Parallel Job Miner

*Note: Sub-titles are not captured in Xplore and should not be used

Jason Saini
CECS
University of Central Florida
Orlando, FL
jason.saini@ucf.edu

line 1: 2nd Given Name Surname
line 2: *dept. name of organization*

(of Affiliation)
line 3: *name of organization (of Affiliation)*
line 4: City, Country
line 5: email address or ORCID

line 1: 3rd Given Name Surname
line 2: *dept. name of organization*

line 1: 5th Given Name Surname
line 2: *dept. name of organization (of Affiliation)*
line 3: *name of organization (of Affiliation)*
line 4: City, Country
line 5: email address or ORCID

(of Affiliation)
line 3: *name of organization (of Affiliation)*
line 4: City, Country
line 5: email address or ORCID

line 1: 4th Given Name Surname
line 2: *dept. name of organization*

line 1: 6th Given Name Surname
line 2: *dept. name of organization (of Affiliation)*
line 3: *name of organization (of Affiliation)*
line 4: City, Country
line 5: email address or ORCID

(of Affiliation)
line 3: *name of organization (of Affiliation)*
line 4: City, Country
line 5: email address or ORCID

***Abstract*— The technology industry encompasses a diverse and continually evolving landscape, presenting a wealth of opportunities for computer science students on the cusp of graduation. For computer science students, the transition from academic study to professional practice is a critical phase of their careers. The job search process for students can be time-consuming and often inefficient, with relevant positions scattered across numerous platforms. Our team decided to make it easier for students to find job applications online with a parallel "job miner", which will consolidate job application data from multiple popular job sites for computer science: LinkedIn, GitHub/Simplify, USAJobs (for those interested in government positions), Monster, and Indeed. We accomplished this by using a process called "web scraping" to gather job opportunity data from the web. In other words, our program gets necessary information about jobs from the sites listed above and provides that information to our users in the form of an xlsx (Excel) file. With this Excel file, users will be able to adopt this strategy of tracking positions they are interested in.**

I. Introduction

Transitioning from studying in college to the ever-changing workforce requires finding jobs and internships within one's field. This process often involves sending many applications and receiving mostly rejections. To make the process easier for students, particularly those majoring in Computer Science and Software Engineering, we have created a web scraping application.

The project is open source, and therefore free, to all that would like to use it. Being open source enables all those who wish to contribute and expand/improve upon it to do so. To get started, users simply need to follow the download and setup instructions on the GitHub page, allowing them to run it as needed. We were able to achieve all of this through using pre-existing Application Programming Interfaces (APIs), creating our own web scraping functions, and storing collected info into a DataFrame from Python's Pandas library.

Identify applicable funding agency here. If none, delete this text box.

II. Problem Statement

A. Problem

Students encounter difficulty when applying to many different positions or companies trying to find relevant opportunities across various websites. After a website with relevant listings and the desired opportunities, they then must go into each card to read the details and click the actual application link if they are interested. Navigating through a multitude of cards on multiple sites for jobs that may not be relevant or applicable to you can quickly become overwhelming.

B. Proposed Solution

To help combat this, our project stores much of the needed information into a single file that is easy to read and contains the application links directly to all jobs found. From this spreadsheet, applicants can track their progress of individual applications using color coding, as demonstrated in the aforementioned commonplace strategy for tracking applications.

III. Related Work

There are many resources available that scrape the sites mentioned above, along with many others, to collect job data. Where our project differs is that we are the only resource which collects job data from multiple sites at once, providing users with a comprehensive list of opportunities without needing much technical knowledge.

To achieve this, we use existing APIs to access content and data. APIs allow our project to interact with the platforms

efficiently and collect the needed information in a straightforward manner. This approach allows us to streamline data retrieval and adapt to anti-scraping measures used by major job finding sites.

As noted above, many job finding companies implement constantly evolving anti-scraping features. These features include bot detection, dynamic loading, and including unique job ids into the URL. Having these measures in place makes basic web scraping challenging, but APIs help combat that challenge and enable our data collection from various sites.

Additionally, our program is easy to use and designed to be efficient. The user only needs to download the files, follow the quick setup on the GitHub page, and run the file with a title to receive 100+ job opportunities from 4 sites related to what they are looking for. The program is made to be efficient by utilizing threads to scrape all the websites at once, instead of one by one. While existing resources do exist, our project offers a unique version by scraping and consolidating information from multiple sites at once.

IV. Technique

Our approach to a multi-threaded web scraper was using BeautifulSoup for the manual web scraping, RapidAPI to get our APIs, and Python's multiprocessing library to run each scraper on a different thread.

BeautifulSoup is a popular Python library used for web scraping. We use it in conjunc tion with the Requests library, which sends a get request to a provided link and returns the contents. After getting the content, you pass that through to BeautifulSoup to parse the content it was given, we choose to use the HTML parser.

Now that you have the HTML content, you can use BeautifulSoup to then traverse the page via its html tags. RapidAPI is a service that allows users to publish their own API and use others' pub lished APIs, either free or for a cost. Most published APIs tend to have multiple price plans which allow them a certain number of requests.

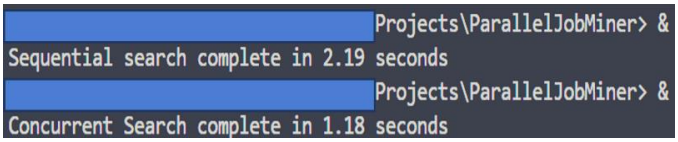
Python's multiprocessing library enables a program to run processes on separate threads concurrently. It allows one to assign a 'Process' to a function with any arguments it may need. When the thread is started, it executes that function with the given arguments.

We use these in conjunction with each other to create our current application. Beautiful Soup allows us to find the needed content on the pages that we manually scrape, RapidAPI returns us the data of jobs that it gets, and multiprocessing allows us to run the scrapers concurrently instead of sequentially.
To ensure the scalability of our program we have a clean and organized file structure that is modular. For example, you can add a scraper to another site by simply adding a file for it and adding the function to the main file.

V. Evaluation

Initially, our team hypothesized that the sequential and concurrent methods of web scraping result in very similar execution times, with the sequential method having the faster runtime around 70% of the time. This, however, was mainly due to the need to finish adding scrap ing/querying functionalities for multiple job sites such as Monster, Glassdoor, and Indeed.

After adding multiple sources for jobs and more job listings in general, we observed that the execution times started to deviate away from each other, with the concurrent method having a faster runtime.



After running our concurrent and sequential algorithms separately, we observed a nearly 86% decrease in runtime from end to end, from processing the user specified job title to generating the spreadsheet. While a noticeably small increase in the quantifiable runtime, a decrease of this magnitude on a much larger scale will be incredibly beneficial to those compiling job applications in larger quantities

VI. Limitations and Challenges

In the early stages of the project, we discovered that websites (i.e. LinkedIn) are weary against web scrapers for reasons including protecting intellectual property, preserving band width and server resources, maintaining data accuracy and quality, protecting user privacy, preventing competitive advantage, avoiding legal complications, etc.

Our quick workaround was to use the official APIs, but of course these come with their own set of limitations. For example, the free LinkedIn API we are using allows 25 calls per month, which will not be serviceable for a full production launch. This is another challenge for scalability and maintenance, and a significant consideration when incorporating a data source into our parallel job scraper.

Another considerable challenge is accessing our pandas dataframe through threads, which requires a thread-safe extended class of the dataframe. A final challenge is find ing ways to elevate the user experience from a heavily manual setup process to a more direct way of accessing our application.

VII. First Milestone

On March 8th, 2024, we reached a significant milestone in our project at this point, using a modular approach to thread different scrapers and API requests, we can quickly compile relevant job application information for prospective applicants.

At the time of writing, these are the following tasks left at hand:

1. Finish scraping/querying remaining sites (Monster, Glassdoor, Indeed)

2. Investigate appending Excel data in multi-threaded way for potential performance improvements

3. Filter out duplicate positions from multiple sites (most likely using the application link to check for duplicate entries)

4. Navigate API token limits (LinkedIn only allows 25 queries a month)

5. Add legend to spreadsheet to allow color-coded status tracking for job applications

6. Adding more parameters to further customize job search

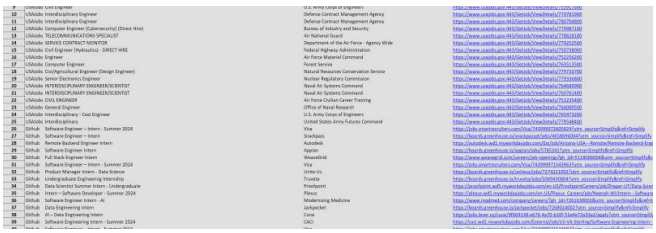
7. Reach goal: Implement a better UI or general UX improvements

Our job miner project represents a significant effort towards easing the job search process for computer science students and professionals. By consolidating job appli cation data from multiple popular job sites into a single Excel file, our tool empowers users to efficiently track and manage their job applications.

Throughout the development process, we have encountered various challenges and lim itations, including navigating anti-scraping measures, managing API token limits, and en hancing the user experience. However, through our innovative approach and collaborative efforts, we have made substantial progress towards overcoming these obstacles.

Moving forward, our focus will be on refining the project, addressing remaining tasks, and incorporating user feedback to enhance usability and functionality. We are committed to finishing the scraping/querying functionalities for remaining sites, optimizing performance through multi-threading, implementing duplicate position filtering, and improving the user interface.

Ultimately, our goal is to provide a valuable resource for the computer science commu nity, facilitating seamless access to job opportunities and empowering individuals in their career pursuits. We are excited about the future of the Parallel Job Miner project and look forward to its continued growth and impact.



VIII. TODO: FILL OUT REST OF PAGES

TODODODODOODODODOO

IX. Conclusion

In conclusion …

NOTES:

The template will number citations consecutively within brackets [1]. The sentence punctuation follows the bracket [2]. Refer simply to the reference number, as in [3]—do not use “Ref. [3]” or “reference [3]” except at the beginning of a sentence: “Reference [3] was the first ...”

Number footnotes separately in superscripts. Place the actual footnote at the bottom of the column in which it was cited. Do not put footnotes in the abstract or reference list. Use letters for table footnotes.

Unless there are six authors or more give all authors’ names; do not use “et al.”. Papers that have not been published, even if they have been submitted for publication, should be cited as “unpublished” [4]. Papers that have been accepted for publication should be cited as “in press” [5]. Capitalize only the first word in a paper title, except for proper nouns and element symbols.

For papers published in translation journals, please give the English citation first, followed by the original foreign-language citation [6].

[1] G. Eason, B. Noble, and I. N. Sneddon, “On certain integrals of Lipschitz-Hankel type involving products of Bessel functions,” Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955. *(references)*

[2] J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[3] I. S. Jacobs and C. P. Bean, “Fine particles, thin films and exchange anisotropy,” in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.

[4] K. Elissa, “Title of paper if known,” unpublished.

[5] R. Nicole, “Title of paper with only first word capitalized,” J. Name Stand. Abbrev., in press.

[6] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, “Electron spectroscopy studies on magneto-optical media and plastic substrate interface,” IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

[7] M. Young, The Technical Writer’s Handbook. Mill Valley, CA: University Science, 1989.

IEEE conference templates contain guidance text for composing and formatting conference papers. Please ensure that all template text is removed from your conference paper prior to submission to the conference. Failure to remove template text from your paper may result in your paper not being published.