Analysis and Management to Hash-Based Graph and Rank

Yangtao Wang[1], Yu Liu[1], Yifei Liu[1], Ke Zhou[1](✉), Yujuan Yang[1], Jiangfeng Zeng[1], Xiaodong Xu[1], and Zhili Xiao[2]

[1] Huazhong University of Science and Technology, Wuhan, China
{ytwbruce,liu_yu,yifeiliu,k.zhou,gracee, jfzeng,xiaodong-xu}@hust.edu.cn
[2] Tencent Inc., Shenzhen, China
tomxiao@tencent.com

Abstract. We study the problem of how to calculate the importance score for each node in a graph where data are denoted as hash codes. Previous work has shown how to acquire scores in a directed graph. However, never has a scheme analyzed and managed the graph whose nodes consist of hash codes. We extend the past methods and design the undirected hash-based graph and rank algorithm. In addition, we present addition and deletion strategies on our graph and rank.

Firstly, we give a mathematical proof and ensure that our algorithm will converge for obtaining the ultimate scores. Secondly, we present our hash based rank algorithm. Moreover, the results of given examples illustrate the rationality of our proposed algorithm. Finally, we demonstrate how to manage our hash-based graph and rank so as to fast calculate new scores in the updated graph after adding and deleting nodes.

Keywords: Analysis · Management · Hash-based · Graph · Rank

1 Introduction

Using graph structure [1,3] to construct data correlation and manage data is a popular method for data experts to mine data and extract knowledge. Calculating the global importance rank for each data contained in a graph has always been an important research topic in data analysis and information retrieval domain [13,14]. Graph-based algorithms have achieved great success in this aspect. Especially, as one of the most important graph-based algorithms, PageRank [11] has been widely applied and extended. However, one of the difficulties is to define the correlation between nodes on a graph. Several previous researches have explored this issue. PageRank [11] considers out-degree of related nodes as impact factor for data rank. [12] applies random walk to ranking community images for searching. [7] introduces the concept of probability to improve the RegEx in PageRank. However, above graph-based rank algorithms all focus on in-degree and out-degree, neglecting the weight on edges, resulting that they are

J. Shao et al. (Eds.): APWeb-WAIM 2019, LNCS 11641, pp. 289–296, 2019.
https://doi.org/10.1007/978-3-030-26072-9_22

not competent for quantization with weighted graphs. TexRank [10] and SentenceRank [5] take the weights on edges into consideration, both of which apply PageRank to improving their respective algorithms, but none of them provide detailed proof of convergence.

Above researchers generally use feature vectors represented by floating point numbers to measure the correlation between nodes while dealing with weighted graphs. However, vast resource cost caused by feature vectors makes it not applicable to utilize this method with the increase of scale of data. Especially for analysis of large-scale image data, the dimension and complexity of image features lead to greater complexity. For example, [2,4] extracted image features through content perception, built the graph using Euclidean or Cosine distance, and further acquired recommended result using improved PageRank algorithm. However, feature vectors will inevitably lead to huge storage overhead. At the same time, the metrics just like Euclidean distance will also bring unacceptable time cost. Moreover, along with massive data quantity expansion recent years, it becomes more and more computationally complex to obtain the correlation between nodes which denote high-dimensional floating point numbers. On the other side, hash techniques are often used in storage and retrieval fields. For example, Hua et al. [6] map data to hash code using Locality-Sensitive Hashing (LSH). Taking advantage of hash in retrieval, they can fast perform some operations like query. Besides, due to the easy "XOR" operation, it will be simple and convenient to measure the correlation between two objects denoted as hash codes.

In this paper, we combine hash with graph structure to establish a semantic information management theory paradigm, which can not only serve for big data analysis but also enrich the operations for graph database. Assuming that we have obtained corresponding hash codes, we can build a undirected weighted hash-based graph by leveraging the Hamming distance [8,15] between nodes. Our defined hash-based graph is a kind of graph of which the node value is a hash value and edge value is the Hamming distance between nodes. Based on this, we design a hash-based rank algorithm which can effectively compute the importance of each node. We will give a complete mathematical proof and analysis of our algorithm. In addition, in order to reduce computational overhead as much as possible when graph changes, we also provide a series of graph data management operations such as addition and deletion.

2 Convergence Analysis

Matrix A and B are both $n \times n$ square matrix and each column sum of them is 1. i and j are positive integers. Generally, we respectively denote A and B as

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \qquad B = \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix} \tag{1}$$

where $\forall\ i \in [1,\, n]$ satisfies $\sum_{j=1}^{n} a_{ji} = 1$ and $\sum_{j=1}^{n} b_{ji} = 1$.

2.1 Matrix Product Convergence

Given matrix $C = AB$, we denote C as $\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix}$. We find that each column sum of C also satisfies: $\forall\ i \in [1,\ n]$,

$$\begin{aligned} \sum_{j=1}^{n} c_{ji} &= \sum_{j=1}^{n} a_{1j}b_{ji} + \sum_{j=1}^{n} a_{2j}b_{ji} + \cdots + \sum_{j=1}^{n} a_{nj}b_{ji} \\ &= b_{1i}\sum_{j=1}^{n} a_{j1} + b_{2i}\sum_{j=1}^{n} a_{j2} + \cdots + b_{ni}\sum_{j=1}^{n} a_{jn} \\ &= b_{1i} + b_{2i} + \cdots + b_{ni} = 1 \end{aligned} \tag{2}$$

2.2 Vector Convergence

R is a column vector whose column sum is r. We denote R as $R = [r_1\ r_2\ \cdots\ r_n]^T$, where $\sum_{j=1}^{n} r_j = r$. Given $R' = AR = [r'_1\ r'_2\ \cdots\ r'_n]^T$, we find that the column sum of R' satisfies:

$$\begin{aligned} \sum_{j=1}^{n} r'_j &= \sum_{j=1}^{n} a_{1j}r_j + \sum_{j=1}^{n} a_{2j}r_j + \cdots + \sum_{j=1}^{n} a_{nj}r_j \\ &= r_1\sum_{j=1}^{n} a_{j1} + r_2\sum_{j=1}^{n} a_{j2} + \cdots + r_n\sum_{j=1}^{n} a_{jn} \\ &= r_1 + r_2 + \cdots + r_n = r \end{aligned} \tag{3}$$

Also, for $\forall\ k \in Z^+$, we can conclude that each column sum of A^kR is also 1. Consequently, A^kR will never diverge as k becomes larger if A^k converges.

3 Hash-Based Graph

Given a graph G consisting of n nodes, each node is denoted as a l-bits hash code. N_* denotes the $*$-th node of Graph G and $H(N_*)$ denote the hash code of N_*. We define XOR operation as $\oplus$ and threshold $\Omega \in [1, l] \cap Z^+$. Different from the work of the predecessors [9], we stipulate that two nodes are connected only if the Hamming distance between them does not exceed threshold Ω. Therefore, the Hamming distance weight on undirected edge between N_i and N_j is defined as:

$$d_{ij} = \begin{cases} H(N_i) \oplus H(N_j) & i \neq j, H(N_i) \oplus H(N_j) \leq \Omega, \\ NULL & otherwise. \end{cases} \tag{4}$$

As a result, our hash-based graph has been established.

4 Hash-Based Rank

In this Section, we demonstrate our designed hash-based rank algorithm on our weighted undirected hash-based graph. Our goal is to calculate the importance score of each node. For $\forall\ i \in [1,\ n]$, T_i is defined as the set including orders of all nodes connected with N_i, where $T_i \subset [1,\ n]$.

As defined in Sect. 3, l denotes the length of hash code and d_{ij} denotes weight of the edge between N_i and N_j. We denote $R(N_*)$ as the importance score of N_*. Referring to PageRank, we also intend to calculate the ultimate $R(N_*)$ by means of iteration. Draw impact factor $I(N_{ij})$ for N_j to N_i which measures how N_j contributes to N_i, where $I(N_{ij})$ is defined as:

$$I(N_{ij}) = \begin{cases} \dfrac{l - d_{ij}}{\sum\limits_{t \in T_j} l - d_{tj}} R(N_j) & \exists d_{ij}, \\ 0 & otherwise. \end{cases} \tag{5}$$

Theoretically, we design Eq. (5) according to two principals. Firstly, the less d_{ij} is, the greater influence N_j contributes to N_i is. Meanwhile, the longer hash code (l) is, the more compact the similarity presented by d_{ij} is. Secondly, PageRank considers all (unweighted) edges as the same, but we extend it to be applied to different weights on edges. Specially, when all weights on edges are the same, our hash-based rank algorithm will turn into undirected PageRank. Consequently, $R(N_i)$ should be equal to the sum of the impact factors of all nodes connected to N_i, where N_i is expressed as: $R(N_i) = \sum\limits_{j=1, j \neq i}^{n} I(N_{ij})$.

Let f_{ij} represent the coefficient of $R(N_j)$ in $I(N_{ij})$, where

$$f_{ij} = \begin{cases} \dfrac{l - d_{ij}}{\sum\limits_{t \in T_j} l - d_{tj}} & \exists d_{ij}, \\ 0 & otherwise. \end{cases} \tag{6}$$

We define coefficient matrix D as $\begin{bmatrix} 0 & f_{12} & \cdots & f_{1n} \\ f_{21} & 0 & \cdots & f_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ f_{n1} & f_{n2} & \cdots & 0 \end{bmatrix}$ and calculate each column sum of D according to Eq. (6), we take the jth column as

$$\begin{aligned} & f_{1j} + f_{2j} + \cdots + f_{nj} \\ = & \frac{l - d_{1j}}{\sum\limits_{t \in T_j} l - d_{tj}} + \frac{l - d_{2j}}{\sum\limits_{t \in T_j} l - d_{tj}} + \cdots + \frac{l - d_{nj}}{\sum\limits_{t \in T_j} l - d_{tj}} \\ = & \sum_{t \in T_j} \frac{l - d_{tj}}{\sum\limits_{t \in T_j} l - d_{tj}} = 1 \end{aligned} \tag{7}$$

Usually, the initial value is set as $R^0 = [R^0(N_1)\ R^0(N_2)\ \cdots\ R^0(N_n)]^T = [1\ 1\ \cdots\ 1\ ]^T$. We draw iteration formula as

$$R^{k+1} = DR^k \tag{8}$$

where R^k=$[R^k(N_1)\ R^k(N_2)\ \cdots\ R^k(N_n)\ ]^T$, and k is the number of iteration rounds. According to Eq. (3), vector R^k will converge when k becomes larger.

We define the termination condition as $R^{k+1}(N_m) - R^k(N_m) \le \varepsilon$, where $m \in [1, n]$. Meanwhile, ε is set to a small constant (say 0.0001).

Thus, our designed hash-based rank algorithm converges. Then we will illustrate the result of the algorithm. As shown in Fig. 1, we use a graph G_1 with 10 nodes to verify our algorithm. Each node is a 48-bits hash code. We set termination condition $\varepsilon = 1.0E{-}8$, threshold $\Omega = 24$ (see Eq. (4)) and $[R^0(N_1)\ R^0(N_2)\ \cdots\ R^0(N_{10})]^T = [1\ 1\ \cdots\ 1]^T$. The score and rank for each node are displayed in Table 1.

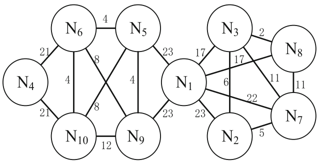

Fig. 1. Example graph G_1 with 10 nodes.

Table 1. Score and rank in graph G_1.

Node	Hash code	Score	Rank
N_1	FFFFFFFFFFFF	1.14788732	1
N_2	FFFFFF800000	1.05633802	6
N_3	FFFFFFFE0000	1.09859154	2
N_4	000000000000	0.38028169	10
N_5	C000007FFFFF	1.0774648	5
N_6	0000001FFFFF	1.09154931	3
N_7	FBFF7F8000E0	1.00704224	9
N_8	FFFFFF7E0080	1.08450703	4
N_9	C0003079FFFF	1.02112677	8
N_{10}	0300001FFE7F	1.03521128	7

As shown in graph G_1, each node is influenced by both of the edges and weights. If a node owns more edges with lower weights, it will obtain higher score and rank. For example, N_1 owns the most connections, so it acquires the highest score and rank. N_3 has the same number of connections as N_9, but the weights on edges connected with N_3 are lower than that of N_9. Thus, N_3 owns a higher rank than N_9. N_4 obtains the lowest score and rank because of fewest connections with high weights. The result in Table 1 is deemed reasonable.

5 Management to Hash-Based Graph and Rank

In this Section, we demonstrate how to manage our hash-based graph and rank algorithm when adding or deleting nodes.

Actually, if we intend to calculate the score and rank for each node in a updated graph, we have to obtain the corresponding updated coefficient matrix D. In Sect. 4, without isolated nodes, the graph consists of n nodes and the coefficient matrix D has been calculated according to Eq. (6). In the next part of this Section, we mainly introduce how to perform minimal change to coefficient matrix D when adding and deleting a node in the graph.

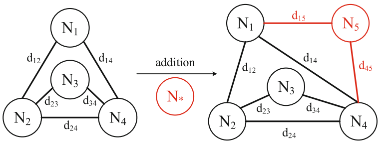

Fig. 2. Example graph G_2 by addition operation.

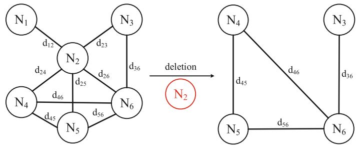

Fig. 3. Example graph G_3 by deletion operation.

5.1 Addition

Generally, when a new node is added to graph G, this node will be marked as N_{n+1} by default if it is connected with one of the n nodes. (As shown in Fig. 2, N_* is added to graph G_2 which contains 4 nodes. We directly mark N_* as N_5 because N_* is connected with N_1 and N_4.) And T_{n+1} (defined in Sect. 4) denotes the set including orders of all nodes connected with N_{n+1} where $T_{n+1} \subseteq [1, n]$. Then we analyze how matrix D will change and calculate the scores and ranks for $n+1$ nodes.

Algorithm 1. Calculate D_{n+1} and scores for $n+1$ nodes when adding a node.

Calculate $d_{i(n+1)}$ and set T_{n+1} for $\forall\ i \in \{1, 2, \cdots, n\}$.
Judge whether T_{n+1} is a empty set and calculate the following D_{n+1}.
Directly Calculate the ith column elements of D_{n+1} based on D_n and update T_i for $\forall\ i \in \{1, 2, \cdots, n\} \cap T_{n+1}$.
Directly Calculate the ith column elements of D_{n+1} based on D_n for $\forall\ i \in \{1, 2, \cdots, n\} \backslash T_{n+1}$.
Calculate the $(n+1)$th column elements of D_{n+1} according to Equation (6).
Calculate scores for $n+1$ nodes according to Equation (8) using D_{n+1}.

For convenience, we denote the $n \times n$ matrix D as D_n. When adding a node, we need to calculate D_{n+1} based on D_n. As shown in Algorithm 1, we describe the steps that calculate matrix D_{n+1} and scores for $n+1$ nodes.

5.2 Deletion

Similarly, if we delete a node from graph G, how can we fast adjust the matrix D_n and calculate score for each node in the new graph? For example, as shown in Fig. 3, N_2 is deleted from G_3 which contains 6 nodes. However, N_1 will be removed from G_4 because N_1 is only connected with N_2. Also, those edges which are connected with N_2 will also disappear. Generally, for $i \in \{1, 2, \cdots, n\}$, once we delete N_i from graph G, those edges connected with N_i will be removed from G. Of course, if N_i is deleted, those isolated nodes will be also removed.

Algorithm 2. Adjust D_n and calculate scores for remaining nodes when deleting a node.

1: Calculate the set I_i which contains the orders of those nodes that are only connected with N_i.
2: Judge whether $(n - |I_i|)$ equals 1 and calculate the following $D_{n-|I_i|-1}$.
3: Directly adjust the tth column elements of D_n for $t \in \{1, 2, \cdots, n\} \cap T_i \backslash I_i$.
4: Directly reserve the tth column elements of D_n for $t \in \{1, 2, \cdots, n\} \backslash T_i$.
5: Calculate the expected $D_{n-|I_i|-1}$ by deleting the tth rows as well as the tth column elements of D_n.
6: Calculate scores for the remaining $n - |I_i| - 1$ nodes according to Equation (8) using $D_{n-|I_i|-1}$.

As shown in Algorithm 2, we describe the steps that analyze how matrix D_n will change and calculate scores for the remaining nodes after deleting N_i.

In this Section, we demonstrate how to manage our hash-based graph and rank when faced with addition and deletion operations by giving fast calculation method of the iteration matrix. Incidentally, the operation of modifying a node is actually such a process that we first delete (Algorithm 2) a node and then add (Algorithm 1) a node. We will not elaborate this process due to limited space.

6 Conclusion

This paper builds a hash-based graph using restricted Hamming distance and proposes an undirected hash-based rank algorithm to calculate importance score for each node. By analyzing the iterative matrix, we give a full mathematical proof to verify that our algorithm will converge. Moreover, we illustrate the rationality of our algorithm. At last, we demonstrate how to manage our hash-based graph and rank by performing the minimal change strategy after adding and deleting a node, which can dynamically and fast compute the score and rank for each node in the updated graph.

Acknowledgements. This work is supported by the Innovation Group Project of the National Natural Science Foundation of China No. 61821003 and the National Key Research and Development Program of China under grant No. 2016YFB0800402 and the National Natural Science Foundation of China No. 61672254. Thanks for Jay Chou, a celebrated Chinese singer whose songs have been accompanying the author.

References

1. Baeza, P.B.: Querying graph databases. In: Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2013, 22–27 June 2013, New York, NY, USA, pp. 175–188 (2013). https://doi.org/10.1145/2463664.2465216
2. Cai, H., Huang, Z., Srivastava, D., Zhang, Q.: Indexing evolving events from tweet streams. In: ICDE, pp. 1538–1539 (2016)
3. Cuzzocrea, A., Jiang, F., Leung, C.K.: Frequent subgraph mining from streams of linked graph structured data. In: Proceedings of the Workshops of the EDBT/ICDT 2015 Joint Conference (EDBT/ICDT), 27 March 2015, Brussels, Belgium, pp. 237–244 (2015). http://ceur-ws.org/Vol-1330/paper-37.pdf
4. Gao, S., Cheng, X., Wang, H., Chia, L.: Concept model-based unsupervised web image re-ranking. In: ICIP, pp. 793–796 (2009)
5. Ge, S.S., Zhang, Z., He, H.: Weighted graph model based sentence clustering and ranking for document summarization. In: ICIS, pp. 90–95 (2011)
6. Hua, Y., Jiang, H., Feng, D.: FAST: near real-time searchable data analytics for the cloud. In: International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2014, 16–21 November 2014, New Orleans, LA, USA, pp. 754–765 (2014). https://doi.org/10.1109/SC.2014.67
7. Lei, Y., Li, W., Lu, Z., Zhao, M.: Alternating pointwise-pairwise learning for personalized item ranking. In: Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, pp. 2155–2158. ACM (2017)
8. Liu, Y., et al.: Deep self-taught hashing for image retrieval. IEEE Trans. Cybern. **49**(6), 2229–2241 (2019)
9. Michaelis, S., Piatkowski, N., Stolpe, M. (eds.): Solving Large Scale Learning Tasks, Challenges and Algorithms - Essays Dedicated to Katharina Morik on the Occasion of Her 60th Birthday. LNCS (LNAI), vol. 9580. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-41706-6
10. Mihalcea, R.: Graph-based ranking algorithms for sentence extraction, applied to text summarization. Unt Sch. Works **170–173**, 20 (2004)
11. Page, L., Brin, S., Motwani, R., Winograd, T.: The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab (1999)
12. Richter, F., Romberg, S., Hörster, E., Lienhart, R.: Multimodal ranking for image search on community databases. In: MIR, pp. 63–72 (2010)
13. Wang, Y., Zhu, L., Qian, X., Han, J.: Joint hypergraph learning for tag-based image retrieval. IEEE Trans. Image Process. **PP**(99), 1 (2018)
14. Yang, J., Jie, L., Hui, S., Kai, W., Rosin, P.L., Yang, M.H.: Dynamic match Kernel with deep convolutional features for image retrieval. IEEE Trans. Image Process. **27**(11), 5288–5302 (2018)
15. Zhou, K., Liu, Y., Song, J., Yan, L., Zou, F., Shen, F.: Deep self-taught hashing for image retrieval. In: Proceedings of the 23rd Annual ACM Conference on Multimedia Conference, MM 2015, 26–30 October 2015, Brisbane, Australia, pp. 1215–1218 (2015). https://doi.org/10.1145/2733373.2806320