Parallel Jaccard and Related Graph Clustering Techniques

Alexandre Fender, Nahid Emad, Serge Petiton, Joe Eaton, Maxim Naumov

Parallel Jaccard and Related Graph Clustering Techniques

Alexandre Fender, Nahid Emad, Serge Petiton, Joe Eaton, Maxim Naumov

Parallel Jaccard and Related Graph Clustering Techniques

Alexandre Fender
Nvidia Corp., Maison de la Simulation
LI-PaRAD - University of Paris-Saclay
afender@nvidia.com

Nahid Emad
Maison de la Simulation
LI-PaRAD - University of Paris-Saclay
nahid.emad@uvsq.fr

Serge Petiton
Maison de la Simulation
University of Lille I, Sci. & Tech.
serge.petiton@univ-lille1.fr

Joe Eaton
Nvidia Corporation
featon@nvidia.com

Maxim Naumov
Nvidia Corporation
mnaumov@nvidia.com

ABSTRACT

In this paper we propose to generalize Jaccard and related measures, often used as similarity coefficients between two sets. We define Jaccard, Dice-Sorensen and Tversky edge weights on a graph and generalize them to account for vertex weights. We develop an efficient parallel algorithm for computing Jaccard edge and PageRank vertex weights. We highlight that the weights computation can obtain more than 10× speedup on the GPU versus CPU on large realistic data sets. Also, we show that finding a minimum balanced cut for modified weights can be related to minimizing the sum of ratios of the intersection and union of nodes on the boundary of clusters. Finally, we show that the novel weights can improve the quality of the graph clustering by about 15% and 80% for multi-level and spectral graph partitioning and clustering schemes, respectively.

CCS CONCEPTS

• **Mathematics of computing** → **Spectra of graphs**; **Graph algorithms**; *Graph theory*;

KEYWORDS

Jaccard, Tversky, Laplacian, spectral, clustering, community detection, graphs, networks, PageRank, parallel, scalable, CUDA, GPU

ACM Reference Format:
Alexandre Fender, Nahid Emad, Serge Petiton, Joe Eaton, and Maxim Naumov. 2017. Parallel Jaccard and Related Graph Clustering Techniques. In *Proceedings of ScalA17: 8th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA17).* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3148226.3148231

1 INTRODUCTION

Many processes in physical, biological and information systems are represented as graphs. In a variety of applications we would like to find a relationship between different nodes in a graph and partition it into multiple clusters. For example, graph matching techniques can be used to build an algebraic multigrid hierarchy and graph clustering can be used to identify communities in social networks.

In this paper we start by reviewing the Jaccard, Dice-Sorensen and Tversky coefficients of similarity between sets [7, 11, 27, 29]. Then, we show how to define graph edge weights based on these measures [26]. Further, we generalize them to be able to take advantage of the vertex weights and show how to compute these using the PageRank algorithm [24]. These modified weights can help to naturally express the graph clustering information. For instance, the graph representing the Amazon book co-purchasing data set [2, 19, 23] with original weights is shown on Fig. 1, while the effect of using modified weights is illustrated on Fig. 2, where thicker connections and larger circles indicate larger Jaccard and PageRank weights, respectively. The graph has two apparently distinct clusters, which are easier to visually identify with Jaccard weights. We will show that they are also algorithmically easier to compute.

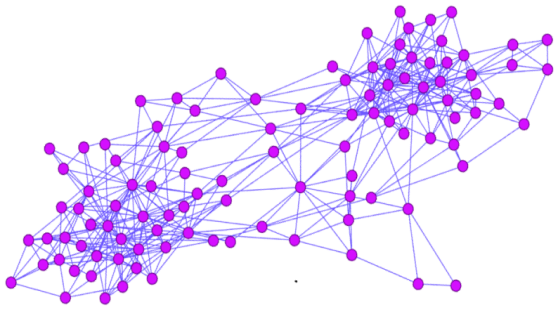

Figure 1: Amazon book co-purchasing original graph

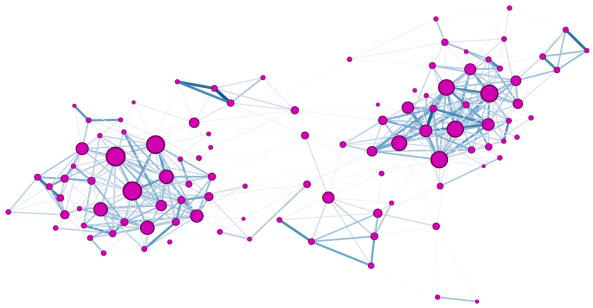

Figure 2: Amazon book co-purchasing graph with Jaccard

ScalA17, November 12–17, 2017, Denver, CO, USA

https://doi.org/10.1145/3148226.3148231

We develop an efficient parallel algorithm for computing Jaccard edge and PageRank vertex weights. We highlight that the Jaccard weights computation can obtain more than 10× speedup on the GPU versus CPU. Also, we show that the modified weights, when combined with multi-level partitioning [15, 16] and spectral clustering schemes [21, 22], can improve the quality of the minimum balanced cut obtained by these schemes by about 15% and 80%, respectively. Finally, we relate the Jaccard weights to the intersection and union of nodes on the boundary of clusters.

In Sections 2 and 3, we define Jaccard and related measures as edge weights. We show how to compute them in parallel in Section 4. In Section 5, we propose to account for vertex weights, which can be computed by PageRank. In Section 6, we show that the combination of these novel weights can improve the spectral clustering of large networks. Finally, we present the experimental results in Section 7.

2 JACCARD AND RELATED COEFFICIENTS

The Jaccard coefficient is often used as a measure of similarity between sets S_1 and S_2 [11, 20]. It is defined as

$$\mathcal{J}(S_1,S_2) = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|} \tag{1}$$

where $|.|$ denotes the cardinality of a set. Notice that $J(S_1,S_2) \in [0,1]$, with minimum 0 and maximum 1 achieved when the sets are disjoint $S_1 \cap S_2 = \{\emptyset\}$ and the same $S_1 \equiv S_2$, respectively. It is closely related to the Tanimoto coefficient for bit sequences [25, 28].

Also, Jaccard coefficient is related to the Dice-Sorensen coefficient [7, 27] often used in ecology and defined as

$$\frac{1}{2}\mathcal{D}(S_1,S_2) = \frac{|S_1 \cap S_2|}{|S_1| + |S_2|} = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2| + |S_1 \cap S_2|} \tag{2}$$

and Tversky index [29] used in psychology and defined as

$$\mathcal{T}_{\alpha,\beta}(S_1,S_2) = \frac{|S_1 \cap S_2|}{\alpha|S_1 - S_2| + \beta|S_2 - S_1| + |S_1 \cap S_2|} \tag{3}$$

where $S_1 - S_2$ is a relative complement of set S_2 in S_1 and scalars $\alpha, \beta \geq 0$. Notice that we may write $\mathcal{T}_{\frac{1}{2},\frac{1}{2}}(S_1,S_2) = \mathcal{D}(S_1,S_2)$ and $\mathcal{T}_{1,1}(S_1,S_2) = \mathcal{J}(S_1,S_2)$.

3 JACCARD AND RELATED EDGE WEIGHTS

Let a graph $G = (V,E)$ be defined by its vertex V and edge E sets. The vertex set $V = \{1,...,n\}$ represents n nodes and edge set $E = \{(i_1,j_1),...,(i_m,j_m)\}$ represents m edges. Also, we associate a nonnegative vertex $v_i \geq 0$ and edge $w_{ij} \geq 0$ weights with every node $i \in V$ and edge $(i,j) \in E$ in a graph, respectively.

Let the adjacency matrix $A = [a_{ij}]$ corresponding to a graph $G = (V,E)$ be defined through its elements

$$a_{ij} = \begin{cases} w_{ij} & \in E \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

We will assume that the graph is undirected, with $w_{ij} \equiv w_{ji}$, and therefore A is a symmetric matrix.

Let us define a neighbourhood of a node i as the set of nodes immediately adjacent to i, so that

$$\mathcal{N}(i) = \{j \mid (i,j) \in E\} \tag{5}$$

For example, for the unweighted graph shown on Fig. 1 the neighbourhood $\mathcal{N}(3) = \{2,4,5\}$.

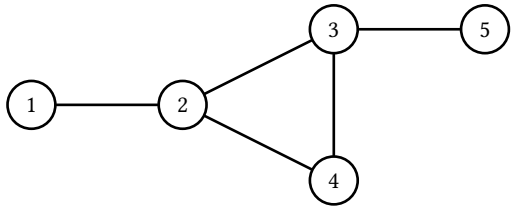

Fig. 3: An example graph $G = (V,E)$

In order to setup Jaccard-based clustering, we propose to define the following intermediate edge weights in the graph. The intersection weight

$$w_{ij}^{(I)} = \sum_{k \in \mathcal{N}(i) \cap \mathcal{N}(j)} v_k \tag{6}$$

the sum weight

$$w_{ij}^{(S)} = \sum_{k \in \mathcal{N}(i)} v_k + \sum_{l \in \mathcal{N}(j)} v_l \tag{7}$$

the complement weight

$$w_{ij}^{(C)} = \sum_{k \in \mathcal{N}(i)} v_k - w_{ij}^{(I)} \tag{8}$$

and the union weight

$$w_{ij}^{(U)} = w_{ji}^{(S)} - w_{ij}^{(I)} \tag{9}$$

$$= w_{ij}^{(C)} + w_{ji}^{(C)} + w_{ji}^{(I)} \tag{10}$$

For instance, for the special case of unweighted graphs, with $v_i = 1$ and $w_{ij} = 1$, we can omit the vertex weight and write these weights as

$$w_{ij}^{(I)} = |\mathcal{N}(i) \cap \mathcal{N}(j)| \tag{11}$$

$$w_{ij}^{(S)} = |\mathcal{N}(i)| + |\mathcal{N}(j)| \tag{12}$$

$$w_{ij}^{(C)} = |\mathcal{N}(i)| - |\mathcal{N}(i) \cap \mathcal{N}(j)| = |\mathcal{N}(i) - \mathcal{N}(j)| \tag{13}$$

$$\begin{aligned} w_{ij}^{(U)} &= |\mathcal{N}(i)| + |\mathcal{N}(j)| - |\mathcal{N}(i) \cap \mathcal{N}(j)| \\ &= |\mathcal{N}(i) - \mathcal{N}(j)| + |\mathcal{N}(j) - \mathcal{N}(i)| + |\mathcal{N}(i) \cap \mathcal{N}(j)| \\ &= |\mathcal{N}(i) \cup \mathcal{N}(j)| \end{aligned} \tag{14}$$

Then, we can define Jaccard weight as

$$w_{ij}^{(\mathcal{J})} = w_{ij}^{(I)} / w_{ij}^{(U)} \tag{15}$$

Dice-Sorensen weight as

$$w_{ij}^{(\mathcal{D})} = w_{ij}^{(I)} / w_{ij}^{(S)} \tag{16}$$

Tversky weight as

$$w_{ij}^{(\mathcal{T})} = w_{ij}^{(I)} / (\alpha w_{ij}^{(C)} + \beta w_{ij}^{(C)} + w_{ij}^{(I)}) \tag{17}$$

For example, for the unweighted graph on Fig. 1 the original adjacency matrix can be written as

$$A^{(O)} = \begin{bmatrix} & 1 & & & \\ 1 & & 1 & 1 & \\ & 1 & & 1 & 1 \\ & 1 & 1 & & \\ & & 1 & & \end{bmatrix} \tag{18}$$

while based on Jaccard weights it can be written as

$$A^{(\mathcal{J})} = \begin{bmatrix} & 0 & & & \\ 0 & & 1/5 & 1/4 & \\ & 1/5 & & 1/4 & 0 \\ & 1/4 & 1/4 & & \\ & & 0 & & \end{bmatrix} \tag{19}$$

Notice that if we simply use the Jaccard weights the new graph might become disconnected. For instance, in our example the intersections of neighborhoods of $\mathcal{N}(1) \cap \mathcal{N}(2)$ and $\mathcal{N}(3) \cap \mathcal{N}(5)$ are empty $\{\emptyset\}$ and consequently nodes 1 and 5 are disconnected from the rest of the graph. While it is possible to work with disconnected graphs, in many scenarios such change in the graph properties is undesirable.

Also, notice that the original weights $w_{ij}^{(O)}$ have arbitrary magnitude, while Jaccard weight $w_{ij}^{(\mathcal{J})} \in [0,1]$. Therefore, adding these weights might result in non uniform effects on different parts of the graph (with small and large original weights) and make these effects scaling dependent.

In order to address these issues we propose to combine Jaccard and original weights in the following fashion

$$w_{ij}^{(*)} = w_{ij}^{(O)} \left(1 + w_{ij}^{(\mathcal{J})}\right) \tag{20}$$

Notice that in this formula the Jaccard weight is used to strengthen edges with large overlapping neighbourhoods.

In the next section we will show how we can efficiently compute Jaccard weights in parallel on the GPU. The Dice-Sorensen and Tversky weights can be computed similarly.

4 PARALLEL ALGORITHM

The graph and its adjacency matrix can be stored in arbitrary data structures. Let us assume that we use the standard CSR format, which simply concatenates all non-zero entries of the matrix in row-major order and records the starting position for the entries of each row. For example, the adjacency matrix (18) can be represented using three arrays

$$\begin{aligned} \texttt{Ap} &= [0,1,4,7,9,10] \\ \texttt{Ac} &= [1;0,2,3;1,3,4;1,2;2] \\ \texttt{Av} &= [1;1,1,1;1,1,1;1,1;1] \end{aligned} \tag{21}$$

where ";" denotes the start of elements in a new row.

Then, the intersection weights in (6) can be computed in parallel using Alg. 1, where the binary search is done according to Alg. 2. Notice that in Alg. 1 we perform intersections on sets corresponding to neighbourhoods of nodes i and j. These sets have potentially different number of elements $N_i = e_i - s_i$ and $N_j = e_j - s_j$. In order to obtain better computational complexity we would like to perform the binary search on the largest set. In the above pseudo-code we have implicitly assumed that the smallest set corresponds to node i. In practice, we can always test the set size by looking at whether $N_i < N_j$ and flip-flop indices i and j if needed.

Algorithm 1 Intersection Weights

1: Let n and m be the # of nodes and edges in the graph.
2: Let Ap, Ac and Av represent its adjacency matrix $A^{(O)}$.
3: Initialize all weights $w_{ij}^{(I)}$ to 0.
4: **for** $i = 1,...,n$ **do in parallel**
5: Set $s_i = \texttt{Ap}[i]$ and $e_i = \texttt{Ap}[i+1]$
6: **for** $k = s_i,...,e_i$ **do in parallel**
7: Set $j = \texttt{Ac}[k]$
8: Set $s_j = \texttt{Ap}[j]$ and $e_j = \texttt{Ap}[j+1]$
9: **for** $z = s_i,...,e_i$ **do in parallel** ▷ Intersection
10: $l = \texttt{binary\_search}(\texttt{Ac}[z], s_j, e_j - 1, \texttt{Ac})$
11: **if** $l \geq 0$ **then** ▷ Found element
12: AtomicAdd($w_{ij}^{(I)}$, $\texttt{Av}[l]$) ▷ Atomic Update
13: **end if**
14: **end for**
15: **end for**
16: **end for**

Algorithm 2 binary_search($i, l, r, \texttt{x}$)

1: Let i be the element we would like to find.
2: Let left l and right r be the end points of a set.
3: Let sorted set elements be located in array x.
4: **while** $l \leq r$ **do**
5: $m = (l + r)/2$ ▷ Find middle of the set
6: $j = \texttt{x}[m]$
7: **if** $j > i$ **then**
8: Set $r = m - 1$ ▷ Move right end point
9: **else if** $j < i$ **then**
10: Set $l = m + 1$ ▷ Move left end point
11: **else**
12: Return m ▷ Done, element found
13: **end if**
14: **end while**
15: Return -1 ▷ Done, element not found

Algorithm 3 Sum Weights

1: Let n and m be the # of nodes and edges in the graph.
2: Let Ap, Ac and Av represent its adjacency matrix $A^{(O)}$.
3: **for** $i = 1,...,n$ **do in parallel**
4: Set $s_i = \texttt{Ap}[i]$ and $e_i = \texttt{Ap}[i+1]$
5: Set $N_i = \texttt{sum}(s_i, e_i, \texttt{Av})$
6: **for** $k = s_i,...,e_i$ **do in parallel**
7: Set $j = \texttt{Ac}[k]$
8: Set $s_j = \texttt{Ap}[j]$ and $e_j = \texttt{Ap}[j+1]$
9: Set $N_j = \texttt{sum}(s_j, e_j, \texttt{Av})$
10: Set $w_{ij}^{(S)} = N_i + N_j$
11: **end for**
12: **end for**

Then, the sum weights in (7) can be computed using the parallel Alg. 3, where the sum operation on line 6 and 10 can be written for general graphs as

$$\texttt{sum}(s, e, \texttt{Av}) = \texttt{Av}[s] + \texttt{Av}[s+1] + \ldots + \texttt{Av}[e] \tag{22}$$

and for unweighted graphs as

$$\texttt{sum}(s, e, \texttt{Av}) = 1 + \ldots + 1 = e - s \tag{23}$$

Finally, the union and the corresponding Jaccard weights can be obtained using (9) and (15), respectively.

Let us assume a standard theoretical PRAM (CREW) model for analysis [12]. Notice that the sequential complexity of Alg. 1 is

$$\sum_{i=1}^{n} \sum_{j=1}^{N_i} N_i \log N_j \tag{24}$$

and, assuming we can store intermediate results, of Alg. 3 is

$$\sum_{i=1}^{n} \log N_i + m \tag{25}$$

where $N_i = |\mathcal{N}(i)|$ is the number of elements in each row. However, the complexity of both algorithms is

$$\max_i \log N_i \tag{26}$$

using $n \max_i N_i^2$ and m processors, respectively, which illustrates the degree of available parallelism.

Also, notice that Alg. 1 can be interpreted as sparse matrix-matrix multiplication AA^T, where only elements that are already present in the sparsity pattern of A are left, in other words, we do not allow any fill-in in the result.

The performance of the parallel implementation in CUDA of the algorithm for computing Jaccard weights on the GPU is shown in Fig. 4. Notice that we compare it with sequential (single thread) as well as openMP (12 threads) implementation of the algorithm on the CPU, with hardware details specified in the numerical experiments section. We often obtain a speedup above 10× on the data sets from Tab. 3. The details of the experiments are provided in Tab. 1.

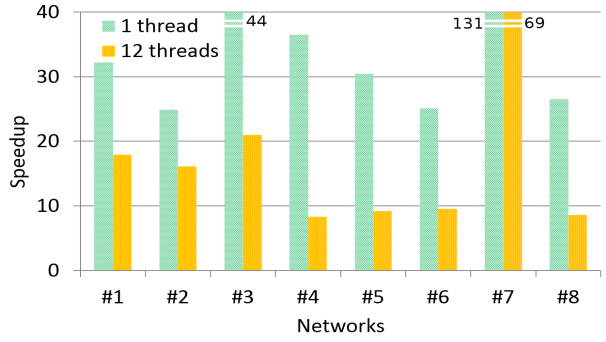

Figure 4: Speedup when computing Jaccard Weights

#	CPU (1 thread)	CPU (12 threads)	GPU
1.	155	86	5
2.	193	125	8
3.	172	82	4
4.	340	77	9
5.	9401	2847	308
6.	13514	5130	538
7.	65582	34646	502
8.	337870	109541	12751

Table 1: Time(ms) needed to compute Jaccard weights

5 PAGERANK AND VERTEX WEIGHTS

The PageRank algorithm measures the relative importance of a vertex compared to other nodes in the graph. Therefore, it is natural to incorporate the vertex weights v_k to measure the importance of neighbourhoods, as shown on Fig. 2.

In this section, we will show how to compute vertex weights based on the PageRank algorithm [24], which has been a key feature of search and recommendation engines for years [5, 18]. Recall that the PageRank algorithm is based on a discrete Markov process (Markov chain), a mathematical system that undergoes transitions from one state to another and where the future states depend only on the current state. It can be represented as a transition matrix $P = [p_{ij}]$, where the off-diagonal elements correspond to the probabilities of transitioning between i-th and j-th states. Therefore, the transition matrix allows us to move from the vector of current $\mathbf{s}_k$ to the vector of next states using $\mathbf{s}_{k+1}^T = \mathbf{s}_k^T P$.

In PageRank the transition matrix

$$P = H + \mathbf{b}\mathbf{w}^T \tag{27}$$

where the matrix $H = [h_{ij}]$ represents the probabilities of following links between connected pages, so that

$$h_{ij} = \begin{cases} 1/\mathrm{d}(i) & \text{if there is an outgoing link from } i \text{ to } j \\ 0 & \text{otherwise,} \end{cases} \tag{28}$$

the bookmark vector $\mathbf{b} = [b_i]$ marks the dangling pages (also called leaf pages) without outgoing links, so that

$$b_i = \begin{cases} 1 & \text{if there are no outgoing links from } i \\ 0 & \text{otherwise,} \end{cases} \tag{29}$$

the probability vector $\mathbf{w} = [w_i]$ satisfies $\mathbf{w}^T\mathbf{e} = 1$ and gives the probability of transitioning to page i from a dangling page, $\mathrm{d}(i)$ denotes the number of outgoing links from i-th page and vector $\mathbf{e}^T = (1, \ldots, 1)$.

The matrix P is often further modified to allow for personalization, so that we finally obtain the Google matrix

$$G = \alpha P + (1 - \alpha)\mathbf{e}\mathbf{p}^T \tag{30}$$

where constant $\alpha \in [0, 1]$ and personalization vector $\mathbf{p} = [p_i]$ satisfies $\mathbf{p}^T\mathbf{e} = 1$ and gives the probability of transitioning to page i at any moment.

Notice that matrix G is row-stochastic. Therefore, it has non-negative elements and satisfies $G\mathbf{e} = \mathbf{e}$, with a comprehensive analysis of its properties done in [3–6, 9]. The rank v_i of page i is

given by the rank vector $\mathbf{v} = [v_i]$ that satisfies

$$\mathbf{v}^T G = \mathbf{v}^T \tag{31}$$

and corresponds to the left eigenvector associated with the largest eigenvalue $\lambda = 1$ of nonsymmetric matrix G. This vector is sometimes referred to as the stationary distribution (or equilibrium) vector of the Markov chain.

The PageRank vector $\mathbf{v}$ is often computed using Power method with G^T [1, 13, 14], as shown in Alg. 4. Notice that in this algorithm we never form the Google matrix explicitly, but rather replicate its action on a vector using matrix H^T and rank-one updates.

Algorithm 4 PageRank

1: Let P be the transition matrix.
2: Let $\mathbf{b}$ be the bookmark vector.
3: Let $\mathbf{w}$ be the probability vectors (often $\mathbf{w} = \frac{1}{n}\mathbf{e}$).
4: Let $\mathbf{p}$ be the personalization vector (may be $\mathbf{p} = \frac{1}{n}\mathbf{e}$).
5: Let $\mathbf{v}_0$ be the initial guess.
6: **for** i=0,1,2,...,convergence **do** ▷ Power method
7: Compute $\beta_i = \alpha \mathbf{b}^T \mathbf{v}_i$
8: Compute $\gamma_i = (1-\alpha)\mathbf{e}^T \mathbf{v}_i$
9: Compute $\mathbf{z}_{i+1} = \alpha H^T \mathbf{v}_i + \beta_i \mathbf{w} + \gamma_i \mathbf{p}$ ▷ $\mathbf{z}_{i+1} = G\mathbf{v}_i$
10: Compute $\mathbf{v}_{i+1} = \mathbf{z}_{i+1}/||\mathbf{z}_{i+1}||_2$
11: **end for**

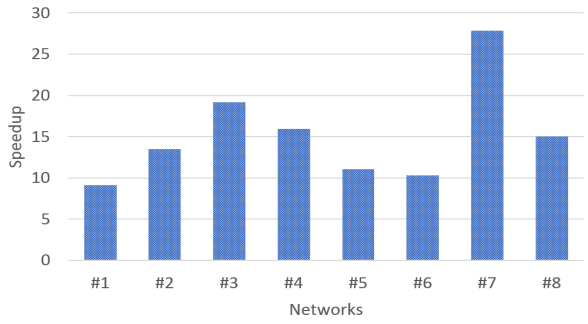

Figure 5: Speedup when computing PageRank

	CPU (12 threads)			GPU		
#	time	it.	$\|\|\mathbf{r}_k\|\|_2/\|\|\mathbf{r}_0\|\|_2$	time	it.	$\|\|\mathbf{r}_k\|\|_2/\|\|\mathbf{r}_0\|\|_2$
1.	61	17	8.3e-06	7	17	8.3e-06
2.	393	76	9.0e-06	29	76	9.0e-06
3.	461	51	9.9e-06	24	51	9.9e-06
4.	470	57	8.9e-06	30	57	8.9e-06
5.	1721	51	2.4e-06	156	51	2.4e-06
6.	1615	53	2.7e-06	157	53	2.7e-06
7.	5228	74	6.6e-06	188	74	6.6e-06
8.	6650	49	1.1e-06	442	49	1.1e-06

Table 2: Time(ms) needed to compute PageRank

The speedup and performance of the parallel CUDA implementation of the algorithm for computing PageRank on the GPU is shown in Fig. 5 and Tab. 2, respectively. In these experiments we compare the parallel CUDA implementation with the Intel MKL implementation on the CPU, using all available threads. Both device and host implementations take advantage of the sparse matrix-vector multiplication building block implemented in the CUSPARSE library and Intel MKL, respectively. Further hardware details are specified in the numerical experiments section.

Notice that we often obtain a speedup above 10× on the realistic data sets from Tab. 3. In both implementations we always use the same initial guess and stopping criteria, based on the maximum number of iterations being < 100 and relative residual $< 10^{-5}$. Therefore, the obtained iteration count (it.) and relative resiudual ($||\mathbf{r}_k||_2/||\mathbf{r}_0||_2$) are equals across CPU and GPU, as shown in Tab. 2.

6 GRAPH CLUSTERING

In graph clustering a vertex set V is often partitioned into p disjoint sets S_k, such that $V = S_1 \cup S_2 ... \cup S_p$ and $S_i \cap S_j = \{\emptyset\}$ for $i \neq j$ [16, 21]. Notice that instead of the original graph $G = (V, E)$ we can use the modified graph $G^{(*)} = (V^{(*)}, E^{(*)})$, with vertex $v_i^{(*)}$ and edge $w_{ij}^{(*)}$ weights computed based on PageRank and Jaccard or related schemes discussed in earlier sections.

6.1 Jaccard Spectral Clustering

Notice that we can define the Laplacian as

$$L^{(*)} = D^{(*)} - A^{(*)} \tag{32}$$

where $D^{(*)} = \text{diag}(A^{(*)}\mathbf{e})$ is the diagonal matrix.

Then, we would minimize the normalized balanced cut

$$\begin{aligned} \tilde{\eta}(S_1, ..., S_p) &= \min_{S_1,...,S_p} \sum_{k=1}^{p} \frac{\text{vol}(\partial(S_k))}{\text{vol}(S_k)} \\ &= \min_{U^T D^{(*)} U = I} \text{Tr}(U^T L^{(*)} U) \end{aligned} \tag{33}$$

where Tr(.) is the trace of a matrix, boundary edges

$$\partial S = \{(i,j) \mid i \in S \wedge j \notin S\} \tag{34}$$

and volume

$$\text{vol}(S) = \sum_{i \in S} w_{ij}^{(*)} \tag{35}$$

$$\text{vol}(\partial S) = \sum_{(i,j) \in \partial(S)} w_{ij}^{(*)} = \sum_{(i,j) \in \partial(S)} w_{ij}^{(O)} \left(1 + \frac{w_{ij}^{(I)}}{w_{ij}^{(U)}}\right)$$

by finding its smallest eigenpairs and transforming them into assignment of nodes into clusters [22]. Notice that Jaccard weights correspond to the last term in the above formula, and are related to the sum of ratios of the intersection and union of nodes on the boundary of clusters.

Also, we point out that we choose to use normalized cut spectral formulation because it is invariant under scaling. Notice that based on (20) the edge weight $w_{ij}^{(*)} \geq w_{ij}^{(O)}$. Therefore, to avoid artificially higher/lower scores when comparing quality, we need to use a metric that is invariant under edge weight scaling. To illustrate this point suppose that for a given assignment of nodes into clusters the edge weights are multiplied by 2. The clustering has not changed and normalized score stays the same, while ratio cut score increases and therefore is not an appropriate metric for our comparisons.

Finally, we show the assignment of nodes into clusters based on Jaccard and PageRank weights for a realistic Amazon book co-purchasing data set [2, 19] on Fig. 6. Notice that use of Jaccard edge and PageRank vertex weights lead to visually intuitive discovery of clusters.

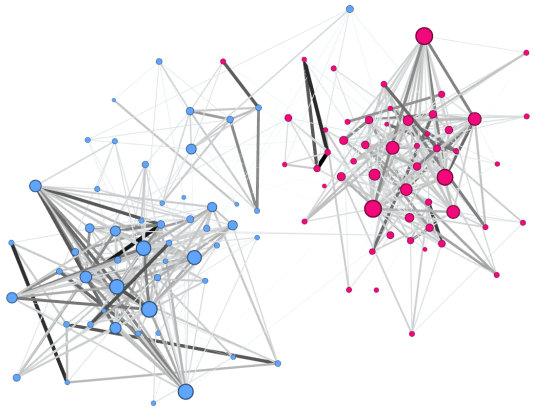

Figure 6: Amazon book co-purchasing graph clustering with PageRank vertex and Jaccard edge weights

6.2 Tversky Spectral Clustering

So far we have essentially defined Tversky clustering for a special cases $\mathcal{T}_{1,1}(S_1, S_2) = \mathcal{J}(S_1, S_2)$. We note that further generalization is possible by introducing

$$A^{(\mathcal{T})} = A^{(I)} \oslash (\alpha L^{(C)} + \beta U^{(C)} + A^{(I)}) \tag{36}$$

where $L^{(C)}$ is lower and $U^{(C)}$ is upper triangular part of the matrix $A^{(C)} = [a_{ij}^{(C)}]$ with elements $a_{ij}^{(C)} = w_{ij}^{(C)}$ corresponding to complement weights, $A^{(I)}$ is a matrix with intersection weights and $\oslash$ operation corresponds to Hadamard (entry-wise) division.

However, we point out that we can only compute Tversky clustering analogously to Jaccard clustering when the scaling parameters $\alpha = \beta$. Notice that if $\alpha \neq \beta$ then the adjacency matrix $A^{(\mathcal{T})}$ and the corresponding Laplacian matrix $L^{(\mathcal{T})}$ are not symmetric. Therefore, the Courant-Fischer theorem [10] is no longer applicable and the minimum of the objective function $\tilde{\eta}$ in (33) no longer corresponds to the smallest eigenvalues of the Laplacian.

6.3 Profiling

Notice that the computation of Jaccard and PageRank weights is often a small fraction $< 20\%$ of the total computation time, see Fig. 7. In fact the profiling of the complete spectral clustering pipeline on the GPU shows that most time $> 80\%$ is actually spent in the eigenvalue solver. In our code we rely on the LOBPCG method [17], which has been shown to be effective for Laplacian matrices [22].

The second most time consuming operation is the computation of PageRank vertex weights. Notice that PageRank also solves an eigenvalue problem, but it finds the largest eigenpairs of the Google matrix and therefore is significantly faster than LOBPCG, which looks for the smallest eigenpairs. We point out that the PageRank computation is optional and can be skipped if needed.

Finally, the computation of Jaccard edge weights is only the third most time consuming operation. Notice that our implementation supports weighted vertices by design, so that there is no extra cost for using the vertex weight resulting from PageRank or any other algorithm.

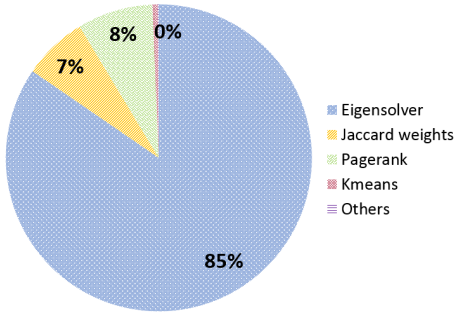

Figure 7: Profile of spectral clustering with PageRank vertex and Jaccard edge weights

7 NUMERICAL EXPERIMENTS

Let us now study the performance and quality of the clustering obtained using Jaccard weights on a sample of graphs from the DIMACS10, LAW and SNAP graph collection [30], shown in Tab. 3.

#	Matrix	$n = \|V\|$	$m = \|E\|$	Application
0.	smallword	100,000	999,996	Artificial
1.	preferentialA...	100,000	499,985	Artificial
2.	caidaRouterLevel	192,244	609,066	Internet
3.	coAuthorsDBLP	299,067	977,676	Coauthorship
4.	citationCiteseer	268,495	1,156,647	Citation
5.	coPapersDBLP	540,486	15,245,729	Affiliation
6.	coPapersCiteseer	434,102	16,036,720	Affiliation
7.	as-Skitter	1,696,415	22,190,596	Internet
8.	hollywood-2009	1,139,905	113,891,327	Coauthorship

Table 3: General information on networks

In our spectral experiments we use the nvGRAPH 9.0 library and let the stopping criteria for the LOBPCG eigenvalue solver be based on the norm of the residual corresponding to the smallest eigenpair $||\mathbf{r}_1||_2 = ||L\mathbf{u}_1 - \lambda_1 \mathbf{u}_1||_2 \leq 10^{-4}$ and maximum of 40 iterations, while for the k-means algorithm we let it be based on the scaled error difference $|\epsilon_l - \epsilon_{l-1}|/n < 10^{-2}$ between consecutive steps and a maximum of 16 iterations [22].

In our multi-level experiments we use the METIS 5.1.0 library and choose the default parameters for it [15]. Also, we plot the quality improvement as a percentage of the original score based on $100\% \times (\tilde{\eta}^{(modified)} - \tilde{\eta}^{(original)})/\tilde{\eta}^{(original)}$.

All experiments are performed on a workstation with Ubuntu 14.04 operating system, gcc 4.8.4 compiler, Intel MKL 11.0.4, CUDA Toolkit 9.0 software and Intel Core i7-3930K CPU 3.2 GHz and NVIDIA Titan Xp GPU hardware. The performance of algorithms was always measured across multiple runs to ensure consistency.

7.1 Multi-level Schemes (CPU)

Let us first look at the impact of using Jaccard weights in popular multi-level graph partitioning schemes, that are implemented in software packages such as METIS [15, 16]. These schemes agglomerate nodes of the graph in order to create a hierarchy, where the fine level represents the original graph and the coarse level represents its reduced form. The partitioning is performed on the coarse level and results are propagated back to the fine level.

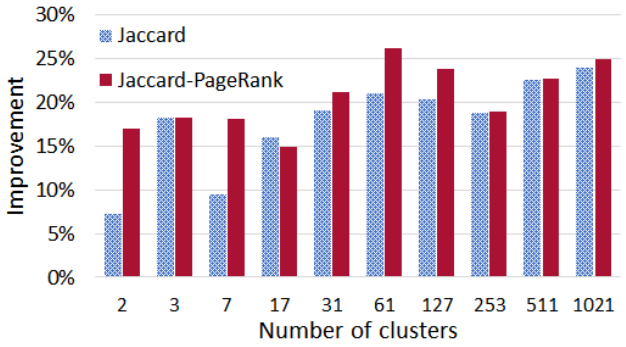

Figure 8: Improvement in the quality of partitioning obtained by METIS, with Jaccard and Jaccard-PageRank for `coPapersCitseer` graph

In our experiments we compute the modified vertex $v_i^{(*)}$ and edge $w_{ij}^{(*)}$ weights ahead of time and supply them to METIS as one of the parameters. We measure the quality of the partitioning using the cost function $\tilde{\eta}$ in (33) and plot it over different number of cluster for the same `coPaperCitseer` network. The obtained improvement in quality when using Jaccard and Jaccard-PageRank versus original weights is shown in Fig. 8.

Notice that using Jaccard and Jaccard-PageRank weights helped improve METIS partitioning by 18% and 21% on average, respectively. This is a moderate but steady amelioration, taking values within a range of 7% to 25% for Jaccard and 15% to 26% with additional PageRank information.

7.2 Spectral Schemes (GPU)

Let us now look at using Jaccard weights in spectral schemes, that are implemented in the nvGRAPH library. These schemes often use the eigenpairs of the Laplacian matrix and subsequent post-processing by k-means to find the assignment of nodes into clusters.

In our experiments we measure the quality of clustering using the cost function $\tilde{\eta}$ in (33) and plot it over different number of cluster for the same `coPapersDBLP` network. The obtained improvement in quality when using Jaccard and Jaccard-PageRank versus original weights is shown in Fig. 9. Notice that in spectral clustering it is possible to compute a smaller number of eigenpairs than clusters [8] and in these experiments we have varied them synchronously until 32, after which we have fixed the number of eigenpairs pairs and increased the number of clusters only. The limit of 32 was chosen somewhat arbitrarily based on tradeoffs between computation time, memory usage and quality.

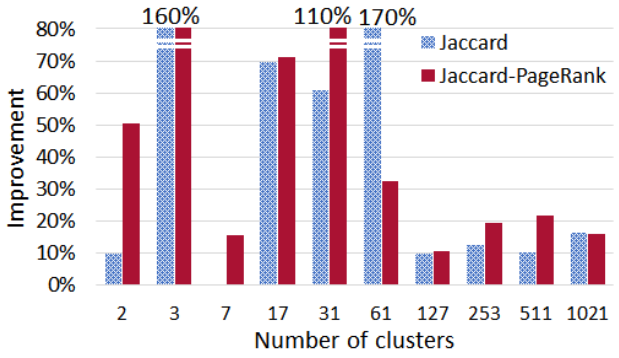

Figure 9: Improvement in the quality of partitioning obtained by nvGRAPH, with Jaccard and Jaccard-PageRank for `coPaperDBLP` graph

Notice that using Jaccard and Jaccard-PageRank weights we often obtain a significant improvement of up to 160% in the quality of clustering up to about 32 clusters. Then, the improvement tails off up to 20% for a larger number of clusters. This happens in part because as mentioned in previous paragraph we do not increase the number of obtained eigenpairs past 32 in the spectral clustering scheme. Therefore, in the latter regime we have essentially already traded off higher performance for lower quality.

Notice that in general using Jaccard and Jaccard-PageRank weights helped improve the spectral clustering quality by 49% and 51% on average, respectively. This is a significant but sometimes irregular amelioration, taking values within a range of −39% to 172% for Jaccard and 11% to 163% with additional PageRank information.

7.3 Quality Across Many Samples

Finally, let us compare the impact of using Jaccard and Jaccard-PageRank weights across samples listed in Tab. 3. In this section we fix the number of clusters to be 31, which is a prime number large enough to be relevant for real clustering applications. We measure quality as described in the previous two sections. The obtained improvement in quality when using Jaccard and Jaccard-PageRank versus original weights is shown in Fig. 10 and Tab. 4.

	M-L (J)	Spect (J)	M-L (J+P)	Spect (J+P)
smallworld	14.0%	9.9%	14.0%	22.9%
coAuthorsDBLP	14.3%	52.0%	15.1%	33.1%
citationCiteseer	2.1%	-9.0%	4.5%	-20.2%
coPapersDBLP	13.1%	61.0%	11.8%	113.8%
coPapersCiteseer	19.1%	237.7%	21.2%	236.5%

Table 4: Improvement in the quality of partitioning obtained by nvGRAPH (Spect) and METIS (M-L), with Jaccard (J) and Jaccard-PageRank (J+P) weights

Notice that for these graphs the Jaccard weights help to improve the multi-level and spectral clustering quality by about 10% and 70% on average, respectively. When using additional PageRank information this improvement rises to about 15% and 80% on average,

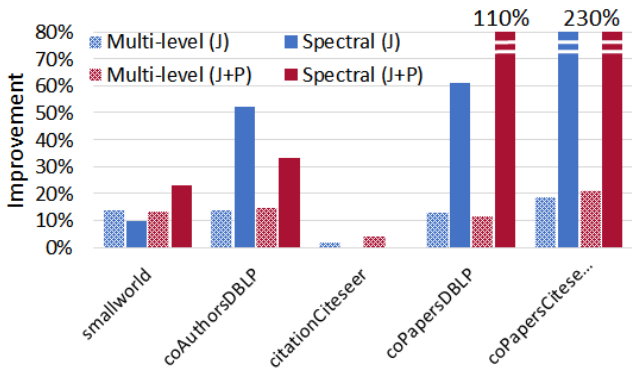

Figure 10: Improvement in the quality of partitioning obtained by nvGRAPH and METIS, with Jaccard and Jaccard-PageRank weights

respectively. However, the improvements are not always regular, and on occasion might result in lower quality clustering.

The spectral clustering has a more intense average amelioration but there is one case that does not benefit from using modified weights. This is consistent with the experiment of Fig. 9. The multi-level clustering has lower average amelioration, but all cases seem to benefit from using Jaccard and Jaccard-PageRank weights.

Finally, we note that using Jaccard or Jaccard-PageRank weights on `coPapersCiteseer` network leads to an improvement over 230% for the spectral clustering approach. In this case, the high amelioration ratio happens because the spectral clustering method struggles to find a good clustering without weights that represent the local connectivity information.

8 CONCLUSION AND FUTURE WORK

In this paper we have extended the Jaccard, Dice-Sorensen and Tversky measures to graphs. We have defined the associated edge weights and we have shown how to incorporate vertex weights into these new graph metrics.

Also, we have developed the corresponding parallel implementation of Jaccard edge and PageRank vertex weights on the GPU. The Jaccard and PageRank implementation has attained a speedup of more than 10× on GPU versus a parallel CPU code. Moreover, we have profiled the entire clustering pipeline and shown that computation of modified weights consumes no more than 20% of the total time taken by the algorithm.

Finally, in our numerical experiments we have shown that clustering and partitioning can benefit from using Jaccard and PageRank weights on real networks. In particular, we have shown that spectral clustering quality can increase by up to 3×, while we also note that the improvements are not uniform across graphs. On the other hand, for multi-level schemes, we have shown smaller but steadier improvement of about 15% on average.

In the future, we would like to explore the distributed implementation of the spectral clustering schemes. For instance, notice that the computation of Jaccard edge weights can be interpreted as matrix-matrix multiplication AA^T without fill-in, while PageRank algorithm relies on the matrix-vector multiplication kernel. It is well known that these operations are well suited for parallelization on distributed platforms, which we plan to explore next.

9 ACKNOWLEDGEMENTS

The authors would like to acknowledge Michael Garland for his useful comments and suggestions.

REFERENCES

[1] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe and H. van der Vorst, *Templates for the solution of Algebraic Eigenvalue Problems: A Practical Guide*, SIAM, Philadelphia, PA, 2000.
[2] M. Bastian, S. Heymann and M. Jacomy, *Gephi: An Open Source Software for Exploring and Manipulating Networks*, Int. AAAI Conf. Web Social Media, 2009.
[3] A. Bourchtein and L. Bourchtein, *On Some Analytical Properties of a General PageRank Algorithm*, Math. Comp. Modelling, Vol. 57, pp. 2248-2256, 2013.
[4] L. Bourchtein and A. Bourchtein, *On Perturbations of Principal Eigenvectors of Substochastic Matrices*, J. Comput. Applied Math., Vol. 295, pp. 149-158, 2016.
[5] C. Brezinski and M. Redivo-Zaglia, *The PageRank Vector: Properties, Computation, Approximation, and Acceleration*, SIAM J. Mat. Anal. Appl., Vol. 28, pp. 551-575, 2006.
[6] A Cicone and S. Serra-Capizzano, *Google PageRanking Problem: The Model and the Analysis*, J. Comput. Applied Math., Vol. 234, pp. 3140-3169, 2010.
[7] L. R. Dice, *Measures of the Amount of Ecologic Association Between Species*, Ecology, Vol. 26, pp. 297-302, 1945.
[8] A. Fender, N. Emad, S. Petiton and M. Naumov, *Parallel Modularity Clustering*, Int. Conf. Comput. Sci. (ICCS), submitted, 2017.
[9] T. Haveliwala and S. Kamvar, *The Second Eigenvalue of the Google Matrix*, Technical Report, 2003-20, Stanford University, 2003.
[10] R. A. Horn and C. R. Johnson, *Matrix Analysis*, Cambridge University Press, New York, NY, 1999.
[11] P. Jaccard, *Lois de Distribution Florale dans la Zone Alpine*, Bull. Soc. Vaud. Sci. Nat., Vol. 38, pp. 69-130, 1902.
[12] J. JaJa, *An Introduction to Parallel Algorithms*, Addison-Wesley, 1992.
[13] S. Kamvar, T. Haveliwala, C. D. Manning and G. Golub, *Extrapolation Methods for Accelerating PageRank Computations*, Proc. 12th Intern. Conf. World Wide Web, pp. 261-270, 2003.
[14] S. Kamvar, T. Haveliwala and G. Golub, *Adaptive Methods for the Computation of PageRank*, Linear Algebra Appl., Vol. 386, pp. 51-65, 2004.
[15] G. Karypis and V. Kumar, *METIS - Unstructured Graph Partitioning and Sparse Matrix Ordering System, V2.0*, 1995.
[16] G. Karypis and V. Kumar, *A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs*, SIAM J. Sci. Comput., Vol. 20, pp. 359-392, 1998.
[17] A. V. Knyazev, *Toward the Optimal Preconditioned Eigensolver: Locally Optimal Block Preconditioned Conjugate Gradient Method*, SIAM J. Sci. Comput., Vol. 23, pp. 517-541, 2001.
[18] A. N. Langville and C. D. Meyer, *Google's PageRank and Beyond: The Science of Search Engine Rankings*, Princeton University Press, Princeton, NJ, 2006.
[19] J. Leskovec, L. Adamic and B. Adamic, *The Dynamics of Viral Marketing*, ACM Trans. Web, Vol. 1, 2007.
[20] M. Levandowsky and D. Winter, *Distance Between Sets*, Nature, Vol. 234, pp. 34-35, 1971.
[21] U. von Luxburg, *A Tutorial on Spectral Clustering*, Technical Report No. TR-149, Max Planck Institute, 2007.
[22] M. Naumov and T. Moon, *Parallel Spectral Graph Partitioning*, NVIDIA Technical Report, NVR-2016-001, 2016.
[23] M. E. J. Newman, *Networks: An Introduction*, Oxford University Press, New York, NY, 2010.
[24] L. Page, S. Brin, R. Motwani, and T. Winograd, *The PageRank Citation Ranking: Bringing Order to the Web*, Technical Report, Stanford InfoLab, 1999.
[25] D. J. Rogers and T. T. Tanimoto *A Computer Program for Classifying Plants*, Science, Vol. 132, pp. 1115-1118, 1960.
[26] J. Santisteban and J. L. T. Carcamo, *Unilateral Jaccard Similarity Coefficient*, Proc. SIGIR Graph Search and Beyond, pp. 23-27, 2015.
[27] T. Sorensen, *A Method of Establishing Groups of Equal Amplitude in Plant Sociology Based on Similarity of Species and its Application to Analyses of the Vegetation on Danish Commons*, Royal Danish Acad. Sci., Vol. 5, pp. 1-34, 1948.
[28] T. T. Tanimoto, *An Elementary Mathematical Theory of Classification and Prediction*, IBM Technical Report, 1958.
[29] A. Tversky, *Features of Similarity*, Psychological Reviews, Vol. 84, pp. 327-352, 1977.
[30] The University of Florida Matrix Collection, http://www.cise.ufl.edu/research/sparse/matrices/