Automated knowledge base construction

2. Design considerations, crawling and scraping

Simon Razniewski
Summer term 2022
Notes

• Central communication: Mailing list

• Assignment results
  • Late submissions, format

• Rooms

• Survey
  • Missed DSAI (57% other)? → MSc. mostly
  • NLP 50/50
  • ML: 75/25
  • Semantic Web, Wikidata: 25/75
  • 80% Python
  • Oral exam 50/50 → assignments, last tutorial test session

• Comments
  • Recordings: Noted
  • Rooms: see above
  • Builds from core?
Outline

1. AKBC - Design considerations
2. Crawling
3. Scraping
AKBC design considerations

Fundamental questions:

1. What should be the output?
2. What is the best suited input?
3. How to get from 2. to 1.?
What should be the output?

• Err, a KB?

• What kind of KB?
  • Canonicalized entities?
  • Canonicalized relations?
  • Importance of precision vs. recall?

• Typically approached as several subtasks
  • Entity extraction
  • Entity canonicalization
  • Entity set expansion
  • Entity typing
  • Relation extraction
  • Relation canonicalization
  • Constraint extraction
  • Knowledge cleaning

...
Outline

1. AKBC design considerations
2. Crawling
3. Scraping
Acknowledgment

- Material adapted from Fabian Suchanek and Antoine Amarilli
Crawling: Task

- **Given**: One or several source URLs
- **Return**: Document corpus obtained by transitive hyperlink closure (bounded)

Donald John Trump (born June 14, 1946) is the 45th President of the United States.

The United States unites lots of states: Some of the cooler ones are California and New York.
A crawler does BFS on URLs

1. Start with queue of important URLs

http://...
http://...
http://...
A crawler does BFS on URLs

http://...

2. Download

http://...

http://...
A crawler does BFS on URLs

http://...

3. If page is “good”, add it to corpus
A crawler does BFS on URLs

4. Find URLs in page, enqueue them
A crawler does BFS on URLs

http://...

5. repeat the process until you covered all pages
   • within a certain depth
   • in a certain domain
   • with certain topics
   • ..
Crawling: The fine print

1. How to find hyperlinks?
2. How to decide when to revisit/how often to revisit?
3. Denial of service
4. Captchas
5. Deep web
6. Existing crawl corpora
Finding new URLs

- In an HTML page
- Hyperlinks `<a href="...">
- Media `<img src="...">, `<audio src="...">, `<video src="...">, `<source src="...">
- Frames `<iframe src="...">
- JavaScript `window.open("...")`
- Page text by regular expressions.
- In other kinds of files (PDFs...).
- In sitemaps provided specifically to crawlers.
Freshness Problem

• Content on the Web changes
• Different change rates:
  online newspaper main page: every hour or so
  published article: virtually no change
• Continuous crawling, and identification of change rates
for adaptive crawling:
  If-Last-Modified HTTP feature (not reliable)
  Identification of duplicates in successive request

Firefox: Developer tools/Network/Response header
https://en.wikipedia.org/wiki(Max_Planck_Institute_for_Informatics)
Freshness problem (2)

• Prediction problem: Estimate page change frequency
  • From previous change behavior
  • Or from page content

• Optimization problem: Decide crawl frequency
  • Fixed budget → How to distribute them
  • Flexible budget → Cost-benefit framework needed
Estimating change frequencies

- Cho and Molina, TOIT 2003
  - Model changes as Poisson processes (i.e., memoryless/statistically independent)
  - Extrapolate change frequency from previous visits
    - Daily visit for 10 days, 6 changes detected
    - Change frequency: 0.6 changes/day?
  - Extrapolation underestimates change frequency due to multiple change possibility

- Wijaya et al., EMNLP 2015
  - Wikipedia-specific
  - Learn state-change-indicating terms
  - E.g., engage, divorce
<table>
<thead>
<tr>
<th>Label</th>
<th>Verb</th>
</tr>
</thead>
<tbody>
<tr>
<td>begin-deathdate</td>
<td>+(arg1) die on (arg2), +(arg1) die (arg2),</td>
</tr>
<tr>
<td></td>
<td>+(arg1) pass on (arg2)</td>
</tr>
<tr>
<td>begin-birthplace</td>
<td>+(arg1) be born in (arg2), +(arg1) bear in (arg2),</td>
</tr>
<tr>
<td></td>
<td>+(arg1) be born at (arg2)</td>
</tr>
<tr>
<td>begin-predecessor</td>
<td>+(arg1) succeed (arg2), +(arg1) replace (arg2),</td>
</tr>
<tr>
<td></td>
<td>+(arg1) join cabinet as (arg2), +(arg1) join as (arg2)</td>
</tr>
<tr>
<td>begin-successor</td>
<td>+(arg1) lose seat to (arg2), +(arg1) resign on (arg2),</td>
</tr>
<tr>
<td></td>
<td>+(arg1) resign from post on (arg2)</td>
</tr>
<tr>
<td>begin-termstart</td>
<td>+(arg1) be appointed on (arg2), +(arg1) serve from (arg2),</td>
</tr>
<tr>
<td></td>
<td>+(arg1) be elected on (arg2)</td>
</tr>
<tr>
<td>begin-spouse</td>
<td>+(arg1) marry on (arg2), +(arg1) marry (arg2),</td>
</tr>
<tr>
<td></td>
<td>+(arg1) be married on (arg2), - (arg1) be engaged to (arg2)</td>
</tr>
<tr>
<td>end-spouse</td>
<td>+(arg1) file for divorce in (arg2), +(arg1) die on (arg2),</td>
</tr>
<tr>
<td></td>
<td>+(arg1) divorce in (arg2)</td>
</tr>
<tr>
<td>begin-youthclubs</td>
<td>+(arg1) start career with (arg2),</td>
</tr>
<tr>
<td></td>
<td>+(arg1) begin career with (arg2), +(arg1) start with (arg2)</td>
</tr>
</tbody>
</table>
Optimization problem
[Razniewski, 2016]

• Resources flexible

• Ingredients:
  • Benefit of an up-to-date website
    • Alternatively: cost of outdated website
  • Cost of a crawl action
  • Decay behavior

→ Page-specific recrawl frequency that maximizes benefit minus cost
Decay behaviour

\[ z_{\text{exp}}(t) = e^{-\lambda_{\text{exp}} t}. \]

\[ z_{\text{lin}}(t) = \max(1 - \lambda_{\text{lin}} t, 0). \]
Observed decay behaviour

Figure 7: Decay behaviour of soccer players at Manchester United (blue) and Bayern München (red), observed (solid lines), and approximated by exponential decay curves with $\lambda = 0.26$ and 0.36, respectively (dashed lines).
Average freshness $F$

$$F(\lambda, u) = \frac{\int_0^u z(t) dt}{u}.$$
Net income $NI$

\[
 NI(u) = B \cdot F(\lambda, u) - \frac{C}{u}.
\]

B...Benefit/time unit  
F...Average freshness  
$\lambda$...decay coefficient  
u...update interval length  
C...cost of an update

$\rightarrow$ Standard algebra:  
Finding function maximum
Examples for address updates: NI over u

Assumption: benefit over one year = 100 x cost of single crawl
Actual ratio magnitudes lower, e.g., 0.003 Cents/crawl
(and for 580 $ on Amazon EC2)
Duplicate pages

- Prevent multiple indexing and penalize content farms.
- Prevent duplicate URLs by canonicalization.
  
  http://example.com:80/foo
  = http://example.com/bar/../foo
  = http://www.example.com/foo

- Detect duplicate pages by using a hash function.
- Detect near-duplicates (dates, etc.) by using a similarity function.
  (e.g., Broder's MinHash from 1997, used in AltaVista and later Google)
Crawl scheduling

• Wait a minimal delay between requests to the same server.
  ➜ Depends on the server (wikipedia.org vs your laptop).
  ➜ Depends on the resource (large files...).
  ➜ Generally, waiting at least one second is preferable.
• Requests to different servers can be parallelized.
• Crawlers represent about 20% of Web traffic.
Crawler traffic

[Yuan et al., CCN 2002]  
"We estimate that approximately 40% of Internet traffic is due to Web crawlers"
Robot control (honor-based)

  - Only at root level (not available for subfolders).
  - Filtering by **User-agent**.
  - **Disallow** directive to forbid certain pages.
  - Also: **Allow**, **Crawl-delay**, **Host**, **Sitemap**.

- **HTTP header**: **X-Robots-Tag** (less support):
  - **X-Robots-Tag**: noindex

- **Meta tag**: `<meta name="robots" content="noindex">`
  - Also **nofollow**, **nosnipped**, **noarchive**...

- **Links**: `<a href="secret/" rel="nofollow">`

- **Engine-specific** interfaces (e.g., Google Webmaster Tools).

  => No guarantees!

[https://www.mpi-inf.mpg.de/robots.txt](https://www.mpi-inf.mpg.de/robots.txt)
[https://www.google.de/robots.txt](https://www.google.de/robots.txt)
Robot control with CAPTCHAs

How can we discriminate against robots?

- Completely Automated Public Turing test to tell Computers and Humans Apart (trademarked by CMU, but patented by AltaVista).
- Making a computer able to recognize humans.
- Can be any AI problem: add two numbers, listen to a word, recognize an animal in an image, etc.
ReCAPTCHAs

CAPTCHAs can be used to
• digitize books
  Show one word that we know (to validate the user),
  and one word that we want to digitize (to digitize the book)

  following  finding

• Show ads
  Ask the user to type a slogan

• Do recognition of street numbers in
  Google street view images
Breaking CAPTCHAs

• Employ humans to remotely solve CAPTCHAs ("sweatshops", hundreds per hour)

• Sometimes there may be no ground truth \(\rightarrow\) Try often enough

• Optical character recognition has improved and can solve some CAPTCHAs
“Robot Control” by Spider Traps

A spider trap (also: crawler trap, robot trap) is a set of web pages that cause a web crawler to make an infinite number of requests or cause a poorly constructed crawler to crash.

[WikiPedia/Spider trap]

Example:

<table>
<thead>
<tr>
<th>January 1st</th>
<th>Spider traps can be intentional or unintentional. Can be used to trap spiders that do not follow robots.txt :-)</th>
</tr>
</thead>
<tbody>
<tr>
<td>• no meetings</td>
<td></td>
</tr>
</tbody>
</table>

http://foo.com/bar/foo/bar/foo/bar/foo/bar/.....
Deep web / dark web

- Pages that have no links to them.
- For instance, result pages from a search.
- 2001 estimate: the deep Web is hundreds of times larger than the reachable Web.
- Web form probing:
  - Need to figure out form constraints.
  - Need to come up with keywords.
  - Idea: feed back words from the website into the form.

We can use an existing Web crawl

<table>
<thead>
<tr>
<th></th>
<th>pages</th>
<th>size</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClueWeb</td>
<td>1b</td>
<td>25 TB</td>
</tr>
<tr>
<td>CommonCrawl</td>
<td>6b</td>
<td>100 TB</td>
</tr>
<tr>
<td>Internet Archive</td>
<td>2b</td>
<td>80 TB</td>
</tr>
<tr>
<td>enWikipedia</td>
<td>5m</td>
<td>30 GB</td>
</tr>
<tr>
<td>Dresden web</td>
<td>125m</td>
<td></td>
</tr>
<tr>
<td>Dresden web</td>
<td></td>
<td>table corpus</td>
</tr>
<tr>
<td>Twitter dumps</td>
<td>280m</td>
<td></td>
</tr>
<tr>
<td>2016 US election</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reddit dumps</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Wikia dumps</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Insights from crawling mpi-inf.mpg.de

• URL ending *inclusion/exclusion* criteria need thought
• Long (*machine-generated URLs*) need exclusion
• Beyond that no issues
• 35 lines in Python
• Sequential runtime for 2000 pages: ~10 minutes
• Completeness?
Outline

1. Design considerations
2. Crawling
3. Scraping
Bilbo was the first hobbit to become famous in the world at large, and one of the few to set foot in the Undying Lands.

Gandalf

- Other names: Oldrin, Mithrandir, Incin, Tharûn, Greyhame, Old Greybeard, The Grey Pilgrim, Stormcrow, White Rider, Lûthspeli, Gandalf the Wandering Wizard


- Birth: Before the Shaping of Arda

- Death: January 26, 3019, Battle of the Peak (physical death only, resurrected): immortal

- Weapon: Glamdring, Narvi, Wizard staff

Bilbo Baggins

- Other names: Mr. Baggins, Bilbo Took (see more)

- Titles: Elf-friend, Ring-bearer, Burglar, The Fly who Stings the Spider, Barrel Rider, etc.

- Birth: 22 September, TA 2980 (SR 1290)

- Death: Unknown (Last sighting on 29 September, TA 3021) (SR 1420)

- Spouse: None

- Weapon: Sting
### Technical Details

<table>
<thead>
<tr>
<th>Product Dimensions</th>
<th>17.6 x 17.6 x 12.5 cm; 442 Grams</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item Weight</td>
<td>442 Grams</td>
</tr>
<tr>
<td>Item volume</td>
<td>420 Millilitres</td>
</tr>
<tr>
<td>Anti-Tick Material</td>
<td>Kristallglas</td>
</tr>
<tr>
<td>Is Assembly Required</td>
<td>No</td>
</tr>
<tr>
<td>Number Of Pieces</td>
<td>4</td>
</tr>
</tbody>
</table>

### Additional Information

<table>
<thead>
<tr>
<th>ASIN</th>
<th>B013KF6YV0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item model number</td>
<td>1172098140</td>
</tr>
<tr>
<td>Date First Available</td>
<td>7 Aug. 2015</td>
</tr>
<tr>
<td>Customer Reviews</td>
<td>4.7 out of 5 stars</td>
</tr>
<tr>
<td>Best Sellers Rank</td>
<td>565 in Home &amp; Kitchen Top 100 in Home &amp; Kitchen 1 in Water Glasses</td>
</tr>
</tbody>
</table>
Generated Web pages

Web page generation is the process of producing several similar Web pages from a KB.
Example: Generated Web pages

![Diagram showing an Amazon product page for "Shrek" by W. Steig, priced at 10 USD.](image)
Scraping aims to reconstruct the KB

10 USD

"Shrek"
by W. Steig
only 10 USD!
Buy it!
Def: Wrapper

A wrapper for a set of pages generated from the same KB is a function that extracts strings from such a page. (Technically, it is the inverse function of the function that generated the page. The strings still have to be disambiguated and put in relation to yield facts. Different applications have different more specific definitions of the "strings").

Rushmerick: Wrapper Induction

"Shrek...", "90 min", "7.9"
Def: XPath

XPath is a formal language for selecting nodes in an XML document.

/ identifies the root node
K/T[i] identifies the i-th child with tag T of the node identified by K
K/T is K/T[1] if K has one T child

<html>
<body>
<h1>Aloha from Hawaii</h1>
<p>This is a really great movie</p>
<p>Stars:<i>Elvis Presley</i></p>
</body>
</html>

[https://www.w3schools.com/xml/xml_xpath.asp]
[https://devhints.io/xpath]
Task: XPath

Write XPath expressions that identify nodes whose text is "Shrek", "W. Steig", and "84 min".

```html
<html>
<body>
<b>Shrek</b>
<ul>
<li>Creator: <b>W. Steig</b></li>
<li>Duration: <i>84 min</i></li>
</ul>
</body>
</html>
```
Scraping: Browser

Website: https://lotr.fandom.com/wiki/Bilbo_Baggins

- “Try XPath” Firefox addin

- //h3[@class='pi-data-label pi-secondary-font']

- Firefox console
  - $x('//h3[@class="pi-data-label pi-secondary-font"]')

- //h3[@class='pi-data-label pi-secondary-font'] | //div[@class='pi-data-value pi-font']
Scraping in Python - XPath

```python
# from https://lxml.de/parsing.html#parsing-html

import requests
import lxml
from lxml import etree

url = 'https://lotr.fandom.com/wiki/Frodo_Baggins'

req = requests.get(url)

html = etree.HTML(req.text)

output = html.xpath('//h3[@class="pi-data-label pi-secondary-font"]/')

for e in output:
    print(e.text)
```

Other names
Titles
Birth
Death
Weapon
Race
Hair
Eyes
Culture
Actor
Def: Wrapper induction

Wrapper induction is the process of generating a wrapper from a set of Web pages with strings to be extracted.

```
+ "Shrek", "7.9"

= /html/body/h1
   /html/body/p[2]/i
```
Detail Pages & List Pages

Wrappers can be learned across several detail pages:

Wrappers can also be learned across items in a list:
ROADRUNNER: Learn types

ROADRUNNER is a system that can learn the Web page structure.
Finds least upper bounds in regex lattice

Page 1:
<ul>
<li>Peanut</li>
</ul>

Page 2:
<ul>
<li>Charles</li>
</ul>

Wrapper:
<ul>
<li>[FIELD]</li>
</ul>

Crescenzi et al., VDLB 2001
ROADRUNNER: Learn types

ROADRUNNER is a system that can learn the Web page structure.

Page 1:
<ul>
<li>Peanut</li>
</ul>

Page 2:
<ul>
<li>Charles</li>
<li>Anne</li>
</ul>

Wrapper:
<ul>
<li>\((\text{li}[^\text{FIELD}]+)\)
</li>
</ul>
Def: Wrapper Application

Wrapper application is the process of extracting its strings from a Web page.

```
<html/body/h1
<html/body/p[2]/i

= "Elvis", "11"

Disambiguation, + relation

hasActor(e42, ElvisPresley)
hasRating(e42, "11.0")
```
Alternative Scraping in Python — BeautifulSoup

• Python library for
  • Treating HTML structure as a Python object
  • Effective search inside this object

```html
<html>
<head>
<title>
The Dormouse's story</title>
</head>
<body>
Once upon a time there were three little sisters; and their names were <a class="sister" href="http://ex.com/elsie" id="link1">Elsie</a>, <a class="sister" href="http://ex.com/lacie" id="link2">Lacie</a> and …

soup.title
# <title>The Dormouse's story</title>

soup.title.string
# 'The Dormouse's story'

soup.title.parent.name
# 'head'

soup.a
# <a class="sister" href="http://ex.com/elsie" id="link1">Elsie</a>

soup.find_all('a')
# [<a class="sister" href="http://ex.com/elsie" id="link1">Elsie</a>,
# <a class="sister" href="http://ex.com/lacie" id="link2">Lacie</a>]
```

Alternative Scraping in Python — BeautifulSoup (2)

```python
from bs4 import BeautifulSoup
import urllib3
import requests
from urllib.request import urlopen

site = "http://en.wikipedia.org/wiki/Max_Planck_Institute_for_Informatics"

page = requests.get(site, verify=False)
soup = BeautifulSoup(page.text, 'html.parser')
table = soup.find('table', class_='infobox vcard')

for tr in table.find_all('tr'):
    if tr.find('th'):
        print(tr.find('th').text + ": " + tr.find('td').text)
```

Abbreviation: MPI-INF
Formation: 1993; 26 years ago (1993)
Type: research institute
Headquarters: Saarbrücken, Saarland, Germany
Website: www.mpi-inf.mpg.de
XPath vs. BeautifulSoup vs ...  

• **XPath**: Generic query language to select nodes in XML (HTML) documents  
  • Queries can be issued from Python, Java, C, ...  

• **BeautifulSoup**  
  • Python library to manipulate/search websites as Python objects  

• **Scrapy**  
  • Python library to crawl websites  

• **Selenium**  
  • Actual scripted browser interaction  
  → To get around Javascript etc.
Assignment 2

• No crawling (practicality/ethics…)
• 1x Wikia infobox extraction
  • XML format, but essential content not structured by XML tags → BeautifulSoup/pattern matching/regex
• 1x LSF-scraping
  • XPath/BeautifulSoup should both work
Take home

1. Considerations about output, input, method go first

2. Crawling
   - BFS to achieve coverage
   - Challenges with captchas, traps, deep web

3. Scraping
   - Reverse-engineering of template-based websites

• Next week: (Textual) entity typing