A New Graph-Partitioning Algorithm for Large-Scale Knowledge Graph

Jiang Zhong(✉), Chen Wang, Qi Li, and Qing Li

College of Computer Science, Chongqing University, Chongqing 400030, China
zhongjiang@cqu.edu.cn

Abstract. Large-scale knowledge graph is finding widely practical applications in many fields such as information retrieval, question answering, health care, and knowledge management and so on. To carry out computations on such large-scale knowledge graphs with millions of entities and facts, partitioning of the graphs is necessary. However, the existing partitioning algorithms are difficult to meet the requirements on both partition efficiency and partition quality at the same time. In this paper, we utilize the community-based characteristic that real-world graphs are mostly power-law distribution, and propose a new graph-partitioning algorithm (called MCS) based on message cluster and streaming partitioning. Compared with the traditional algorithms, MCS is closer to or even surpasses Metis package in the partition quality. In the partition efficiency, we use the PageRank algorithm in the spark cluster system to compute the Twitter graph data, and the total time of MCS is lower than that of Hash partitioning. With an increasing number of iterations, the effect is more obvious, which proves the effectiveness of MCS.

Keywords: Large-scale knowledge graph · Graph partitioning
Streaming partitioning · Community detection · Parallel computing

1 Introduction

In 2012, Google proposed knowledge graph to enhance its search engine's search results [1, 2]. Knowledge graphs usually contain two layers: schema and instances. The schema layer contains concepts, property and relationships between concepts. While the instances layer contains entities and relationships between different entities. Schema is usually expressed as an ontology. While instances can be represented as a set of RDF [3] (Resource Description Framework, a standard model for data interchange on the Web) triples, which constitute many large graphs.

Large-scale knowledge graphs contain massive entities and abundant relations among the entities, and they can be widely applied to many practical applications such

This work gets supports from the National Key Research and Development Program of China under Grant 2017YFB1402401, Technology Innovation and Application Demonstration project of Chongqing under Grant cstc2018jszx-cyzdX0086, Fundamental Research Funds for the Central University under Grant 2018CDYJSY0055 and CERNET Next Generation Internet Technology Innovation Project under Grant NGII20150706.

as semantic search, question answering and recommender systems, information retrieval, health care, knowledge management and so on [4].

By the end of 2012, the knowledge graph of Google has contained over 570 million entities and more than 18 billion facts about relationships between different entities [5]. Other public knowledge graphs, such as DBpedia [6] and YAGO [7] contain millions of entities and hundreds of millions of facts. All the above famous knowledge graphs would continuously grow, as more facts would be automatically discovered from the underlying sources or manually created by human.

The fast evolving on large-scale knowledge graphs poses some challenges to the effective usage of them. One challenge is how to carry out effectively computations on large-scale knowledge graphs, such as knowledge fusion, knowledge acquisition and knowledge reasoning.

To carry out computations on such large-scale knowledge graphs with millions of entities and facts, the first step is partition the graph. In order to partition such large-scale knowledge graphs, distributed iterative processing systems (e.g., Spark [8], Pregel [9], and GiraphLab [10]) have been developed. These systems mainly adopt the Hash function to send the vertices to each processing unit, which refers to the computing units in cluster system. Although this method has low time complexity, the communication traffic among processing units will be large in the iterative process. If using an algorithm with better partition quality (as Metis [11]) instead of the Hash method, for the high time complexity of Metis, the total cost of time is much larger than suing the Hash method. Therefore, the more efficient partitioning algorithm becomes the existing urgent problem in the distributed graph computing system.

The graph partitioning belongs to NP complete problem [12]. The current research work mainly covers into two categories: centralized partitioning and streaming partitioning. The centralized partitioning algorithm has been studied for a long time, but with its high time complexity, it can handle graphs with a few vertices and edges. The biggest advantage of streaming partitioning is that the streaming method only processes one vertex at a time. The information used is neighbors, so the efficiency of centralized partitioning algorithm is lower than that of streaming partitioning.

However, the existing partitioning methods have ignored the structure of the graph itself. The paper utilizes the community-based feature of most graphs to propose a message cluster and streaming partitioning (MCS) algorithm, which has great improvement in partition quality and efficiency compared with the traditional graph partitioning algorithms.

The main contributions of the paper are as follows:

(1) We introduce a message cluster and streaming partitioning (MCS) algorithm by using characteristics of the community in most graphs and the low complexity of streaming algorithm. The partition efficiency is obviously improved.
(2) We apply the idea of label propagation to preprocessing of graph partitioning with the design of restrictive propagation process, and quantitatively analyze its stability.
(3) We demonstrate the experimental results on different types of real and synthetic graphs by comparison with traditional algorithms in different aspects, to prove the effectiveness of the MCS algorithm.

2 Streaming Partitioning

In order to increase the efficiency of streaming partitioning, we try to use distributed streaming algorithm to partition the graph [13]. This approach increases the complexity of hardware and program. Restreaming partitioning algorithm is provided forward in [14] and its partition quality has obviously increase after several iterations.

The graph data $G = (V, E)$, V and E denote sets of vertices and edges respectively. $|V| = n. |E| = m$. k is the number of subsets. *N (v)* refers to the set of vertices that v neighbors. A k-partitioning of G can be defined as a mapping (partition function) $\pi : V \rightarrow \{1, 2, \ldots, k\}$ that distributes the vertices of V among k disjoint subsets $(S_1 \cup S_2 \cup \ldots \cup S_k = V)$ of roughly equal size and make the number of edges cut (edges whose endpoints belong to different subsets) as little as possible. Equation 1 demonstrates the load coefficient (ρ) and the fraction of edges cut (λ), *n/k* is the average load.

$$\lambda = \frac{\#\text{the number of edges cut}}{\#\text{total edges}} \tag{1}$$

$$\rho = \frac{\#\text{maximum load}}{\#(n/k)} \tag{2}$$

S_i^t represents the set of vertices in subset $S_i (i \in [1, k])$ at time t. Streaming algorithm is to traverse the graph and send the vertex v to the subset S_i one by one according to streaming function. Different algorithms has different heuristic rules. For example, the heuristic rule of the Non-Neighbors algorithm is minimum $|S_i \backslash N(v)|$. The heuristic rule of Deterministic Greedy is maximum $|N(v) \cap S_i|$. The heuristic rule of Exponentially Weighted Deterministic Greedy is maximum $|N(v) \cap S_i|(1 - \exp(|S_i| - n/k))$.

The following details describe the common streaming algorithm. The specific process of these algorithms in [6] are as following:

(1) Hash (H): $(|\mathrm{V}| \bmod \mathrm{k}) + 1$
(2) Balance (B): $(|V| \bmod k) + 1$
(3) Exponentially Weighted Deterministic Greedy (EDG): $max_{i \in [1,k]}\{|S_i^t \cap N(v)| \times (1 - exp(|S_i| - c))\}$
(4) Exponentially Weighted Triangles (ET): $max\{|S_i^t| \times (1 - exp(|V_i| - c))\}$
(5) Linear Weighted Triangles (LT): $max\{|S_i^t| \times (1 - |V_i|/c)\}$
(6) Triangles (T): $max\{|S_i^t|\}$
(7) Chunk (C): $\lceil |V|/c \rceil$

Stanton and Kliot analyzed [15] the performance of a series of heuristic streaming algorithms. The best is Linear Weighted Deterministic Greedy (LDG). Equation 2 demonstrates its rule of sending the vertex v to the subset S_{ind} one by one.

$$S_{ind} = \underset{i \in \{1, \ldots, k\}}{\arg\max} \{ \left| N(v) \cap S_i^t \right| w(t, i) \} \tag{3}$$

$$w(t, i) = 1 - \left| S_i^t \right| / (n/k) \tag{4}$$

here $|N(v) \cap S_i^t|$ represents the number of v neighbors in S_i at time t. In order to reach load balancing, it adds penalty function $w(t, i)$ to punish the subset which has too many vertices. In the initial phase of the algorithm, many vertices' value calculated by function is zero, because their neighbors have not been calculated.

3 Method Description

3.1 MCS Algorithm Description

We introduce the MCS algorithm in three steps. We cluster the input graph and sort the clustering results, and then select the vertices in the order of community to send them to the corresponding processing unit. The following steps describe the specific details.

Step one: Graph Clustering. The Label Propagation algorithm (LPA) [15] is used for community detection. Here, we use it for two reasons. First, LPA time complexity is very low and is close to the linear complexity t × |E|, where t denotes the number of iterations. Second, LPA can discover dense structures in complex networks quickly. We carefully analyzed the propagation process of LPA and found that most vertex labels do not change after the fourth iteration. Figure 1 records the percentage change in vertex labels for each iteration.

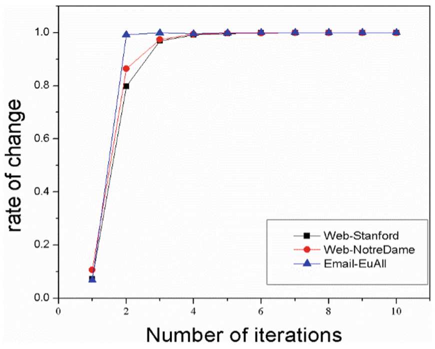

Fig. 1. Convergence of real-world graphs

According to this feature, we set iteration time $t \leq 4$. Due to the randomness of LPA, it is very easy to create a large community, which is not allowed. Therefore, mechanisms must be adopted to limit the size of the community. We set the size of the single community $|F_i| \leq |V|/a \times k(a \in N^+)$. Two properties are added to each vertex. The first is the vertex label (*flag Message*) and the second is the vertex status (*state Message*). In the initial step of MCS algorithm. The content of *flag Message* is the vertex number itself (label = Id) and the content of *state Message* is false (lock = false). A vertex in the state that lock is false can send its *flag Message* to its neighbors and receive *flag Message* from its neighbors. When a vertex in the state that lock is true, the vertex can neither send nor receive *flag Message*. Each vertex takes the maximum number of the same label as its own label according to its *flag Message* received. When the size of one community is $|F_i| = |V|/2 \times k$, change the *state Message* of its vertices to true.

The pseudo code is shown below.

```
Algorithm 1. Graph Clustering Algorithm
Input: Initial Graph G = $(V, E)$
Output: Clustering result $\{F_1, F_2, \ldots, F_n\}$
Initialize the parameters: the number of iterations Iter, the number of processing units
k, initiate the flag Message and state Message of all vertices.
Begin
    1. repeat
    2.  Iter: = Iter + 1
    3.  for each vertex in Graph
    4. { Each vertex takes the maximum number of the same label as its own label
       according to the flagMessage received }
    5.    if $|F_i| = |V|/2 \times k$
    6.      each vertex in $F_i$
    7.        the vertex's lock changes from false to true
    8.    else if
    9.  end for
    10.  until stop condition not met
```

Step 2: Determining the community sequence. $edge_{i,j}$ denotes the number of edges between the different communities F_i and F_j $(i \neq j)$. The average capacity of each unit is $c = \lceil |V|/k \rceil$. c_{left} is defined as the remaining capacity of the processing unit. The order of communities is determined as follows:

(1) Randomly select one community Fa as the first sequence item, and follow Step 3 to load it into the processing unit.
(2) If $c_{left} \geq max_p\{edge_{p,F_a}\}(p \neq F_a)$, select the next community $|F_b| = max_p \{edge_{p,F_a}\}$.
(3) If $c_{left} < min_p\{edge_{p,F_a}\}(p \neq F_a)$, select the next community $|F_b| = min_p \{edge_{p,F_a}\}$.

Step 3: Streaming partitioning. For vertices in the same community, loaded the same unit in principle is optimal. Because of the load balancing requirement, vertices in the same community may not be in same processing unit. We use LDG algorithm to further optimize the fraction of edges cut.

Figure 2 depicts the MCS partitioning process (a) and the LDG partitioning process (b), $k = 2$. Our method detect the dense structure by graph clustering first and then using streaming algorithm to partition the specified community sequence. MCS algorithm is easier to partition the dense structure into subsets compared with LDG. The superiority of the proposed method is obvious from Fig. 2.

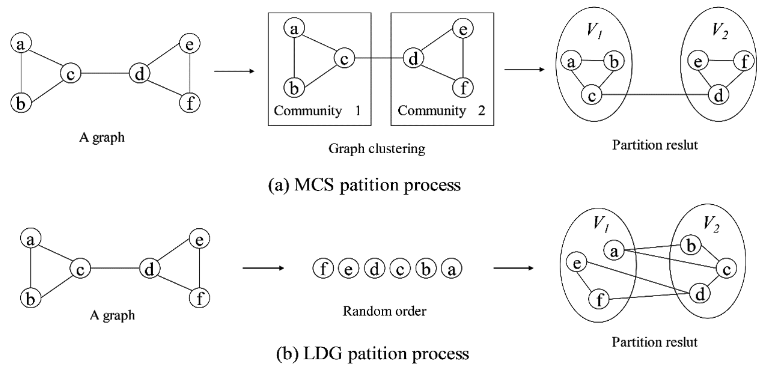

Fig. 2. The partitioning process of MCS algorithm and LDG algorithm.

3.2 Time Complexity Analysis

We analyze the time complexity for the MCS algorithm. In the first step, the complexity for traversing the whole graph is approximately O ($t \times |E|$), where t denotes the number of iterations ($t \leq 4$). In the second step, the determination of the community sequence requires time complexity at less than O (log ($|V|$)). In the third step, the streaming algorithm requires O (log ($|E|$)), and the time complexity of the whole MCS algorithm requires $O(t \times |E| + \log(|V| \times |E|))$.

4 Experimental Results and Analysis

4.1 Experimental Setup

The information of the real-world graph and the synthetic graph used in the experiment is shown in Table 1. All the experiments were carried out in a stand-alone Intel Xeon processor with 2.67 GHz, 32 G memory.

4.2 Algorithm Comparison

It has been generally observed that presenting the graph data in either breadth-first search (BFS), depth-first search (DFS), or Random search does not greatly alter performance [15]. Of these orders, a random ordering is the simplest to guarantee in large-scale streaming data scenarios, and so we restrict our analysis to only consider random vertex orders for simplicity.

Figure 3 shows the partitioning results of Amazon0312 (k = 32). For the fraction of edges cut, we can see that MCS is better than the traditional approaches and is closer to the result of Metis. The MCS equilibrium coefficient is 1.00.

Table 1. Datasets used in our experiments.

Graph	Nodes	Edges	Type
PL	1000	9895	Synthetic
ER	5000	1247739	Synthetic
amazon0312	400727	3200440	Co-purchasing
amazon0302	262111	1234877	Co-purchasing
amazon0505	410236	3356824	Co-purchasing
amazon0601	403394	3387388	Co-purchasing
Wiki-Talk	2394385	5021410	Communication
email-EuAll	265214	420045	Communication
web-NotreDame	325729	1117563	Web
Web-Stanford	281903	1992636	Web
soc-LiveJournal1	4847571	68993773	Social
Twitter-2010	41652230	1468365182	Social

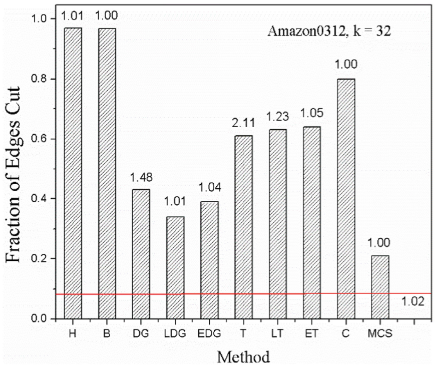

Fig. 3. The partitioning results of Amazon0312, k = 32. (Color figure online)

The red line represents the result of Metis (0.08%). The numerical values on the bar chart represent the equilibrium coefficients of the corresponding algorithms. The equilibrium coefficient 2.11 of triangle algorithm (T) is the maximum. The equilibrium coefficient of the proposed algorithm is 1.00, and its fraction of edges cut is 21%, which is the closest to that of Metis.

We use the graphs with different characteristics to analyze the applicability of the MCS algorithm. The power-law graph (PL), dense graph and non-power-law graph (ER) generated by NetworkX software package. The upper right corners of Fig. 4 shows the frequency of degree. Figure 4 shows the partitioning results of the various heuristic algorithms on the ER (left) and PL (right) graph, respectively. The vertical axis shows the fraction of edges cut. The red line shows the partitioning results of Metis. A closer value to Metis means the partitioning is of better quality. The partitioning result of ER is shown in Fig. 4(left), and the quality of the Linear Weight Deterministic Greedy algorithm is the best, which reached 83%. The hash result of 92%

is the worst, and our proposed algorithm in this paper is 85%, with Metis at 81%. Figure 4(right) shows the PL graph partitioning result. The fraction of edges cut by our method is 49%, with Metis at 50%. Because the characteristic 'world let' of the power-law network graphs are very suitable for the initial clustering, our method is very suitable for the partitioning of the power-law graphs.

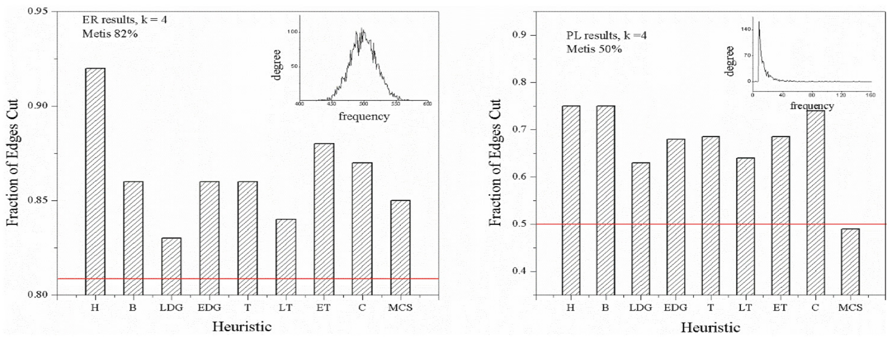

Fig. 4. ER (left) and PL (right) results, k = 4.

The first step of the MCS algorithm is the label selection problem for the vertex. If the same number of neighbors labels more than one category, which includes randomness. Therefore, we need to verify whether the randomness has an influence on the partitioning results. In Fig. 5, we show the stability results of the MCS algorithm on three real-world graphs (amazon0505, amazon0312, and amazon0601). The vertical axis shows the Difference = $\lambda_t - \lambda_{t+1}$ (the difference between the fraction of edges cut from the previous experiment and the fraction of edges cut from the next experiment), and the abscissa indicates the number of running times. It is obvious that the fluctuation rate is small and it can even be negligible for large-scale graphs from Fig. 5.

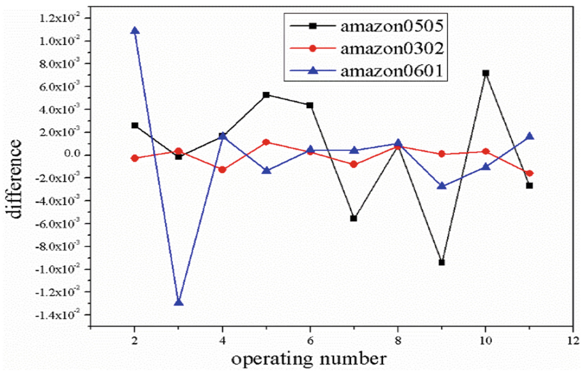

Fig. 5. Stability testing of our proposed approach.

Finally, the efficiency we need to verify of our algorithm mainly refers to the time of graph partitioning. However, because the time of partitioning is obviously higher than that of Hash and the purpose of the algorithm is to reduce the time of graph computation. The efficiency here is mainly embodied in the total time of the three phases of the graph loading, the graph partitioning and the graph computation. If the total time is lower than the most widely used Hash partitioning, then the effectiveness of the algorithm can be proved. The results in Table 2 show the total time for different iterative times of PageRank to compute the Twitter graph. The choice of PageRank was mainly because PageRank is familiar and the procedure can easily be achieved, and the iteration needs to traverse every edge and every vertex in the graph for each time. Through the optimization, we can determine whether the assumption is valid by computational time. The results in Table 2 also show the relationship between the computation of the total time and edges cut. For the same size graph, the less number of edges cut, the less computation time of the graph under the same condition.

Table 2. Performance of the partitioning algorithms.

Data set	Machines(k)	Iterations	Runtime (min)	
			Hash	MCS
Twitter-2010	20	6	39.2	**37.8**
		9	65.42	**61.7**
		12	85.8	**76.69**
		15	98.4	**83.21**
	50	6	17.72	21.7
		9	26.16	**26.63**
		12	34.09	**32.09**
		15	42.3	**39.6**

From Table 2, it is obvious that when the number of iterations is small, the total time is more than the hash method due to the high initial partition time complexity of MCS method. However, as the number of iterations increases, the advantage of the less number of edges cut (traffic) become more apparent relative to hash method. For example, when the number of iterations is 15 on 20 machines, the total time is reduced by nearly 15%. The experimental results formulate the verity the effectiveness of the MCS algorithm. The total time for different iterative times of PageRank computation on the Twitter-2010 graph, Hash and MCS. The difference between the two experiments gradually increases. With the increase in the number of iterations, the advantages of MCS is more obvious.

5 Conclusion

In this paper, for the computation of large-scale knowledge graph, we propose a new graph-partitioning algorithm (MCS) based on message cluster and streaming partitioning. The process of MCS algorithm is to cluster the original input graphs quickly and then using the streaming algorithm to send the vertices to the specified processing unit. MCS is a great improvement on the traditional partitioning algorithms in terms of the fraction of edges cut as well as the low time complexity. Compared with the traditional algorithms, in partition quality MCS is closer to or even surpasses Metis package. In partition efficiency, we use the PageRank algorithm in the spark cluster system to compute the Twitter graph data, and the total time of MCS is less than that of Hash partitioning. With an increasing number of iterations, the advantage is more obvious.

Research and the widespread application of the distributed system have posed a severe challenge to the effect and efficiency of the graph-partitioning algorithm. In future research, we will continue to study the graph partitioning, hoping to improve the fraction of edges cut and the time complexity.

References

1. Wei, Y., Luo, J., Xie, H.: KGRL: an OWL2 RL reasoning system for large scale knowledge graph. In: 12th International Conference on Semantics, Knowledge and Grids, pp. 83–89. IEEE, Piscataway (2017)
2. Chen, J., Chen, Y., Du, X., et al.: SEED: a system for entity exploration and debugging in large-scale knowledge graphs. In: 32nd IEEE, International Conference on Data Engineering, pp. 1350–1353. IEEE, Piscataway (2016)
3. Passant, A.: dbrec—Music Recommendations Using DBpedia. The Semantic Web. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17749-1_14
4. Tan, Z., Zhao, X., Fang, Y., et al.: GTrans: generic knowledge graph embedding via multi-state entities and dynamic relation spaces. IEEE Access **6**(99), 8232–8244 (2018)
5. Dong, X., Gabrilovich, E., Heitz, G., et al.: Knowledge vault: a web-scale approach to probabilistic knowledge fusion. In: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 601–610. ACM, New York (2014)
6. Auer, S., Bizer, C., Kobilarov, G., et al.: Dbpedia: A Nucleus for a Web of Open Data. The Semantic Web. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-76298-0_52
7. Hoffart, J., Suchanek, F.M., Berberich, K., et al.: YAGO2: A spatially and temporally enhanced knowledge base from Wikipedia. Artif. Intell. **1**(194), 28–61 (2013)
8. Zaharia, M., Chowdhury, M., Franklin, M.J., et al.: Spark: cluster computing with working sets. HotCloud **10**(95), 10 (2010)
9. Malewicz, G., Austern, M.H., Bik, A.J., et al.: Pregel: a system for large-scale graph processing. In: Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, pp. 135–146. ACM, New York (2010)
10. Low, Y., Gonzalez, J.E., Kyrola, A., et al.: Graphlab: a new framework for parallel machine learning. arXiv preprint arXiv,1408-2041 (2014)
11. Karypis, G., Kumar, V.: A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM J. Sci. Comput. **1**(20), 359–392 (1998)

12. Dutt, S.: New faster kernighan-lin-type graph-partitioning algorithms. In: Proceedings of 1993 International Conference on Computer Aided Design (ICCAD), pp. 370–377. IEEE, Piscataway (1993)
13. Battaglino, C., Pienta, P., Vuduc, R.: GraSP: distributed streaming graph partitioning. In: Proceeding of first High performance Graph Mining Workshop, Sydney (2015)
14. Nishimura, J., Ugander, J.: Restreaming graph partitioning: simple versatile algorithms for advanced balancing. In: Proceedings of 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1106–1114. ACM, New York (2013)
15. Stanton, I., Kliot, G.: Streaming graph partitioning for large distributed graphs. In: Proceedings of 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1222–1230. ACM, New York (2012)
16. Chen, L., Li, X., Sheng, Q.Z., et al.: Mining health examination records—a graph-based approach. IEEE Trans. Knowl. Data Eng. **9**(28), 2423–2437 (2016)