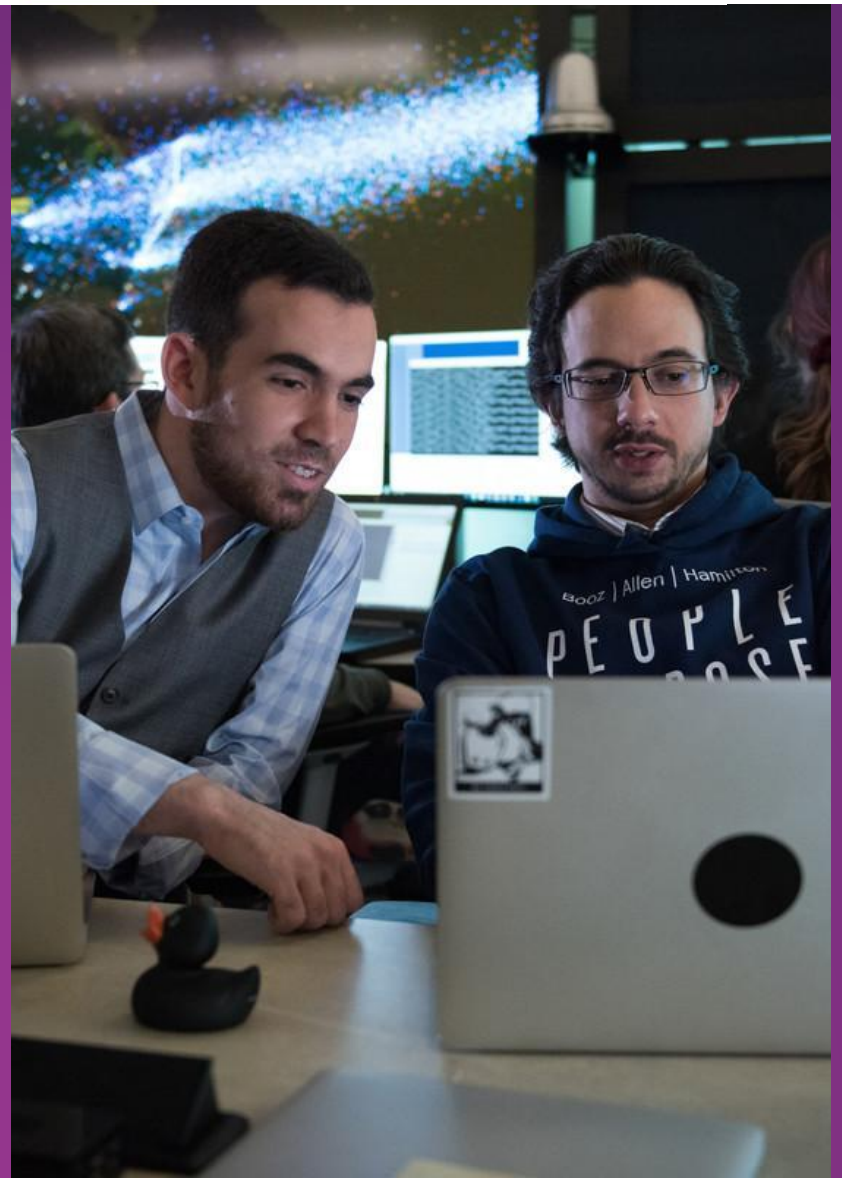
NVIDIA

UMBC

Booz | Allen | Hamilton



GPU SEMIRING PRIMITIVES FOR SPARSE NEIGHBORHOOD METHODS

Corey J. Nolet, Divye Gala, Edward Raff, Joe Eaton, Brad Rees, John Zedlewski, Tim Oates

AUGUST 2022

Publicly Available Libraries for Computing Sparse Distances are Inflexible & Inefficient

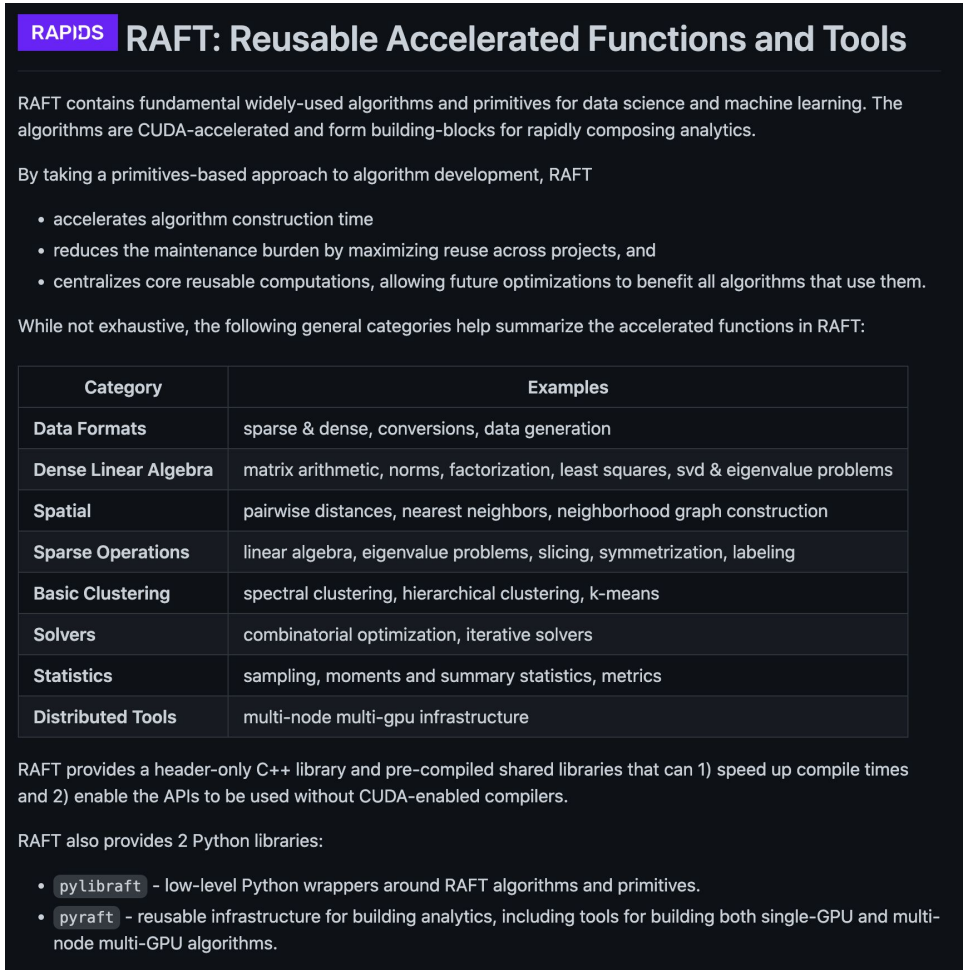
- Computing distances between vectors is core to many machine learning algorithms
  - Computing them efficiently for sparse datasets can be very hard
  - Little easier when it can computed with standard matrix multiplication (i.e., Euclidean)

- There are many distance metrics people want to use and a large variety of sparsity patterns and interactions to contend with

Publicly Available Libraries for Computing Sparse Distances are Inflexible & Inefficient

- We present a single unified framework for computing several important sparse pairwise distances
  - Fast and memory efficient across many different sparsity patterns.
  - Provides reusable building blocks for composing many different important metrics in ML.
  - Can be extended to different execution patterns by optimizing specific sparsity patterns.
  - Already available to you in RAPIDS!

https://github.com/rapidsai/raft



RAPIDS **RAFT: Reusable Accelerated Functions and Tools**

RAFT contains fundamental widely-used algorithms and primitives for data science and machine learning. The algorithms are CUDA-accelerated and form building-blocks for rapidly composing analytics.

By taking a primitives-based approach to algorithm development, RAFT

- accelerates algorithm construction time
- reduces the maintenance burden by maximizing reuse across projects, and
- centralizes core reusable computations, allowing future optimizations to benefit all algorithms that use them.

While not exhaustive, the following general categories help summarize the accelerated functions in RAFT:

Category	Examples
Data Formats	sparse & dense, conversions, data generation
Dense Linear Algebra	matrix arithmetic, norms, factorization, least squares, svd & eigenvalue problems
Spatial	pairwise distances, nearest neighbors, neighborhood graph construction
Sparse Operations	linear algebra, eigenvalue problems, slicing, symmetrization, labeling
Basic Clustering	spectral clustering, hierarchical clustering, k-means
Solvers	combinatorial optimization, iterative solvers
Statistics	sampling, moments and summary statistics, metrics
Distributed Tools	multi-node multi-gpu infrastructure

RAFT provides a header-only C++ library and pre-compiled shared libraries that can 1) speed up compile times and 2) enable the APIs to be used without CUDA-enabled compilers.

RAFT also provides 2 Python libraries:

- `pylibraft` - low-level Python wrappers around RAFT algorithms and primitives.
- `pyraft` - reusable infrastructure for building analytics, including tools for building both single-GPU and multi-node multi-GPU algorithms.

Semirings and Relation To Matrix Multiplication

- A ***monoid*** contains an associative binary relation, such as addition ($\oplus$), and an identity element ($id_{\oplus}$)

- A ***semiring***, denoted (S,R,{$\oplus$,$id_{\oplus}$},{$\otimes$,$id_{\otimes}$}), is a tuple containing additive ($\oplus$) and multiplicative ($\otimes$) monoids where
  1. $\oplus$ is commutative, distributive, and has an identity element 0
  2. $\otimes$ distributes over $\oplus$

- Given two sparse vectors $a,b \in R^k$, a semiring with (S,R,{$\oplus$,0},{$\otimes$,1}) and $annihilator_{\otimes} = 0$ is a **standard matrix multiplication**.

- Sparse Matrix-Vector multiplication (***SPMV***) is fundamental low-level BLAS routine in sparse matrix multiplication. Our contribution is a CUDA-accelerated Sparse Matrix-Sparse Vector (***SPSV***) multiplication primitive.

The Euclidean Semiring

- Let vector *a* = [1,0,1] and *b* = [0,1,0]
- Take the formula for computing Euclidean distance

$$\sqrt{\sum_{i=1}^{n} |x_i - y_i|^2}$$

- We can use the distributive property to compute more efficiently in parallel:
  - x^2 - 2<x,y> + y^2
  - Can compute with a simple dot product (plus-times semiring) and L2 norms of *x* and *y*.

$$\sum(|a-b|) = \tag{4}$$

$$\sum([|1-0|, |0-1|, |1-0|]) = \tag{5}$$

$$\sum([1,1,1]) = 3 \tag{6}$$

$$\sum(|a-b|) = \tag{7}$$

$$\sum([|1-0|, |0-1|, |1-0|]) = \tag{8}$$

$$\sum([0,0,0]) = 0 \tag{9}$$

The Manhattan Semiring and Non-Annihitilating Multiplicative Monoid (NAMM)

- Let vector *a* = [1,0,1] and *b* = [0,1,0]
- We take the sum of the absolute value of their differences (eqs 4, 5, 6)
- Semiring libraries rely on the detail that the multiplicative annihilator is equal to the additive identity.
  - If we follow this detail in our example, we end up with the following result of Eqs. 7, 8, 9 (if any side is 0, the arithmetic evaluates to 0).
- What we need here instead is for the multiplicative identity to be ***non-annihilating***, evaluating to the other side when either side is zero and evaluating to 0 only in the case where both sides have the same value. i.e., :

```
|1 - 0| = 1
|0 - 1| = 1
|0 - 0| = 0
|1 - 1| = 0
```

$$\sum(|a-b|) = \tag{4}$$

$$\sum([|1-0|, |0-1|, |1-0|]) = \tag{5}$$

$$\sum([1,1,1]) = 3 \tag{6}$$

$$\sum(|a-b|) = \tag{7}$$

$$\sum([|1-0|, |0-1|, |1-0|]) = \tag{8}$$

$$\sum([0,0,0]) = 0 \tag{9}$$

Semirings of Several Important Distances

Distance	Formula	NAMM	Norm	Expansion
Correlation	$1 - \frac{\sum_{i=0}^{k}(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum_{i=0}^{k}x_i-\bar{x}^2}^2\sqrt{\sum_{i=0}^{2}y_i-\bar{y}^2}^2}$		L_1,L_2	$1 - \frac{k\langle x\cdot y\rangle - \|x\|\|y\|}{\sqrt{(k\|x\|_2-\|x\|^2)(k\|y\|_2-\|y\|^2)}}$
Cosine	$\frac{\sum_{i=0}^{k}x_iy_i}{\sqrt{\sum_{i=0}^{k}x_i^2}\sqrt{\sum_{i=0}^{k}y_i^2}}$		L_2	$1 - \frac{\langle x\cdot y\rangle}{\|x\|_2^2\|y\|_2^2}$
Dice-Sorensen	$\frac{2\lvert\sum_{i=0}^{k}x_iy_i\rvert}{(\sum_{i=0}^{k}x)^2+(\sum_{i=0}^{k}y)^2}$		L_0	$\frac{2\langle x\cdot y\rangle}{\lvert x\rvert^2+\lvert y\rvert^2}$
Dot Product	$\sum_{i=0}^{k}x_iy_i$			$\langle x\cdot y\rangle$
Euclidean	$\sqrt{\sum_{i=0}^{k}\lvert x_i-y_i\rvert^2}$		L_2	$\|x\|_2^2 - 2\langle x\cdot y\rangle + \|y\|_2^2$
Canberra	$\sum_{i=0}^{k}\frac{\lvert x_i-y_i\rvert}{\lvert x_i\rvert+\lvert y_i\rvert}$	$\{\frac{\lvert x-y\rvert}{\lvert x\rvert+\lvert y\rvert}, 0\}$		
Chebyshev	$\sum_{i=0}^{k}\max(x_i-y_i)$	$\{\max(x-y), 0\}$		
Hamming	$\frac{\sum_{i=0}^{k}x_i\neq y_i}{k}$	$\{x\neq y, 0\}$		
Hellinger	$\frac{1}{\sqrt{2}}\sqrt{\sum_{i=0}^{k}(\sqrt{x_i}-\sqrt{y_i})^2}$			$1-\sqrt{\langle\sqrt{x}\cdot\sqrt{y}\rangle}$
Jaccard	$\frac{\sum_{i=0}^{k}x_iy_i}{(\sum_{i=0}^{k}x_i^2+\sum_{i=0}^{k}y_i^2-\sum_{i=0}^{k}x_iy_i}$		L_0	$1-\frac{\langle x\cdot y\rangle}{(\|x\|+\|y\|-\langle x\cdot y\rangle)}$
Jensen-Shannon	$\sqrt{\frac{\sum_{i=0}^{k}x_i\log\frac{x_i}{\mu_i}+y_i\log\frac{y_i}{\mu_i}}{2}}$	$\{x\log\frac{x}{\mu}+y\log\frac{y}{\mu}, 0\}$		
KL-Divergence	$\sum_{i=0}^{k}x_i\log(\frac{x_i}{y_i})$			$\langle x\cdot\log\frac{x}{y}\rangle$
Manhattan	$\sum_{i=0}^{k}\lvert x_i-y_i\rvert$	$\{\lvert x-y\rvert, 0\}$		
Minkowski	$(\sum_{i=0}^{k}\lvert x_i-y_i\rvert^p)^{1/p}$	$\{\lvert x-y\rvert^p, 0\}$		
Russel-Rao	$\frac{k-\sum_{i=0}^{2}x_iy_i}{k}$			$\frac{k-\langle x\cdot y\rangle}{k}$

SPSV CUDA Kernel: Load-Balanced Hybrid CSR+COO

1. **Load-balancing** using a row index array in coordinate format (COO) for B, coalescing the loads from each vector from A

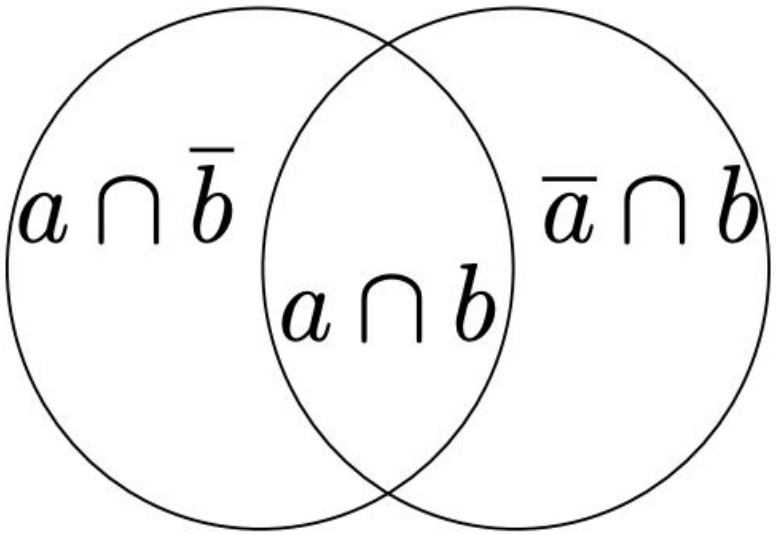
2. Lowered memory footprint by **removing the need to transpose B**.

3. **Two-pass execution**

Algorithm 3 Load-balanced Hybrid CSR+COO SPMV.

Input: $A_i, B, product\_op, reduce\_op$
Result: $C_{ij} = d(A_i, B_j)$
read A_i into shared memory
cur_row=rowidx[ind]
ind = idx of first elem to be processed by this thread
c = product_op(A[ind], x[colidx[ind]])
for $i \leftarrow 1$ **to** nz_per_chunk; **by** $warp\_size$ **do**
    next_row = cur_row + $warp\_size$
    if $next\_row$!= cur_row —— is_final_iter? **then**
        v = segmented_scan(cur_row, c, product_op)
        if $is\_segment\_leader$? **then**
            atomic_reduce(v, reduce_op)
        end
        c = 0
    end
    cur_row = next_row
    ind += $warp\_size$
    c = product_op(A[ind], x[colidx[ind]])
end

Implement NAMM with Mutliple Passes of an SPMV



$$a \cup b = \{a \cap b\} \cup \{\bar{a} \cap b\} \cup \{a \cap \bar{b}\}$$

- A single pass computes the intersection $a \cap b$ between nonzero columns from each vector *a*, and *b* so long as $\otimes$ is applied to all nonzero columns of *b*
- A second pass can compute the remaining symmetric difference required for the full union between non-zero column
- $id_{\otimes}$ in *B* is skipped in the second pass.

Performance- It's Fast But Also Memory Efficient

- Benchmarks were performed on a DGX1 containing dual 20-core Intel Xeon ES-2698 CPUs (80 total threads) at 2.20GHZ and a Volta V100 GPU running CUDA 11.0 for both the driver and toolkit.
- Each benchmark performs a k-nearest neighbors query to test our primitives end-to-end and allow scaling to datasets where the dense pairwise distance matrix may not otherwise fit in the memory of the GPU
- We used the brute-force *NearestNeighbors* estimator from RAPIDS cuML for the GPU benchmarks since it makes direct use of our primitive
- We used Scikit-learn's corresponding brute-force *NearestNeighbors* estimator as a CPU baseline and configured it to use all the available CPU cores

Performance- Fast And Memory Efficient

Table 3: Benchmark Results for all datasets under consideration. All times are in seconds, best result in **bold**. The first italicized set of distances can all be computed as dot products, which are already highly optimized for sparse comparisons today. This easier case we are still competitive, and sometimes faster, than the dot-product based metrics. The Non-trivial set of distances that are not well supported by existing software are below, and our approach dominates amongst all these metrics.

		MovieLens		scRNA		NY Times Bag of Words		SEC Edgar	
	Distance	Baseline	RAFT	Baseline	RAFT	Baseline	RAFT	Baseline	RAFT
Dot Product Based	*Correlation*	130.57	**111.20**	**207.00**	235.00	**257.36**	337.11	134.79	**87.99**
	Cosine	131.39	**110.01**	**206.00**	233.00	**257.73**	334.86	127.63	**87.96**
	Dice	130.52	**110.94**	**206.00**	233.00	**130.35**	335.49	134.36	**88.19**
	Euclidean	131.93	**111.38**	**206.00**	233.00	**258.38**	336.63	134.75	**87.77**
	Hellinger	129.79	**110.82**	**205.00**	232.00	**258.22**	334.80	134.11	**87.83**
	Jaccard	130.51	**110.67**	**206.00**	233.00	**258.24**	336.01	134.55	**87.73**
	Russel-Rao	130.35	**109.68**	**206.00**	232.00	**257.58**	332.93	134.31	**87.94**
Non-Trivial Metrics	Canberra	3014.34	**268.11**	4027.00	**598.00**	4164.98	**819.80**	505.71	**102.79**
	Chebyshev	1621.00	**336.05**	3907.00	**546.00**	2709.30	**1072.35**	253.00	**146.41**
	Hamming	1635.30	**229.59**	3902.00	**481.00**	2724.86	**728.05**	258.27	**97.65**
	Jensen-Shannon	7187.27	**415.12**	4257.00	**1052.00**	10869.32	**1331.37**	1248.83	**142.96**
	KL Divergence	5013.65	**170.06**	4117.00	**409.00**	7099.08	**525.32**	753.56	**87.72**
	Manhattan	1632.05	**227.98**	3904.00	**477.00**	2699.91	**715.78**	254.69	**98.05**
	Minkowski	1632.05	**367.17**	4051.00	**838.00**	5855.79	**1161.31**	646.71	**129.47**

It Is Used In The RAPIDS cuML Library

- Enables several clustering and manifold learning algorithms to accept sparse inputs.
- Also being used in cuML's Sparse k-Nearest Neighbors estimator.
- Already available in current RAPIDS, no hard work required.

https://github.com/rapidsai/cuml

```
from cuml.neighbors import
 ↪  NearestNeighbors
nn = NearestNeighbors().fit(X)
dists, inds = nn.kneighbors(X)
from cuml.metrics import
 ↪  pairwise_distances
dists = pairwise_distances(X,
 ↪  metric='cosine')
```

Figure 2: Excluding data loading and logging, all the code needed to perform the same GPU accelerated sparse distance calculations done in this paper are contained within these two snippets. Top shows k-NN search, bottom all pairwise distance matrix construction. These are the APIs that most would use.

RAFT Library Provides C++ API for Defining New Distance Semirings

Just define monoids!

```
#include
↪  <raft/sparse/distance/coo_spmv.cuh>
#include <raft/sparse/distance/operators.h>

using namespace raft::sparse::distance

distances_config_t<int, float> conf;

// Use conf to set input data arguments...

balanced_coo_pairwise_generalized_spmv(
    out_dists, conf, coo_rows_a,
    AbsDiff(), Sum(), AtomicSum());

balanced_coo_pairwise_generalized_spmv_rev(
    out_dists, conf, coo_rows_b,
    AbsDiff(), Sum(), AtomicSum());
```

Conclusion / Questions?

- Semirings provide us a framework for unifying many important distances in ML applications.
- Our SPSV kernel is state of the art in performance, efficiency and flexibility







Check out the paper for details!

@cjnolet

https://github.com/rapidsai/raft
https://github.com/rapidsai/cuml