Efficient Crawling Through URL Ordering

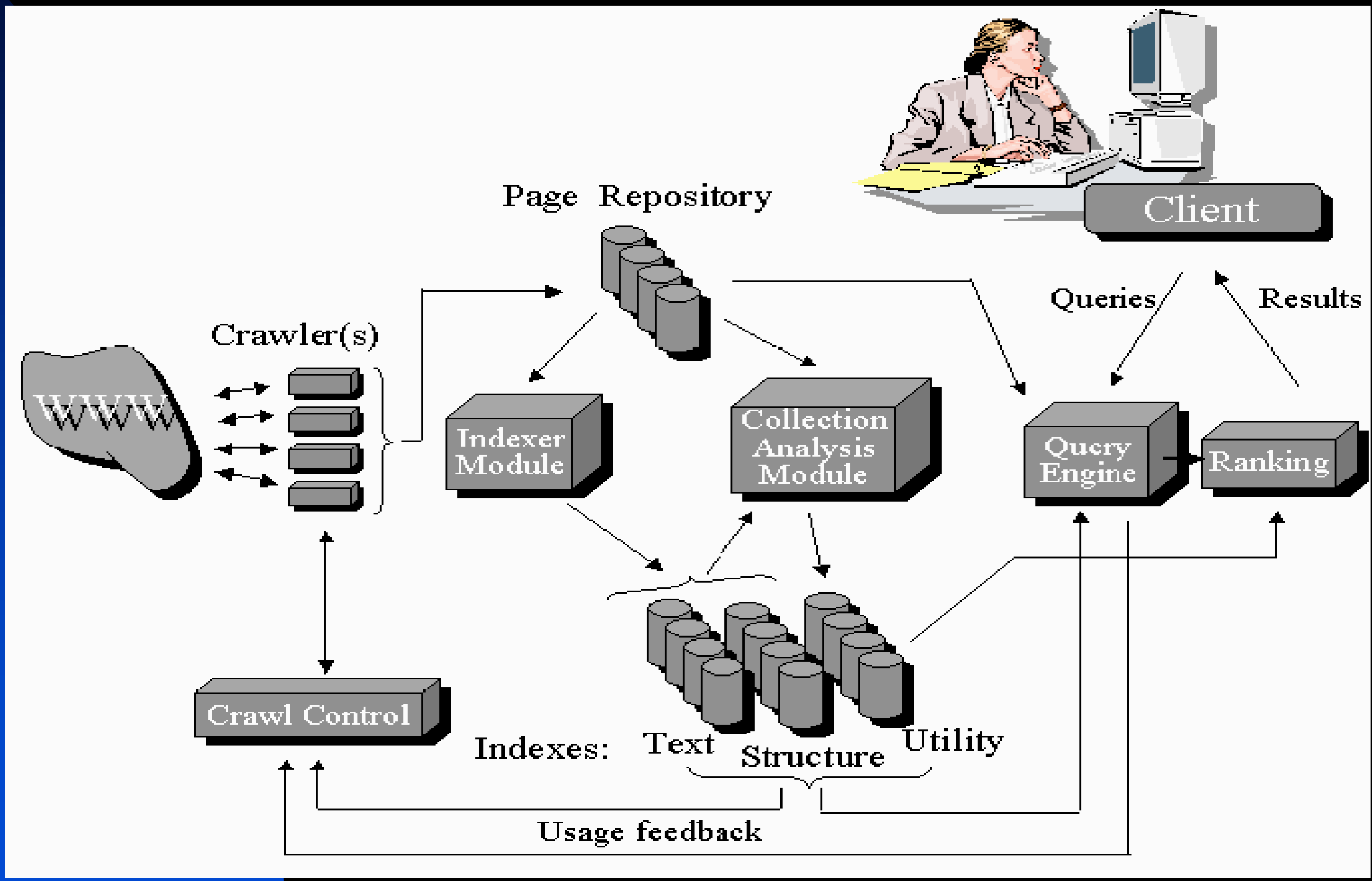
Junghoo Cho Hector Gracia-Molina Lawrence Page

Presented By: Issam Al-Azzoni

Outline

- Crawlers and Search Engines
- “Importance” Metrics
- Crawler Performance
- Crawler Models
- Experiments and Observations
- Conclusion
- Discussion

Search Engine Architecture



Crawling Web Pages

- What pages should the crawler download?
- How should the crawler refresh pages?
- How should the load on the visited Web sites be minimized?
- How should the crawling process be parallelized?

What Pages Should the Crawler Visit First?

- "important" pages
- Different importance metrics:
  - Similarity to a Driving query
  - Backlink Count
  - PageRank
  - Location
  - Forward Link Count

The Similarity Metric

- Given a query *q*, the importance of a page *p* is defined to be the "textual similarity" between *p* and *q* [Arasu et al.]

$$I(p) = IS(p, q)$$

The Backlink Count Metric

- The importance of a page *p* is defined to be the number of links to *p* that appear over the entire Web

$$I(p) = IB(p)$$

- $IB'(p)$ denotes the estimated value of $IB(p)$

The PageRank Metric

- The importance of a page *p is defined to be the weighted sum of the importance of the pages that have backlinks to p*

$$I(p) = IR(p)$$

- *IR'(p)* denotes the estimated value of IR(p)

$$IR(p) = (1-d) + d\,[\,IR(t_1)/c_1 + \ldots + IR(t_n)/c_n\,]$$

Other Importance Metrics

- Forward Link Count
- Location Metric
- Combinations of metrics

$$I(p) = k_1 . IS(p, q) + k_2 . IB(p)$$

The Question is

How to design a crawler that if possible visits high *I(p) pages before lower ranked ones, for some definition of I(p)*

- *Crawler performance*
- *Crawler models*
  - *Crawl and Stop*
  - *Crawl and Stop with Threshold*
  - *Limited Buffer Crawl*

Crawl and Stop

- The crawler starts at an initial page and stops after visiting *K* pages

$$\underbrace{r_1, \ldots, r_M}_{I(r_i) \geq I(rk)}, \ldots, r_K$$

$P_{ST}(C) = M / K$

$P_{ST}(\text{ideal crawler}) = 1$

$P_{ST}(\text{random crawler}) = K / T$, where T is the total number of pages in the Web

Crawl and Stop with Threshold

- The crawler starts at an initial page and stops after visiting *K* pages
- Any page *p* with $I(p) \geq G$ (where *G* is a given importance target) is considered hot
- $P_{ST}(C) = \dfrac{\text{Total number of visited hot pages}}{\text{Total number of hot pages } (H)}$

$$P_{ST}(\textit{ideal crawler}) = \begin{cases} K/H, & \text{if } K < H \\ 1 & \text{otherwise} \end{cases}$$

$$P_{ST}(\textit{random crawler}) = \begin{cases} K/T, & \text{if } K < T \\ 1 & \text{otherwise} \end{cases}$$

Limited Buffer Crawl

- The crawler can keep at most *B* pages in its buffer

$$P_{BC} = \frac{\text{Total number of hot pages in the buffer}}{B}$$

Ordering Metrics

- A crawler keeps a queue of URLs it has seen during a crawl
- The crawler must select from this queue the next URL to visit
- An ordering metric O is used for this selection
- The chosen O metric should be suitable for the importance metric in mind

Experiments

- Define the entire Web to be a portion of the Stanford University Web pages (179,000 pages)
- Measure the performance of various ordering metrics for the importance metric *IB(p)*
- Measure the performance of various ordering metrics for the importance metric *IS(p, Q)*

Backlink-based Crawlers

Algorithm 5.1 Crawling algorithm (backlink based)

Input: `starting_url`: seed URL

Procedure:

```
enqueue(url_queue, starting_url)
while (not empty(url_queue))
   url = dequeue(url_queue)
   page = crawl_page(url)
   enqueue(crawled_pages, (url, page))
   url_list = extract_urls(page)
   foreach u in url_list
      enqueue(links, (url, u))
      if (u∉url_queue and (u,-)∉crawled_pages)
         enqueue(url_queue, u)
   reorder_queue(url_queue)
```

Function description:

- `enqueue(queue, element)`: append `element` at the end of `queue`
- `dequeue(queue)`: remove the element at the beginning of `queue` and return it
- `reorder_queue(queue)`: reorder `queue` using information in `links` (refer to Figure 2)

Backlink-based Crawlers

(1) breadth first

```
do nothing (null operation)
```

(2) backlink count, $IB'(p)$

```
foreach u in url_queue
  backlink_count[u] = number of terms (-,u) in links
sort url_queue by backlink_count[u]
```

(3) PageRank $IR'(p)$

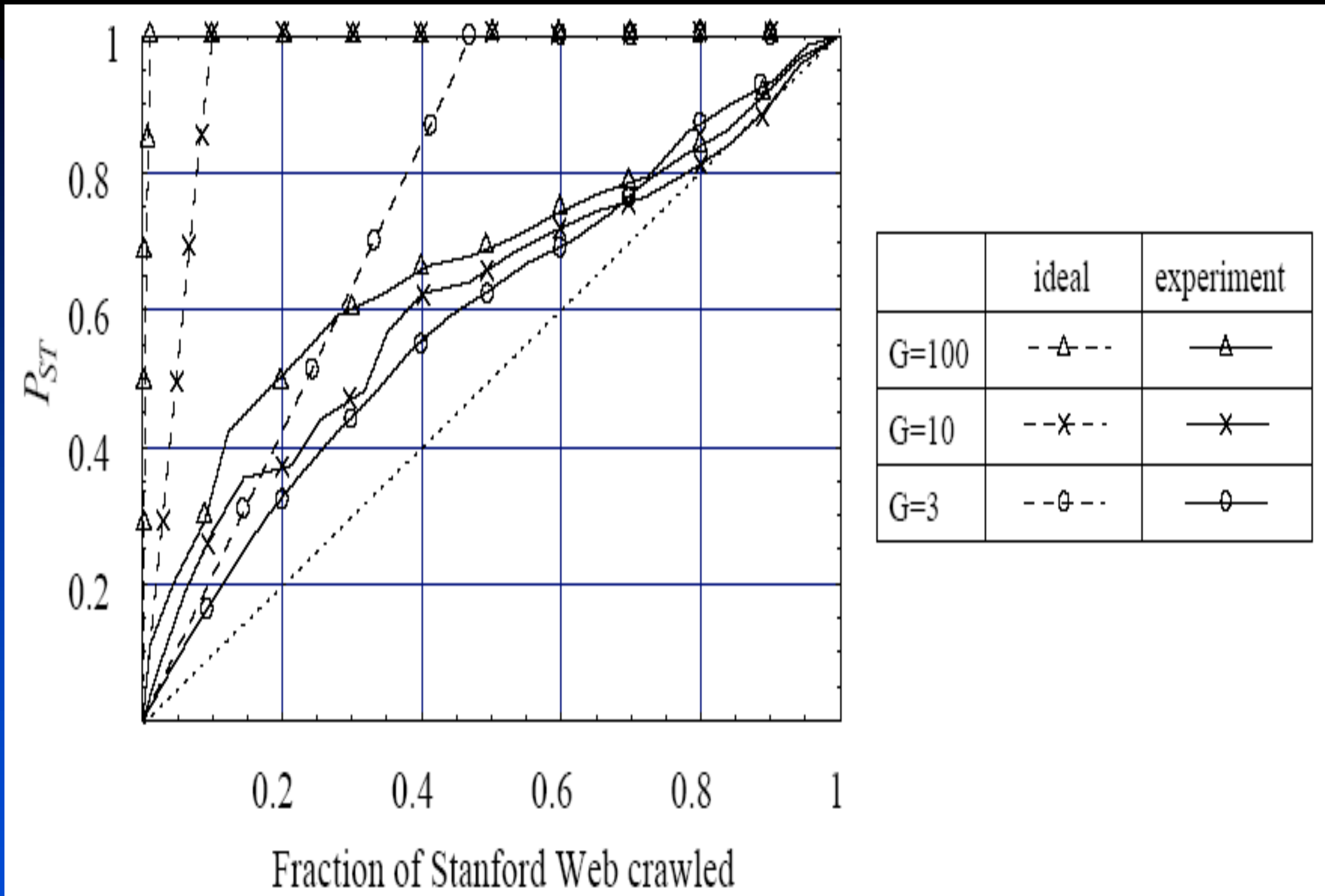
solve the following set of equations:

$IR[u] = (1 - 0.9) + 0.9 \sum_i \frac{IR[v_i]}{c_i}$, where

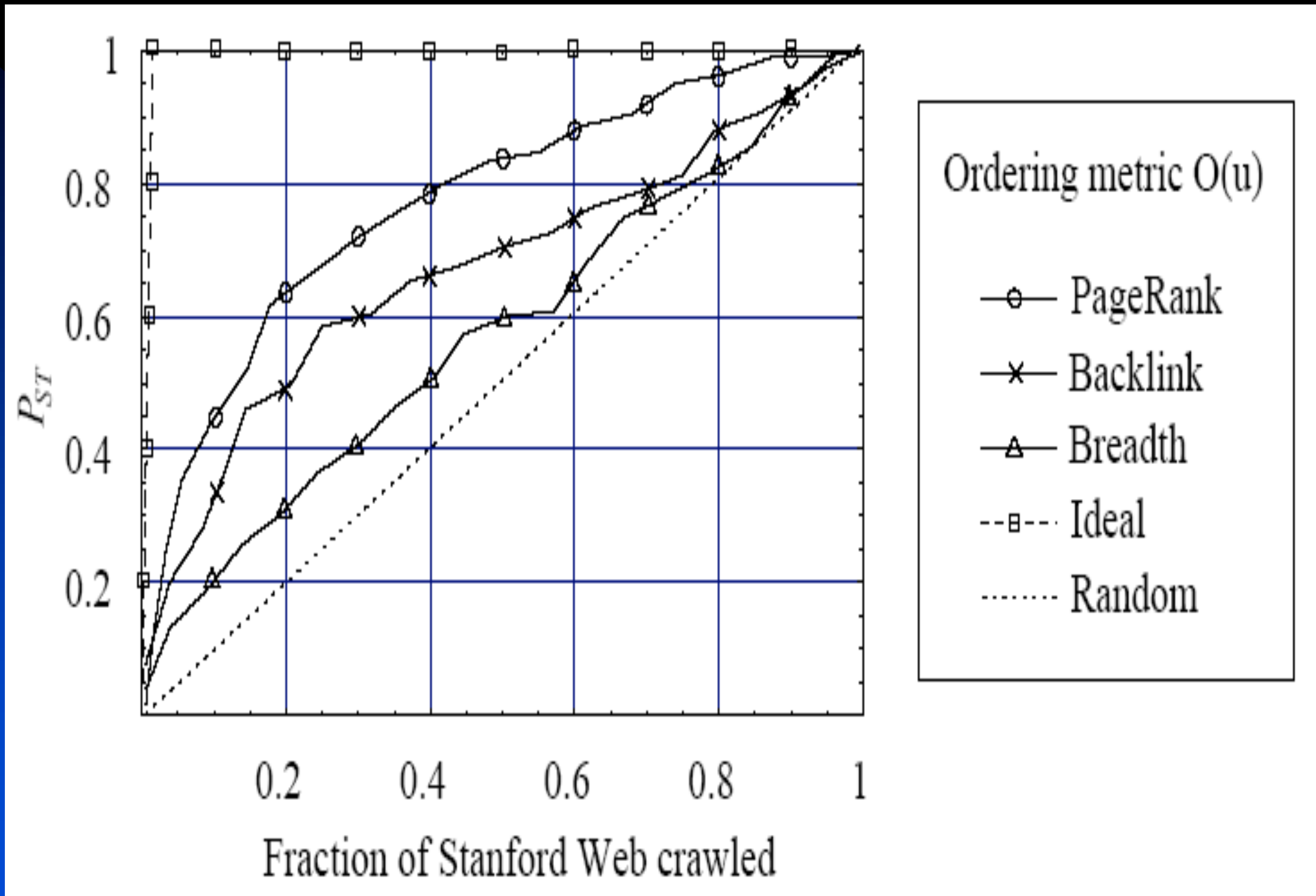
$(v_i, u) \in$ links and c_i is the number of links in the page v_i

sort url_queue by $IR(u)$

Backlink-based Crawlers



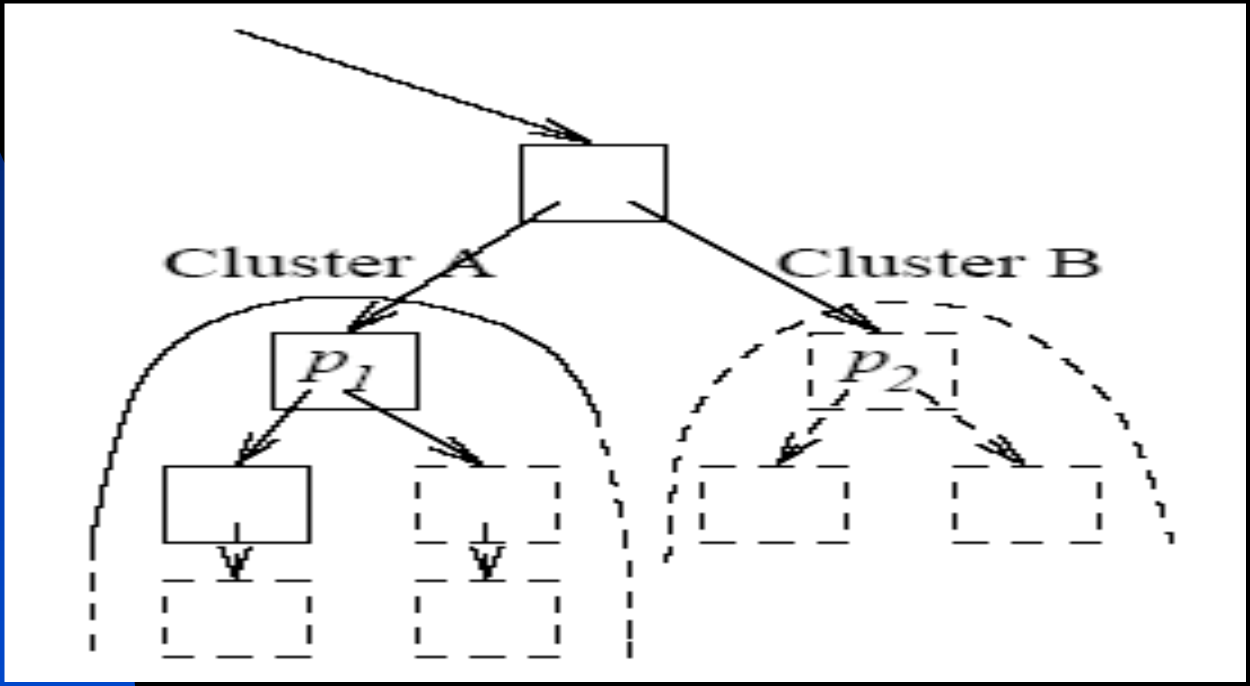
Backlink-based Crawlers



Observations

- *Using an ordering metric outperforms random (or breadth) ordering*
- *The ordering metric IR'(p)* outperforms the IB'(p) one, even when the importance metric is IB(p)

IR'(p) vs IB'(p)



Similarity-based Crawlers

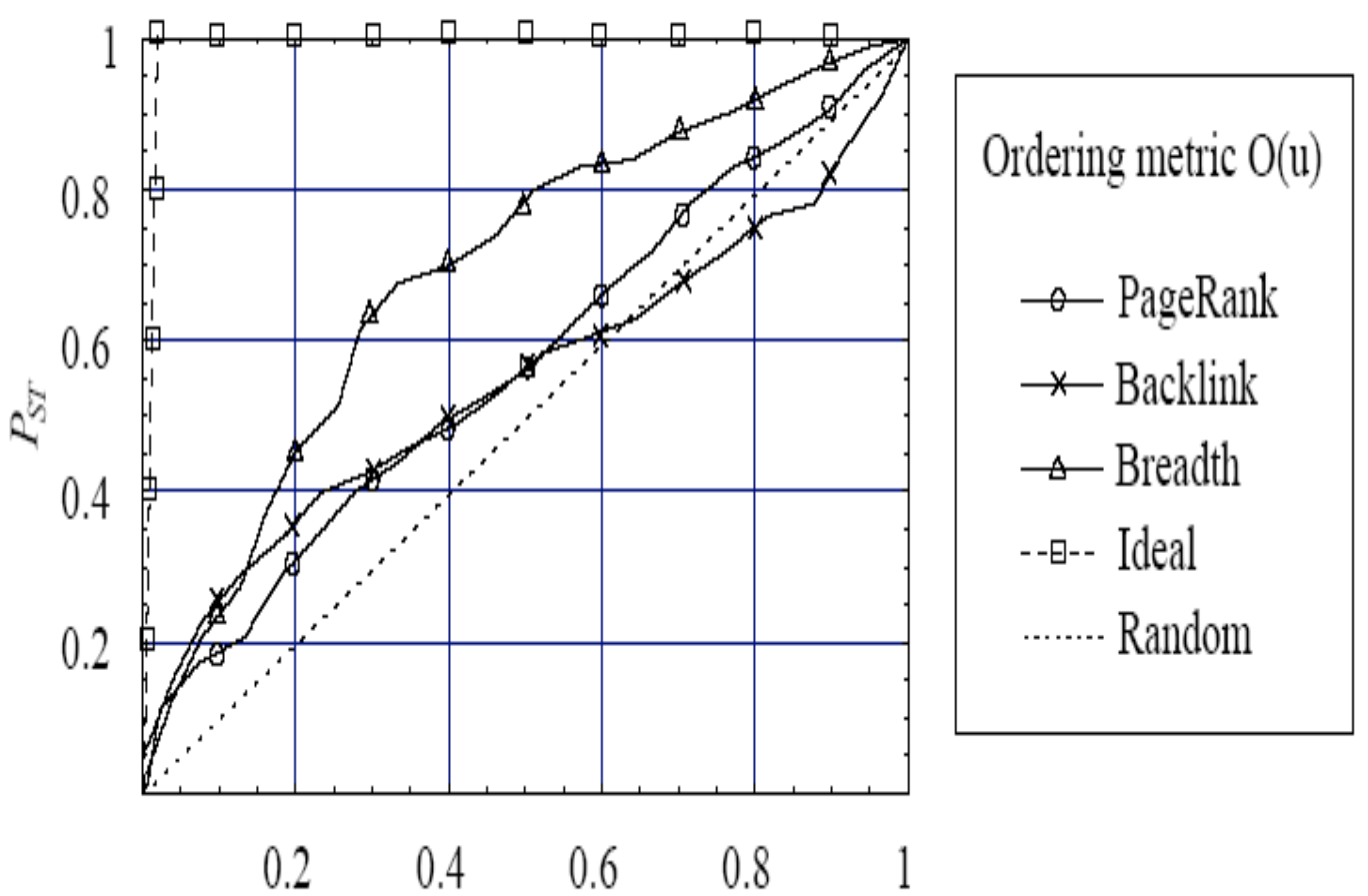
Algorithm 5.2 Crawling algorithm (modified similarity based)
Input: starting_url: seed URL
Procedure:

```
enqueue(url_queue, starting_url)
while (not empty(hot_queue) and not empty(url_queue))
   url = dequeue2(hot_queue, url_queue)
   page = crawl_page(url)
   enqueue(crawled_pages, (url, page))
   url_list = extract_urls(page)
   foreach u in url_list
      enqueue(links, (url, u))
      if (u∉url_queue and u∉hot_queue and (u,-)∉crawled_pages)
         if (u contains computer in anchor or url)
            enqueue(hot_queue, u)
         else
            enqueue(url_queue, u)
   reorder_queue(url_queue)
   reorder_queue(hot_queue)
```

Function description:

```
dequeue2(queue1, queue2): if (not empty(queue1)) dequeue(queue1)
                          else dequeue(queue2)
```

Similarity-based Crawlers



Similarity-based Crawlers

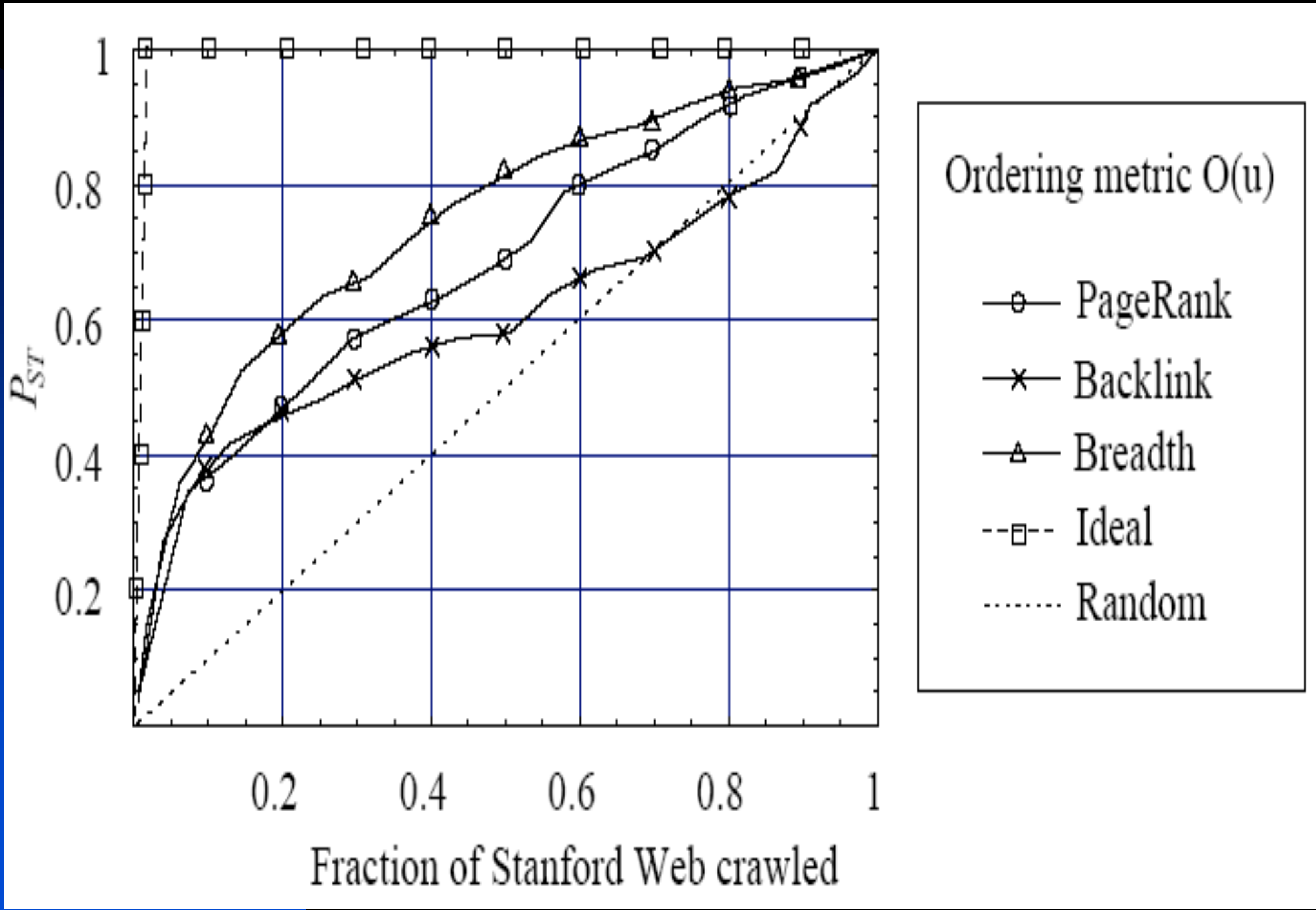
Algorithm 5.3 Crawling algorithm (similarity based)

Input: starting_url: seed URL

Procedure:

```
enqueue(url_queue, starting_url)
while (not empty(hot_queue) and not empty(url_queue))
  url = dequeue2(hot_queue, url_queue)
  page = crawl_page(url)
  if (page contains 10 or more computer in body
        or one computer in title)
    hot[url] = True
  enqueue(crawled_pages, (url, page))
  url_list = extract_urls(page)
  foreach u in url_list
    enqueue(links, (url, u))
    if (u∉url_queue and u∉hot_queue and (u,-)∉crawled_pages)
      if (u contains computer in anchor or url)
        enqueue(hot_queue, u)
      else if (distance_from_hotpage(u) < 3)
        enqueue(hot_queue, u)
      else
        enqueue(url_queue, u)
  reorder_queue(url_queue)
  reorder_queue(hot_queue)
```

Similarity-based Crawlers



Observations

- *When similarity is important, it is effective to use an ordering metric that considers 1) the contents of anchors and URLs and 2) the distance to the hot pages that have been discovered*

Conclusion

- With a good ordering strategy, we can build crawlers that can obtain a significant portion of the hot pages relatively early

Limitations

- Experiments run only over a portion of the Stanford pages
- Is the definition of crawler performance adequate?
- What about other ordering metrics or combinations of metrics
- What about other types of crawlers e.g. focused crawlers

References

- A. Arasu, J. Cho, H. Garcia-Molina, A. Paepcke, and S. Raghavan, Searching the Web, *ACM Trans. Internet Tech.*, 1(1), 2001.
- Yuan Wang, David J. DeWitt: Computing PageRank in a Distributed Internet Search Engine System. VLDB 2004: 420-431
- Marc Najork and Janet L. Wiener. Breadth-first search crawling yields high-quality pages. In *Proc. 10th World Wide Web Conference*, pages 114–118, 2001.
- Junghoo Cho. Crawling the Web: Discovery and Maintenance of a Large-Scale Web Data. Ph.D. thesis, Stanford University.