergraph Motif Counting Algorithm Framework Optimization

For the remaining hypergraph motifs that cannot be optimized, we further reduce the overall complexity of the algorithm by preserving the hyperedge pair intersections in the preprocessing stage. **1) Constructing Projected Graph.** As a preprocessing step (*Lines* 1–7), Algorithm 1 builds a complete hyperedge projected graph for subsequent motif counting. It first clears H for recording hyperedge pairs (*Line* 1). Then it finds all neighbors of each hyperedge e_i (*Lines* 2–4). E_v is used to denote the set of all hyperedges containing the vertices v. Finally it stores the hyperedge pair in H (*Line* 6). At the same time, it pre-stores the intersection (set of vertices) of the corresponding hyperedge pairs in $\overline{H}$ for computing acceleration (*Line* 7). The time complexity of this preprocessing step is $O(\sum_{(e_i,e_j)\in H} |e_i \cap e_j|)$. In fact, it needs to compute $e_i \cap e_j$ to find the neighbor e_j of hyperedge e_i, hence it does not affect the time complexity of the algorithm by pre-storing $e_i \cap e_j$ in $\overline{H}$. **2) Motif Counting.** Algorithm 1 (*Lines* 8–12) first finds two neighbors of each hyperedge e_i to form a hyperedge triple (*Lines* 8–9). H_{e_i} is used to represent all neighbors of hyperedge e_i in PG. Then it determines whether the three hyperedges belong to a particular motif (*Line* 10). If the corresponding hyperedge triple belongs to DSI or SDI or SSI or open motif, we use the function $\overline{h}(\{e_i, e_j, e_k\})$ to determine which motif it belongs to and accumulate at the corresponding position of M (*Line* 11). Since $\overline{h}(\{e_i, e_j, e_k\})$ does not need to compute e$_i$ ∩ e$_j$ ∩ e$_k$, the time complexity of $\overline{h}(\{e_i, e_j, e_k\})$ is $O(1)$. For the remaining motifs, we use function $h(\{e_i, e_i, e_i\}, \overline{H})$ to calculate (*Line* 12). Since the algorithm pre-stores the hyperedge pair intersections in $\overline{H}$, the time complexity of $h(\{e_i, e_i, e_i\}, \overline{H})$ is $O(min(|e_i \cap e_j|, |e_j \cap e_k|, |e_i \cap e_k|))$. In conclusion, the time complexity of our algorithm is better than that of existing algorithm $(O(min(|e_i|, |e_j|, |e_i|))$ in [6]).

Algorithm 1: The Framework For Hypergraph Motif Counting

```
Input  : Hypergraph $G = (V, E)$, hyperedge projected graph $PG = (E, H)$
Output: Exact count of each hypergraph motif in $M$
1  initialize: Set $H := \emptyset$ ;
2  foreach hyperedge $e_i \in E$ do
3      foreach vertex $v \in e_i$ do
4          foreach hyperedge $e_j \in E_v$ do
5              if $j > i$ then
6                  $H \leftarrow (e_i, e_j)$ ;
7                  $\overline{H} \leftarrow e_i \cap e_j$ ;
8  foreach hyperedge $e_i \in E$ do
9      foreach unordered hyperedge pair $e_j \in H_{e_i}$ and $e_k \in H_{e_i}$ do
10         if $\{e_i, e_j, e_k\} \in$ DSI or SDI or SSI or open motif then
11             $M[\overline{h}(\{e_i, e_j, e_k\})] + +$ ;
           else
12             $M[h(\{e_i, e_j, e_k\}, \overline{H})] + +$ ;
13 return $M$ ;
```

5 Experimental Settings and Results Analysis

5.1 Experimental Settings

1) Competitive Algorithms. The first is a native algorithm for hypergraph motif counting. This algorithm does not employ any optimization techniques. We call this algorithm **HMC** for short, and we regard **HMC** as a basic method. The second algorithm **HMCO** can be seen as **HMC** with optimization techniques only for the open motif and it is actually the exact motif algorithm in [6]. The third algorithm **HMCA** can be seen as **HMC** with optimization techniques for DSI, SDI, SSI and open motifs. Our algorithm **HMCP** can be seen as **HMCA** adding preserving intersections techniques for remaining hypergraph motifs. **2) Experiment Environment**. We obtained the source code of **HMCO** from the authors of [6]. The compiler for compiling source code is $g{+}{+}$ 4.9.3$-O3\ flag$. We conduct all experiments on a PC machine with equipment of $Intel\ i5$ 3.20 GHz and 16 GB RAM. **3) Metrics**. We measure the execution time in milliseconds (ms). **4) Datasets**. We use 8 real-world datasets (http://konect.cc/) to evaluate the algorithms. The specific information of all real-world datasets is given in Table 1.

Table 1. Real-World Datasets Statistics

Data	$\|V\|$	$\|H\|$	H_Avg	H_Max	V_Avg	$Edge\ of\ BiGraph$
unicodelang	254	614	2.04	141	4.94	1,225
edit-crwiki	1,188	2,071	10.63	248	19.11	22,700
filmtrust	1,508	30,087	17.14	1,044	23.54	35,494
escorts	10,106	6,624	7.64	615	5.01	50,632
wang-amazon	26,112	799	36.37	812	1.11	29,062
tripadvisor	145,316	1,759	99.92	2,138	1.21	175,765
bag-kos	3,430	6,906	67.73	2,123	136.36	467,714
flickr	395,979	103,631	82.46	34,989	21.58	8,545,307

5.2 Experimental Results Analysis

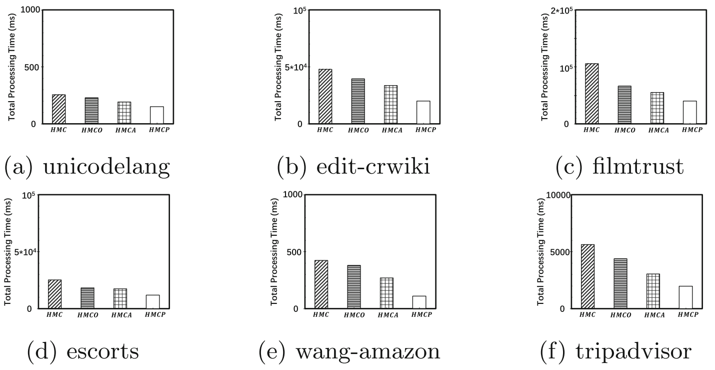
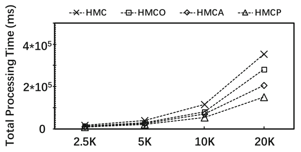
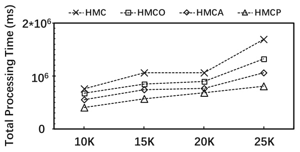


Fig. 3. Total Processing Time on Different Datasets (Vary Algorithm).

1) Total processing time. Figures 3(a)–(f) show the total time when processing the corresponding dataset. Based on the experimental results, we can obtain the following conclusions. 1) **HMC** performs the worst on all datasets, because it employs the brute force policy and lacks optimization method. 2) Simplifying the computation by considering only special motifs can also lead to better speedups. **HMCA** outperforms existing methods **HMCO**. This is because a large number of hyperedge inclusion relations are actually contained in the real-world hypergraph. Our optimization technique exploits these relationships to greatly reduce computational overhead. 3) **HMCP** always maintains the advantage on all datasets. The reason is twofold. One is to use Theorems 1–3 to reduce redundant intersection calculations. The second is that preserving the hyperedge pair intersections in the preprocessing stage provides speedup for computing the remaining hypergraph motifs. In general, **HMCP** is more than 2 times faster than existing method **HMCO**. In dataset wang-amazon, **HMCP** can bring a maximum speedup of four times. **2) Scalability**. To test the scalability of our algorithm, we use larger datasets. By varying the number of edges added to the hypergraph, we compare the performance of the four algorithms as shown in Figs. 4(a)–(b). The conclusion is that **HMCP** has better scalability than other algorithms. This is because our algorithm fully considers the hyperedge relationship to provide speedup. It is worth noting that the degree of the hyperedge increases as the number of edges increases. This will lead to more hyperedge inclusion relations, so the advantage of **HMCP** is more obvious.



(a) bag-kos



(b) flickr

Fig. 4. Processing time on different datasets (vary number of edges)

6 Conclusion

In this paper, we propose effective techniques for accelerating hypergraph motif counting based on hyperedge relations. In our work, we classify hypergraph motifs with different hyperedge relations and demonstrate different optimization methods. For the remaining hypergraph motifs that cannot be optimized, we further reduce the overall complexity of the algorithm. Extensive experiments on real datasets show that our method is superior to the existing solutions.

Acknowledgements. This work is supported by the National Nature Science Foundation of China (62072083) and the Fundamental Research Funds of the Central Universities (N2216017).

References

1. Bressan, M., Leucci, S., Panconesi, A.: Motivo: fast motif counting via succinct color coding and adaptive sampling. Proc. VLDB Endow. **12**(11), 1651–1663 (2019)
2. Han, G., Sethu, H.: Waddling random walk: fast and accurate mining of motif statistics in large graphs. In: 2016 IEEE 16th International Conference on Data Mining (ICDM), pp. 181–190. IEEE (2016)
3. Huang, S., Li, Y., Bao, Z., Li, Z.: Towards efficient motif-based graph partitioning: an adaptive sampling approach. In: International Conference on Data Engineering (2021)
4. Jiang, J., Wei, Y., Feng, Y., Cao, J., Gao, Y.: Dynamic hypergraph neural networks. In: IJCAI, pp. 2635–2641 (2019)
5. Klamt, S., Haus, U.U., Theis, F.: Hypergraphs and cellular networks. PLoS Comput. Biol. **5**(5), e1000385 (2009)
6. Lee, G., Ko, J., Shin, K.: Hypergraph motifs: concepts, algorithms, and discoveries. Proc. VLDB Endow. **13**(11), 2256–2269 (2020)
7. Liu, J., Shao, Y., Su, S.: Multiple local community detection via high-quality seed identification over both static and dynamic networks. Data Sci. Eng. **6**(3), 249–264 (2021). https://doi.org/10.1007/s41019-021-00160-6
8. Pinar, A., Seshadhri, C., Vishal, V.: ESCAPE: efficiently counting all 5-vertex subgraphs. In: Proceedings of the 26th International Conference on World Wide Web, pp. 1431–1440 (2017)
9. Yang, D., Qu, B., Yang, J., Cudre-Mauroux, P.: Revisiting user mobility and social relationships in LBSNs: a hypergraph embedding approach. In: The World Wide Web Conference, pp. 2147–2157 (2019)
10. Yoon, S.E., Song, H., Shin, K., Yi, Y.: How much and when do we need higher-order information in hypergraphs? A case study on hyperedge prediction. In: The Web Conference 2020 (WWW 2020) (2020)