

Optimizing CV Matching with Job Vacancies Using the Boyer-Moore Algorithm

Andrew Christofer Hadiwinata¹, Eunike Endariahna Surbakti²

^{1,2} Informatics Department, Faculty of Engineering & Informatics,
Universitas Multimedia Nusantara Tangerang, Indonesia

¹andrew.hadiwinata@student.umn.ac.id, ²eunike.endariahna@umn.ac.id

Accepted 20 Mei 2025

Approved 28 Mei 2025

Abstract— According to a survey conducted by a public survey agency, there was a decline in the percentage from 37% to 20%, indicating that a gap still exists between the skills required by the job market and those possessed by job seekers. To address this issue, this study aims to assess the alignment between data from curriculum vitae (CV) and job vacancies. The Boyer-Moore algorithm is implemented through a web-based system. The system extracts text from a PDF file, which is then used in the application of the Boyer-Moore algorithm. The system also retrieves selected job vacancy data and generates keywords using YAKE (Yet Another Keyword Extractor). Before processing with the Boyer-Moore algorithm, all data undergoes pre-processing. The algorithm's output is either a "match found" or "no match found." The similarity score is determined by dividing the number of matching keywords by the total number of keywords. Additionally, the system recommends other job options, aiming to suggest alternative vacancies that may better match the CV. These recommendations are based on the highest percentage of keyword matches from all job vacancy data stored in the system, which are sorted accordingly. The Boyer-Moore algorithm was successfully implemented in the job vacancy system, and the system's performance evaluation, using 100 job vacancy data entries, yielded an average processing time of 2.84438 seconds.

Index Terms— Boyer-Moore, curriculum vitae, Similarity score.

I. INTRODUCTION

According to data from the Central Statistics Agency (BPS), 937,176 people registered or applied for jobs in Indonesia in 2022. Job seekers' educational backgrounds play a crucial role in the application process. Various job vacancies in Indonesia require different minimum education levels, ranging from high school to bachelor's degrees. However, the proportion of university-educated workers in the labor force is only around 9.92%. This discrepancy highlights the gap between required job skills and the qualifications possessed by job seekers. Notably, a significant number of workers (up to 63%) end up in jobs unrelated to their field of study. To address this, an

effective system is needed one that helps job seekers identify relevant skills for their desired positions and assists recruiters in selecting candidates whose CVs align with job criteria. Unlike some other algorithms, Boyer-Moore doesn't require preprocessing, making it efficient for large-scale searches [1]. Boyer-Moore performs pattern matching by sliding the pattern over the text. It starts matching from the last character of the pattern, which allows for faster processing. The algorithm combines two heuristics: the Bad Character Heuristic and the Good Suffix Heuristic. When a mismatch occurs, the algorithm identifies the "bad character" (a character in the text that doesn't match the current pattern character). It then shifts the pattern to align with the last occurrence of the bad character in the pattern. This heuristic improves efficiency by skipping unnecessary comparisons [2]. The Boyer-Moore algorithm also considers the context of matching characters. If a mismatched character in the text occurs somewhere in the pattern, its index is used to skip over more characters, further reducing the number of comparisons. Boyer-Moore can be applied to strings of varying lengths and characters, making it suitable for CVs and job vacancy matching [3].

The research aims to implement the Boyer-Moore algorithm for assessing the suitability of CVs to specific job vacancies. It evaluates the efficiency of the algorithm based on computation time and focuses on English language patterns in PDF files. The goal is to assist HR professionals in streamlining the process and achieving more precise job candidate matches.

II. METHODOLOGY

A. Literature Studies

Literature studies aim to determine the development of knowledge, theories, concepts, methods, and current findings related to the problem that you want to research. Literature study includes searching, reading, analyzing, and evaluate relevant scientific sources about

algorithm implementation Boyer-Moore towards a system. There are references to previous research regarding implementation of the Boyer-Moore algorithm taken from a written journal by Sara Nasr. In This research, found the best candidates using CV and process fuzzy or uncertain information with berth algorithm [4]. The Boyer-Moore algorithm is an efficient string search method. Discovered by Bob Boyer and J. Strother Moore, it has become a standard in string search literature [1] [6]. Key characteristics of the Boyer-Moore algorithm include right-to-left string matching. This approach allows the algorithm to skip further when encountering mismatches, avoiding unnecessary character comparisons [6].

The *Boyer Moore* algorithm is a *string* search algorithm that matches characters from right to left, and uses two rules to shift the pattern to the right if a mismatch occurs, namely the *bad character rule* and the *good rule suffix* [1]. This algorithm requires two pre-search tables, namely the *bmBc* table and the *bmGs* table, which store the distance between each character or suffix and the end of the pattern. The mathematical formula for the *Boyer Moore* algorithm is

$$s = \max\{bmBc[\text{text}[i]] - m + 1 + j, bmGs[j]\} \quad (1)$$

B. Dataset

Data collection involves obtaining relevant data from both primary and secondary sources. It must be done in a valid, reliable, and ethical manner. In this research, a dataset named “Job Dataset” from Kaggle is used for job vacancy data [5]. The relevant fields include company name, city, country, job title, role, salary, description, skills, and requirements.

C. Requirements

The system requirements for implementing the Boyer-Moore algorithm include allowing users to view all job postings. Users can scan their CV’s against specific job criteria and match their CV data with job descriptions and skills requirements. The matching process involves extracting keywords using YAKE (Yet Another Keyword Extractor) from job descriptions and skills requirements, followed by applying the Boyer-Moore algorithm for accurate matching. The system should display the match percentage for the selected job and provide alternative options with the top 5 match percentages.

D. Planning

Planning refers to the process of creating a plan or sketch for the functions, features, and technologies to be used in a system under development. It involves creativity, innovation, and alignment with user specifications and needs. The design phase encompasses

system blueprints, user and system requirements, sitemaps, flowcharts, and low-fidelity prototypes. Here is the flowchart that defines the web-based system process.

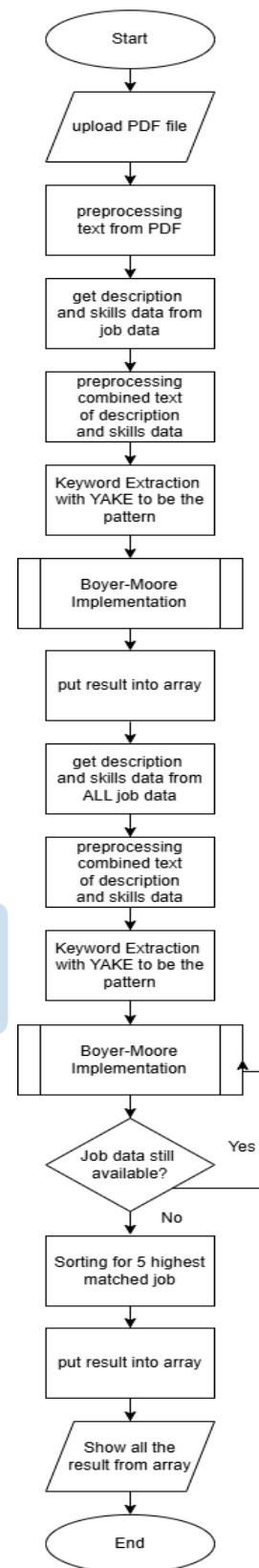


Fig. 1. System Flowchart

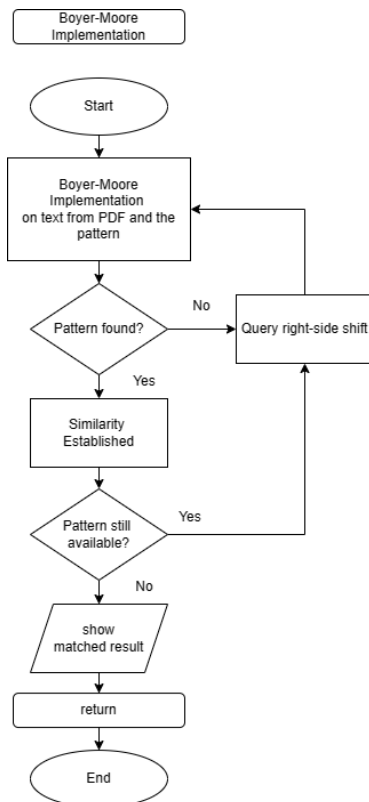


Fig. 2. Boyer-Moore Implementation Flowchart

The system design at Figure 1 System flowchart and Figure 2 Boyer Moore Implmentation, outlines the process for implementing the Boyer-Moore algorithm. It begins by requesting user input in the form of a PDF file. The text within the PDF is then extracted and tokenized, followed by pre- processing (including removing non-alphanumeric characters and converting to uppercase). Next, the system combines job description data with skills data based on the selected job. The resulting combined text is tokenized and preprocessed as well.

For string matching using the Boyer-Moore algorithm, both the text and pattern are needed. The text comes from the extracted and preprocessed data from the PDF, while the pattern is derived from the combined job description and skills text. To obtain the pattern, keyword extraction using YAKE is performed [8]. The Boyer-Moore string matching process begins, adding match percentages when a pattern is found. If no match occurs, a query right-side shift is applied. This process continues until the entire pattern is exhausted. After Boyer-Moore completes, the match percentage is displayed. For example, if 20 keywords were extracted, but only 15 matched, the system shows a 75% match. These percentages are stored in an array.

In the recommendation system, sorting is crucial to display the most relevant results to users. The system proceeds to obtain the top 5 recommendations. Sorting involves taking the match percentages generated by

Boyer-Moore and arranging them from highest to lowest. The same process is applied to all job postings, resulting in 5 additional recommendations. These top 5 matches are also stored in an array. Once all steps are completed, the system displays the array results.

III. RESULT

A. Implementation

The implementation of the Boyer-Moore algorithm within a Python-based web system requires the Flask library. Flask handles user input from the website, processes data using the algorithm, and returns the results to display on the website. The system follows the flowchart design outlined in Figure 1, involving steps such as selecting job postings, uploading documents, processing data, and displaying results along with recommendations.

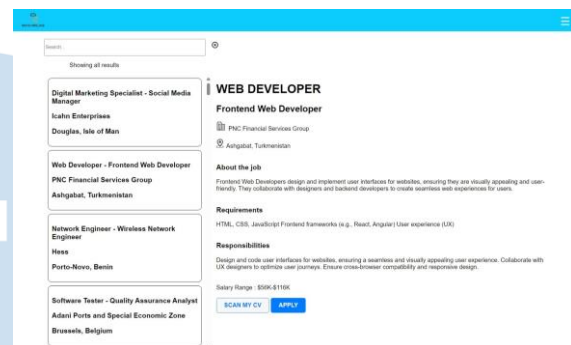


Fig. 3. Job Listings Page

Figure 2 Job Listings Page is the job listing page allows users to view all available job postings. It displays job titles, roles, and locations. When a specific job is selected, detailed information including job descriptions and skills requirements is shown on the right side of the page. Below this information, a "SCAN MY CV" button redirects users to the document upload page.



Fig. 4. Upload Document Page

The "Upload Document" page, as depicted in Figure 4, allows users to upload PDF files related to job applications. It displays job titles, roles, and locations. After selecting a file, users can press the "Scan Now" button to initiate the process. To extract text from a PDF, libraries like PyPDF2 are commonly used [7]. Preprocessing involves tokenization (splitting text into

individual words) and removing stop words (common words with little meaning). The extracted text is combined and then processed further. YAKE (Yet Another Keyword Extractor) is used for keyword extraction [8].

The Boyer-Moore algorithm, an efficient string search method, doesn't require extensive preprocessing, making it suitable for large-scale searches. It compares text and pattern from right to left. Two key heuristics are employed: the bad character rule (adjusting the pattern based on the last mismatched character) and the good suffix rule (shifting the pattern if the mismatched character doesn't appear elsewhere in the pattern). By combining these heuristics, Boyer-Moore achieves efficient pattern matching.

```
class BoyerMooreSearch:
    def __init__(self, text, pattern):
        self.text = text
        self.pattern = pattern
        self.m = len(pattern)
        self.n = len(text)
        self.skip = []
        for i in range(256): self.skip.append(-1)
        for i in range(self.m): self.skip[ord(pattern[i])] = i

    def bad_character_heuristic(self):
        i = 0
        while i <= self.n - self.m:
            j = self.m - 1
            while j >= 0 and self.pattern[j] == self.text[i + j]:
                j -= 1
            if j < 0:
                return [i]
            else:
                i += max(1, j - self.skip[ord(self.text[i + j])])
        return []


bm_search = BoyerMooreSearch(text, word)
positions = bm_search.bad_character_heuristic()
```

Fig. 5. Boyer-Moore Algorithm

BoyerMooreSearch(text, word) : This is the creation of a BoyerMooreSearch object with the text and words (pattern) you want to search for. bad_character_heuristic(): This function is an implementation of the bad character heuristic in the Boyer-Moore algorithm. This function returns the position where the word (pattern) is found in the text. __init__(self, text, pattern): This is the class constructor. This function takes two arguments, text and pattern, which are respectively the text on which the search is performed and the pattern to search for. This function also initializes some variables used in the algorithm. bad_character_heuristic(self): This is a method that implements the bad character heuristic of the Boyer-Moore algorithm. This heuristic speeds up the search by skipping characters that do not match in the pattern. The first loop (while i != self.n - self.m:) moves the search window through the text. The

second loop (while j != 0 and self.pattern[j] == self.text[i + j]:) compares the characters in the pattern and text from right to left. If a pattern is found (if j != 0), the starting index of the pattern in the text is returned. If a non-matching character is found, the search window advances a maximum distance between 1 and j - self.skip[ord(self.text[i + j])].

The results of string matching using the Boyer-Moore algorithm display job titles, positions, company names, locations, and match percentages. After completing the BoyerMooreSearch process, the results are available only for the selected job. To obtain job recommendations with the highest match percentages, testing is performed across all job data stored in the database. The process is similar to before, but without using a unique ID since the query aims to select all data. Iterating through all data, job descriptions and skills requirements are combined and pre-processed. Keyword extraction is then applied, followed by matching using the Boyer-Moore algorithm, resulting in the same format. Additional job recommendations are displayed to provide alternative options that may have higher match percentages than the selected job. Sorting with a Sortation Algorithm which is Timsorts determines the highest match percentage recommendations [9] [10].



Scan Results	
These are the outcomes of the selected job	
Web Developer - Frontend Web Developer by PNC Financial Services Group (Atlanta, Tennessee), matched 48.0%	
These more suggestions might work well on your CV	
Software Engineer - Backend Developer by C# Search (Singapore, Hong Kong), matched 48.0%	
Web Developer - Frontend Web Developer by PNC Financial Services Group (Atlanta, Tennessee), matched 48.0%	
UI Developer - Front-End Developer by Bopcodes (Austin, Texas), matched 48.0%	
Sales Manager - Regional Sales Director by Enterprise Products Partners (Houston, Texas), matched 37.0%	
Sales Manager - Regional Sales Director by Bopcodes (Austin, Texas), matched 37.0%	

Fig. 6. Result Page

Once the top recommendations are obtained, the results are stored in an array and sent to the page depicted in Figure 6, the "Results Page," using the render template function in Flask. The "Results Page" displays job titles, roles, locations, and match percentages. The top result corresponds to the selected job, while the lower section shows additional recommendations with the highest match percentages.

B. Testing

The testing process for the Boyer-Moore algorithm applied to CV data matching with job vacancies. The testing process involves several steps:

- 1) File Upload: Begin by uploading a simple CV in PDF format.
- 2) Processing: Extract the text from the file and convert it into string tokens.
- 3) Text Pre-processing: Perform pre-processing on the extracted text.
- 4) Select Job Vacancy Data: Choose a specific job vacancy to match against.
- 5) Combine Job Description and Skills

Requirements: Pre- process the text from the job description and skills requirements, combining them into a single text.

- 6) Keyword Extraction with YAKE: Use the YAKE algo- rithm to extract keywords. Set the parameter 'n' to 1 to ensure that the generated keywords consist of single words.
- 7) String Matching with Boyer-Moore: Apply the Boyer- Moore algorithm to search for the extracted keywords. The algorithm provides accurate outputs for both "match found" and "no match found" cases.
- 8) Calculate Matching Score: Based on manual testing, there was an 8 out of 20 keyword match, resulting in a 40% match percentage. Additionally, processing one CV against one job vacancy took less than 1 second.

Overall, the system is categorized as fast in displaying results

C. Evaluation

Evaluating an algorithm multiple times is crucial to obtain accurate results. Various factors, such as operating system conditions and processor load, can affect execution time. Therefore, conducting multiple tests ensures consistent and reliable outcomes. While there is no strict rule about the exact number of evaluations, averaging results over multiple runs reduces variability and provides a more accurate performance assessment.

The primary reason for measuring computational time during algorithm evaluation, especially for the efficient Boyer- Moore string search algorithm, is to assess its reliability and speed in real-world applications. In unsupervised learning contexts, where algorithms work with unlabeled data, speed and reliability are critical. Algorithms must identify patterns or structures in data without external assistance.

Computational time measurement is crucial for systems dealing with large or real-time data processing. Reliable and fast algorithms enable accurate recommendations or results within short timeframes. This is valuable in practical applications like job matching, text analysis, and recommendation systems. Therefore, evaluating the Boyer-Moore algorithm considers both accuracy and speed. The following are the table of average evaluation results.

TABLE I
AVERAGE EVALUATION TIME RESULT

Data	CV AndrewChristofer (Time Taken)	CV Lydia (Time Taken)
100	3.05692 s	2.84438 s
200	5.78710 s	5.36939 s
500	15.19051 s	12.92706 s
1000	28.64033 s	26.95702 s

The increase in processing from 100 data to 200 data nearly doubles the required time. From the data, we can infer that processing 100 data takes approximately 3 seconds. Below is the graph generated based on the average evaluation results from Table 1.

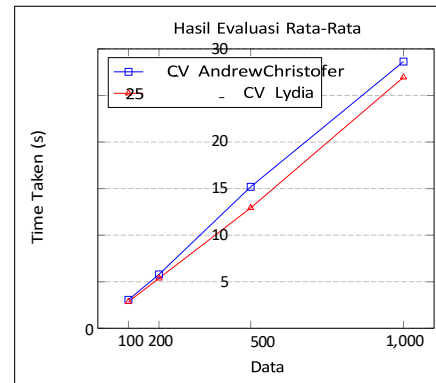


Fig. 7. Average Evaluation Time Result Graph

The graph illustrates the average evaluation results for these two CVs. The x-axis represents the processed data volume, while the y-axis shows the time required for processing. As expected, processing time increases with larger data volumes a common occurrence in computing.

For CV AndrewChristofer (indicated by the blue line), the processing time was approximately:

- 1) 100 data: 3.06 seconds
- 2) 200 data: 5.79 seconds
- 3) 500 data: 15.19 seconds
- 4) 1000 data: 28.64 seconds

Similarly, for CV Lydia (indicated by the red line), the processing time was approximately:

- 1) 100 data: 2.84 seconds
- 2) 200 data: 5.37 seconds
- 3) 500 data: 12.93 seconds
- 4) 1000 data: 26.96 seconds

The difference in processing time between the two CVs diminishes as the data volume increases. This suggests that both CVs perform similarly for large-scale data processing. In summary, while the number of words and characters does not significantly impact computational time, the data volume does affect processing time

IV. CONCLUSIONS

Implementation of the Boyer-Moore algorithm within a website-based system for checking the compatibility of curriculum vitae (CV) with job vacancies is successful. The implementation process involved several key steps. Literature Study, prior to implementation, thorough literature research was conducted to ensure that the chosen approach addressed the problem statement effectively. Data Collection and Analysis, a dataset of job vacancies served as a

reference for evaluating CV compatibility. Additionally, requirements analysis and system design formed the foundation for the website-based system. Algorithm Integration the Boyer-Moore algorithm was integrated into the system, aiming to assess the suitability between CV documents and job vacancy data. Additional algorithms, such as YAKE for keyword extraction and Timsort for sorting, were also utilized. Manual Testing and Evaluation, manual testing verified whether the system's processed data matched the expected output. Finally, performance evaluation measured the time required to process a PDF CV against job vacancy data and provide recommendations based on the highest match percentage.

The evaluation revealed that while word count and character length had minimal impact on computational time, the data volume significantly affected processing time.

REFERENCES

- [1] R. S. Boyer and J. S. Moore, "A Fast String Searching Algorithm," *Communications of the ACM*, vol. 20, no. 10, pp. 762–772, 10 1977
- [2] W. Rytter, "Correctness of the Boyer-Moore Algorithm," *Information Processing Letters*, vol. 9, no. 5, pp. 232–234, 12 1979
- [3] OpenCV. (2024, 2) "Feature Matching". [Online; accessed 17-Feb-2024].
- [4] S. Nasr and O. German, "Resume searching to decide best candidate based on relief method," *Open Science Journal*, vol. 5, no. 2, 2020
- [5] R. S. RANA, "A Comprehensive Job Dataset for Data Science, Research, and Analysis." [Online]. Available: <https://www.kaggle.com/datasets/ravindrasinghrana/job-description-dataset>
- [6] Fitriyah, "The Implementation of Boyer-Moore Algorithm in WEB Based Computer and Informatic Terms Dictionary," in 2020 4th International Conference on Vocational Education and Training (ICOVET), 9 2020
- [7] T. Aggarwal, "PyPdf2: A comprehensive guide to mastering pdf manipulation with python," 2023
- [8] A. M. J. A. J. Ricardo Campos, Ga el Dias, "YAKE: Yet Another Keyword Extractor"
- [9] S. Valdarrama, "Sorting Algorithms in Python"
- [10] GeeksforGeeks, "TimSort - Data Structures and Algorithms Tutorials."
- [11] S. I. Hakak, A. Kamsin, S. Palaiahnakote, and G. A. Gilkar, "Exact String Matching Algorithms: Survey, Issues, and Future Research Directions," *ResearchGate*, 2019
- [12] A. K. Singh, A. K. Singh, and A. K. Singh, "A String Matching Algorithm for Job Searching and Skill Analysis," *International Journal of Computer Applications*, vol. 117, no. 16, pp. 10–14, 5 2015
- [13] R. S. Patil and S. S. Sherekar, "A Novel Approach for Skill Matching in Job Recruitment System Using String Matching Algorithm," *International Journal of Computer Science and Information Technologies*, vol. 5, no. 3, pp. 3627–3630, 2014
- [14] A. K. Singh, A. K. Singh, and A. K. Singh, "A Novel Approach for Word Search Puzzle Game Using String Matching Algorithm," *International Journal of Computer Applications*, vol. 120, no. 19, pp. 1–4, 6 2015.
- [15] OpenCV. (2024, 2) "Feature Matching".
- [16] CodeCrucks. (2024, 2) "String Matching Algorithms - CodeCrucks"
- [17] D. Gusfield, *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge, UK: Cambridge University Press, 1997.
- [18] A. P. A. M. J. C. N. A. J. Ricardo Campos, Vitor Mangaravite, "Unsupervised Multiword Extraction for Keyword Generation."

UMN