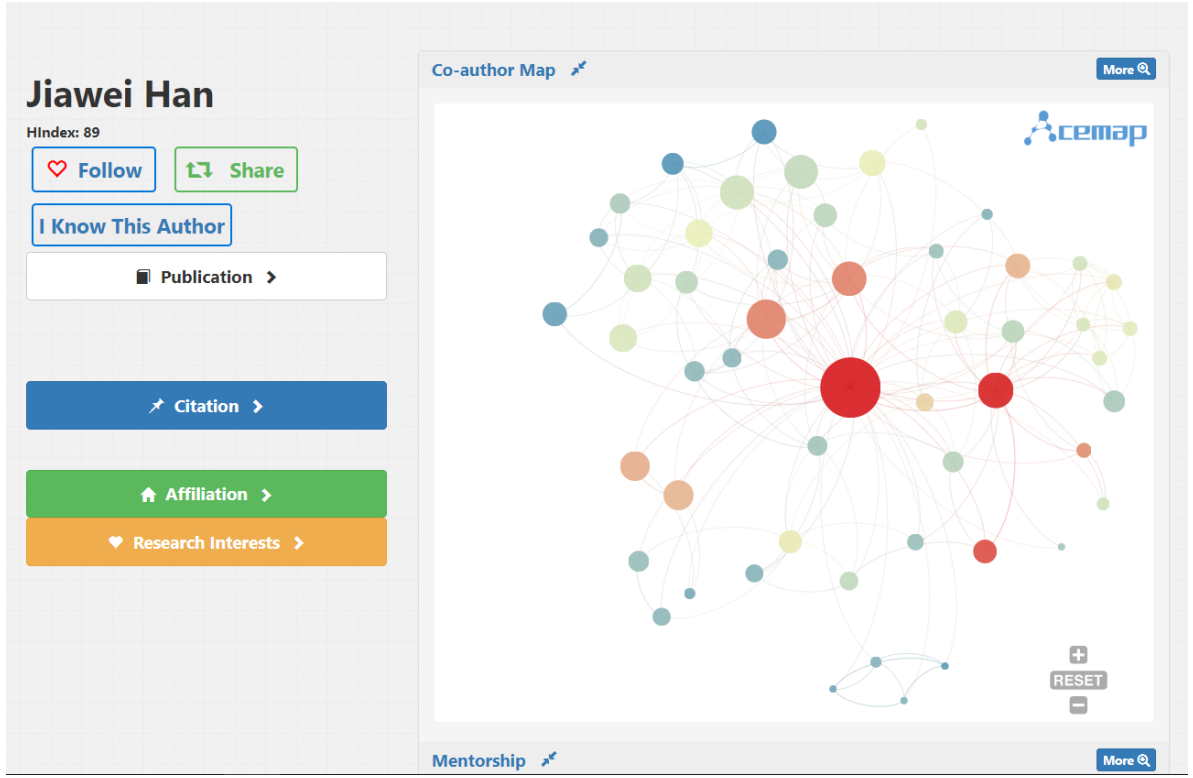
Scholar Photo Mining

Ruiliang Lyu
515030910208

Background



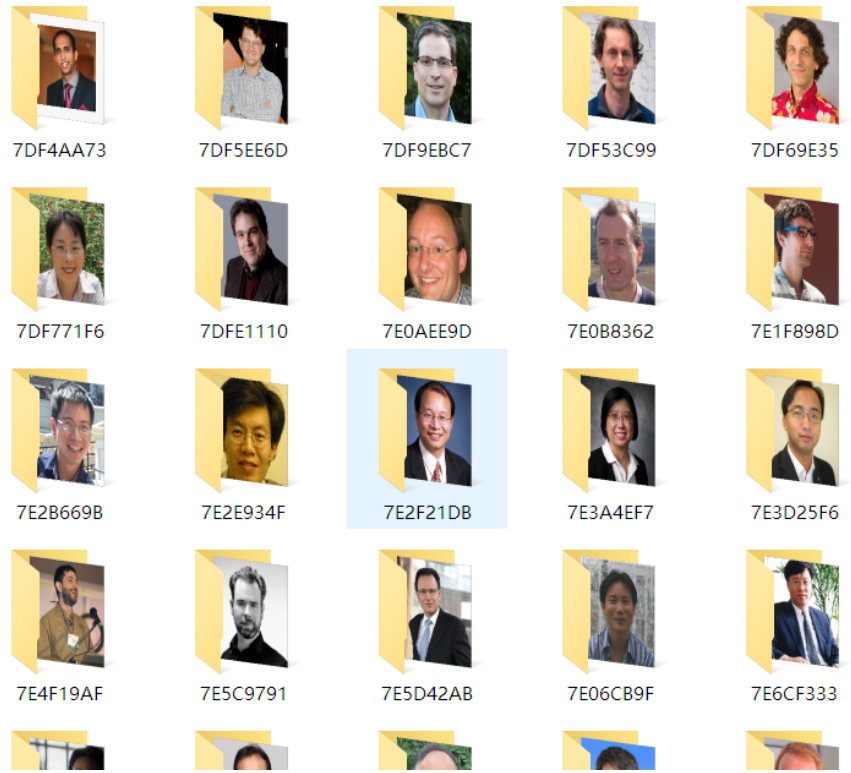
- Previously, there is **no photo** on the author profile page of Acemap (http://acemap.sjtu.edu.cn/)
- This is the **first** project to mine scholar photo from the Internet

Task Introduction

- **Input**
  - a list of CS top authors
  - with name, id (unique in Acemap system) and affiliation

	A	B	C
1	jiawei han	7E7A3A69	university of illinois at urbana champaign
2	hanspeter seidel	7F425224	max planck society
3	philip s yu	7EAA8442	ibm
4	gerhard weikum	1EAFC7AC	max planck society
5	don towsley	7945B8D1	university of massachusetts amherst
6	avi wigderson	01DE22A8	hebrew university of jerusalem
7	michael i jordan	7F8038BA	university of california berkeley
8	hector garciamolina	0E26BDB3	stanford university
9	christos faloutsos	8311D172	carnegie mellon university
10	oded goldreich	03E80959	weizmann institute of science

- **Output**
  - Corresponding photos of each scholar



Several Challenges

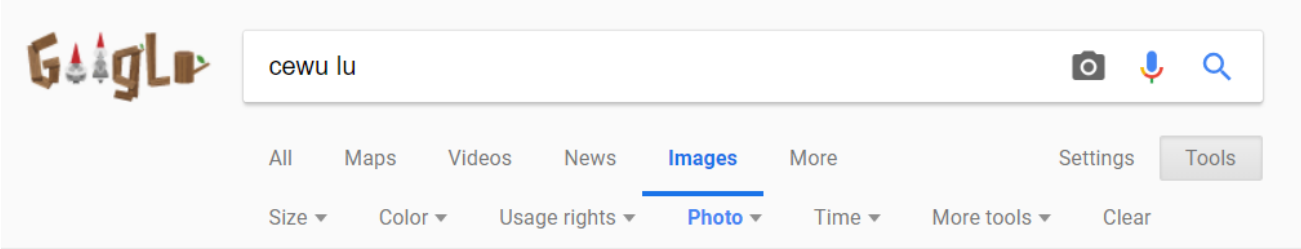
- Large scale of data
  - More than 200,000 scholars in computer science related areas
- Lack of ground-truth
  - Unsuitable to use supervised learning approach
- Name confliction
  - Scholars may share the same name with famous stars or other scholars

Approach

- STEP 1: Building Photo Library
  - Obtain a set of photos for each scholar in the scholar list
- STEP 2: Photo Cleaning
  - Analyze whether a photo is valid and remove invalid photos
- STEP 3: Photo selection
  - Select the best photo for each scholar

STEP 1: Building Photo Library

- **Objective:** download a set of photos for each scholar
- **Techniques:** Search engine, Python crawler, Remote server
- **Approach:**
  - Use Google searching for image

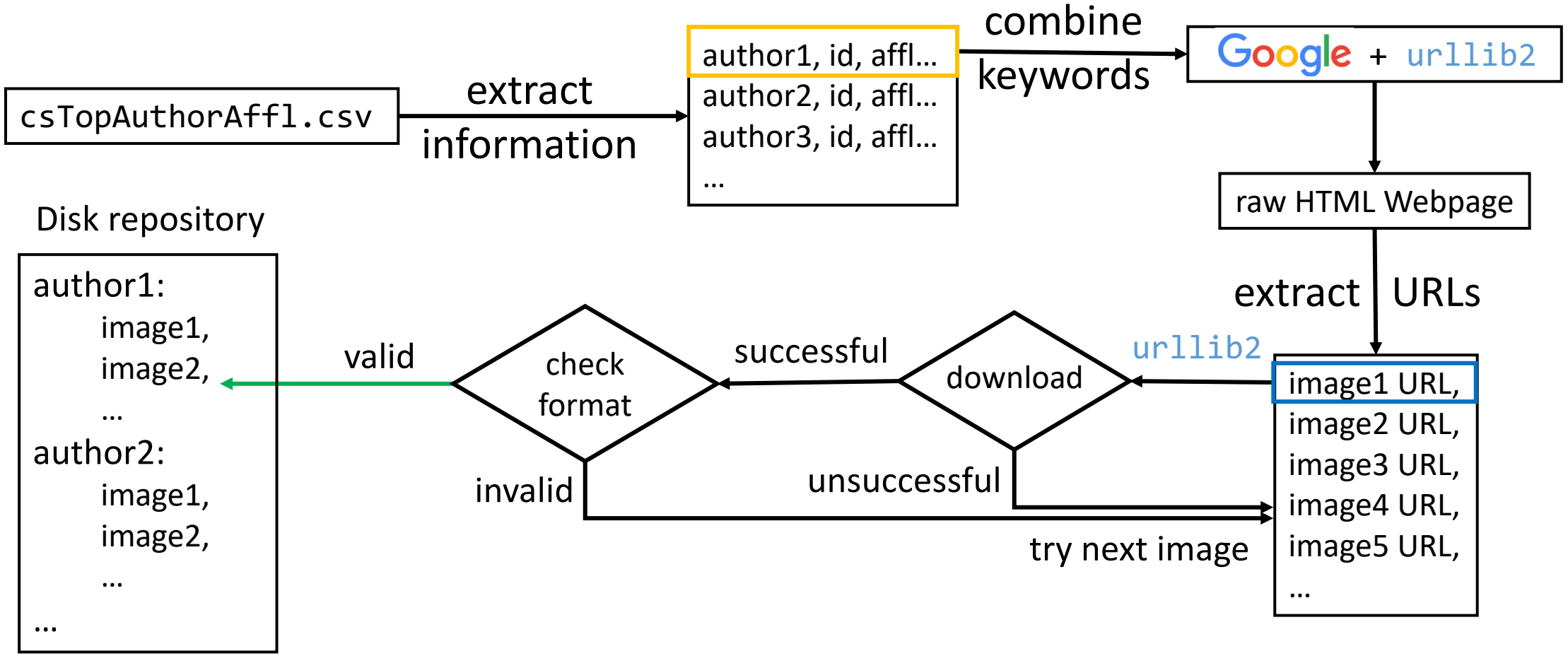


(tip: select the image type -> Photo)

  - Extract image URLs from webpage source code
  - Download images using Python module `urllib2`

STEP 1: Building Photo Library

- Framework overview:



STEP 1: Building Photo Library

- Implementation Details:
- 1. Using Google via VPN is slow
  - ==> deploy my program on a remote foreign server
- 2. Robustness of code
  - Handle various kinds of Exceptions
  - Use signal module to set timeout
  - Set checkpoint and build logs

```
class TimeoutError(Exception):
    pass

def handler(signum, frame):
    raise TimeoutError()
```

```
# set alarm to avoid timeout
signal.signal(signal.SIGALRM, handler)
signal.alarm(30)
try:
```

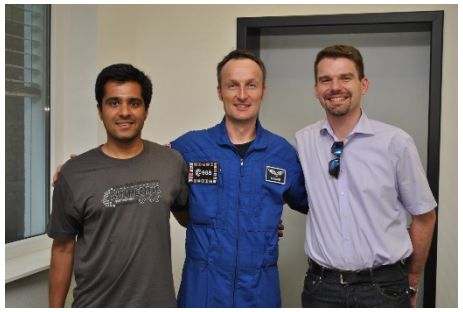
```
    except UnicodeEncodeError as e:
        download_status = 'fail'
        download_message = "UnicodeEncodeError on an image...trying next one..." + " Error: " + str(e)

    except TimeoutError as e:
        download_status = 'fail'
        download_message = "TimeoutError on an image...trying next one..." + " Error: " + str(e)

except HTTPError as e:
    download_status = 'fail'
    download_message = "HTTPError on an image...trying next one..." + " Error: " + str(e)

except URLError as e:
    download_status = 'fail'
    download_message = "URLError on an image...trying next one..." + " Error: " + str(e)

except ssl.CertificateError as e:
    download_status = 'fail'
    download_message = "CertificateError on an image...trying next one..." + " Error: " + str(e)

except IOError as e:
    download_status = 'fail'
    download_message = "IOError on an image...trying next one..." + " Error: " + str(e)

except IncompleteRead as e:
    download_status = 'fail'
    download_message = "IncompleteRead...trying next one..." + " Error: " + str(e)
```

STEP 2: Photo Cleaning

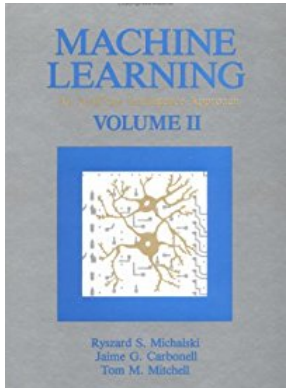
- **Objective:** remove improper images and crop single-face photos
- **Techniques:** Face Detection
- **Approach:**
  - Count faces in an image using Python module `face_recognition`
  - Remove images with 0 face and multiple faces (group photo)
  - crop images with 1 face (keep the original copy)

STEP 2: Photo Cleaning

- `face_recognition.face_locations(image)` could list the co-ordinates of each face
- examples:



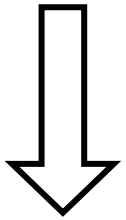
multi-face



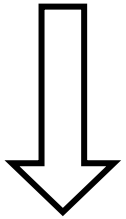
zero-face

single-face

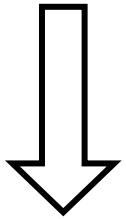
crop



remove



remove



keep

STEP 3: Photo Selection

- **Objective:** select the best photo from remaining photos
- **Techniques:** Face Recognition
- **Approach:**
  - Encoding faces into vectors using `face_recognition.face_encodings()`
  - Calculate similarity between every pair of images $\text{sim}_{ij} = V_1 \cdot V_2$
  - For every photo, calculate the metric $s_i = \sum_{j=1}^{N} \text{sim}_{ij}$
  - Pick the one with the highest score

STEP 3: Photo Selection

- Face Recognition vs. Face Detection
- Clustering algorithm vs. picking by score
  - Typical face clustering algorithm is Chinese Whispers (k-means not applicable)
  - Clustering needs iteration, therefore is slower
  - Clustering over meets the requirement and bring redundancy
  - Picking by score is faster

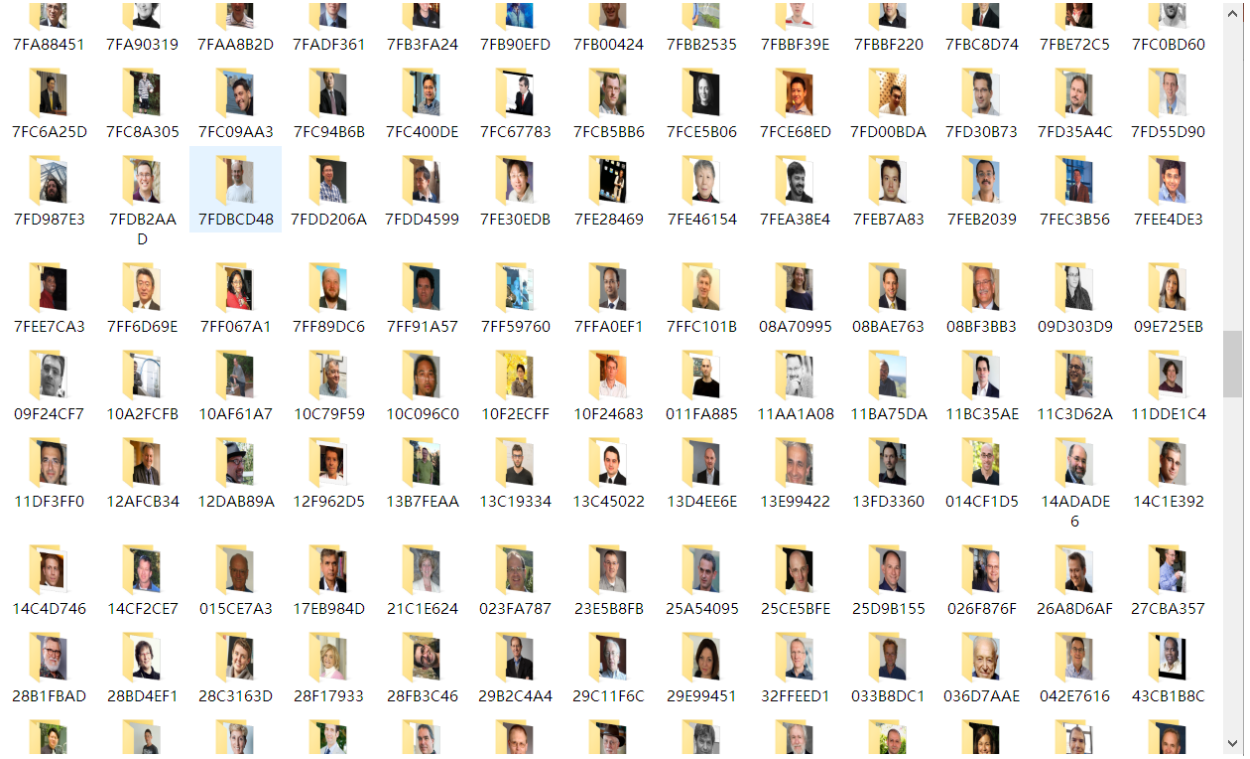
Solutions to Challenges

- Large scale of data
  - run code on a remote server 24 hours/day
- Lack of ground-truth
  - Use unsupervised methods
- Name confliction
  - Add affiliation to search term
  - typically 10 images by name and 5 images by name + affiliation

Results

- Downloaded more than 100,000 photos, 30+ GB data
- Selected more than 10,000 scholars' photos
- Evaluation:
  - compared with photos crawled

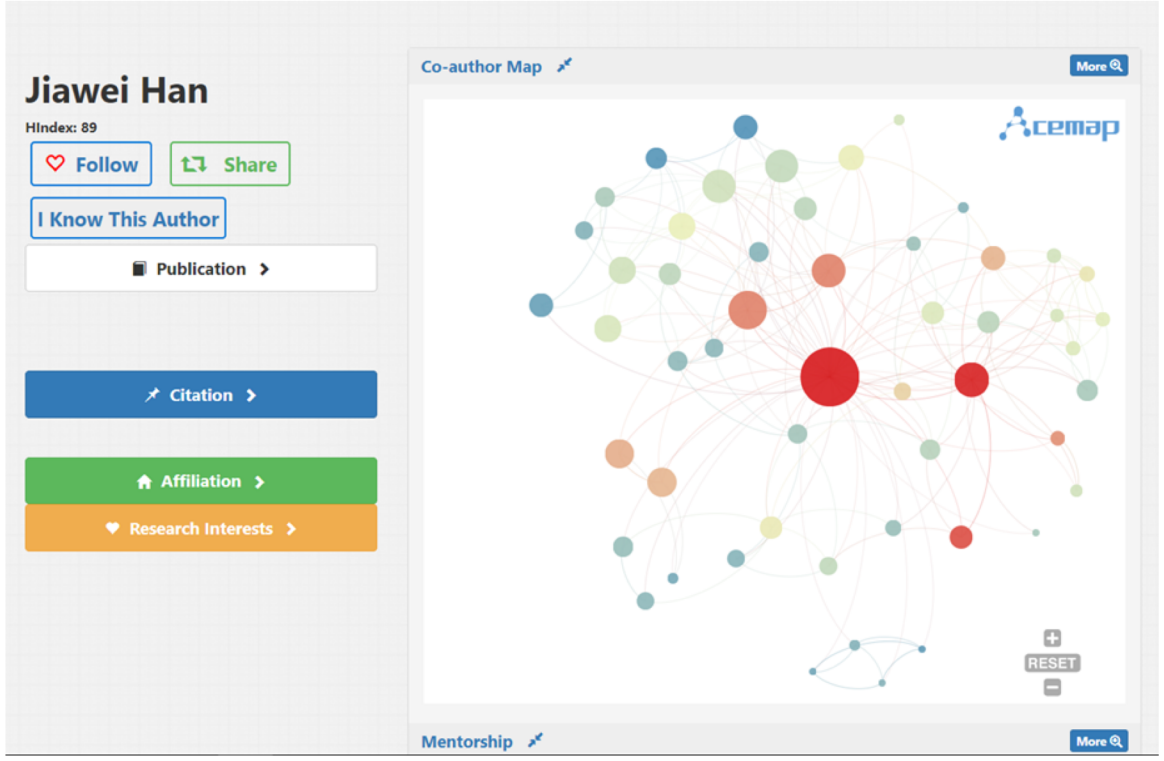
  from the home page of scholar
  - achieve an accuracy higher than 95%



Results

- submitted part of the photos to Acemap (http://acemap.sjtu.edu.cn/)

Before



After