

Introduction to Information Retrieval

<http://informationretrieval.org>

IIR 20: Crawling

Hinrich Schütze

Institute for Natural Language Processing, Universität Stuttgart

2012-01-11

Outline

- 1 Recap
- 2 A simple crawler
- 3 A real crawler

Search engines rank content pages *and* ads

Web Images Maps News Shopping Gmail more

Sign in



discount broker

Search

Advanced Search
Preferences

Web Results 1 - 10 of about 807,000 for discount broker [definition]. (0.12 seconds)

Discount Broker Reviews

Information on online **discount brokers** emphasizing rates, charges, and customer comments and complaints.

www.broker-reviews.us/ - 94k - [Cached](#) - [Similar pages](#)

Discount Broker Rankings (2008 Broker Survey) at SmartMoney.com

Discount Brokers. Rank/ **Brokerage**/ Minimum to Open Account, Comments, Standard Commis- sion*, Reduced Commission, Account Fee Per Year (How to Avoid), Avg. ...

www.smartmoney.com/brokers/index.cfm?story=2004-discount-table - 121k - [Cached](#) - [Similar pages](#)

Stock Brokers | Discount Brokers | Online Brokers

Most Recommended. Top 5 **Brokers** headlines. 10. Don't Pay Your **Broker** for Free Funds May 15 at 3:39 PM. 5. Don't **Discount** the Discounters Apr 18 at 2:41 PM ...

www.fool.com/investing/brokers/index.aspx - 44k - [Cached](#) - [Similar pages](#)

Discount Broker

Discount Broker - Definition of **Discount Broker** on Investopedia - A stockbroker who carries out buy and sell orders at a reduced commission compared to a ...

www.investopedia.com/terms/d/discountbroker.asp - 31k - [Cached](#) - [Similar pages](#)

Discount Brokerage and Online Trading for Smart Stock Market ...

Online stock **broker** SogoTrade offers the best in **discount brokerage** investing. Get stock market quotes from this internet stock trading company.

www.sogotrade.com/ - 39k - [Cached](#) - [Similar pages](#)

15 questions to ask discount brokers - MSN Money

Jan 11, 2004 ... If you're not big on hand-holding when it comes to investing, a **discount broker** can be an economical way to go. Just be sure to ask these ...

moneycentral.msn.com/content/Investing/StartInvesting/P66171.asp - 34k - [Cached](#) - [Similar pages](#)

Sponsored Links

Rated #1 Online Broker

No Minimums. No Inactivity Fee
Transfer to Firsttrade for Free!
www.firsttrade.com

Discount Broker

Commission free trades for 30 days.
No maintenance fees. Sign up now.
TDAMERITRADE.com

TradeKing - Online Broker

\$4.95 per Trade, Market or Limit
SmartMoney Top **Discount Broker** 2007
www.TradeKing.com

Scottrade Brokerage

\$7 Trades, No Share Limit. In-Depth
Research. Start Trading Online Now!
www.Scottrade.com

Stock trades \$1.50 - \$3

100 free trades, up to \$100 back
for transfer costs, \$500 minimum
www.sogotrade.com

\$3.95 Online Stock Trades

Market/Limit Orders, No Share Limit
and No Inactivity Fees
www.Marsco.com

INGDIRECT | ShareBuilder

Google's second price auction

| advertiser | bid | CTR | ad rank | rank | paid |
|------------|--------|------|---------|------|-----------|
| A | \$4.00 | 0.01 | 0.04 | 4 | (minimum) |
| B | \$3.00 | 0.03 | 0.09 | 2 | \$2.68 |
| C | \$2.00 | 0.06 | 0.12 | 1 | \$1.51 |
| D | \$1.00 | 0.08 | 0.08 | 3 | \$0.51 |

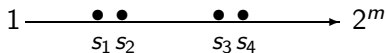
- **bid**: maximum bid for a click by advertiser
- **CTR**: click-through rate: when an ad is displayed, what percentage of time do users click on it? **CTR is a measure of relevance.**
- **ad rank**: $\text{bid} \times \text{CTR}$: this trades off (i) how much money the advertiser is willing to pay against (ii) how relevant the ad is
- **paid**: Second price auction: **The advertiser pays the minimum amount necessary to maintain their position in the auction (plus 1 cent).**

What's great about search ads

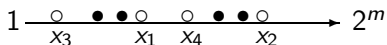
- Users only click if they are interested.
- The advertiser only pays when a user clicks on an ad.
- Searching for something indicates that you are more likely to buy it ...
- ...in contrast to radio and newspaper ads.

Near duplicate detection: Minimum of permutation

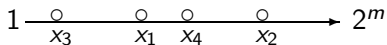
document 1: $\{s_k\}$



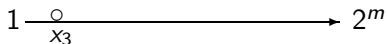
$$x_k = \pi(s_k)$$



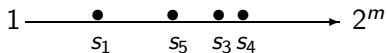
x_k



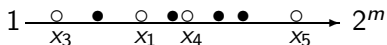
$$\min_{s_k} \pi(s_k)$$



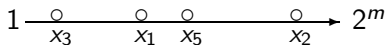
document 2: $\{s_k\}$



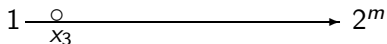
$$x_k = \pi(s_k)$$



x_k



$$\min_{s_k} \pi(s_k)$$



Roughly: We use $\min_{s \in d_1} \pi(s) = \min_{s \in d_2} \pi(s)$ as a test for: are d_1 and d_2 near-duplicates?

Example

$$d_1 \quad d_2$$

$$s_1 \quad 1 \quad 0$$

$$s_2 \quad 0 \quad 1$$

$$s_3 \quad 1 \quad 1$$

$$s_4 \quad 1 \quad 0$$

$$s_5 \quad 0 \quad 1$$

$$h(x) = x \bmod 5$$

$$g(x) = (2x + 1) \bmod 5$$

$$\min(h(d_1)) = 1 \neq 0 =$$

$$\min(h(d_2))$$

$$\min(g(d_1)) = 2 \neq 0 =$$

$$\min(g(d_2))$$

$$\hat{J}(d_1, d_2) = \frac{0+0}{2} = 0$$

| | d_1 slot | d_2 slot |
|------------|------------|------------|
| | ∞ | ∞ |
| | ∞ | ∞ |
| $h(1) = 1$ | 1 1 | - ∞ |
| $g(1) = 3$ | 3 3 | - ∞ |
| $h(2) = 2$ | - 1 | 2 2 |
| $g(2) = 0$ | - 3 | 0 0 |
| $h(3) = 3$ | 3 1 | 3 2 |
| $g(3) = 2$ | 2 2 | 2 0 |
| $h(4) = 4$ | 4 1 | - 2 |
| $g(4) = 4$ | 4 2 | - 0 |
| $h(5) = 0$ | - 1 | 0 0 |
| $g(5) = 1$ | - 2 | 1 0 |

final sketches

Outline

- 1 Recap
- 2 A simple crawler
- 3 A real crawler

How hard can crawling be?

- Web search engines must crawl their documents.

How hard can crawling be?

- Web **search engines must crawl** their documents.
- Getting the content of the documents is easier for many other IR systems.

How hard can crawling be?

- Web [search engines must crawl](#) their documents.
- Getting the content of the documents is easier for many other IR systems.
 - E.g., indexing all files on your hard disk: just do a recursive descent on your file system

How hard can crawling be?

- Web **search engines must crawl** their documents.
- Getting the content of the documents is easier for many other IR systems.
 - E.g., indexing all files on your hard disk: just do a recursive descent on your file system
- Ok: for web IR, getting the content of the documents takes longer ...

How hard can crawling be?

- Web **search engines must crawl** their documents.
- Getting the content of the documents is easier for many other IR systems.
 - E.g., indexing all files on your hard disk: just do a recursive descent on your file system
- Ok: for web IR, getting the content of the documents takes longer ...
- ... because of latency.

How hard can crawling be?

- Web **search engines must crawl** their documents.
- Getting the content of the documents is easier for many other IR systems.
 - E.g., indexing all files on your hard disk: just do a recursive descent on your file system
- Ok: for web IR, getting the content of the documents takes longer ...
- ... because of latency.
- But is that really a design/systems challenge?

Basic crawler operation

- Initialize queue with URLs of known seed pages

Basic crawler operation

- Initialize queue with URLs of known seed pages
- Repeat

Basic crawler operation

- Initialize queue with URLs of known seed pages
- Repeat
 - Take URL from queue

Basic crawler operation

- Initialize queue with URLs of known seed pages
- Repeat
 - Take URL from queue
 - Fetch and parse page

Basic crawler operation

- Initialize queue with URLs of known seed pages
- Repeat
 - Take URL from queue
 - Fetch and parse page
 - Extract URLs from page

Basic crawler operation

- Initialize queue with URLs of known seed pages
- Repeat
 - Take URL from queue
 - Fetch and parse page
 - Extract URLs from page
 - Add URLs to queue

Basic crawler operation

- Initialize queue with URLs of known seed pages
- Repeat
 - Take URL from queue
 - Fetch and parse page
 - Extract URLs from page
 - Add URLs to queue
- Fundamental assumption: The web is well linked.

Exercise: What's wrong with this crawler?

```
urlqueue := (some carefully selected set of seed urls)
while urlqueue is not empty:
    myurl := urlqueue.getlastanddelete()
    mypage := myurl.fetch()
    fetchedurls.add(myurl)
    newurls := mypage.extracturls()
    for myurl in newurls:
        if myurl not in fetchedurls and not in urlqueue:
            urlqueue.add(myurl)
    addtoinvertedindex(mypage)
```

What's wrong with the simple crawler

- Scale: we need to **distribute**.
- We can't index everything: we need to **subselect**. How?
- Duplicates: need to integrate **duplicate detection**
- Spam and spider traps: need to integrate **spam detection**
- **Politeness**: we need to be “nice” and space out all requests for a site over a longer period (hours, days)
- **Freshness**: we need to recrawl periodically.
 - Because of the size of the web, we can do frequent recrawls only for a small subset.
 - Again, subselection problem or **prioritization**

Magnitude of the crawling problem

- To fetch 20,000,000,000 pages in one month . . .

Magnitude of the crawling problem

- To fetch 20,000,000,000 pages in one month ...
- ...we need to fetch almost 8000 pages per second!

Magnitude of the crawling problem

- To fetch 20,000,000,000 pages in one month . . .
- . . . we need to fetch almost 8000 pages per second!
- Actually: many more since many of the pages we attempt to crawl will be duplicates, unfetchable, spam etc.

What a crawler must do

Be polite

What a crawler must do

Be polite

- Don't hit a site too often

What a crawler must do

Be polite

- Don't hit a site too often
- Only crawl pages you are allowed to crawl: robots.txt

What a crawler must do

Be polite

- Don't hit a site too often
- Only crawl pages you are allowed to crawl: robots.txt

Be robust

What a crawler must do

Be polite

- Don't hit a site too often
- Only crawl pages you are allowed to crawl: robots.txt

Be robust

- Be immune to spider traps, duplicates, very large pages, very large websites, dynamic pages etc

Robots.txt

- Protocol for giving crawlers (“robots”) limited access to a website, originally from 1994

Robots.txt

- Protocol for giving crawlers (“robots”) limited access to a website, originally from 1994
- Examples:

Robots.txt

- Protocol for giving crawlers (“robots”) limited access to a website, originally from 1994
- Examples:
 - User-agent: *
 - Disallow: /yoursite/temp/

Robots.txt

- Protocol for giving crawlers (“robots”) limited access to a website, originally from 1994
- Examples:
 - User-agent: *
Disallow: /yoursite/temp/
 - User-agent: searchengine
Disallow: /

Robots.txt

- Protocol for giving crawlers (“robots”) limited access to a website, originally from 1994
- Examples:
 - User-agent: *
Disallow: /yoursite/temp/
 - User-agent: searchengine
Disallow: /
- Important: cache the robots.txt file of each site we are crawling

Example of a robots.txt (nih.gov)

```
User-agent: PicoSearch/1.0
Disallow: /news/information/knight/
Disallow: /nidcd/
...
Disallow: /news/research_matters/secure/
Disallow: /od/ocpl/wag/

User-agent: *
Disallow: /news/information/knight/
Disallow: /nidcd/
...
Disallow: /news/research_matters/secure/
Disallow: /od/ocpl/wag/
Disallow: /ddir/
Disallow: /sdminutes/
```

What any crawler should do

- Be capable of **distributed** operation

What any crawler should do

- Be capable of **distributed** operation
- Be scalable: need to be able to increase crawl rate by adding more machines

What any crawler should do

- Be capable of **distributed** operation
- Be scalable: need to be able to increase crawl rate by adding more machines
- Fetch pages of higher quality first

What any crawler should do

- Be capable of **distributed** operation
- Be scalable: need to be able to increase crawl rate by adding more machines
- Fetch pages of higher quality first
- Continuous operation: get fresh version of already crawled pages

Outline

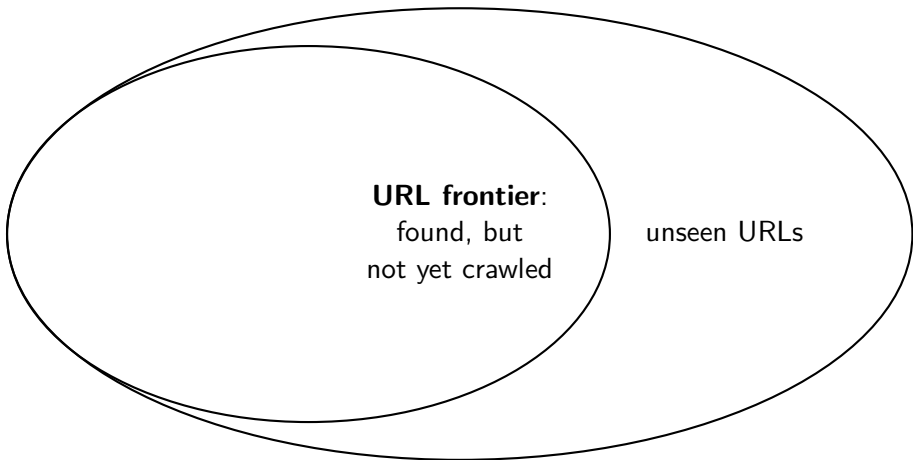
- 1 Recap
- 2 A simple crawler
- 3 A real crawler

URL frontier

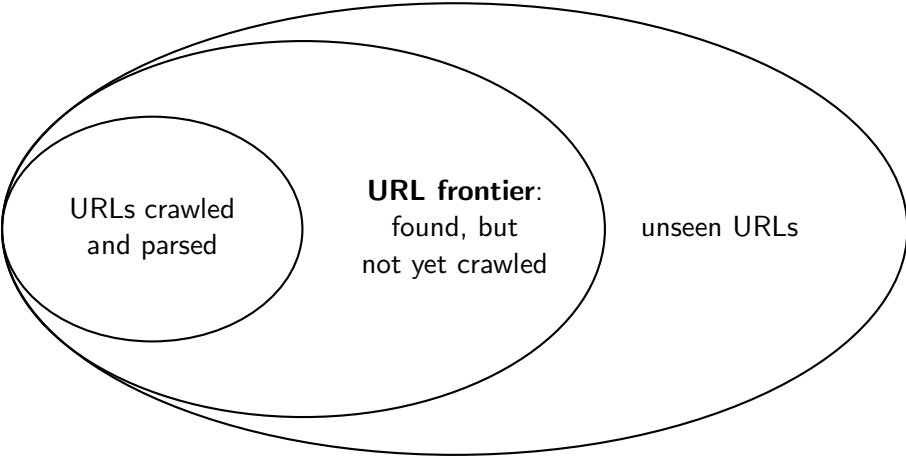


unseen URLs

URL frontier



URL frontier



A Venn diagram illustrating the states of a crawler's URL frontier. It consists of three nested ellipses. The innermost ellipse on the left is labeled 'URLs crawled and parsed'. The middle ellipse, which overlaps with the first, is labeled 'URL frontier: found, but not yet crawled'. The outermost ellipse, which overlaps with both the first and second, is labeled 'unseen URLs'.

URLs crawled
and parsed

URL frontier:
found, but
not yet crawled

unseen URLs

URL frontier

- The URL frontier is the data structure that holds and manages URLs we've seen, but that have not been crawled yet.

URL frontier

- The URL frontier is the data structure that holds and manages URLs we've seen, but that have not been crawled yet.
- Can include multiple pages from the same host

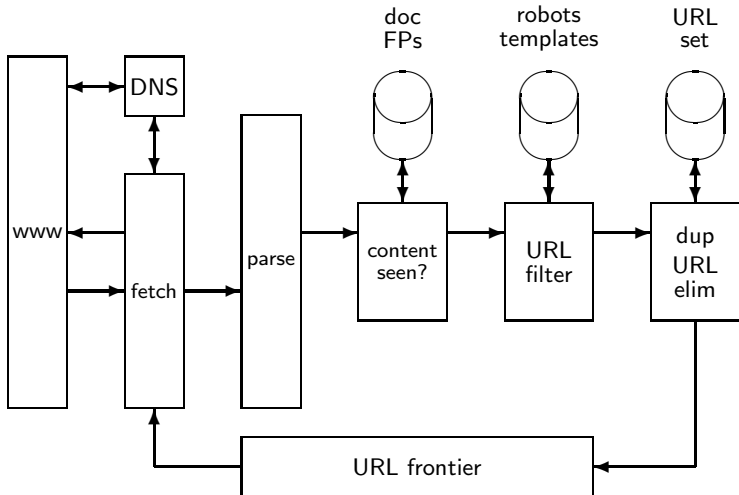
URL frontier

- The URL frontier is the data structure that holds and manages URLs we've seen, but that have not been crawled yet.
- Can include multiple pages from the same host
- Must avoid trying to fetch them all at the same time

URL frontier

- The URL frontier is the data structure that holds and manages URLs we've seen, but that have not been crawled yet.
- Can include multiple pages from the same host
- Must avoid trying to fetch them all at the same time
- Must keep all crawling threads busy

Basic crawl architecture



URL normalization

- Some URLs extracted from a document are **relative** URLs.
- E.g., at `http://mit.edu`, we may have `abouthe.html`
 - This is the same as: `http://mit.edu/abouthe.html`
- During parsing, we must normalize (expand) all relative URLs.

Content seen

- For each page fetched: check if the content is already in the index
- Check this using document fingerprints or [shingles](#)
- Skip documents whose content has already been indexed

Distributing the crawler

- Run multiple crawl threads, potentially at different nodes

Distributing the crawler

- Run multiple crawl threads, potentially at different nodes
 - Usually geographically distributed nodes

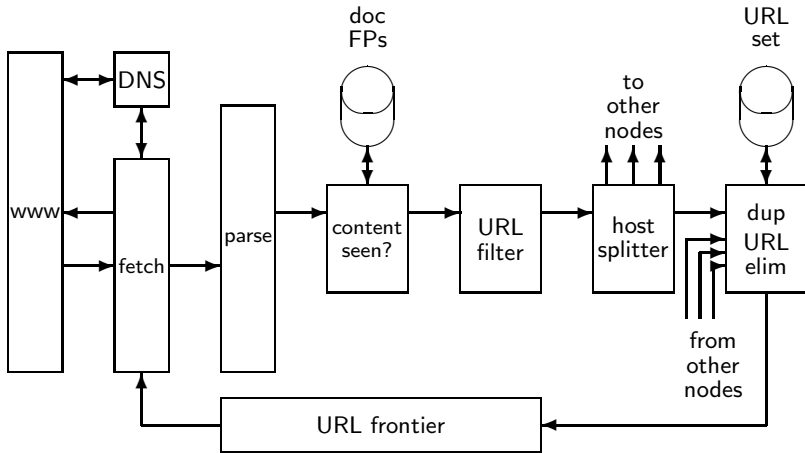
Distributing the crawler

- Run multiple crawl threads, potentially at different nodes
 - Usually geographically distributed nodes
- Partition hosts being crawled into nodes

Google data centers (wayfaring.com)



Distributed crawler



URL frontier: Two main considerations

- Politeness: Don't hit a web server too frequently

URL frontier: Two main considerations

- Politeness: Don't hit a web server too frequently
 - E.g., insert a time gap between successive requests to the same server

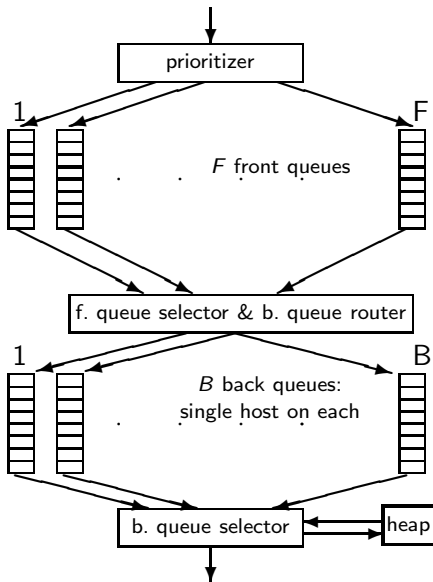
URL frontier: Two main considerations

- Politeness: Don't hit a web server too frequently
 - E.g., insert a time gap between successive requests to the same server
- Freshness: Crawl some pages (e.g., news sites) more often than others

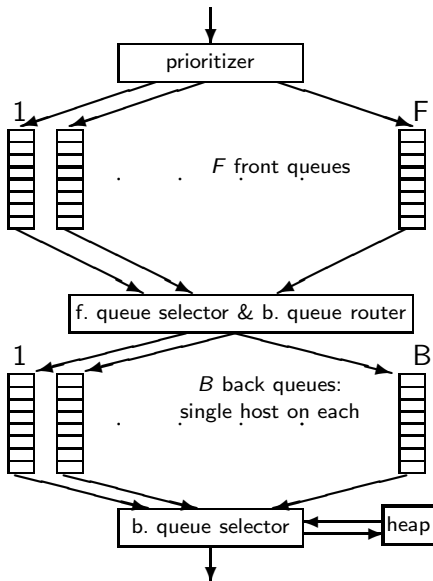
URL frontier: Two main considerations

- Politeness: Don't hit a web server too frequently
 - E.g., insert a time gap between successive requests to the same server
- Freshness: Crawl some pages (e.g., news sites) more often than others
- Not an easy problem: simple priority queue fails.

Mercator URL frontier

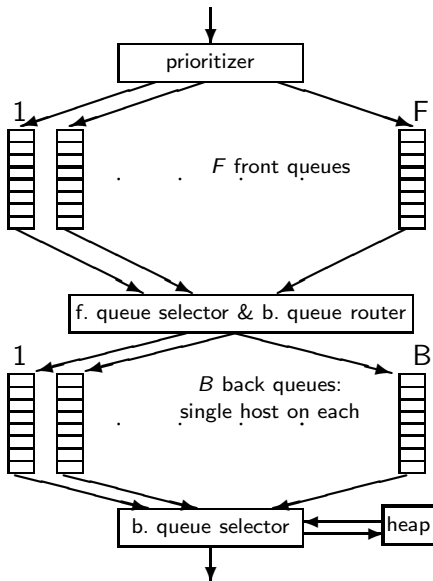


Mercator URL frontier



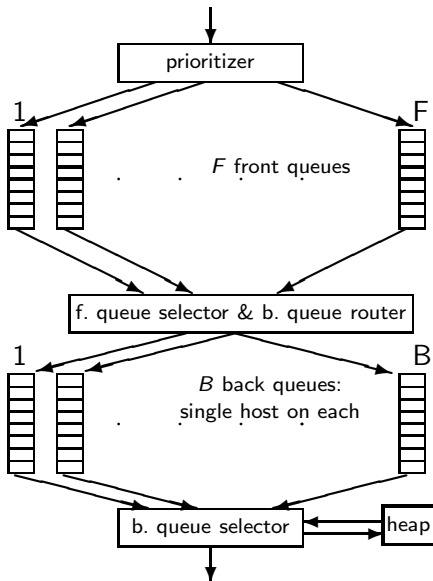
- URLs flow in from the top into the frontier.

Mercator URL frontier



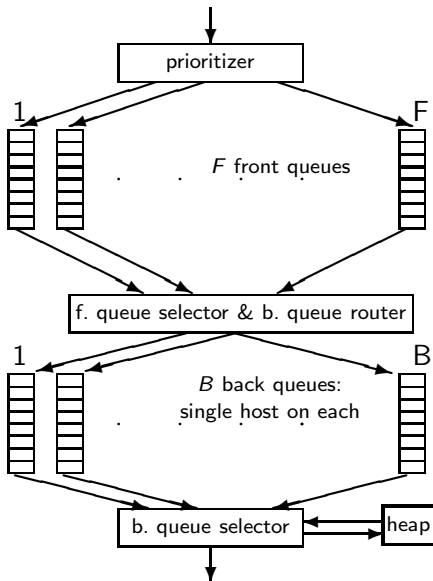
- URLs flow in from the top into the frontier.
- Front queues manage prioritization.

Mercator URL frontier



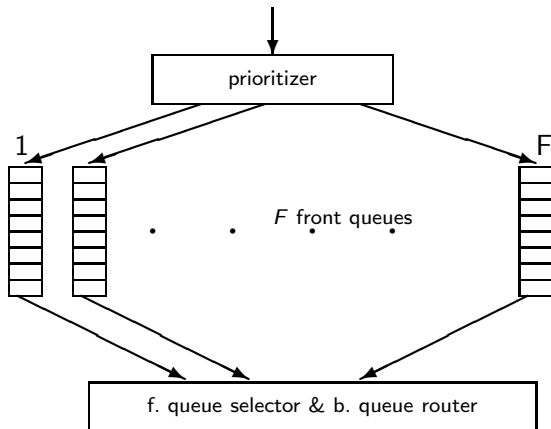
- URLs flow in from the top into the frontier.
- Front queues manage prioritization.
- Back queues enforce politeness.

Mercator URL frontier

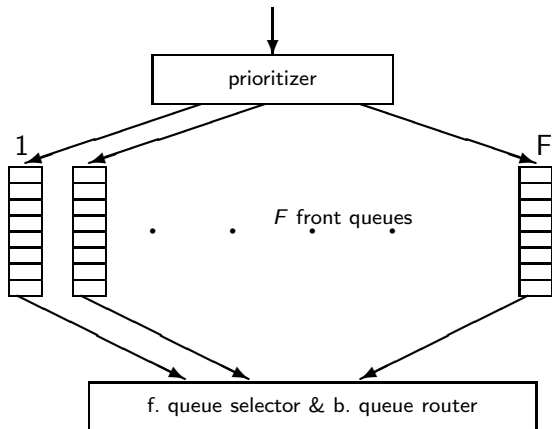


- URLs flow in from the top into the frontier.
- Front queues manage prioritization.
- Back queues enforce politeness.
- Each queue is FIFO.

Mercator URL frontier: Front queues

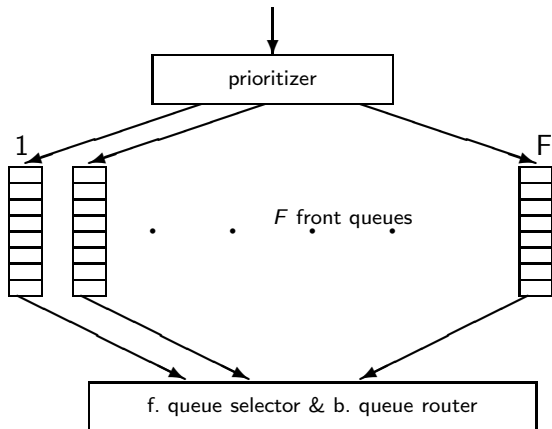


Mercator URL frontier: Front queues



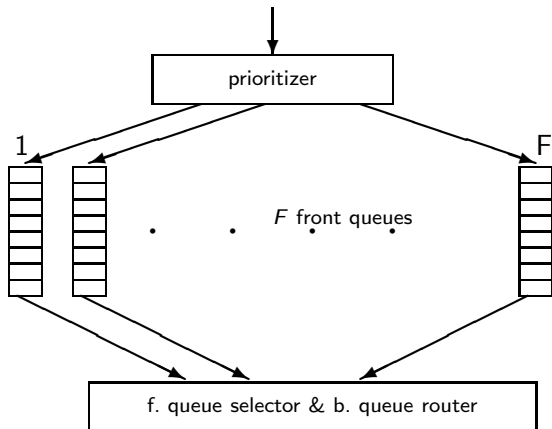
- Prioritizer assigns to URL an integer priority between 1 and F .
- Then appends URL to corresponding queue

Mercator URL frontier: Front queues



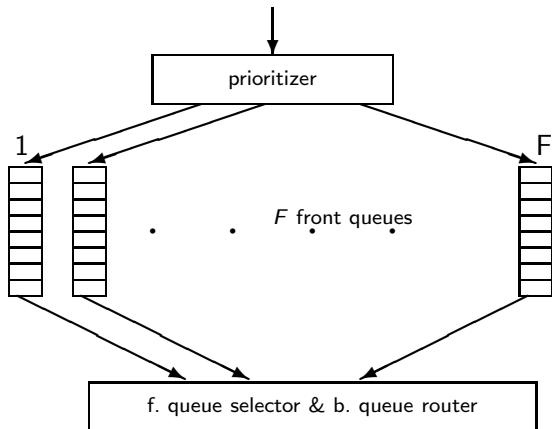
- Prioritizer assigns to URL an integer priority between 1 and F .
- Then appends URL to corresponding queue
- Heuristics for assigning priority: refresh rate, PageRank etc

Mercator URL frontier: Front queues



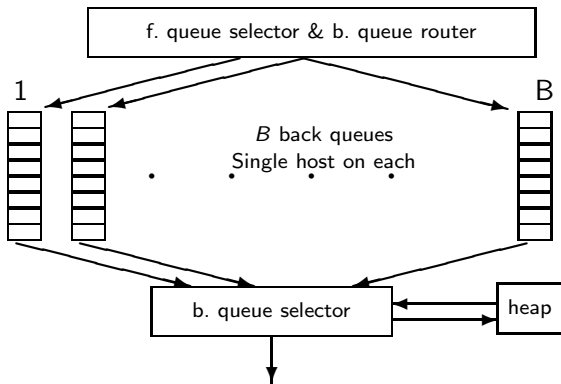
- Prioritizer assigns to URL an integer priority between 1 and F .
- Then appends URL to corresponding queue
- Heuristics for assigning priority: refresh rate, PageRank etc

Mercator URL frontier: Front queues

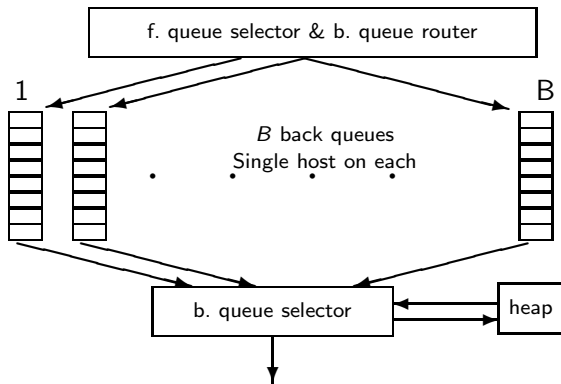


- Selection from front queues is initiated by back queues
- Pick a front queue from which to select next URL: Round robin, randomly, or more sophisticated variant
- **But with a bias** in favor of high-priority front queues

Mercator URL frontier: Back queues

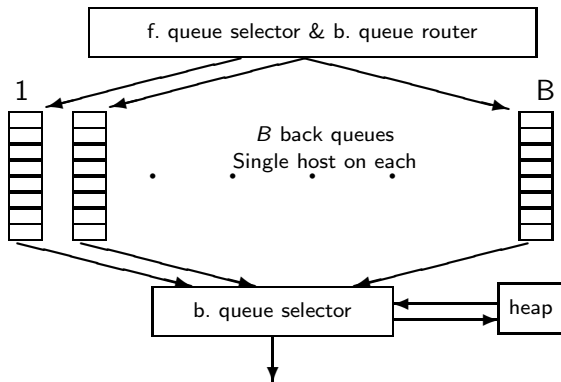


Mercator URL frontier: Back queues



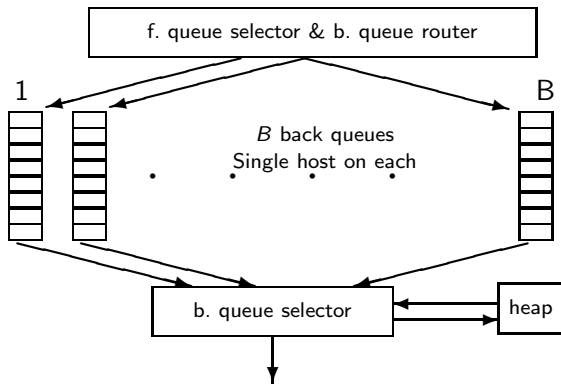
- **Invariant 1.** Each back queue is kept non-empty while the crawl is in progress.
- **Invariant 2.** Each back queue only contains URLs from a single host.
- Maintain a table from hosts to back queues.

Mercator URL frontier: Back queues



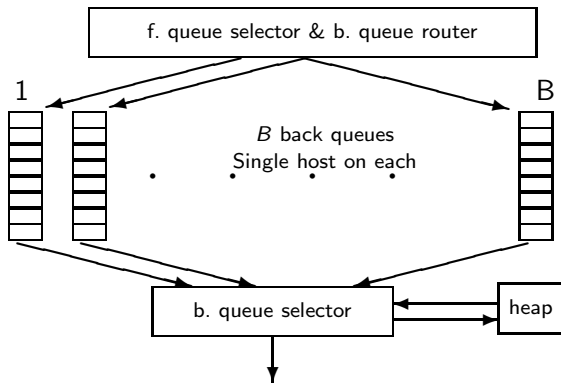
- In the heap:
- One entry for each back queue
- The entry is the earliest time t_e at which the host corresponding to the back queue can be hit again.
- The earliest time t_e is determined by (i) last access to that host
(ii) time gap heuristic

Mercator URL frontier: Back queues



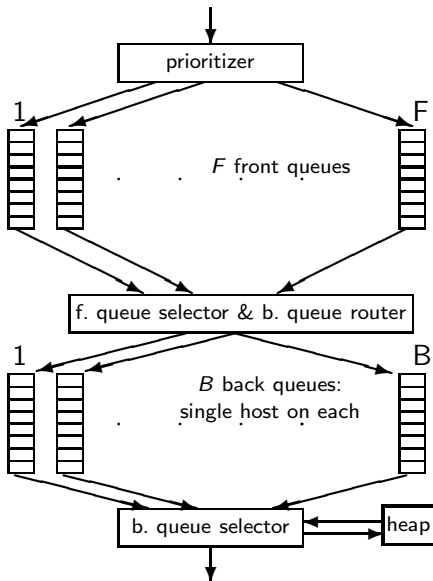
- How fetcher interacts with back queue:
- Repeat (i) extract current root q of the heap (q is a back queue)
- and (ii) fetch URL u at head of q ...
- ... until we empty the q we get.
- (i.e.: u was the last URL in q)

Mercator URL frontier: Back queues



- When we have emptied a back queue q :
- Repeat (i) pull URLs u from front queues and (ii) add u to its corresponding back queue ...
- ... until we get a u whose host does not have a back queue.
- Then put u in q and create heap entry for it.

Mercator URL frontier



- URLs flow in from the top into the frontier.
- Front queues manage prioritization.
- Back queues enforce politeness.

Spider trap

- Malicious server that generates an infinite sequence of linked pages
- Sophisticated spider traps generate pages that are not easily identified as dynamic.

Resources

- Chapter 20 of IIR
- Resources at <http://ifnlp.org/ir>
 - Paper on Mercator by Heydon et al.
 - Robot exclusion standard