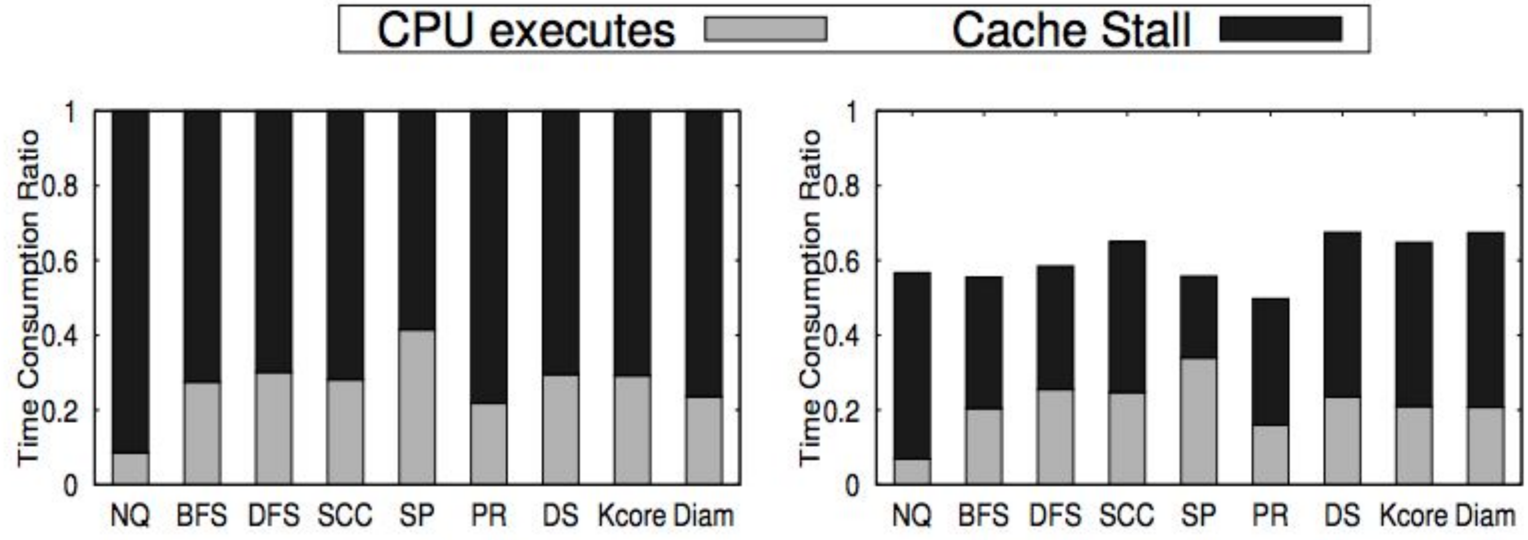
Speedup Graph Processing by Graph Ordering

Hao Wei, Jeffrey Xu Yu, Can Lu, Xuemin Lin

Presented by: Bishesh Khadka
MIT 6.886 - Graph Analytics

Motivation

- Graphs are important
- CPU cache performance is key issue in efficiency in DBS



(a) The original order (b) Gorder

Motivation

- Graphs are important
- CPU cache performance is key issue in efficiency in DBS
  - Cache stalls take a large proportion of time
- Can better locality via ordering help?
  - Store frequently accessed nodes close in memory
- How can a generalized solution reduce cache stall rates?

Graph Access Patterns

- Most common access pattern:
    - **for each node** $v \in N_O(u)$ **do**
      the program segment to compute/access v
- Locality between neighboring nodes are important
- Locality among sibling nodes even more important
    - $\binom{d_O(u)}{2} \gg d_O(u)$
- Let “closeness” heuristic be S(u, v) = S_s(u, v) + S_n(u, v)

Graph Partitioning isn't sufficient

- Real graphs have poor edge cuts b/c power law degree distributions
  - Nodes w/ high degrees
- Fixed sized caches
  - What partition size?
- Data alignment

Assume a cache line holds 3

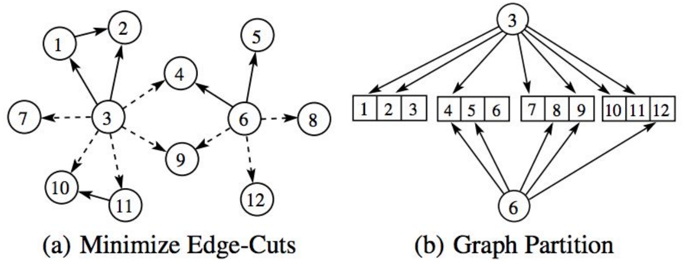


Figure 3: By Graph Partitioning

Graph Ordering does better

- Optimal permutation ϕ among
- Frequently accessed nodes within window w
- Reorder graph id's
- Sort in all adj. lists

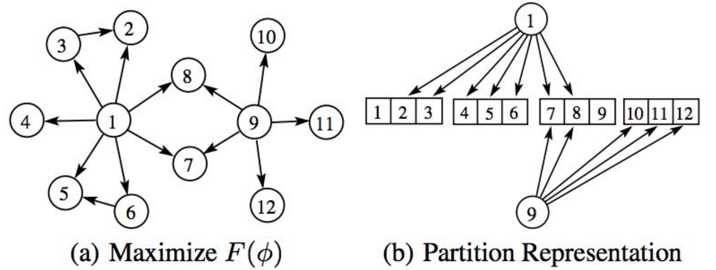


Figure 4: By Graph Ordering

Graph Ordering does better cont'd

- Locality is continuous for any sliding window
  - Assumes little of data alignment
- Considers sibling and neighbor locality

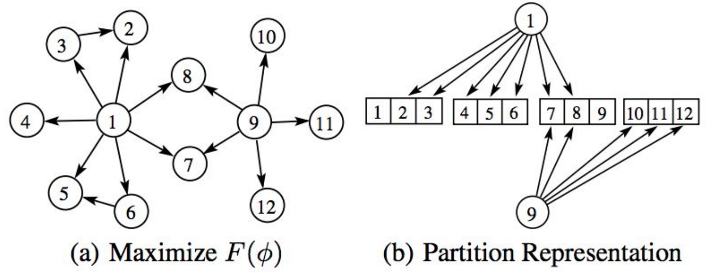


Figure 4: By Graph Ordering

Problem Statement

- Find the optimal permutation ϕ that maximizes aggregate locality defined by F(ϕ) for all sliding windows of size w
-

$$F(\phi) = \sum_{0<\phi(v)-\phi(u)\leq w} S(u,v) \quad (2)$$

$$= \sum_{i=1}^{n} \sum_{j=\max\{1,i-w\}}^{i-1} S(v_i, v_j) \quad (3)$$

Key Contributions

- Locality scoring function
- Prove NP-hardness of graph ordering
  - Graph ordering is a variant of maximum TSP
    - Maximize reward for sliding windows w
- Propose two algorithms for graph ordering
  - GO
  - GO-PQ
- Evaluation of improved efficiency

GO algorithm

Algorithm 1 *GO* $(G, w, S(\cdot,\cdot))$

select a node v as the start node, $P[1] \leftarrow v$;
$V_R \leftarrow V(G) \setminus \{v\}, i \leftarrow 2$;
while $i \leq n$ **do**
    $v_{max} \leftarrow \emptyset, k_{max} \leftarrow -\infty$;
    for each node $v \in V_R$ **do**
        $k_v \leftarrow \sum_{j=\max\{1,i-w\}}^{i-1} S(P[j], v)$;
        if $k_v > k_{max}$ **then**
            $v_{max} \leftarrow v, k_{max} \leftarrow k_v$;
    $P[i] \leftarrow v_{max}, i \leftarrow i + 1$;
    $V_R \leftarrow V_R \setminus \{v_{max}\}$;

GO algorithm

- Greedily maximize F(ϕ) by inserting v with the largest aggregate S() in previous window w
- Randomly select starting node
- Redundantly computes eq. 4 w-times for same pair (v_j, v) while in same window
- Scans through even nodes w/o neighbor/sibling relationships

Eq. 4

$$k_v = \sum_{j=\max\{1,i-w\}}^{i-1} S(v_j, v)$$

Fgo is GO result
Fw is upper bound of optimal locality score

	$w=3$		$w=5$		$w=7$	
	F_{go}	$\overline{F}_w$	F_{go}	$\overline{F}_w$	F_{go}	$\overline{F}_w$
Facebook	149,073	172,526	231,710	275,974	308,091	373,685
AirTraffic	2,420	3,468	2,993	4,697	3,465	5,545

Table 1: F_{go} and $\overline{F}_w$

GO-PQ algorithm

Algorithm 2 *GO-PQ* $(G, w, S(\cdot,\cdot))$

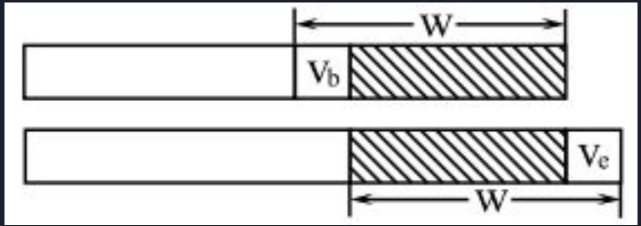
```
for each node v ∈ V(G) do
    insert v into Q such that key(v) ← 0;
select a node v as the start node, P[1] ← v, delete v from Q;
i ← 2;
while i ≤ n do
    v_e ← P[i − 1];
    for each node u ∈ N_O(v_e) do
        if u ∈ Q then Q.incKey(u);
    for each node u ∈ N_I(v_e) do
        if u ∈ Q then Q.incKey(u);
        for each node v ∈ N_O(u) do
            if v ∈ Q then Q.incKey(v);
    if i > w + 1 then
        v_b ← P[i − w − 1];
        for each node u ∈ N_O(v_b) do
            if u ∈ Q then Q.decKey(u);
        for each node u ∈ N_I(v_b) do
            if u ∈ Q then Q.decKey(u);
            for each node v ∈ N_O(u) do
                if v ∈ Q then Q.decKey(v);
    v_max ← Q.pop();
    P[i] ← v_max, i ← i + 1;
```

GO-PQ algorithm

- Similar to GO
- Uses PQ to maintain sliding window
- Q[v] = k_v as computed by Eq. 4
- When V_e joins, v in W increment their keys if there is a neighbor and/or sibling relation
- V_b leaves, v w/ relations decrements key
- Pops largest key as V_b

Eq. 4

$$k_v = \sum_{j=\max\{1,i-w\}}^{i-1} S(v_j, v)$$

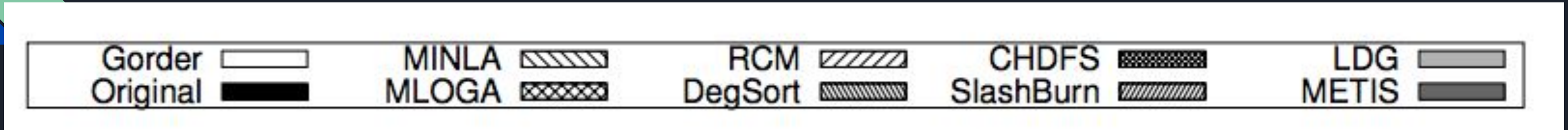


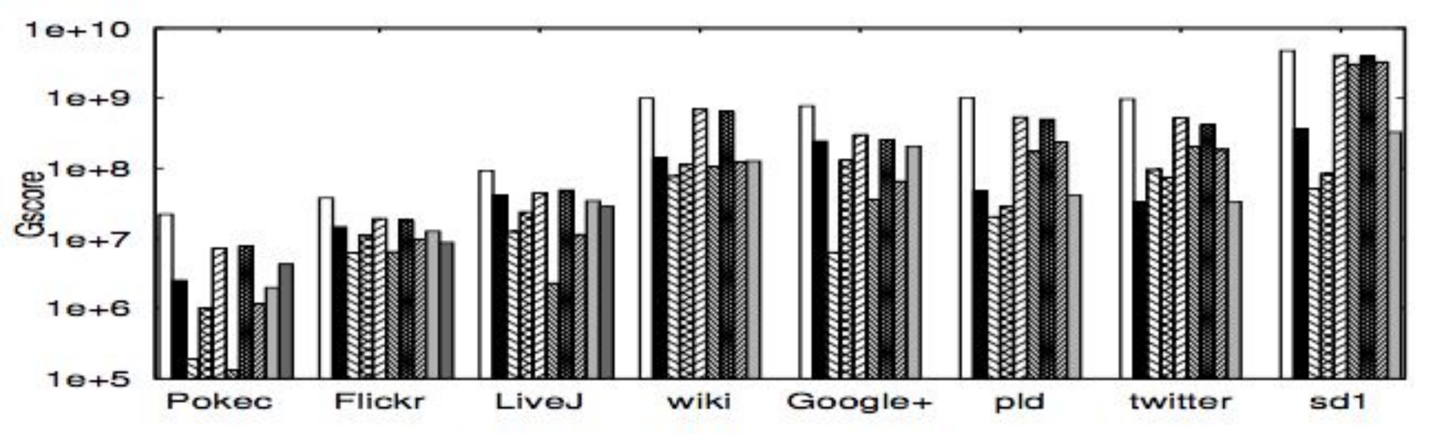
Time complexities

Theorem 3.2: *The GO Algorithm 1 is in* $O(w \cdot d_{max} \cdot n^2)$*, where* d_{max} *denotes the maximum in-degree of the graph* G*.*

Theorem 3.3: *The time complexity of the GO-PQ algorithm is* $O(\mu \cdot \sum_{u \in V}(d_O(u))^2 + n \cdot \varrho)$*, where* μ *denotes the time complexity for the updates* (incKey(·) *and* decKey(·)) *and* ϱ *denotes the time complexity for finding the max node* (pop()).

Evaluation





(a) $F(\cdot)$ by Different Orderings

Evaluation

Order	L1-ref	L1-mr	L3-ref	L3-r	Cache-mr
Original	11,109M	52.1%	2,195M	19.7%	5.1%
MINLA	11,110M	58.1%	2,121M	19.0%	4.5%
MLOGA	11,119M	53.1%	1,685M	15.1%	4.1%
RCM	11,102M	49.8%	1,834M	16.5%	4.1%
DegSort	11,121M	58.3%	2,597M	23.3%	5.3%
CHDFS	11,107M	49.9%	1,850M	16.7%	4.4%
SlashBurn	11,096M	55.0%	2,466M	22.2%	4.3%
LDG	11,112M	52.9%	2,256M	20.3%	5.4%
METIS	11,105M	50.3%	2,235M	20.1%	5.2%
Gorder	11,101M	**37.9%**	**1,280M**	**11.5%**	**3.4%**

Table 3: Cache Statistics by *PR* over Flickr **(M = Millions)**

Evaluation

Order	NQ	BFS	DFS	SCC	SP	PR	DS	Kcore	Diam
Original	76.5	20.0	9.4	13.0	17.5	58.4	21.7	20.0	17.5
MINLA	76.0	22.7	10.2	12.8	20.7	62.5	21.8	20.5	18.3
MLOGA	76.0	21.7	9.4	12.3	19.8	62.1	21.8	20.6	18.5
RCM	61.6	14.4	7.5	8.7	**8.9**	44.9	18.2	17.5	11.7
DegSort	59.3	18.7	8.0	12.1	16.6	55.1	21.9	16.9	15.5
CHDFS	50.0	14.2	5.1	8.3	13.2	38.0	18.4	16.1	10.4
SlashBurn	56.6	16.8	6.7	9.3	10.2	44.5	18.9	16.8	13.5
LDG	74.7	22.7	10.0	13.6	18.7	58.4	22.0	20.3	17.9
Gorder	**40.0**	**12.1**	**4.6**	**7.2**	10.8	**31.5**	**16.9**	**14.5**	**9.5**

Table 7: L1 Cache Miss Ratio on sd1-arc (in percentage %)

Evaluation

- Applying Gorder to distributed graph systems is complicated b/c unclear how graph partitioning happens

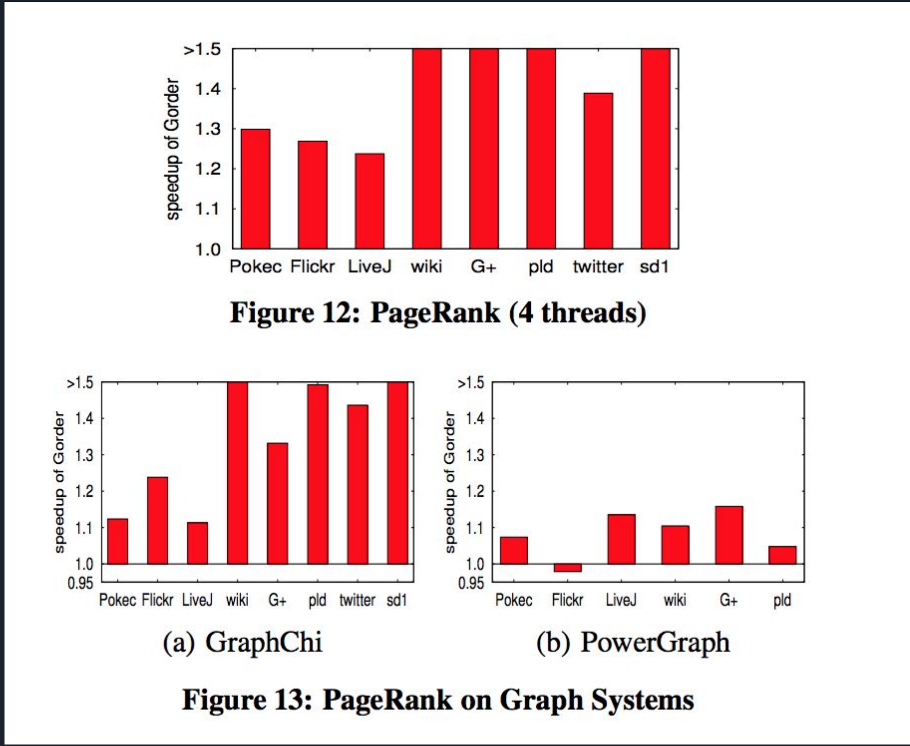


Figure 12: PageRank (4 threads)

Figure 13: PageRank on Graph Systems

Conclusion

- CPU stalling is important barrier to efficiency
- This paper presents a generalized optimization for graph algorithms with the common access pattern
    - **for each node** $v \in N_O(u)$ **do**
      the program segment to compute/access v

References

- Hao Wei, Jeffrey Xu Yu, Can Lu, Xuemin Lin
  Speedup Graph Processing by Graph Ordering