An Implementation of the HITS Algorithm for LAGraph

Aurko Routh

December 9, 2023

Abstract

In this study, we introduce a detailed implementation of the Hyperlink-Induced Topic Search (HITS) algorithm within the LAGraph framework, and we benchmark its performance against the LAGraph implementation of the PageRank algorithm using a suite of social network matrices.

1 Introduction

The Hyperlink-Induced Topic Search (HITS) algorithm, initially proposed by Jon Kleinberg in 1999, represents a cornerstone in the field of web link analysis. Conceptualized in the early days of the World Wide Web, the HITS algorithm emerged as a pioneering method for determining the relative "authority" and "hub" values of web pages. An authority is a page containing valuable information on a specific topic, while a hub is a page that links to multiple authorities. The reciprocal relationship between hubs and authorities allows the HITS algorithm to effectively rank web pages based on their topical relevance and interconnectedness.

The burgeoning volume of web data and the increasing complexity of web structures necessitate efficient and scalable methods for executing graph-based algorithms like HITS. In this scenario, SuiteSparse GraphBLAS emerges as an essential tool. SuiteSparse GraphBLAS, a specific implementation of the GraphBLAS API, excels in handling sparse matrices, a key aspect of graph computations. GraphBLAS is a specification that defines standard building blocks for graph algorithms in the language of linear algebra. Combined with LAGraph, a library built on SuiteSparse GraphBLAS, it provides algorithms, utilities, and best practices for graph analytics, enhancing the accessibility and efficiency of complex graph algorithms.

The LAGraph library further extends these capabilities. It offers a collection of algorithms, utilities, and best practices for graph analytics, making complex graph algorithms more accessible and efficient. The integration of SuiteSparse GraphBLAS into LAGraph paves the way for more efficient graph analytics, especially for algorithms like HITS, which inherently depend on matrix operations.

This paper aims to explore the implementation of the HITS algorithm within the LAGraph library using GraphBLAS. We intend to demonstrate how the abstract algebraic operations defined in GraphBLAS can be utilized to realize the iterative calculations of the HITS algorithm. By doing so, we expect to achieve a more scalable and

efficient approach to web link analysis, suitable for the vast and growing graph datasets of today's internet landscape.

2 Algorithm Explained

The central idea of HITS is rooted in the observation that some web pages (authorities) are valuable sources of information, while others (hubs) serve as aggregators of these resources. The algorithm operates on a directed graph where nodes represent web pages, and edges denote hyperlinks between them. Through an iterative process, each node is assigned two scores: an authority score, signifying the value of the information contained within the page, and a hub score, indicating the quality of its links to authority pages. These scores are updated iteratively, with each node's authority score being determined by the sum of the hub scores of pages linking to it, and each hub score computed from the authority scores of the pages it links to. Throughout the iterative process, the hub and authority scores undergo normalization to maintain consistency. Convergence for the hub and authority scores is achieved when the absolute value of the difference between these scores, across consecutive iterations, falls below a predetermined tolerance threshold. The pseudo-code provided below offers a representation of this algorithm's standard implementation.

Algorithm 1 HITS Algorithm

Input: Graph $G(V, E)$ with vertices V and edges E
Output: Hub scores $h(v)$ and Authority scores $a(v)$ for each vertex $v \in V$
procedure HITS(G, *tolerance*, *maxIterations*)
    Initialize $h(v) \leftarrow 1$ for all $v \in V$
    Initialize $a(v) \leftarrow 1$ for all $v \in V$
    Initialize *iteration* $\leftarrow 0$
    Initialize *converged* $\leftarrow$ False
    while not *converged* and *iteration* ¡ *maxIterations* **do**
        $h_{\text{old}}(v) \leftarrow h(v)$ for all $v \in V$
        $a_{\text{old}}(v) \leftarrow a(v)$ for all $v \in V$
        for each vertex $v \in V$ **do**
            $a(v) \leftarrow \sum_{u \in \text{Neighbors}(v)} h_{\text{old}}(u)$
            $h(v) \leftarrow \sum_{u \in \text{Neighbors}(v)} a_{\text{old}}(u)$
        end for
        Normalize $a(v)$ for all $v \in V$
        Normalize $h(v)$ for all $v \in V$
        Check for convergence based on *tolerance*
        iteration $\leftarrow$ *iteration* $+ 1$
    end while
end procedure

3 Algorithm implementation

This code implements the Hyperlink-Induced Topic Search (HITS) algorithm using the SuiteSparse GraphBLAS API.

The function `LAGr_HITS` is defined to calculate the hub and authority scores of nodes in a graph. It takes several parameters, including pointers to vectors for hubs and authorities, the graph `G`, tolerance for convergence `tol`, maximum number of iterations `itermax`, and a message string `msg`.

```
int LAGr_HITS(
    GrB_Vector * hubs,
    GrB_Vector* authorities,
    int * iters,
    const LAGraph_Graph G,
    float tol,
    int itermax,
    char *msg
) {
    ...
}
```

The code checks the structure of the graph `G`. If the graph is undirected or has a symmetric structure, it uses the adjacency matrix `G->A` to store the transpose matrix. Otherwise, it uses the pre-cached transpose adjacency matrix `G->AT`.

```
    GrB_Matrix AT ;
    if (G->kind == LAGraph_ADJACENCY_UNDIRECTED ||
        G->is_symmetric_structure == LAGraph_TRUE)
    {
        AT = G->A ;
    }
    else
    {
        AT = G->AT ;
        LG_ASSERT_MSG (AT != NULL,
            LAGRAPH_NOT_CACHED, "G->AT is required") ;
    }
```

It initializes the number of nodes `n`, hub and authority vectors `h` and `a`, and their previous values `h_old` and `a_old`. It sets the initial values of hubs and authorities to `1/n`.

```
    GRB_TRY (GrB_Vector_new (&h_old, GrB_FP32, n)) ;
    GRB_TRY (GrB_Vector_new (&a_old, GrB_FP32, n)) ;
    GRB_TRY (GrB_Vector_new (&h, GrB_FP32, n));
    GRB_TRY (GrB_Vector_new (&a, GrB_FP32, n)) ;

    float defaultValue = 1.0/n;
    GRB_TRY(GrB_assign(a, NULL, NULL, defaultValue, GrB_ALL, n, NULL));
    GRB_TRY(GrB_assign(h, NULL, NULL, defaultValue, GrB_ALL, n, NULL));
```

We then calculate the number of non-zero entries in both the in-degree and out-degree vectors. We can assume for this implementation the in-degree and out-degree vectors are cached in the LAGraph object. Depending on if the sum of these 2 entries is greater than one-sixteenth of the number of nodes in the graph, we will be using a different internal kernel for the hubs and authorities calculation.

```
    int indegree, outdegree;

    GrB_Vector_nvals(&indegree, G->in_degree);
    GrB_Vector_nvals(&outdegree, G->out_degree);

    bool flag = (indegree + outdegree) > n/16.0;
```

The loop starts with initializing the iteration counter and the variable `rdiff`, which is used to track the difference between the scores in successive iterations for convergence checking.

```
for((*iters) = 0; (*iters) < itermax && rdiff > tol; (*iters)++) {
    ...
}
```

For each iteration, matrix-vector multiplications are used to update the hub and authority scores: $a = A^T \cdot h_{\text{old}}$ and $h = A \cdot a_{\text{old}}$, where A is the adjacency matrix, and A^T its transpose. The following code represents two possible branches to compute the hubs and authorities values. In the first case, we force the `h` and `a` vectors to 0 and use the accumulator operator when performing the matrix multiplication. This allows us to avoid computing the sparsity pattern overlap between vectors, which is more efficient for dense or semi-dense graphs. For extremely sparse graphs, we can perform the matrix-vector multiplication while only storing the present entries in the `h` and `a` vectors. This design choice will result in a different internal algorithm being used for the sparse case.

```
    if(flag) {
        //a = 0
        GRB_TRY(GrB_assign(a, NULL, GrB_PLUS_FP32, 0.0, GrB_ALL, n, NULL));
        //h = 0
        GRB_TRY(GrB_assign(h, NULL, GrB_PLUS_FP32, 0.0, GrB_ALL, n, NULL));
        // a += AT . h
        GRB_TRY(GrB_mxv(a, NULL,NULL, LAGraph_plus_second_fp32, AT, h_old, NULL));
        // h += A . a
        GRB_TRY(GrB_mxv(h, NULL,NULL, LAGraph_plus_second_fp32, G->A, a_old, NULL));
    } else {
        // a = AT . h
        GRB_TRY(GrB_mxv(a, NULL,NULL, LAGraph_plus_second_fp32, AT, h_old, NULL));
        // h = A . a
        GRB_TRY(GrB_mxv(h, NULL,NULL, LAGraph_plus_second_fp32, G->A, a_old, NULL));
    }
```

The function employs the L1 Norm for normalizing the hub and authority scores. Although Kleinberg's original algorithm utilizes the L2 Norm, this implementation aligns with the NetworkX Python implementation of the HITS algorithm, which adopts the L1 Norm for normalization in each iteration. The verification of the algorithm's correctness was conducted by comparing the output hub and authority scores against those generated by the NetworkX implementation.

```
float sumA;
// sumA = sum(a)
GRB_TRY(GrB_reduce(&sumA, NULL, GrB_PLUS_MONOID_FP32, a, NULL));
a /= sumA
GRB_TRY(GrB_assign(a, NULL, GrB_DIV_FP32, sumA, GrB_ALL, n, NULL));

float sumH;
// sumH = sum(h)
GRB_TRY(GrB_reduce(&sumH, NULL, GrB_PLUS_MONOID_FP32, h, NULL));
// h /= sumH
GRB_TRY(GrB_assign(h, NULL, GrB_DIV_FP32, sumH, GrB_ALL, n, NULL));
```

The loop checks for convergence by calculating the sum of the absolute differences between the new and old scores for both hubs and authorities. The convergence criterion is that this sum divided by two (`rdiff`) is less than the specified tolerance (`tol`).

```
// a_old -= a
GRB_TRY (GrB_assign(a_old, NULL, GrB_MINUS_FP32, a, GrB_ALL, n, NULL));
// a_old = abs(a_old)
GRB_TRY(GrB_apply (a_old, NULL, NULL, GrB_ABS_FP32, a_old, NULL));
// rdiff = sum(a_old)
GRB_TRY(GrB_reduce (&rdiff, NULL, GrB_PLUS_MONOID_FP32, a_old, NULL));
// h_old -= h
GRB_TRY (GrB_assign (h_old, NULL, GrB_MINUS_FP32, h, GrB_ALL, n, NULL));
// h_old = abs(h_old)
GRB_TRY (GrB_apply (h_old, NULL, NULL, GrB_ABS_FP32, h_old, NULL));
// rdiff += sum(h_old)
GRB_TRY (GrB_reduce (&rdiff, GrB_PLUS_FP32, GrB_PLUS_MONOID_FP32, h_old, NULL)) ;
// rdiff = rdiff/2
rdiff /= 2;
```

4 Performance Benchmark

In this analysis, we explore the performance of two principal link analysis algorithms, Hyperlink-Induced Topic Search (HITS) and PageRank, as applied to a suite of social network matrices. These matrices, representative of complex social networks, serve as a testbed to evaluate the computational efficiency and scalability of the algorithms implemented within the LAGraph framework.

PageRank, known for its application in search engine ranking, assigns a global score to each page in the graph based on the stationary distribution of a random walk. It

essentially measures the likelihood of arriving at a particular page through random clicks.

The experiments were conducted on a system with 24 CPUs, leveraging parallel processing capabilities. The following table represents the results of the performance test:

Table 1: Comparison of LAGraph HITS and PageRank Algorithms

Dataset	Runtime (seconds)		Iterations until convergence		Time per iteration (seconds)	
	HITS	PageRank	HITS	PageRank	HITS	PageRank
GAP Twitter	220.279	29.302	70	22	3.15	1.33
GAP Kron	407.086	66.6646	33	13	12.34	5.13
GAP Road	33.7223	1.857	295	39	0.11	0.048
GAP Web	363.293	15.8632	319	32	1.14	0.50

From the performance data, it is evident that HITS typically exhibits longer runtimes in comparison to PageRank. This can largely be ascribed to the algorithmic composition of HITS, which necessitates a pair of matrix multiplications per iteration to update both hub and authority scores. PageRank, conversely, requires only a single matrix multiplication per iteration to update page scores.

Additionally, the iterative nature of these algorithms highlights another dimension of performance. HITS generally demands a greater number of iterations to achieve convergence across all the datasets examined. This characteristic, coupled with the more computationally intensive nature of its iterations, accounts for the observed performance disparity.

In the comparison of the LAGraph HITS and PageRank algorithms, we observe that the HITS algorithm generally requires roughly twice the amount of time per iteration compared to the PageRank algorithm, which is consistent with our observations above.

References

[1] Kleinberg, J. M. (1999). Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5), 604–632.

[2] Page, L., Brin, S., Motwani, R., & Winograd, T. (1999). The PageRank citation ranking: Bringing order to the web. *Stanford InfoLab*.

[3] Mattson, T. G., Bader, D. A., & Kepner, J. (2013). GraphBLAS: Graph algorithms in the language of linear algebra. *ACM SIGPLAN Notices*, 48(8), 121–132.

[4] Davis, T. A., & Hu, Y. (2011). The University of Florida Sparse Matrix Collection. *ACM Transactions on Mathematical Software*, 38.