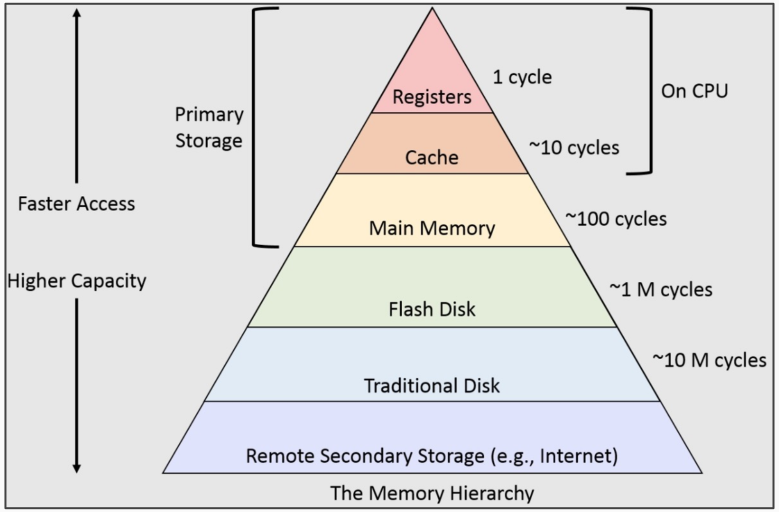
Locality Analysis of Graph Reordering Algorithms

By Mohsen Koohi Esfahani, Peter Kilpatrick, Hans Vandierendonck

Presented by Nick Dow

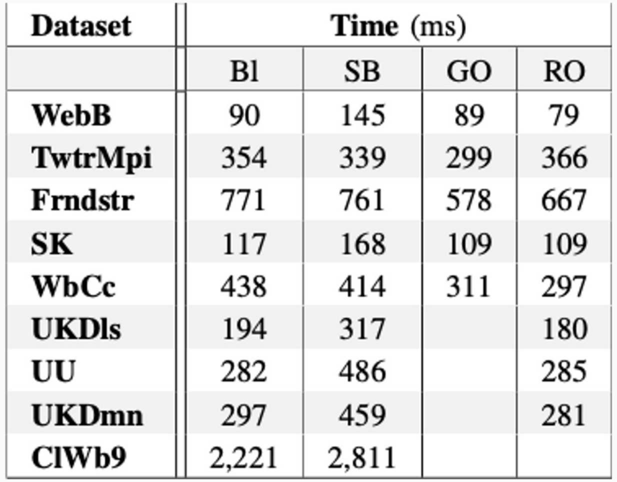
Problem: Graphs traversal is not sequential!

- Structure of graphs make vertex data accesses essentially random as any vertex can have an edge to any other.
- Recall the hierarchical memory model: random accesses are bad for cache.
- More caches misses means algorithms take more time to compute.
- Is there a way to improve these random accesses?



The Solution: Reordering Algorithms(RAs)

- **The Idea**: Relabel the vertices to give vertex data accesses better locality.
- RAs are experimentally shown to increase performance and lower cache misses on some graphs.
- But RAs make performance **worse** on others…why?
- There is little understanding of how RAs affect the structure of graphs



Dataset	Time (ms)			
	Bl	SB	GO	RO
WebB	90	145	89	79
TwtrMpi	354	339	299	366
Frndstr	771	761	578	667
SK	117	168	109	109
WbCc	438	414	311	297
UKDls	194	317		180
UU	282	486		285
UKDmn	297	459		281
ClWb9	2,221	2,811		

Above: RA graph performance

The Solution to The Solution: Analysis!

- We want to find the “why” of how RAs works across different RAs and graph types.
- **Key Questions:**
  - How much locality do natural graphs already have?
  - How do different RAs affect that initial locality?
  - For what types of locality are the graphs improved?
  - What vertices get better locality (LDV vs. HDV; in-hubs vs. out-hubs)?
- Apply these ideas to choose appropriate RAs and even modify them

Contributions

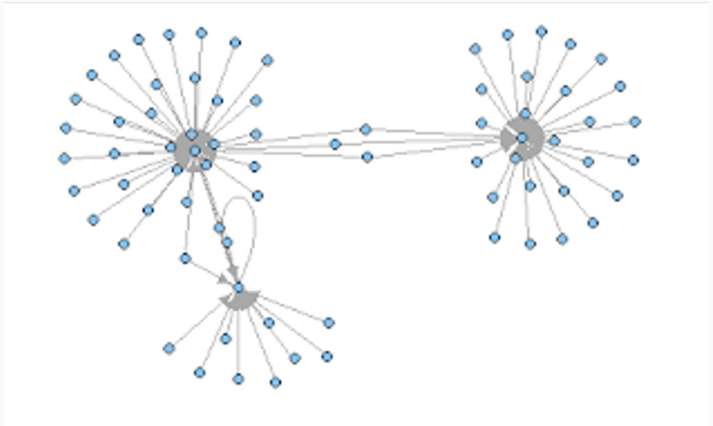
- Locality types in a parallel graph traversal
- Introducing the Neighbour to Neighbour Average ID Distance (N2N AID)
- Using degree distributions to study impacts of RAs on vertex classes,
- Degree range decomposition and degree distribution of asymmetricity to provide structural analysis of different graph types
- How locality manifests itself differently in a push traversal vs a pull traversal.

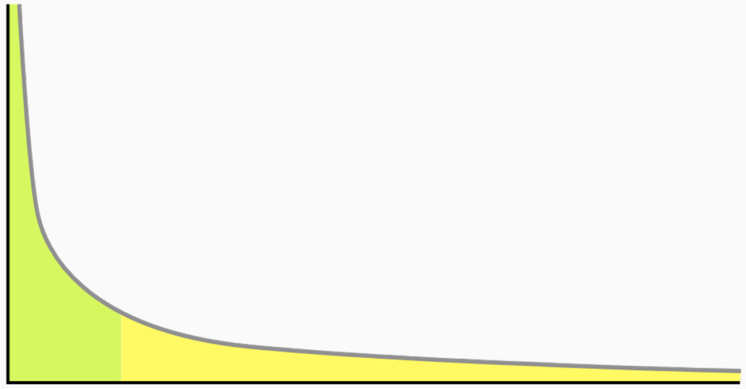
Overview:

Background Concepts

Graph Structure:

- Natural graphs have power law degree distributions.
- This means they have few well-connected high degree vertices (HDV) and many low degree vertices (LDV)
- Hubs are HDV with edges greater than square root of |V|


Example Natural Graph

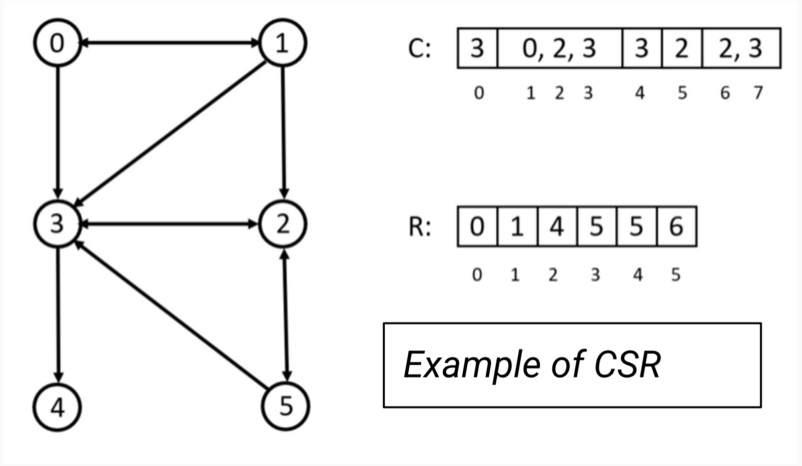

Power Law Distribution

Representation: Adjacency Arrays

- **Core idea**: Have each vertex have a list of its neighbors in order of ID.
  - For directed graphs, this list can be in-neighbors or out-neighbors

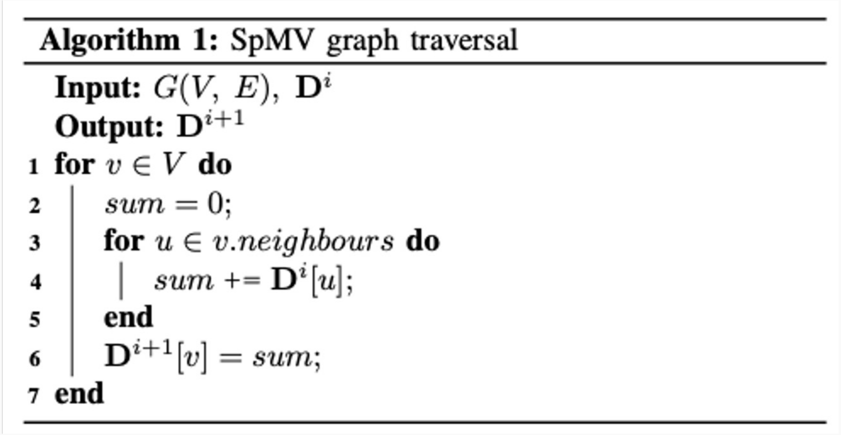
Compressed Sparse Column(CSC):
- Vertex ID indexes into a list of the vertex's in-neighbors.

Compressed Sparse Row(CSR):
- Vertex ID indexes into a list of the vertex's out-neighbors



Graph Traversal - SpMV

- SpMV can go in **pull** or **push** direction depending on representation (CSC v. CSR)
- Cache locality:
  - Edge data (topological) is only read once.
  - Vertex data is reused many times dependent on the vertex’s in or out degree.
  - **Pull** → randomly access old in-neighbors
  - **Push** → randomly access new out-neighbors



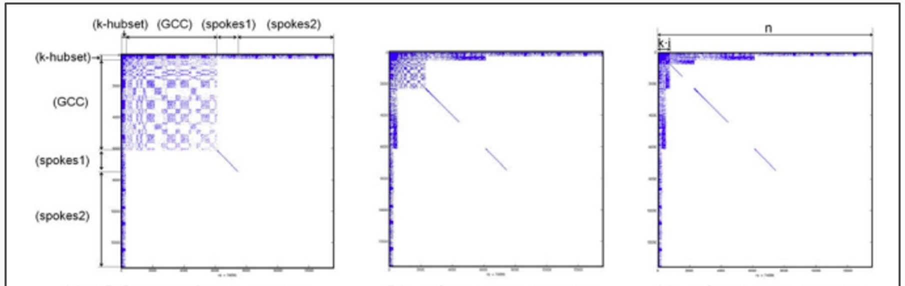
Algorithm 1: SpMV graph traversal

```
Input: $G(V, E)$, $\mathbf{D}^i$
Output: $\mathbf{D}^{i+1}$
for $v \in V$ do
    $sum = 0$;
    for $u \in v.neighbours$ do
        $sum \mathrel{+}= \mathbf{D}^i[u]$;
    end
    $\mathbf{D}^{i+1}[v] = sum$;
end
```

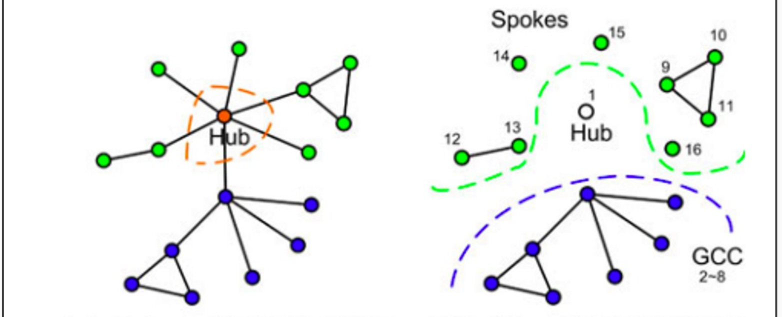
Reordering Algorithms

RA Example: Slash-Burn

- **Intuition**: Imagine the graph as consisting of hubs of HDV and spokes off the hubs, and the hubs being spokes of larger hubs and so on recursively.
- This structure stems from the power law properties of natural graphs.
- Theoretically gives better locality for the spokes of the graph.



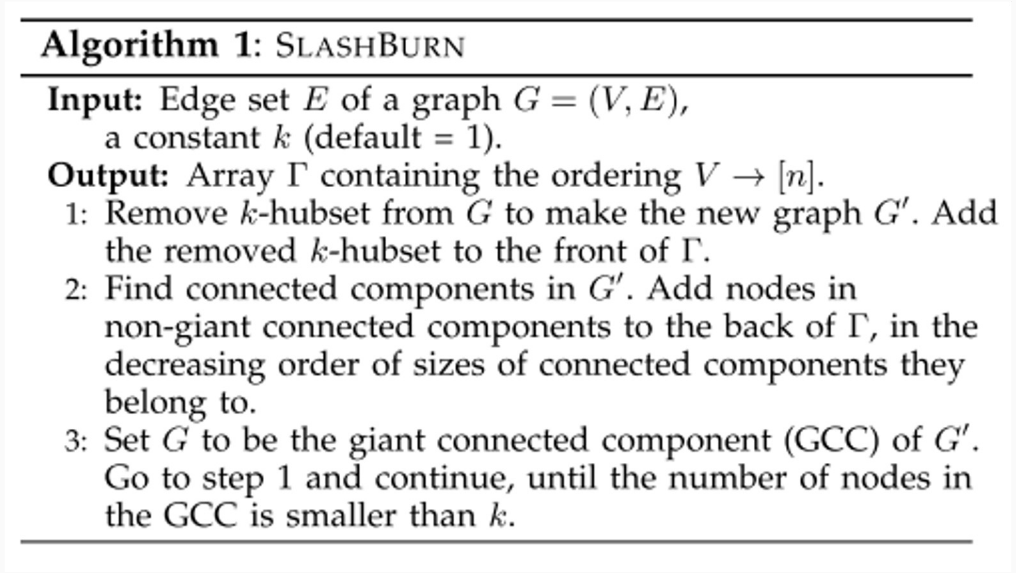
(a) AS-Oregon after 1 iteration (b) .. after 1 more iteration (c) .. after 1 more iteration



(a) Before SLASHBURN (b) After SLASHBURN

Slash-Burn Pseudocode

- *K-hubset* - a set of *k* candidate hub vertices.
- *Giant Connected Component* - the largest spoke to be recursively broken down.



```
Algorithm 1: SLASHBURN
Input: Edge set $E$ of a graph $G = (V, E)$,
    a constant $k$ (default = 1).
Output: Array $\Gamma$ containing the ordering $V \to [n]$.
 1: Remove $k$-hubset from $G$ to make the new graph $G'$. Add
    the removed $k$-hubset to the front of $\Gamma$.
 2: Find connected components in $G'$. Add nodes in
    non-giant connected components to the back of $\Gamma$, in the
    decreasing order of sizes of connected components they
    belong to.
 3: Set $G$ to be the giant connected component (GCC) of $G'$.
    Go to step 1 and continue, until the number of nodes in
    the GCC is smaller than $k$.
```

RabbitOrder

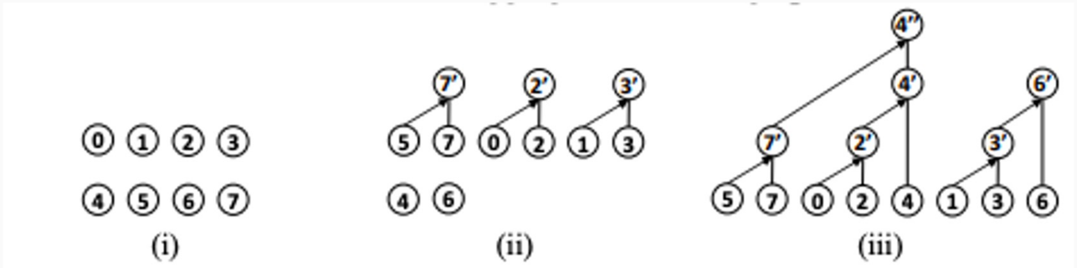
RabbitOrder:

- RO tries to increase locality by merging low degree vertices together recursively to construct local communities.
- Then RO performs parallel DFS on the tree of merges for each communities to number the vertices.
- When merging, uses the gain function: $\boxed{\Delta Q_{u,v} = 2(\frac{w_{uv}}{2|V|} - \frac{deg_u deg_v}{(2|V|)^2})}$
  - $w_{u,v}$= weight of edge (u, v); deg_v = degree of v
  - Merged vertices merge common edge weights and vertex weights.

RabbitOrder

RabbitOrder:

- RO tries to increase locality by merging low degree vertices together recursively to construct local communities.



(b) Dendrogram construction

GOrder

GOrder:

- GO increases locality by sequentially labeling vertices that share many in-neighbors and have a short path.
    - Heuristic: $S(v, u) = S_n(v, u) + S_s(v, u)$
    - $S_n(v, u)$ = # of in-neighbors shared between vertices u and v
    - $S_s(v, u)$ = # of edges between v and u
- GO chooses the next vertex to label by considering a sliding window of previous vertices and comparing with new vertices.
- By labeling vertices this way, GO aims to increase temporal locality.

Analytical Tools

Types of Locality

- **Spatial locality (Type I)**
  - Neighbors are loaded to cache together
- **Temporal locality (Type II)**
  - Subsequent vertices share neighbors in common
- **Spatio-Temporal locality (Type III)**
  - Subsequent vertices have different neighbors on the same cache line as previous vertices
- **Concurrent processing temporal locality (Type IV)**
  - Neighbor of a vertex is already loaded into cache by another thread
- **Concurrent processing spatio-temporal locality (Type IV)**
  - Neighbor shares a cache line with vertex already loaded into cache by another thread

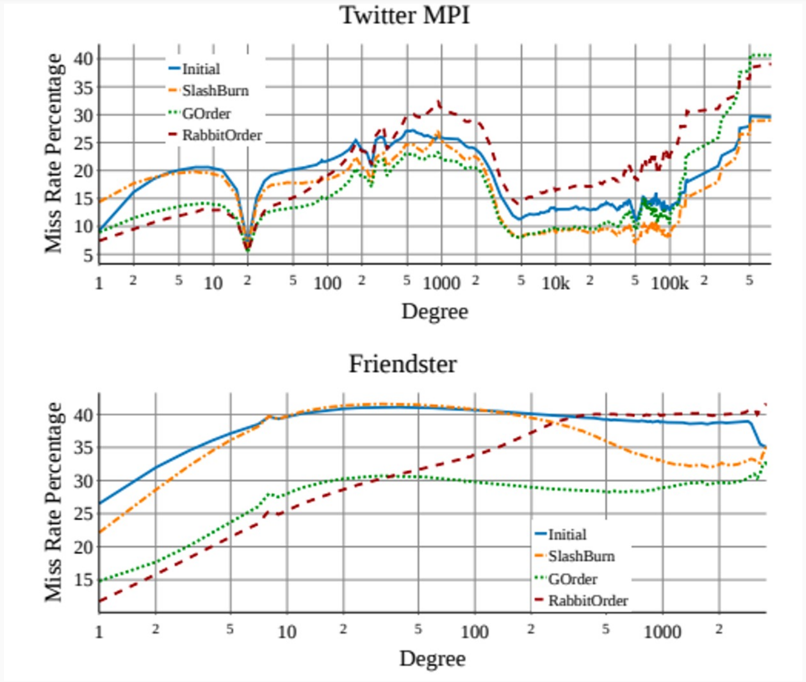
Neighbor to Neighbor Average ID Distance

- New Metric: N2NAID
- Meant to measure how close neighbors' IDs are
- Lower N2NAID intuitively results in better **Spatial Locality (Type I)**
- Useful to think as "average gap profile" in the CSR or CSC representation

$$AID_v = \frac{\sum_{i=2}^{i=|N_v|} |N_{v,i} - N_{v,i-1}|}{|N_v|}$$

Cache Miss Rate Degree Distribution

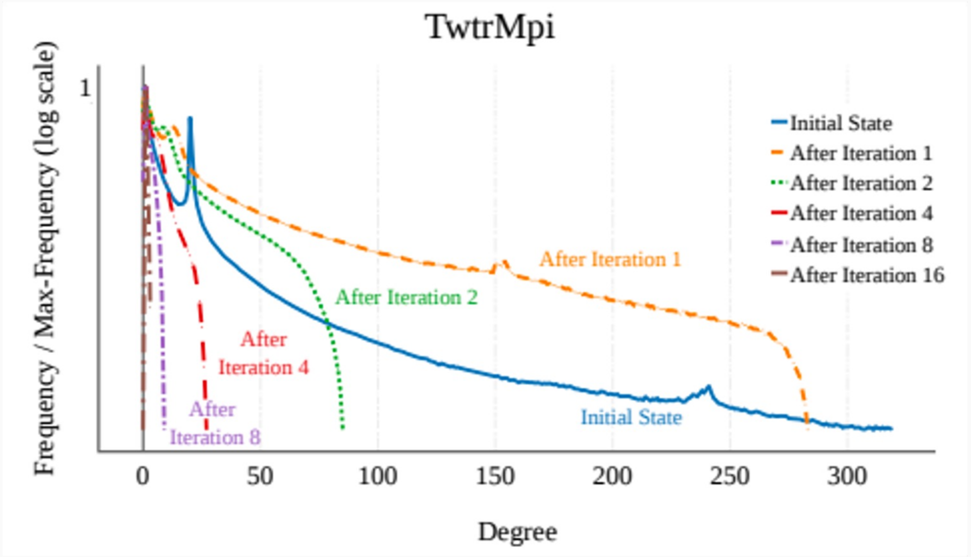
- Meant to quantify cache misses as it relates to type of vertex (HDV or LDV)
- Locality might be prioritized for LDV as they are the most common, or for HDV as they are needed the most often.



Analysis of RAs

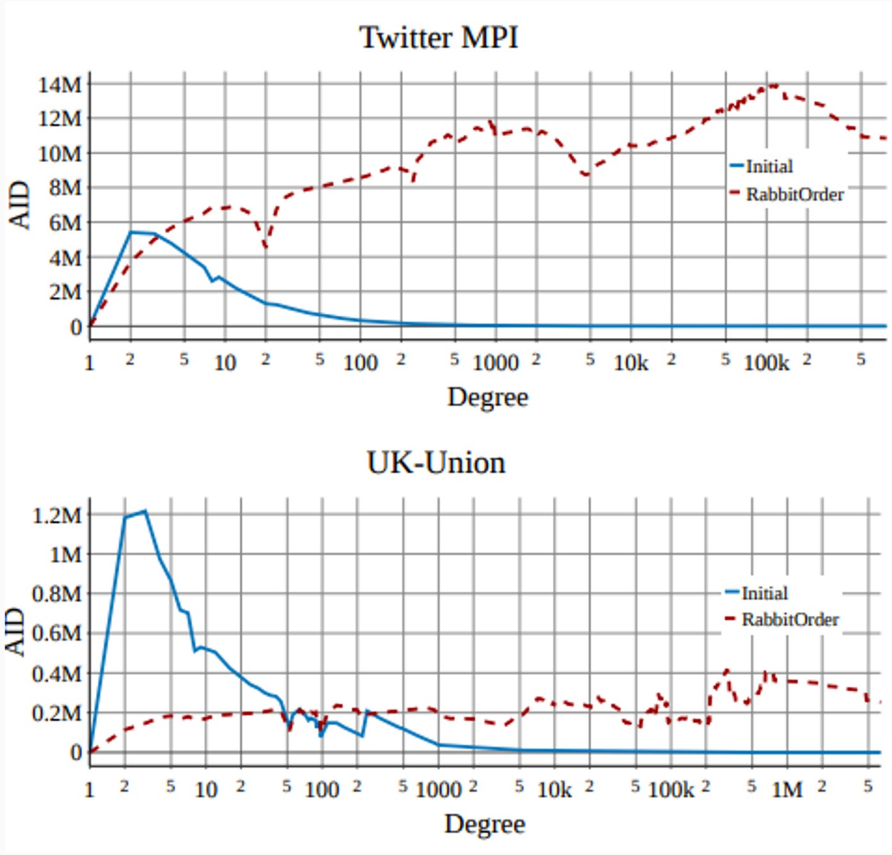
Slash-Burn

- Recall that Slash-Burn relies on the graph’s power law property to increase locality.
- The graph on the right shows that this property disappears quickly in successive iterations.
- After a certain point, SB **separates** LDV from neighbors, **decreasing** Type I and III locality.
- SB **increases** locality type II and III for HDV of out-hubs by grouping their IDs. This locality is useful in *pull* SpMV.



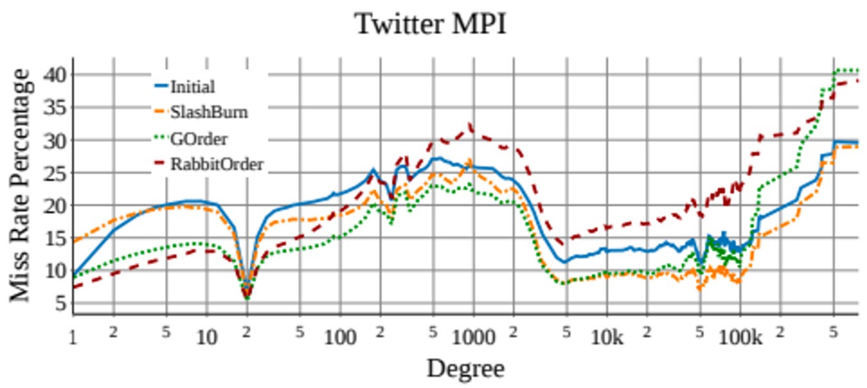
Rabbit-Order

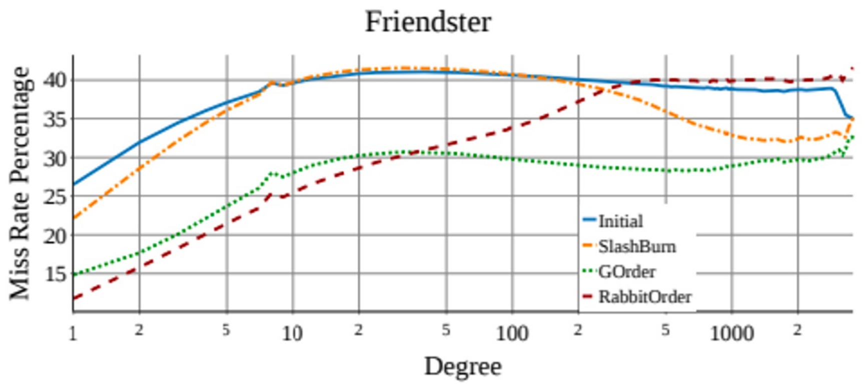
- The DFS assignment of neighbors reduced the N2NAID of LDV.
- However, as the number of neighbors a vertex has increases, consecutive IDs are less likely to be assigned to itself and other neighbors.
- This results in poor locality for HDV.



GOrder

- GOrder numbered vertices based on in-degree neighbors and proximity, aiming to increase Type II and III locality.
- GO was found to reduce cache misses for HDV but not for LDV. They reasoned this was due to the scoring heuristic and the size of the sibling window being evaluated.
- The ordering also occupied more of the cache with LDV rather than HDV.





Complete Results

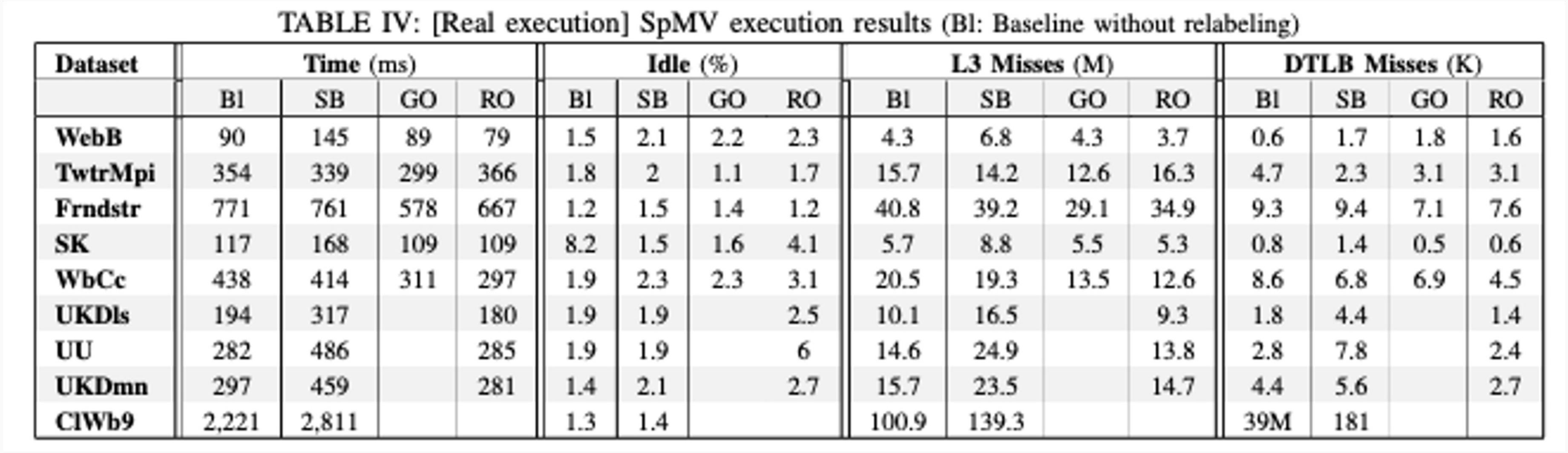


TABLE IV: [Real execution] SpMV execution results (Bl: Baseline without relabeling)

Dataset	**Time** (ms)				**Idle** (%)				**L3 Misses** (M)				**DTLB Misses** (K)			
	Bl	SB	GO	RO	Bl	SB	GO	RO	Bl	SB	GO	RO	Bl	SB	GO	RO
WebB	90	145	89	79	1.5	2.1	2.2	2.3	4.3	6.8	4.3	3.7	0.6	1.7	1.8	1.6
TwtrMpi	354	339	299	366	1.8	2	1.1	1.7	15.7	14.2	12.6	16.3	4.7	2.3	3.1	3.1
Frndstr	771	761	578	667	1.2	1.5	1.4	1.2	40.8	39.2	29.1	34.9	9.3	9.4	7.1	7.6
SK	117	168	109	109	8.2	1.5	1.6	4.1	5.7	8.8	5.5	5.3	0.8	1.4	0.5	0.6
WbCc	438	414	311	297	1.9	2.3	2.3	3.1	20.5	19.3	13.5	12.6	8.6	6.8	6.9	4.5
UKDls	194	317		180	1.9	1.9		2.5	10.1	16.5		9.3	1.8	4.4		1.4
UU	282	486		285	1.9	1.9		6	14.6	24.9		13.8	2.8	7.8		2.4
UKDmn	297	459		281	1.4	2.1		2.7	15.7	23.5		14.7	4.4	5.6		2.7
ClWb9	2,221	2,811			1.3	1.4			100.9	139.3			39M	181		

Web-Graph & Social Graph Structures

- As we saw in previous results, RO produced better results on web graphs than GO, and the same went for GO on social graphs. Why?
- Social Graphs have highly-symmetrical in-hubs, while web graphs do not.
- GO performed better with social graphs due to this symmetry; HDVs have many HDV neighbors.
- RO performed better with web graphs as HDVs overwhelmingly have LDV neighbors so LDV locality was more important.

Push vs. Pull Locality

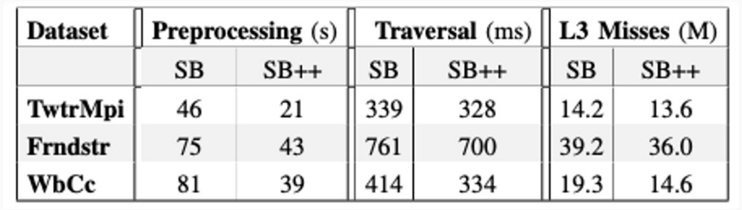
- The direction of traversal also leverages the structure for locality.
- Pull works better for web-graph because of the asymmetrical out-hubs; that data is used many times.
- Push works better for social graphs due to the high in-hubs.

Applying the Analysis

RA improvements

Slash-Burn++:

- Avoid pitfalls of SB by stopping early, when power law stops holding.
- Good results!



Dataset	Preprocessing (s)		Traversal (ms)		L3 Misses (M)	
	SB	SB++	SB	SB++	SB	SB++
TwtrMpi	46	21	339	328	14.2	13.6
Frndstr	75	43	761	700	39.2	36.0
WbCc	81	39	414	334	19.3	14.6

Limit reordering in RO:

- Essentially, find a range of vertex degrees that RO is not effective for, and have a quick special case.
- Found pre-processing time was reduced by 4x on some graphs.

Future Work

Dynamically-sized Window for GO:

- Size of the window would be large for LDVs, and small for HDV.
- This would better reflect the heuristic GO uses for ordering.

Combining Rabbit-Order & GOrder:

- Suggest transitioning from RO to GO when going from LDV to HDV
- Could have best of both worlds potentially.