Bipartite Matching as a Graph Benchmark

***Abstract*—There are presently several graph benchmarks in the literature, some with hundreds of published processing reports. They all, however, have several characteristics that make them of academic, but not necessarily real-world interest. This paper proposes bipartite matching as a benchmark with real-world relevance. Variations in algorithms and implementations are discussed, with an emphasis on a reference implementation and possible scaling via parallelism.**

***Keywords*-Graphs algorithms, Jaccard coefficients, Scalability**

I. Introduction

A graph is of a set of objects (vertices), and links (edges) between pairs of objects that represent some sort of relationships. Computing over such graphs is of increasing importance to a wide spectrum of application areas ranging from "conventional" communication and power networks, transport, and scheduling, to emerging applications such as social networks, medical informatics, genomics, and cybersecurity.

While there are several current graph benchmarks, some with hundreds of reported implementations, most of them are based on "academic" graph problems, and often have little direct value to real-world applications, especially when we want to understand the relative efficiency of different hardware architectures and configurations. Further, given that many of these graphs are growing in size, it is critical that we understand how to do such processing in parallel in an efficient manner. A companion paper [1] suggests three alternatives selected to overcome the issues with the current suite. These include computation of the overlap in neighbors between pairs of vertices, determination of a set of edges that form matches between vertices, and stateful random walks.

This paper focuses on the second: identifying matches between vertices in a very large bipartite graph. ... In the real world, ... Common examples studied in the literature involve ... .

Another complex example ....

In addition to the kernel, a graph benchmark with real-world relevance should represent accurately how the graph is represented and provided to the underlying implementation. A *batch* implementation must read in something akin to the edge list of a graph and generate all possible coefficients in the graph. This corresponds to the way most graph benchmarks are defined today. An *in-memory* implementation instead may start with a graph already constructed as a directly accessible data structure, and may be more relevant to real-world applications that want to perform a variety of on-demand analytics. Two versions of this are relevant here; one where a query requests all coefficients within the graph as in a batch implementation, and one where the coefficients for only a specific subset of vertices (perhaps only one) is requested. In either case, implementations must be capable of handling multiple concurrent queries. A third implementation version of particular real-world relevance is one that *streams* updates into an in-memory graph, and requests a list of what coefficients may change as a result.

This paper defines a formal benchmark for the core of such problems. The rest of this paper is organized as follows. Section II introduces the problem formally and defines metrics. Section III discusses possible algorithms when all coefficients are desired. Section IV discusses streaming implementations. Section V discusses variations and heuristics that may help performance. Section VI overviews related work. Section VII discusses a sequential reference implementation. Section VIII discusses expression of the problem in a variety of graph programming paradigms. Section IX discusses parallel implementations and their expected scaling characteristics. Section X concludes. The Appendix lists some alternative implementations. Two companion papers [2], [3] cover the other two benchmarks proposed in [1] in a format similar to this paper.

II. Problem and Metrics

III. Algorithms

Algorithm 1 Sequential Hopcroft-Karp:
L = set of "left" vertices
R = set of "right" vertices
E = $\{(u, v)|$ u in L, v in $R\}$
M = set of edges in matching
A = $|L|$ by $|R|$ matrix where $A[i, j] = 1$ if (i, j) in E

```
1: procedure HK1(A)
2:     M ← ∅
3:     repeat
4:         for u in L do
5:             if u not in M and u starts some P then
6:                 M ← P_u ⊕ M
7:         end for
8:     until No augmenting paths exist
9:     end while
```

IV. Streaming

A final real-world variation may involve building algorithms that are **streaming**, that is some aspect of the graph changes with time. This may include the incremental addition or removal of vertices or edges. Given that adding a vertex adds no new γs, and a vertex cannot be removed until all edges

to it have been removed, we will concentrate only on edge addition and removal.

V. Variations and Heuristics

VI. Related Work

VII. Reference Implementation

A particular maximum bipartite matching algorithm of interest is that formulated by Hopcroft and Karp [4], due to its desirable asymptotic time complexity of $O(\sqrt{V}E)$ relative to many other maximum matching algorithms. The algorithm utilizes the concept of vertex disjoint augmenting paths. An augmenting path is formed by an alternating path, a path that alternates between matched and unmatched edges, with the additional requirement of starting and ending on vertices that are not part of M. We define an augmenting path P_u starting from u and ending at v. It can be shown that exchanging unmatched and matched edges along P_u, $P_u \oplus M$, always results in adding an edge to the matching, $|M| = |M| + 1$.

The Hopcroft-Karp algorithm may be implemented with a combination of breadth-first search (BFS) and depth-first search (DFS). A BFS creates a layered graph level by level with the first layer starting from unmatched vertices in L. The edges in the layered graph alternate between matched and unmatched, and the levels alternate between vertices in L and R. The BFS terminates at the first level when an unmatched vertex in R is found. If augmenting paths are found by the BFS, a DFS is used to find the maximal set of vertex disjoint augmenting paths. The DFS begins at the unmatched vertices in the first level in the layered graph and traverses layer by layer until an unmatched vertex in R is found. The edges along the path are then exchanged between matched and unmatched, increasing $|M|$. This process repeats until no augmenting paths are found by the BFS, meaning the maximum matching has been achieved.

NEEDS WRITEUP

A. Test Cases

A generator for bipartite graphs has been built in Python that allows separate specification of the size of L and R as well as the average degree. It then uses R-MAT, an algorithm based on that used in the Graph500 benchmark [5], but with edges only from L to R, and a separate set of probabilities for both the source vertices and the destination vertices. The original R-MAT algorithm produces oscillations within the degree distribution, and is not guaranteed to be monotonically decreasing. Random noise was added to the probabilities during the graph generation which has been shown to dampen these oscillations [6], producing distributions similar to those seen in real-world graphs. When generating large-scale graphs, no two consecutive graphs are identical due to the randomness introduced in the R-MAT algorithm.

NEEDS MORE DISCUSSION

VIII. Alternative Representations

IX. Scaling and Parallelism

Parallelism comes in three forms: shared memory multi-threading in an environment where all threads can see all the data structures, distributed memory multi-processing where the data structures are partitioned across different *nodes* (and explicit "messages" need be sent for interactions between them), and hybrid codes where both are employed. OpenMP is a typical programming environment for the former, and MPI for the middle.

X. Conclusions

Acknowledgment

This work was supported in part by NSF grant CCF-1642280, and in part by the University of Notre Dame.

Appendix

A. GraphBLAS

References

[1] P. M. Kogge, N. V. Chawla, D. Thain, B. A. Page, and N. A. Butcher, "Realistic computationally stressing graph benchmarks," In Preparation, 2018.

[2] ——, "Stateful random walks as a graph benchmark," In Preparation, 2018.

[3] ——, "Jaccard coefficients as a graph benchmark," In Preparation, 2018.

[4] J. E. Hopcroft and R. M. Karp, "A $n^{5/2}$ algorithm for maximum matchings in bipartite," in *12th Annual Symposium on Switching and Automata Theory (swat 1971)*, Oct 1971, pp. 122–125.

[5] D. Chakrabarti, Y. Zhan, and C. Faloutsos, "R-mat: A recursive model for graph mining," in *In In Fourth SIAM International Conference on Data Mining*, 2004.

[6] C. Seshadhri, A. Pinar, and T. G. Kolda, "An in-depth analysis of stochastic kronecker graphs," *J. ACM*, vol. 60, no. 2, pp. 13:1–13:32, May 2013. [Online]. Available: http://doi.acm.org/10.1145/2450142.2450149