THE UNIVERSITY OF TEXAS AT AUSTIN

Using Cache Algorithms to Choose Shortcut Links

Justin Brickell

Inderjit S. Dhillon Dharmendra S. Modha

WebKDD 2006 Workshop on Knowledge Discovery on the Web, Aug. 20, 2006, at KDD 2006, Philadelphia, PA, USA

THE UNIVERSITY OF TEXAS AT AUSTIN

Using Cache Algorithms to Choose Shortcut Links (Outline)

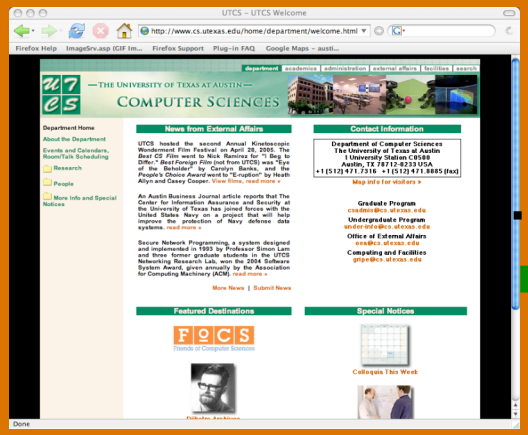
- Introduction
- A simple algorithm for choosing shortcuts
- Caching analogy
- Experimental Results
- Shortcuts on the front page
- Conclusions

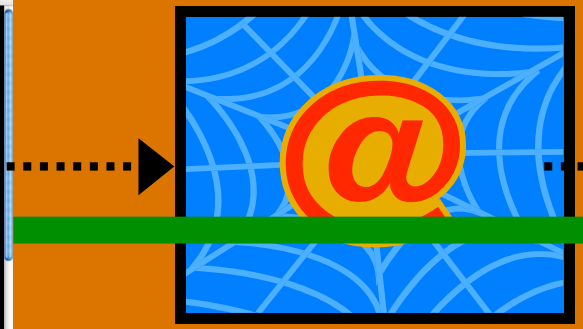
Motivation

- Visitors to websites do not always find what they need on the first page they load
- Navigational links move visitors from their current location to their desired destination
- These links are chosen manually by the author of each page
- Can we supplement these manually chosen links by adding dynamic links automatically?

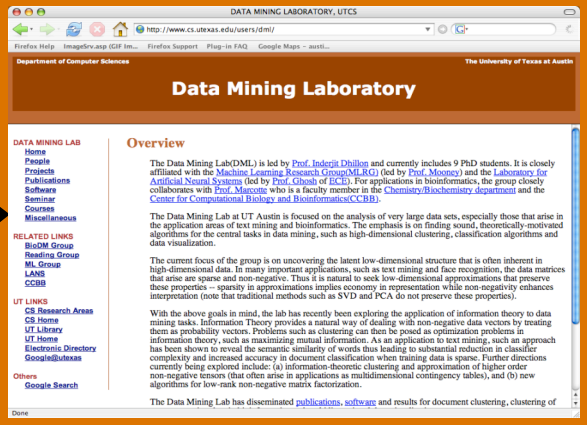
Shortcutting

Page *p*





Page *q*



- Add links based on recent access patterns

Selecting Shortcut Links

- Shortcuts on page *p* should point to pages *q* accessed *after p* within the same session
- Adding all such pages *q* is not a good solution
  - Users would be overwhelmed with thousands of links
  - Need to limit the number of shortcuts on each page
- What features characterize a good shortcut?
  - Recency
  - Frequency

A Naïve Shortcut Selection Algorithm

1. Initialize a 2-D array of counters, with one row and one column for each page.
   - A[i][j] is the number of times page *j* is accessed **after** page *i*
2. For each page *p* in each visit, find all pages *q* that occur **after** *p*. If edge *pq* is not a permanent webgraph edge, increment A[*p*][*q*]
3. For each page, add links to the *k* pages in its row with the highest counts

- This algorithm was suggested by Perkowitz in his PhD thesis
- Transformation is performed nightly and website is updated
- Uses $O(n^2)$ memory

Improving the Naïve Algorithm

- Problem: pages that are infrequently accessed may wind up with poorly-selected shortcuts, or no shortcuts
- Solution: rather than replace all shortcuts each day, replace individual shortcuts when a new shortcut is added
  - Choosing which shortcut to replace is analogous to the *cache-replacement problem*

The Cache Analogy

- Users sessions ⇔ Processes
- Web pages ⇔ Memory locations
- Shortcut destinations ⇔ Cache
- Shortcut quality ⇔ Hit ratio

A Cache-Based Shortcut Selection Algorithm

1. Initialize an array of caches of size *k*, with one cache for each page
2. For each page *p* in each visit, find all pages *q* that occur *after p*.
    1. If the edge *pq* is not a permanent webgraph edge, then register a hit for page *q* on the cache for page *p* (may involve replacement)
    2. Update the links on page *p* to reflect the new cache contents

- *Any* replacement policy will work
- Replacement policies retain pages most likely to be accessed in the future
- Uses O(*n*) memory

Improvement: Batched Caching

- Problem: Caching algorithms update cache on every miss
  - This is too frequent for shortcuts
- Solution: Delay updates
  - "Virtual" cache is updated normally
  - "Real" cache is copied from virtual cache periodically

Improvement: Shadow Caching

- Memory constraints are less restrictive than in a typical caching application
- Can make the virtual cache *larger* than the real cache
- When real cache is updated, populate it with the *k* "best" virtual cache items
- How do we choose the "best" items?
  - Simple: access count from prior time period
  - Better: linear combination of old score and access count from prior time period

Experiments

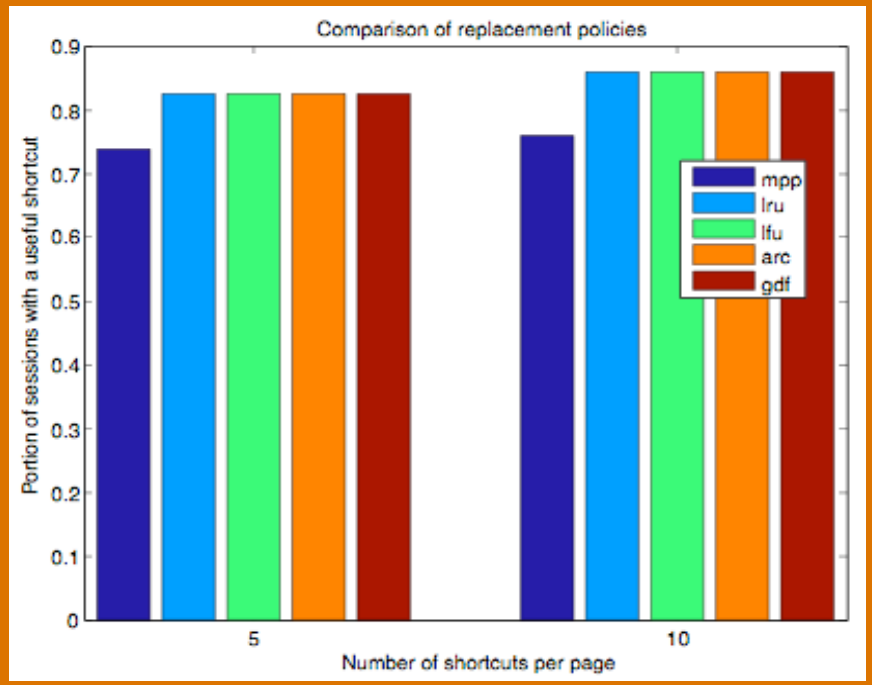
- UTCS access logs from Apr 17 - May 25
  - Robot accesses are removed
  - Long sessions with over 50 pages removed
  - Short sessions with under 3 pages removed
  - 89,000 sessions
  - 3.5 million edges in the sessions
    - Length *k* session has (*k* choose 2) edges
  - 336,000 distinct urls

Replacement Policies Tested

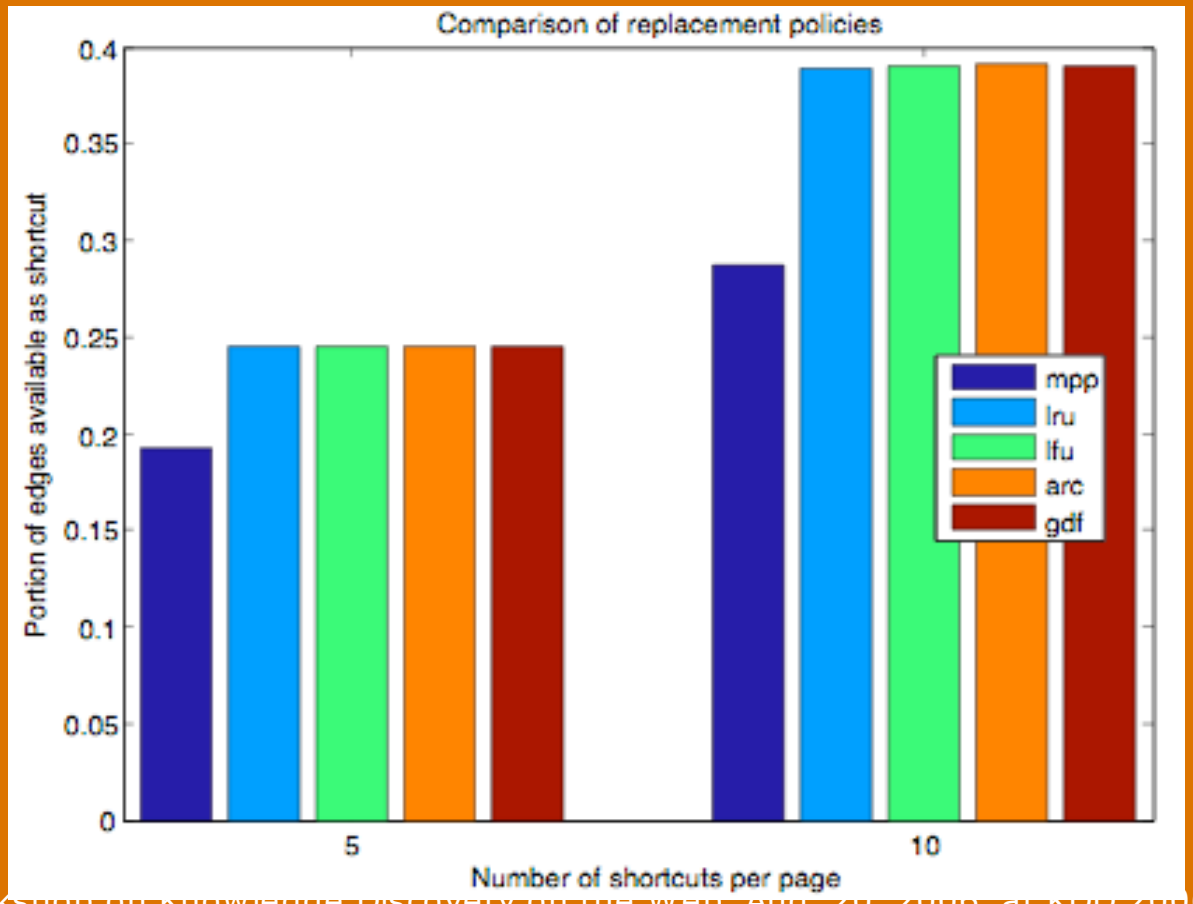
- LRU – Least Recently Used
- LFU – Least Frequently Used
- ARC – Adaptive Replacement Cache
  - Maintains two caches to balance between frequently used and recently used pages
- GDF – Greedy Dual Frequency
  - Like LFU, but with some recency information
- MPP – Most Popular Policy
  - This is the naïve popularity algorithm

Results: Most sessions benefit from shortcuts

- Caching selection outperforms naïve popularity selection



Results: Many edges traversed are available as shortcuts

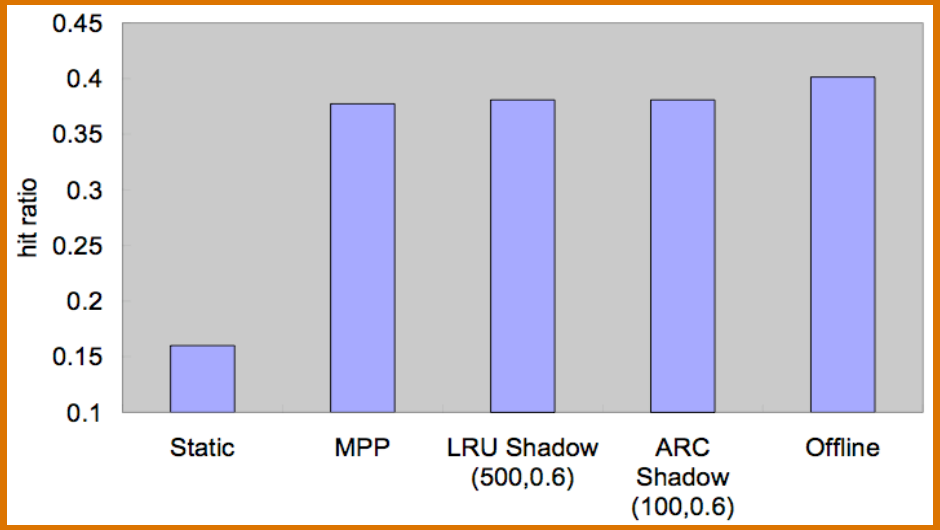


Shortcuts on the Front Page

- The front page serves as a portal
  - Users who load the front page may be interested in *any* content on the site
- Ignore sessions, build shortcuts from **all** pages that are accessed
- Rate success by portion of pages accessed that were shortcut linked on front page

Example of Front Page Shortcuts

Front Page Results



- "Static" refers to the original UTCS front page content
- Naïve mpp performs well, since the top pages receive many hits during each time period
  - Still requires $O(n^2)$ memory
- "Offline" chooses the best possible shortcuts with knowledge of the future

Conclusions

- Shortcutting is a simple, effective way of helping site visitors find the information they need
- Adding only a few links provides connections to almost every page a visitor would want to visit
- Our algorithms are memory efficient and outperform the basic popularity algorithm

Future Work

- How quickly can users get to their intended destination?
  - This assumes that there is a single intended destination, and that we can identify it
- How often are shortcut links actually used?
  - Deployment, and user study

Questions?