L2AP: Fast Cosine Similarity Search With Prefix L-2 Norm Bounds

David C. Anastasiu and George Karypis

Karypis Lab, Computer Science & Engineering, University of Minnesota, Twin Cities

http://cs.umn.edu/~dragos/l2ap

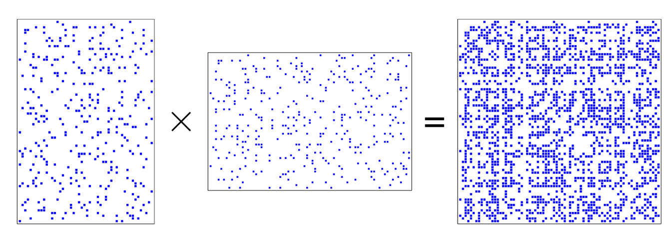
Introduction

- **All-Pairs Similarity Search (APSS):** For each object in a set, find all other objects within the same set with a similarity value of at least t.
- Crucial component in many data mining algorithms, e.g. near duplicate document detection, recommender systems, and clustering.
- `L2AP` leverages the Cauchy–Schwarz inequality to prune more of the search space than previous methods.
- Preliminaries:
  - Let objects be vectors, rows in a sparse matrix, $\mathbf{x}$ is row x in $\mathbf{D}$, $\mathbf{D} \in \mathbb{R}^{n \times m}$. Assume all rows normalized, $||\mathbf{x}|| = ||\mathbf{x}||_2 = 1,\ \forall\ x$ in $\mathbf{D}$. x_j is the value of feature j in $\mathbf{x}$.
  - We focus on cosine similarity between objects, thus $\text{sim}(\mathbf{x}, \mathbf{y}) = \mathbf{x}\mathbf{y}^T = \sum_{j=1}^{m} x_j \times y_j$.
  - If $\text{sim}(\mathbf{x}, \mathbf{y}) > t$, we say that $\mathbf{x}$ and $\mathbf{y}$ are neighbors.
  - We index $\mathbf{x}''$, the suffix of $\mathbf{x}$. We note by $\mathbf{x}''_p = \langle 0, \ldots, 0, x_p, \ldots, x_m \rangle$ the suffix of $\mathbf{x}$ starting at feature p. Similarly, $\mathbf{x}'_p = \langle x_1, \ldots, x_{p-1}, 0, \ldots, 0 \rangle$ is its prefix ending at $p-1$, and $\mathbf{x}'$ is the un-indexed portion of $\mathbf{x}$. $\Rightarrow \mathbf{x} = \mathbf{x}' + \mathbf{x}''$.

Naïve solution and initial extensions

- Compute similarity of each object with all others, keep results $\geq t$.
- Equivalent to sparse matrix-matrix multiplication: APSS $\sim \mathbf{D}\mathbf{D}^T. \geq t$

for each row $x = 1, \ldots, n$ **do**
  for each row $y = 1, \ldots, n$ **do**
    if $x \neq y$ & $\text{sim}(\mathbf{x}, \mathbf{y}) > t$ **then**
      Add $\{x, y, \text{sim}(\mathbf{x}, \mathbf{y})\}$ to result



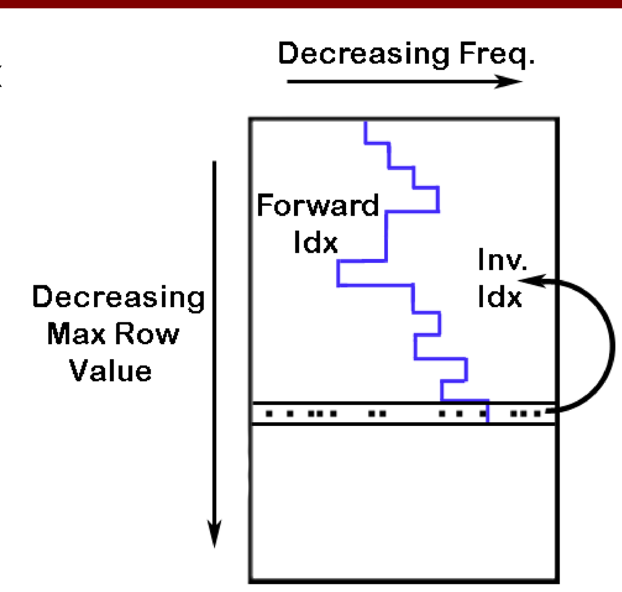
Extensions:

- Leverage sparsity in $\mathbf{D}$. Build an *inverted index*, a sparse column-wise representation of $\mathbf{D}$, then traverse the inverted lists for only $\mathbf{x}$'s features to find its possible neighbors.
- Leverage commutativity of $\cos(\mathbf{x}, \mathbf{y})$. Compute $\text{sim}(\mathbf{x}, \mathbf{y})$ only among vectors $\mathbf{y}$ with id less than x (Sarawagi and Kirpal, 2004).
- Build a partial index. We only need to index enough features of $\mathbf{x}$ to ensure its discovery as a potential neighbor (similarity candidate) during candidate generation for subsequent rows. (Chaudhuri et al., 2006)
- Leads to the straight-forward and practical `AllPairs` framework (Bayardo et al., 2007):

`AllPairs`:
for each rows $x = 1, \ldots, n$ **do**
  Find similarity candidates for $\mathbf{x}$ using current inverted index (candidate generation)
  Complete similarity computation and prune unpromising candidates (candidate verification)
  Index enough of $\mathbf{x}$ to ensure all valid similarity pairs are discovered (index construction)

Index construction

- Add a minimum number of non-zero features j of $\mathbf{x}$ to the inverted index lists I_j (index filtering).
- If we can guarantee that $\text{sim}(\mathbf{x}'_j, \mathbf{y}) < t,\ \forall y > x$, any such $\mathbf{y}$ *must* have at least one feature in common with $\mathbf{x}''_j$ if they are neighbors.
- Ordering $\mathbf{D}$ columns in decreasing frequency order heuristically leads to smaller index sizes.

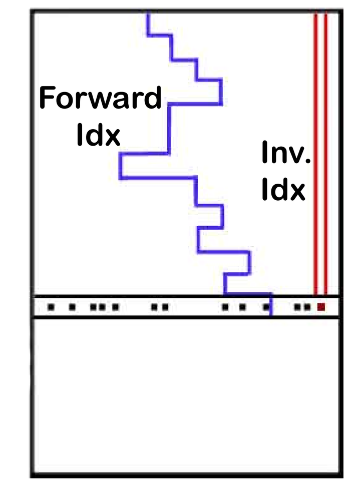


for each column $j = 1, \ldots, m$ s.t. $x_j > 0$ **do**
  if $\text{sim}(\mathbf{x}'_{j+1}, \mathbf{y}) \geq t,\ \forall y > x$ **then**
    $I_j \leftarrow I_j \cup \{(x, x_j)\}$

- Let $\mathbf{w} = \langle \max_z z_1, \ldots, \max_z z_m \rangle$, the vector of max column values in $\mathbf{D}$. $z = 1, \ldots, n$ is some row in $\mathbf{D}$. We can estimate $\text{sim}(\mathbf{x}'_{j+1}, \mathbf{y}) \leq \text{sim}(\mathbf{x}'_{j+1}, \mathbf{w})$.
- Leverage a permutation of $\mathbf{D}$'s rows. Permute rows in decreasing max row value ($||x||_\infty$) order. Let $\hat{\mathbf{w}} = \langle \min(x_1, \max_z z_1), \ldots, \min(x_m, \max_z z_m) \rangle$. Then $\text{sim}(\mathbf{x}'_{j+1}, \mathbf{y}) \leq \text{sim}(\mathbf{x}'_{j+1}, \hat{\mathbf{w}})$, since the y's we seek follow x in the row order (bound b_1, Bayardo et al., 2007).
- By the *Cauchy–Schwarz inequality*, $\text{sim}(\mathbf{x}, \mathbf{y}) = \mathbf{x}\mathbf{y}^T \leq ||\mathbf{x}|| \times ||\mathbf{y}||$, which also holds for $\text{sim}(\mathbf{x}'_{j+1}, \mathbf{y}) \leq ||\mathbf{x}'_{j+1}|| \times ||\mathbf{y}|| = ||\mathbf{x}'_{j+1}||$, since $||\mathbf{y}|| = 1$ (bound b_3). We store $||\mathbf{x}'_j||$ along with x_j in the index to use for later pruning.
- We use the minimum of the two bounds, $\min(b_1, b_3)$, and store $ps[x] \leftarrow \min(\text{sim}(\mathbf{x}'_j, \hat{\mathbf{w}}), ||\mathbf{x}_j||)$ to use in later candidate pruning.

Candidate generation

- During the candidate generation stage, we traverse the inverted index lists I_j corresponding to non-zero features j of $\mathbf{x}$ and keep track of a partial dot product ($A[y]$) for the candidates encountered.

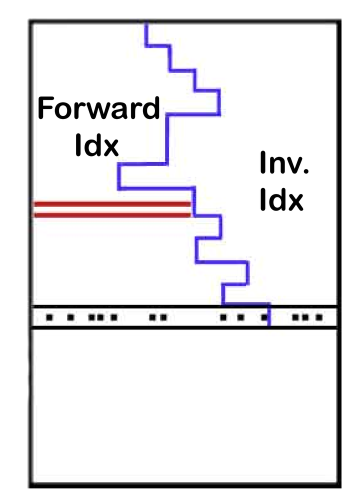


for each column $j = m, \ldots, 1$ s.t. $x_j > 0$ **do**
  for each $(y, y_j) \in I_j$ **do**
    if $A[y] > 0$ **or** $\text{sim}(\mathbf{x}'_j, \mathbf{y}) \geq t$ **then**
      $A[y] \leftarrow A[y] + x_j \times y_j$
      $A[y] \leftarrow 0$ **if** $A[y] + \text{sim}(\mathbf{x}, \mathbf{y}'_j) < t$

- Leverage t to prune potential candidates (residual filtering). Note that we are accumulating the *suffix dot product*, $\text{sim}(\mathbf{x}'', \mathbf{y})$. If the prefix similarity $\text{sim}(\mathbf{x}', \mathbf{y})$ is below t, and $A[y] = 0$, $\mathbf{y}$ cannot be a neighbor of $\mathbf{x}$.
- Given $\mathbf{w}$ defined as before, $\text{sim}(\mathbf{x}', \mathbf{y}) \leq \text{sim}(\mathbf{x}', \mathbf{w})$ (bound rs_1, Bayardo et al., 2007). We pre-compute $rs_1 = \mathbf{x}\mathbf{w}^T$, and roll back the computation as we process each inverted index column j. We stop accumulating *new candidates* once $rs_1 < t$.
- Candidates can only be those vectors with lower ids. We can thus improve rs_1 by using max column values of processed columns instead, $\widetilde{\mathbf{w}} = \langle \max_{z<x} z_1, \ldots, \max_{z<x} z_m \rangle$ (bound rs_3).
- By the *Cauchy–Schwarz inequality*, $\text{sim}(\mathbf{x}'_j, \mathbf{y}) \leq ||\mathbf{x}'_j|| \times ||\mathbf{y}|| = ||\mathbf{x}'_j||$, since $||\mathbf{y}|| = 1$ (bound rs_4). Once the ℓ^2 norm of $\mathbf{x}'_j$ falls below t, we ignore potential candidates $\mathbf{y}$ if $A[y] = 0$.
- While rs_4 is superior in most cases, it is not theoretically guaranteed to be so. We thus choose the best of both worlds, and check $\min(rs_3, rs_4) < t$ during residual filtering.
- We estimate $\text{sim}(\mathbf{x}, \mathbf{y}'_j) \leq ||\mathbf{y}_j||$, which is stored in the index, to prune some of the candidates (bound $l2cg$).

Candidate verification

- During the candidate verification stage, we use the forward index to finish computing the dot products for the encountered candidates, vectors with $A[y] > 0$.



for each $\mathbf{y}$ s.t. $A[y] > 0$ **do**
  next y **if** $A[y] + \text{sim}(\mathbf{x}, \mathbf{y}') < t$
  for each column j s.t. $y_j > 0 \wedge y_j \notin I_j \wedge x_j > 0$ **do**
    $A[y] \leftarrow A[y] + x_j \times y_j$
    next y **if** $A[y] + \text{sim}(\mathbf{x}, \mathbf{y}'_j) < t$
  Add $\{x, y, A[y]\}$ to result **if** $A[y] > t$

- Leverage t and the stored `pscore` $ps[y]$ obtained when indexing $\mathbf{y}$ to prune candidates (pscore filtering). Note that, after candidate generation, $A[y] = \text{sim}(\mathbf{x}, \mathbf{y}'')$. $ps[y]$ is an estimate of $\text{sim}(\mathbf{z}, \mathbf{y}'), \forall z > y$, including x, i.e., $\text{sim}(\mathbf{x}, \mathbf{y}') \leq ps[y]$. Prune if $A[y] + ps[y] < t$.
- While computing the final dot product, we estimate $\text{sim}(\mathbf{x}, \mathbf{y}'_j) \leq ||x'_j|| \times ||y'_j||$ and use it to prune additional candidates. (bound $l2cv$).
- Additional pruning obtained via `dpscore` and `minsize` filtering is described in the paper.

Approximate extension

- `BayesLSH-Lite` (Satuluri and Parthasarathy, 2012) finds the probability that $\text{sim}(\mathbf{x}, \mathbf{y}) > t$, conditional on observed LSH hash matches, after checking h hashes.
- We created two approximate APSS methods by combining `BayesLSH-Lite` with `L2AP`:
  - `L2AP+BayesLSH-Lite` - replace candidate verification with `BayesLSH-Lite`
  - `L2AP-approx` - replace only $l2cv$ bound pruning with `BayesLSH-Lite`

Baseline methods

- `IdxJoin` builds an inverted index and uses it to find $\text{sim}(\mathbf{x}, \mathbf{y}), \forall y < x$, without pruning.
- `AllPairs` uses max vector $\mathbf{w}$ in similarity estimates (bounds b_1 and rs_1)
- `MMJoin` (Lee et al., 2010) enhances `AllPairs` by adding *length filtering* and a tighter `minsize` bound. *Length filtering* estimates $\text{sim}(\mathbf{x}'_j, \mathbf{y}) \leq \frac{1}{2}||\mathbf{x}'_j||^2 + \frac{1}{2}||\mathbf{y}||^2 = \frac{1}{2}||\mathbf{x}'_j||^2 + \frac{1}{2}$, which is not as tight as our ℓ^2 norm estimate, $\text{sim}(\mathbf{x}'_j, \mathbf{y}) \leq ||\mathbf{x}'_j||$, especially for low t values.
- `AllPairs+BayesLSH-Lite` and `LSH+BayesLSH-Lite` are variants of `BayesLSH` that take as input the candidate set generated by `AllPairs` and `LSH`, respectively.
- Source code for exact methods, along with `L2AP` and `L2AP-approx`, available at `http://cs.umn.edu/~dragos/l2ap`.

Experimental Evaluation

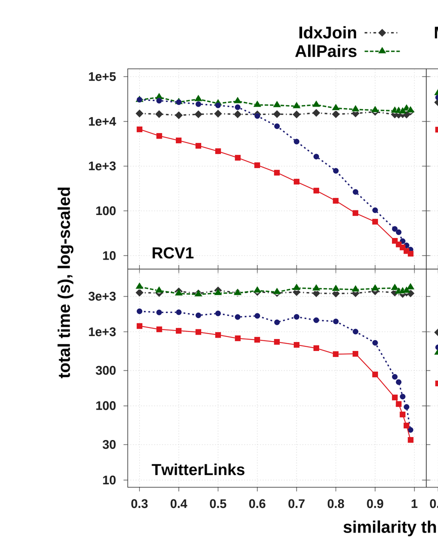
Datasets:

- **RCV1**: standard corpus of $> 800,000$ newswire stories.
- **WikiWords500k**: Wikipedia articles, min length 200.
- **WikiWords100k**: Wikipedia articles, min length 500.
- **TwitterLinks**: *follow* relationships of Twitter users that follow min 1,000 other users.
- **WikiLinks**: directed graph of hyperlinks between Wikipedia articles.
- **OrkutLinks**: Orkut friendship network.

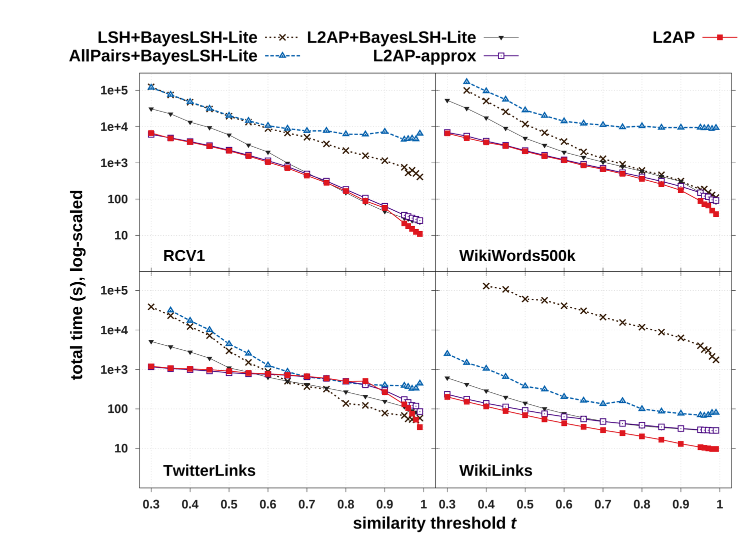
Dataset Statistics

Dataset	n	m	nnz
RCV1	804414	43001	61e6
WikiWords500k	494244	343622	197e6
WikiWords100k	100528	339944	79e6
TwitterLinks	146170	143469	200e6
WikiLinks	1815914	1648879	44e6
OrkutLinks	3072626	3072441	223e6

Efficiency testing



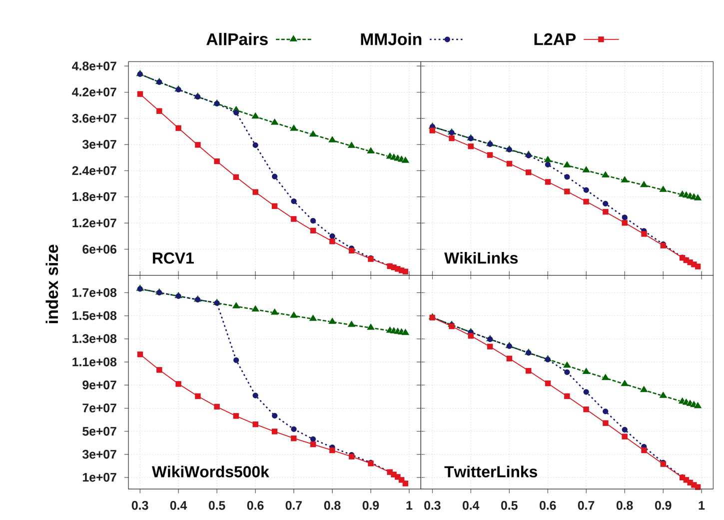
Execution times in seconds, exact methods



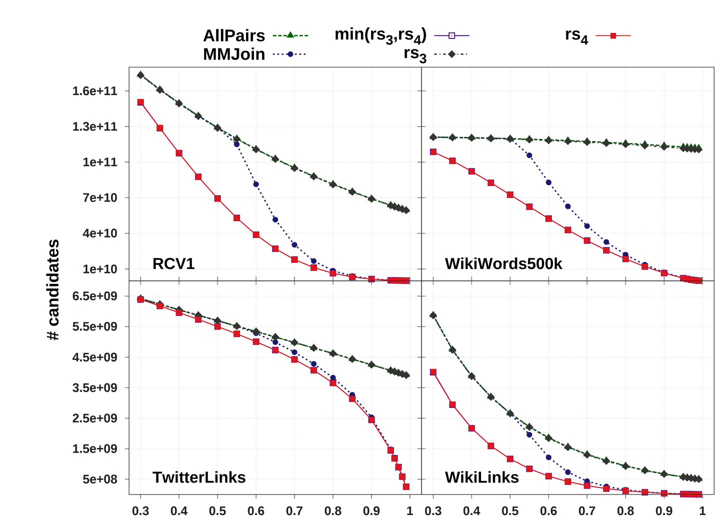
Execution times in seconds, approximate methods

- `L2AP` outperforms exact baselines in most cases and achieves significant speedups, up to 1600x against `AllPairs`, and 2x-13x in general over the best exact baseline.
- `L2AP`'s much smaller index and effective candidate pruning strategies allow it to finish the similarity search in a few seconds for high values of t, while `AllPairs` and `IdxJoin` spend hours to accomplish the same task.
- `L2AP` generally outperforms approximate baselines, especially at low similarity thresholds. It even outperforms `L2AP-approx` in most cases. `L2AP` is able to prune most candidates before the approximate `BayesLSH-Lite` candidate pruning step in `L2AP-approx`.

Effectiveness testing

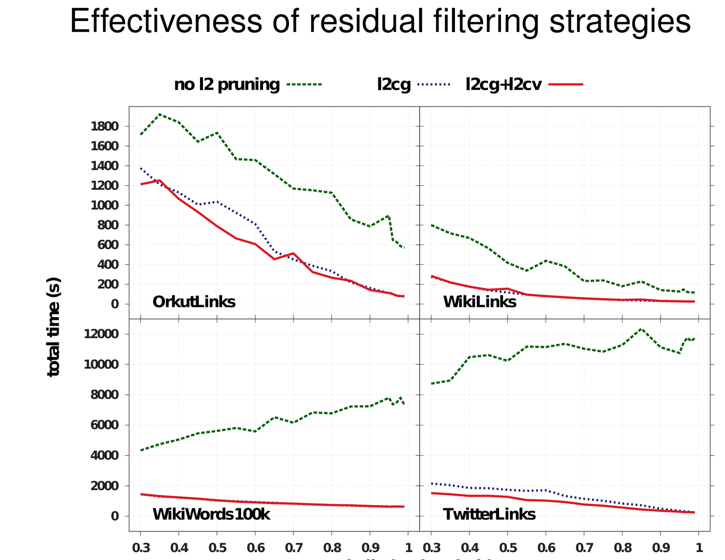


Index size reduction vs. previous methods



Effectiveness of residual filtering strategies

- `L2AP` produces significantly smaller indexes than previous methods. While `MMJoin` achieves similar index sizes at high t, its performance degrades to that of `AllPairs` as $t \to 0.5$.
- ℓ^2-norm filtering drastically reduces the number of generated candidates. The majority of the pruning happens in the candidate generation step, and most of the cost associated with this bound is in the initial computation of the prefix ℓ^2-norm.



Effectiveness of the new ℓ^2-norm filtering

Acknowledgements

This work was supported in part by NSF (IOS-0820730, IIS-0905220, OCI-1048018, CNS-1162405, and IIS-1247632) and the Digital Technology Center at the University of Minnesota. Access to computing and research facilities provided by the Digital Technology Center and the Minnesota Supercomputing Institute.