Chapter 1

Jaccard Coefficients

Contributed by Neil Butcher

1.1 Introduction

Jaccard Coeffcients is a proposed High Performance Computing (HPC) metric that is used in a wide variety of real world applications. The Jaccard metric is designated as a way to define similarity between the neighborhood of two nodes. The Jaccard metric was originally introduced as a way to detect communities in botanical species [3]. This idea has been further expanded to other community detection algorithms[5] [1] [5], and well as other purposes. The Jaccard coefficient has been used by Wikipedia [2] to determine the relationship between web-pages based upon common authors of pages.

The example in figure 1.1 shows a problem that is best solved by using Jaccard coefficients. Suppose a insurance company is attempting to determine reliability of a person, as well as people they cohabitate with. In order to do this they have to determine the shared addresses. This can easily be represented as a Jaccard computation when configuring the graph in a manner similar to what is shown in the figure. [4]

1.2 Basic Definition

The Jaccard coefficient is quite simple. Given a pair of vertices U and V we represent Jaccard as the intersection of the neighborhoods of U and V, divided by the union of the neighborhoods of the two vertices. From a computational standpoint most work comes from computing the intersection of U and V. This is because to compute the union you can simply take the size of the neighborhoods and subtract by the intersection, as to not count the same vertex twice.

For examples look at 1.2. The figure shows a simple graph in which it is easy to demonstrate Jaccard computations. For example we demonstrate how to compute the Jaccard for vertex A and vertex D. The intersection of their neighborhoods is the single node, vertex B. The size of the union is two, nodes C and B. Note that despite that both A and D are neighbors of B, we only count B as one node in the union. This makes the Jaccard value 1/2. This simple example demonstrates the computation needed to compute a Jaccard value.

1.3 How to Compute Jaccard

The most obvious way to compute the Jaccard coefficient is to just brute force compare all the neighborhoods of all pairs of vertices. This of course will have a rather large complexity. This can be simplified greatly by first computing all of the two hop paths from each vertex. If two vertices don't contain a two hop path then their Jaccard coefficient will be zero, since there intersection must the empty. Given two hop paths simplifies the amount of work you have to perform, especially in a sparse matrix.

The problem of computing a single Jaccard coefficient is as stated earlier, is a problem of computing the intersection of two nodes. Computing the intersection of two nodes is essentially just comparing two lists and looking for common intersection. This becomes much simpler if the lists are sorted, based off vertex number. This reduces the amount of work required to compute the intersection by a factor of M, where M is the average out-degree of vertices. Finding the intersection of a two sorted lists is obviously easier, because it only requires going through both lists once. If the list is not sorted it becomes more complicated because you have to check all pairs in the lists and is N squared comparisons.

The complexity of computing Jaccard in this way is fairly straightforward. If just computing a single Jaccard element we have to at least do O(M) work assuming the list is already sorted. If we have to sort the lists first the work is O(MlogM) since that is the complexity of sorting. The last options is to just not sort the lists and that results in a O(M$\hat{2}$) complexity. This is obviously the worst case, however the constant from sorting a fairly short list may not be worth the cost in a real application.

Jaccard can also be computed by using the Graph Basic Linear Algebra Subroutines (GraphBLAS). The GraphBLAS library is a simple C interface that represents graphs as matrices and performs matrix operations to perform graph algorithms. Jaccard is a clear candidate for GraphBLAS, since it can compute the intersection of a list of nodes by multiplying the graph by itself. This results in entries that represent the intersection of the vertices that the rows/columns represent. This implementation is straightforward and has a potential to be highly parallelized.

1.4 Parallizing Jaccard

There are a wide variety of problems the can utilize the structure of a Jaccard coefficient. Since each Jaccard coefficient can be computed independently of the other, parallelizing the computation is fairly straightforward. There however can obviously be multiple caveats to computing the Jaccard coefficient. One simple and effective way to parallelize the computation is the use Hadoop MapReduce algorithms.

1.5 Real World Data Sets

In conducting experiments to observe the performance characteristics of different Jaccard algorithms there are a wide variety of data sets to choose from. The simplest choice is the RMAT graphs. These graphs are a dramatic simplification of real world problem, but are easy to demonstrate strong scaling behaviour. The RMAT graphs are artificial graphs produced for benchmarks, most notably Graph500. This makes RMAT graphs an interesting data point because one of the goals of developing Jaccard codes is to implement in as a benchmark alongside the BFS benchmark in Graph500.

Jaccard was initially developed as a community detection metric. This makes it an obvious candidate for graphs with clear and noticable communities. Previous work computes the similarity between Wikipedia pages. These data sets are available and make an obvious comparision point for any new work that is performed. There are also many other SNAP datasets that exist that have many clearly defined networks and make a useful point of comparison.

1.6 Next Steps

There are a wide variety of Jaccard algorithms that exist, many of them are quite clever. The goal of coming up with a new way of improving how parallel the computation is of great interest. The paper shows a great deal of work in computing triangle counting while keeping the working sets minimal. We think it would be interesting to adapt their algorithm towards Jaccard and tailor it towards to utilize the multi-channel DRAM (MCDRAM) found in Intel's Knight's Landing chip as well as perhaps GPU algorithms. We think that adapting this triangle counting algorithm will require careful consideration but will provide benefit when implemented correctly.

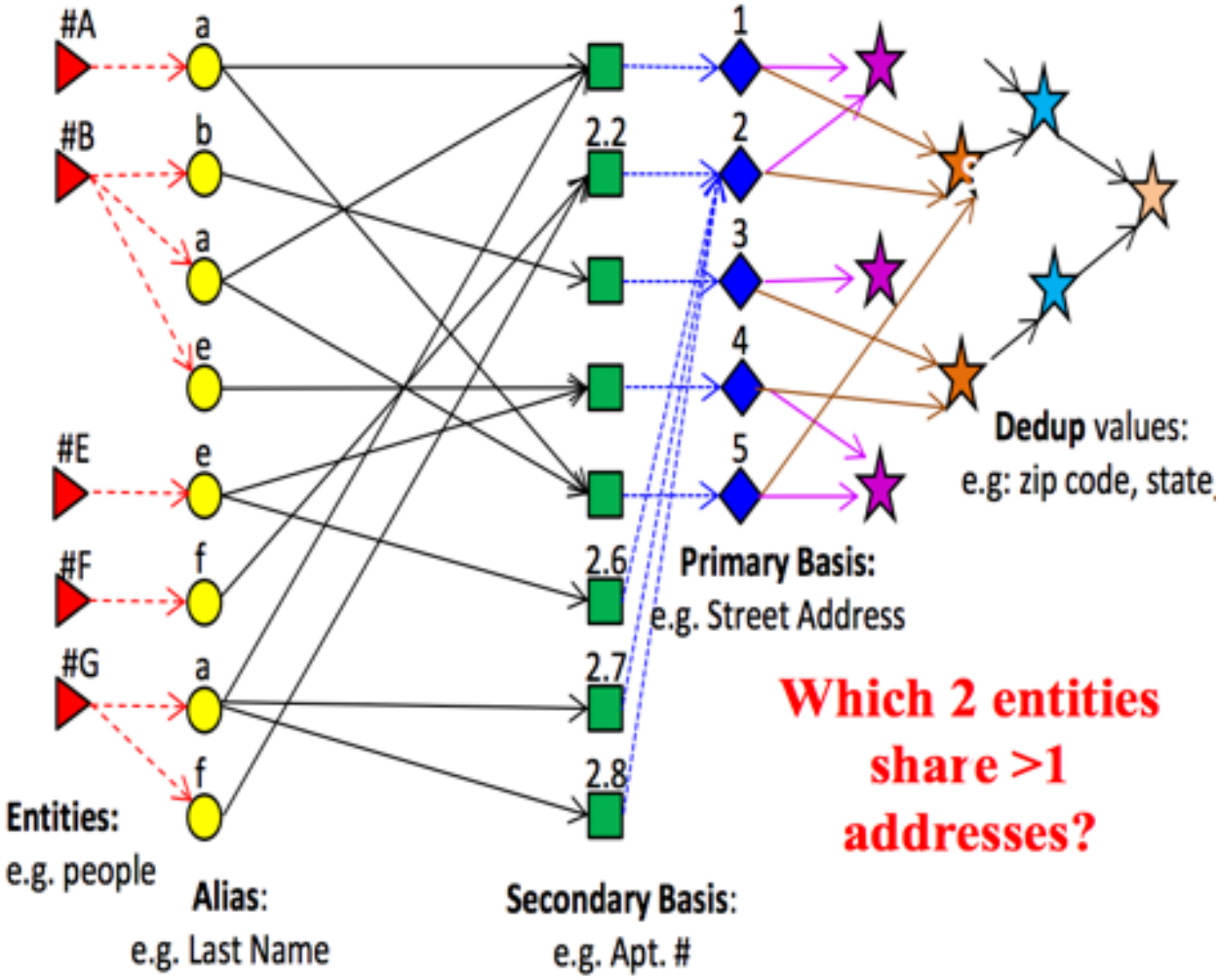

Figure 1.1: Example in which Jaccard Computations a relevant

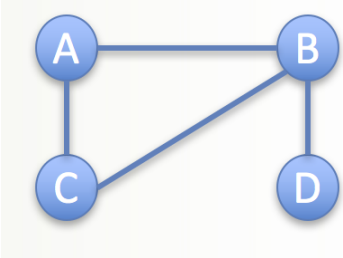

Figure 1.2: Small graph with simple Jaccard computations

Bibliography

[1] Brian Ball, Brian Karrer, and Mark EJ Newman. Efficient and principled method for detecting communities in networks. *Physical Review E*, 84(3):036103, 2011.

[2] Jacob Bank and Benjamin Cole. Calculating the jaccard similarity coefficient with map reduce for entity pairs in wikipedia. *Wikipedia Similarity Team*, pages 1–18, 2008.

[3] Paul Jaccard. The distribution of the flora in the alpine zone. 1. *New phytologist*, 11(2):37–50, 1912.

[4] Peter M Kogge. Jaccard coefficients as a potential graph benchmark. In *Parallel and Distributed Processing Symposium Workshops, 2016 IEEE International*, pages 921–928. IEEE, 2016.

[5] Chayant Tantipathananandh, Tanya Berger-Wolf, and David Kempe. A framework for community identification in dynamic social networks. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '07, pages 717–726, New York, NY, USA, 2007. ACM.