Partitioned Multi-Indexing: Bringing Order to Social Search
Bahman Bahmani, Ashish Goel
Stanford University

Problem Statement
- Real-Time Social Search: Find a piece of content that is exciting to my extended network right now and matches my search criteria
- Hard technical problem: imagine building 100M real-time indices over real-time content

Current Status: No Good Solution...

… Even without the Real-Time Component

Overview of Partitioned Multi-Indexing
- Maintain a small number (e.g., 100) of indexes of real-time content, and a same small number of graph sketches
- Content indexes can be updated in real-time. Graph sketches batched
- Real-time efficient querying
- Implementable on a simple distributed stream-processing system

Algorithms

Algorithm 1 Distance Sketching Algorithm.
1. Input: Undirected graph $G$, $k \geq 1$, $0 \leq \alpha \leq \log_2 n$
2. Let $h = k(r + 1)$
3. for $i = 0$ to $h - 1$
   4. Sample, uniformly at random, a subset $S_i \subseteq V$ of size $|S_i| = 2^i \mod (r + 1)$
   5. Do a BFS from $S_i$, and compute, for all $u \in V$, $L_i[u]$ = $\arg\max_{v \in S_i} \{d(u, v)\}$, and $D_i[u] = d(u, L_i[u])$
   6. end for
7. $\forall u \in V$, let $\forall v \in V$, let $E[u] = \langle (L_0[u], D_0[u]), \ldots, (L_{h-1}[u], D_{h-1}[u]) \rangle$
8. $d(u, v) \leq \min\{D_i[u] + D_j[v] \mid 0 \leq i < h, L_i[u] = L_j[v]\}$

Algorithm 2 Partitioned Multi-Indexing Algorithm.
1. Input: Social graph $G$, corpus $C$, and the distance sketches $E[u] \forall u \in V$
2. Initialize the map $PMI$: $\forall i \leq h$, $z \in S_i$, and word $\omega$, $PMI[i, z, \omega]$ is an empty priority queue.
3. for $v \in V$
   4. for $0 \leq i < h$, $\omega \in C_0$
      5. Insert $v$ into $PMI[i, L_i[v], \omega]$ with priority $D_i[v]$
   6. end for
7. end for

Algorithm 3 Partitioned Multi-Index Query Algorithm.
1. Input: Distance sketches $E[u] \forall u \in V$, Partitioned multi-index $PMI$, and a query $(u_i, J)$
2. $\forall 0 \leq i < h$, $p_i = 1$
3. Initialize $H$ to be an empty priority queue.
4. $\forall 0 \leq i < h$
   5. Insert $x_{t_1}^{i, p_i}([\omega])$ into $H$ with priority $D_i[u] + D_i[p_i]([\omega])$
   6. end for
7. $J = J + 1$
8. while $(1 \leq J)$
   9. Pop the node with the smallest priority from $H$, and let $s = \arg\min_{u \in S_i} \{D_i[u] + D_i[s]([\omega])\}$
   10. if $(\forall j' \leq j \exists x_{t_1}^{i, p_i}([\omega]) \neq v_{j'})$
      11. $v_{j'} = x_{t_1}^{i, p_i}([\omega])$
      12. $j' = j + 1$
      13. end if
   14. $p_{s} = p_{s} + 1$
   15. Insert $x_{t_1}^{s}([\omega])$ into $H$ with priority $D_i[u] + D_i[p_{s}([\omega])$
   16. end while
17. return $\{v_{j} \mid 1 \leq j \leq \text{search result length}\}$ as the ranked list of search results

Mixing in Other Relevance Measures
- Examples of other important relevance measures: PageRank, tf-idf, and recency
- Rank based on:
  - The approximate score decomposes:
    $\hat{s}_{u, \omega}(v) = \lambda d(u, v) + (1 - \lambda)\alpha_{\omega}(v)$
  - And leads to same guarantees.

Distributed Implementation
- Sketching easily doable on MapReduce
- Indexing, updates, and search queries can be implemented on a DHT like Memcached or an Active DHT like Yahoo! S4 or Twitter Storm
- Only 2 network accesses per query/update
- Total network communication almost constant per update or per search result

Experiments
- Networks: Grid and bilateral Twitter undirected, Forest Fire and Twitter directed
- Queries synthesized using random walks to simulate user behavior
- Measures:
  - $F_P = \sum_{Q \in Q} |P| = |Q| \sum_{Q \in Q} |F_Q|$
  - $F_D = \sum_{Q \in Q} |P| = |Q| \sum_{Q \in Q} |F_D|$

Theorems
1. A small poly-logarithmic number of sketches is enough to get an almost constant approximation guarantee
2. Index time and update (insert, delete) time almost constant per word.
3. The search algorithm ranks the results correctly.
4. Query time almost constant per search result.

Table 3: Total query time (sec) over 20000 queries.