Published online on the journal's webpage: **http://journal.ittelkom-pwt.ac.id/index.php/dinda**

Journal of Dinda

Data Science, Information Technology, and Data Analytics

Vol. 5 No. 1 (2025) 36 - 44 E-ISSN: 2809-8064

Optimizing Search Efficiency in Ordered Data: A Hybrid Approach Using Jump Binary Search

Gabriella Youzanna Rorong[1], Syafrial Fachri Pane[2], M Amran Hakim Siregar[3]

[1,2]Applied Bachelor Program of Informatics Engineering, University of Logistics and Business International

[3]Computer Science, Information Technology and Actuarial Science, Universitas Mitra Bangsa

[1]gabriellayouzanna@gmail.com, [2]syafrial.fachri@ulbi.ac.id, [3]amranhakimsiregar@umiba.ac.id

Abstract

This research presents the development of a hybrid algorithm called Jump Binary Search (JBS), which integrates jump search and binary search techniques to improve search efficiency in sorted data distributions. JBS is designed to accelerate the search process using a jump technique to find the target block, after the block is identified, it is followed by a binary search to narrow down the search space. The results of this study show that the performance of JBS is superior compared to Jump Linear Search (JLS) when applied to non-uniform and ordered categorical data distributions. At 400 elements, JBS requires an execution time between 0.008-0.012 ms and at 3200 elements, JBS maintains efficiency with an execution time of 0.010-0.020 ms. The performance of JBS in handling unsorted datasets yields interesting results, compared to JLS which experiences a significant increase in execution time. By minimizing unnecessary data access, JBS becomes the right solution for finding target elements in sorted data distribution.

Keywords: *Jump Search, Binary Search, Linear Search*

1. Introduction

Data archives play an important role in organizations, serving to store information [1]. With the rapid advancement of technology, data archiving has shifted from traditional paper formats to digital formats, enabling more efficient data management and retrieval processes. Digital data archives are organized and structured effectively, significantly improving the speed and quality of information retrieval, which en hances service quality and decision-making processes [2]. Despite these advancements, the challenge of efficiently searching for data through large volumes of information remains, especially with traditional search methods that involve looking one by one.

The inefficiency of manual data collection methods becomes apparent as the volume of data increases [3]. Search ing for data through archives one record at a time is not only time-consuming but also prone to errors and data inconsistencies [4]. Such inefficiencies can cause delays in accessing important information, which ultimately affects the performance and responsiveness of an organization [5]. To address this challenge, the integration of search algorithms into digital archive management systems is becoming increasingly important. These algorithms are designed to streamline the search process, making it faster.

Among various search algorithms, binary search stands out due to its efficiency and simplicity [6]. Binary search performs a search by dividing the data set into smaller parts, allowing for a targeted search within sorted data [7]. With a temporal complexity of O($log_2 n$), the binary search method is highly effective for big data distribution sets [8]. However, its effectiveness can only be used on sorted data, which becomes a limitation in certain case studies. With these limitations, researchers have explored hybrid algorithms, such as Interpolated Binary Search (IBS), which combine the advantages of binary search with interpolation search.

Based on previous research, discussing Interpolated Bi nary Search (IBS), a method that blends interpolation search and binary search. This algorithm is designed to work efficiently on various data distributions, especially with small to medium-sized data. IBS estimates the target element's location using an interpolation technique, then employs binary search to narrow down the search space by looking for the desired target [9].

Received: 22-01-2025 | Accepted: 05-02-2028 | Published: 08-02-2025

The IBS algorithm is very effective for datasets with varied distributions, especially those of small to medium size. Furthermore, this study suggests fusing the binary search algorithm with the jump search algorithm to produce a hybrid solution that capitalizes on both of their advantages. Jump search is effective in minimizing data access by skipping several elements in the data set to find the target block [10]. While binary search efficiently narrows down the target block for the target search [11].

This research aims to explore a hybrid algorithm between jump search and binary search, referred to as Jump Binary Search (JBS), by conducting a comparative study between Jump Binary Search and Jump Linear Search to identify the most efficient method for fast data retrieval. By conducting this research, it is hoped that a more efficient hybrid algorithm solution can be provided.

2. Proposed Jump Binary Search

Two fundamental algorithms for locating a target element in a set of data distributions with variable orders are jump search and binary search. Every algorithm has benefits. The purpose of jump search is to minimize the quantity of com parisons required to locate the target element by jumping forward with a fixed step in the dataset [12]. Using $\sqrt{n}$ of the array's total number of items, this approach first chooses a block step size. The algorithm then uses that block step to skip over the data set, examining each block's components to see if the target is inside it. A binary search is conducted within the block to determine the target's position once the block where the target may be found has been determined.

Binary search repeatedly splits the data into two halves and is intended for a sorted dataset. [13]. Binary search contrasts the step's middle element with the desired value [14]. If the target equals the middle element, the search is complete. If the target is smaller than the center element, the search shifts to the left; if it is larger, the search shifts to the right. With a complexity of O($log_2 n$), binary is perfect for data distribution sets with different orders.

The Jump Binary Search (JBS) algorithm is specifically designed to enhance search efficiency by minimizing the number of data accesses required to find a target within a sorted data distribution set. This algorithm combines the advantages of the jump search and binary search techniques. Jump search is very effective in narrowing down the search space by skipping data with a fixed step, it drastically lowers the quantity of components that must be examined.

$$\text{Step} = | \sqrt{n} |$$

n is the number of index elements in the array. After the block containing the target is found, binary search is used to determine the exact location of the target by comparing the target value with the middle element.

$$\text{Mid} = \text{left} + | \frac{right-left}{2} |$$

Suppose JBS is searching for the name parameter "ma sum" in the sorted data. JBS first performs a search using the jump search approach, then JBS runs a binary search on the remaining index, which most likely contains the desired target. Here is the code for JBS.

Algorithm 1 Jump Binary Search

```
def jump_binary_search(arr, target):
    n = len(arr)
    step = int(math.sqrt(n)) # Jump size
    prev = 0

    # Jump through arrays
    while prev < n and arr[min(step, n)- 1]
        < target:
        prev = step
        step += int(math.sqrt(n))
        if prev >= n:
            return -1

     # Binary Search on found blocks
     left, right = prev, min(step, n)- 1
     while left <= right:
         mid = left + (right- left) // 2
         if arr[mid] == target:
             return mid
          elif arr[mid] < target:
             left = mid + 1
          else:
             right = mid - 1

     return -1
```

The code above shows the implementation of a hybrid algorithm, namely Jump Binary Search (JBS), to efficiently find a target within the elements of a sorted array. The search process begins with the initialization of variables: n represents the length of the elements, and the step size is determined. This step size determines how far the algorithm jumps over elements during the initial search, with the aim of quickly identifying the block where the target element is located, thereby reducing the overall search space.

At the beginning of the search, the jump search algorithm is used as shown in lines 7-11. The algorithm checks each block's final constituent before moving on to the next one. If this element is smaller than the target, the algorithm searches the next block on the right side. This process continues until a block is found where the element equals the target, indicating that the target might be within that block. If the algorithm runs out of items and doesn't find a suitable block, it will return-1 indicating that the array does not contain the target key.

After a block is identified, the algorithm switches to binary search within that block, as shown in lines 14-24. Binary search focuses on the identified block, where the block's borders are shown by left and right, and the mid index is calculated to divide the block into two parts. If the goal is at the mid-index, the algorithm returns this index. If the target is greater than the element at mid, the search moves to the right part of the block. On the other hand, the search moves to the left if the target is smaller. This technique effectively focuses the search to pinpoint the target's precise position.

Combination Jump Binary Search offers significant advantages in terms of efficiency. By using jump search to eliminate most of the elements in the array, the method lowers the amount of components that require close inspection [15]. Meanwhile, binary search within the identified block ensures that the search process runs quickly and accurately [16][17]. This approach is very beneficial in large datasets where traditional search methods would be too slow [18]. The hybrid algorithm effectively balances the need for speed with the accuracy of search results.

Table 1 Step Execution of Jump Binary Search

Index Start	Index End	Middle Index	Key (Target)	Element at Middle Index	Result
0	19	-	masum	adah	key > element
20	39	-	masum	alam setiawan	key > element
40	59	-	masum	asep	key > element
60	79	-	masum	ciah	key > element
80	99	-	masum	emin	key > element
100	119	-	masum	hamid	key > element
120	139	-	masum	ika rahmawati	key > element
140	159	-	masum	juju	key > element
160	179	-	masum	lili riawinata	key > element
180	199	-	masum	mariam	key > element
200	220	210	masum	mintarsih	key < element
200	209	204	masum	masum	key == element

3. Complexity of Jump Binary Search

When the Jump Binary Search (JBS) algorithm is executed, to locate the target in a sorted dataset, it combines binary and jump search methods [19]. Initially, jump search is used which has a complexity of $\sqrt{n}$ [20]. At this stage, data is jumped with fixed steps, and the jump size utilizes the complexity of jump search.

The following step, binary search, has a complexity of $O(\log n)$ [21]. Targeting the targeted element by repeatedly dividing the block in half reduces the search space by utilizing the complexity of binary search [21]. The total difficulty of JBS is a hybrid complexity of $O(\log n)$, which combines the complexity of binary search with the jump search approach. Binary Search's time complexity is greater than Jump Binary Search's overall time complexity because Binary Search is used to refine the search space before Binary Search.

Binary and ternary work on sorted data. Unlike binary, ternary divides the array into three parts to narrow down the search space [22]. The way ternary works is the same as binary, only differing in dividing the array into three parts instead of two, which reduces time complexity [22]. The complexity of ternary search is $O(Log_3 n)$ [22].

4. Limitations of Jump Binary Search

Although Jump Binary Search offers a more efficient search, JBS has some limitations. JBS is highly dependent on a sorted dataset [23]. If the dataset is not fully ordered or contains minor errors in the sequence, JBS efficiency may decrease [23]. This may be due to misidentifying the target block, which can lead to longer search times. In that case, another search method is more flexible, such as linear search.

The distribution and size of the dataset are very important for the efficiency of JBS. The overhead generated from calculating jump steps and transitioning to binary search can be greater for smaller datasets, making traditional binary search more efficient [23]. If the data has a highly skewed distribution, where the target element is concentrated in a specific area, a fixed jump step may not be optimal. In that case, the algorithm that might be more effective for the data distribution is Interpolated Search.

JBS does not require much memory because it is similar to the standard binary search. Because JBS starts with a jump step, the memory access pattern may be less optimal. In other words, the computer may need to retrieve data from a more distant part of the memory, which can slow down the process in some systems [23]. The speed and efficiency of memory in JBS depend on the type of data and hardware used. Although JBS reduces the number of steps, slow memory access can decrease efficiency.

5. Used of The Jump Binary Search

In Table, the Jump Binary Search (JBS) algorithm is shown using a sorted dataset with the parameter "name" [24]. The table outlines the search process by displaying the attributes of index start, index end, middle index, key (target), element at middle index, and result at each step of the search. The start index marks the beginning of the block being examined, while the end index marks the end of the block. The middle index is very important because it helps divide the block into two parts, thereby efficiently narrowing the search space [25].

The target element referred to as the key is the focus of the search [26][27]. The element at the middle index is compared with the key to determine the direction of the next search step. A key that is larger than the middle index element is searched on the right side of the index. But if the key is smaller than the element, the search shifts to the left. When the key corresponds to the element at the middle index, the search is successful and the algorithm delivers the target element's index.

Example search "masum", the JBS algorithm starts by skipping the dataset with a step of 19 indices.

$$\text{Step} = |\sqrt{n}|$$

$$\text{Step} = |\sqrt{399}| = 19{,}9 = 19$$

That stage aims to find the block where the target is located. After the target block is identified, binary search continues the search. In this example, the search narrows down to the block starting at index 200 and ending at index 220. In this case, the middle index is calculated to be 210, because the key is smaller than the element at this index, the search continues in the left part of the block.

$$\text{Mid} = \text{left} + \left|\frac{right-left}{2}\right|$$

$$\text{Mid} = 200 + \left|\frac{220-200}{2}\right|$$

$$= 200 + \frac{20}{2}$$

$$= 200 + 10 = 210$$

When the search is conducted within the index between 200 and 209, the middle index is calculated to be 204.

$$\text{Mid} = \text{left} + \left|\frac{right-left}{2}\right|$$

$$\text{Mid} = 200 + \left| \frac{209 - 200}{2} \right|$$

$$= 200 + \frac{9}{2}$$

$$= 200 + 4.5 = 204.5 = 204$$

When the middle index equals the target, the key equals the element, the search is successfully found by returning the index of the target element "masum". Proving that JBS effectively combines jump search and binary search to minimize search time. The table shows that the middle index and comparison results change as the search gets closer to the target.

6. Comparison of JBS And JLS

The Jump Linear Search algorithm, or JLS, is an innovative approach designed to enhance search efficiency by combining jump search and linear search. JLS operates on unordered data, which distinguishes it from the Jump Binary Search (JBS) algorithm that requires an ordered dataset [28]. The JLS algorithm starts by skipping elements with a fixed step, searching for a block that contains the target element [29]. Once the right block has been found, JLS searches linearly inside it to determine the target's precise location. This method leverages the simplicity of linear search with a jump technique on an unordered dataset.

One of the striking differences between JLS and JBS is how each algorithm handles the identified blocks [30]. JLS, after finding the target block, performs a linear search without dividing the block into smaller parts [31]. Unlike JBS, which uses binary search to divide the block into two parts, optimizing the search process. The JBS algorithm is very efficient for sorted data distribution.

The main advantage of JLS is its flexibility and simplicity. This is beneficial in cases of unordered data due to time constraints. Making JLS valuable in the search process. Here is the Python code that implements JLS and JBS.

Algorithm 2 Jump Binary Search

```
def jump_binary_search(arr, target):
    n = len(arr)
    step = int(math.sqrt(n)) # Jump size
    prev = 0

    # Jump through arrays
    while prev < n and arr[min(step, n)- 1]
        < target:
        prev = step
        step += int(math.sqrt(n))
        if prev >= n:
            return -1

    # Binary Search on found blocks
    left, right = prev, min(step, n)- 1
    while left <= right:
        mid = left + (right- left) // 2
        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            left = mid + 1
        else:
            right = mid - 1

    return -1
```

Algorithm 3 Jump Linear Search

```
def jump_linear_search_unordered(arr, target):
    n = len(arr)
    step = int(math.sqrt(n)) # Jump size
    prev = 0

    # Jump through arrays
    while prev < n:
        # Check each element in the current
            block
        for i in range(prev, min(prev + step ,
            n)):
            if arr[i] == target:
                return i # Element found

        # Move to the next block
        prev += step

    # Element not found
    return -1
```

7. Results And Comparison

The algorithm in this research was implemented in Python using Google Colab as the platform to run and test the code. The experiment focused on the distribution of sorted data with random search keys [32]. Jump Binary Search, the data needs to be sorted to maximize the efficiency of the algorithm [33]. While Jump Linear Binary Search is used on unsorted data [34]. The dataset used has the same size and quantity to ensure a fair comparison [35]. In this case study, the elements consist of small to medium datasets ranging from 400 to 3200 elements in the array.

To measure the performance difference between JBS and JLS, execution time measurements were made [36]. Execution time is used to measure how quickly an algorithm can complete its search task [9][37]. Execution time is generated based on the size of the data and the performance of the algorithm [9][38]. In this case study, repeated measurements were conducted to minimize errors and obtain more accurate results.

6.1. *Performance Test On Non-Uniform Distribution*

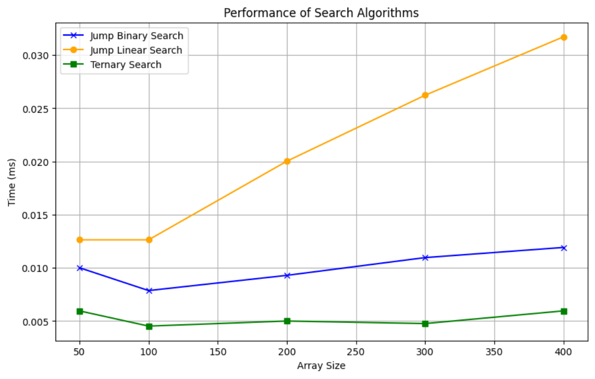


Figure 1 Execution-time of JBS, JLS and Ternary on a small array size

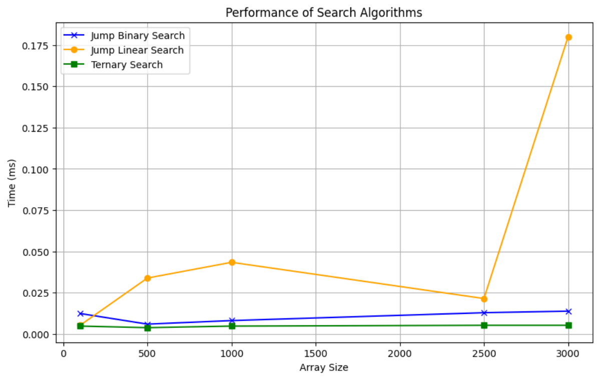


Figure 2 Execution-time of JBS, JLS and Ternary on a large array size

Figures 1 and 2, show a comparison of execution times for jump binary search, jump linear search, and ternary search. In Figure 1, JBS is better than JLS, with an execution time of 0.008-0.012 ms. This indicates that JBS is more efficient in handling sorted data than JLS, which takes longer to check elements one by one, with an execution time range of 0.013-0.030 ms. Meanwhile, ternary search shows the best performance among the three algorithms, with lower execution time.

Figure 2, shows the performance of the three algorithms on a larger dataset. As the dataset size increases, JLS shows a significant increase in execution time, but JBS remains stable and efficient with consistently low execution times, slightly higher than ternary search but much better than JLS. This demonstrates JBS's superiority in handling sorted datasets, where its optimal and stable search strategy excels.

In figure 3, when tested on unordered data, JBS remains efficient. In this case, JBS usually requires sorted data, but it shows stable execution times in the range of 0.010-0.015 ms, which is almost the same as ternary search and much better than linear jump search. This shows that JBS is still a suitable choice for some unordered datasets, offering a balance between speed and accuracy.

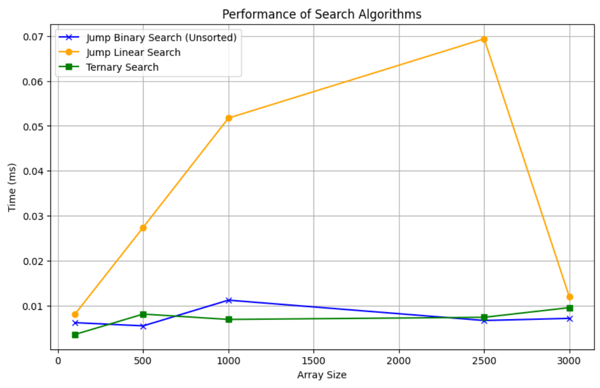


Figure 3 Execution time of JBS, JLS, and Ternary on large unsorted array size

6.2. *Performance Test On Numerical Distribution*

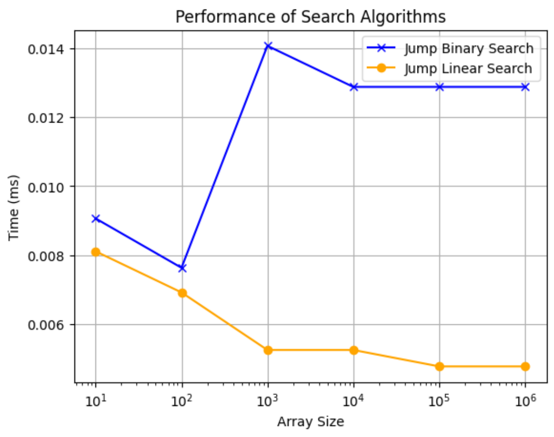


Figure 4 Execution-time of JBS and JLS on numerical distribution

Figure 4, shows the execution time of JBS and JLS when applied to a numerical distribution. The results show that JLS is slightly superior to JBS in terms of speed. JLS completed its execution in the range of 0.003-0.004ms, while JBS took slightly longer, with execution times ranging from 0.004-0.0011ms. The difference indicates the advantage of using JLS for numerical distribution, where its search strategy is slightly more optimal.

6.3. *Performance Test On Categorical Distribution*

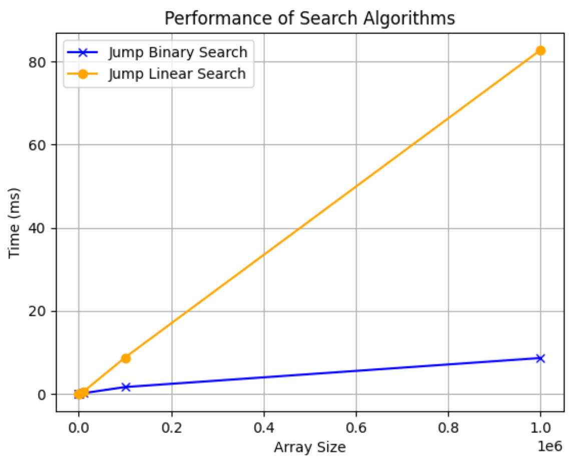
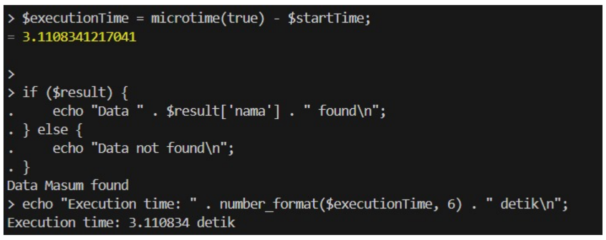


Figure 5 Execution-time of JBS and JLS on categorical distribution

Figure 5, shows the execution time of JBS and JLS when applied to a categorical distribution. The results show that JBS is superior to JLS in terms of speed. JBS completes its execution in the range of 0-10ms, while JLS takes much longer, with execution times ranging from 0-80ms. The difference indicates the advantage of using JBS for the distribution of sorted categorical data.

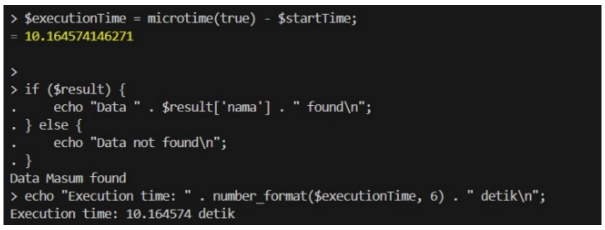
8. Case Study on Implementation JBS



```
> $executionTime = microtime(true) - $startTime;
= 3.1108341217041

>
> if ($result) {
.     echo "Data " . $result['nama'] . " found\n";
. } else {
.     echo "Data not found\n";
. }
Data Masum found
> echo "Execution time: " . number_format($executionTime, 6) . " detik\n";
Execution time: 3.110834 detik
```

Figure 6 Execution Time Using The JBS Algorithm



```
> $executionTime = microtime(true) - $startTime;
= 10.164574146271

>
> if ($result) {
.     echo "Data " . $result['nama'] . " found\n";
. } else {
.     echo "Data not found\n";
. }
Data Masum found
> echo "Execution time: " . number_format($executionTime, 6) . " detik\n";
Execution time: 10.164574 detik
```

Figure 7 Execution Time Without Using The JBS Algorithm

This case study applies the jump binary search algorithm to the data management system to improve the efficiency of searching for social assistance recipients data. This algorithm begins by calculating the step size as the square root of the number of elements in the array that will be jumped over. Then, at each jump, the algorithm checks the last element of the block that was skipped. The algorithm continues to the next block if that element is smaller than the target until it finds a block that might contain the target. After the block is precisely defined, the search switches to binary search within that block to find the exact position of the target.

This implementation shows a significant increase in efficiency, with an execution time of 3.11s to find social assistance recipient data with the name "Masum", while the time required to find data without JBS is 10.16s. The comparison shows the superiority of JBS in improving search efficiency and indicates that JBS is the best choice for searching within the dataset for data management systems. JBS improves the performance of the data management system by combining the speed of jump search with the accuracy of binary search.

9. Conclusion

This research introduces a hybrid algorithm called Jump Binary Search (JBS), which effectively combines jump search and binary search techniques. Designed for sorted data distributions, JBS uses the jump search method to quickly identify blocks that are likely to contain the target element. After the block is found, the algorithm switches to binary search to narrow down the space by dividing the block into two parts. This approach significantly improves the speed and accuracy of the search, making JBS an excellent choice for large sorted datasets.

The research results show that the speed of JBS is superior in non-uniform distributions and categorical distributions. The execution time of JBS, which ranges from 0.008-0.012ms in non-uniform distribution and 0-10ms in categorical distribution, demonstrates its efficiency compared to the Jump Linear Search algorithm. By minimizing unnecessary data access, JBS becomes an appropriate algorithmic solution for finding target elements in non-uniform and categorical data distributions.

Acknowledgments

The researcher would like to thank the supervisor for his guidance during the experiments and compiling the journal, many suggestions helped the researcher in compiling this research.

References

[1] A. R. Kunduru and R. Kandepu, “Data archival

methodology in enterprise resource planning applications (Oracle ERP, Peoplesoft)," *J. Adv. Math. Comput. Sci.*, vol. 38, no. 9, pp. 115–127, 2023.

[2] H. Benmakhlouf and A. Chouaou, "Electronic document, information, and archive management systems in economic institutions: A descriptive study of the onbase system," *Int. J. Prof. Bus. Rev. Int. J. Prof. Bus. Rev.*, vol. 9, no. 6, p. 11, 2024.

[3] A. G. Putrada, N. Alamsyah, S. F. Pane, M. N. Fauzan, and D. Perdana, "VANET Severity Classification in BSM Messages with a Novel Na{\"\i}ve Bayes Feature Selection," in *2023 International Conference on Advancement in Data Science, E-learning and Information System (ICADEIS)*, 2023, pp. 1–6.

[4] M. Devan, L. Shanmugam, and M. Tomar, "AI-powered data migration strategies for cloud environments: Techniques, frameworks, and real-world applications," *Aust. J. Mach. Learn. Res. \& Appl.*, vol. 1, no. 2, pp. 79–111, 2021.

[5] M. O. Ezeh, A. D. Ogbu, and A. Heavens, "The Role of Business Process Analysis and Re-engineering in Enhancing Energy Sector Efficiency." 2023.

[6] M. Macedo *et al.*, "Overview on binary optimization using swarm-inspired algorithms," *IEEE Access*, vol. 9, pp. 149814–149858, 2021.

[7] X. Bai and C. Coester, "Sorting with predictions," *Adv. Neural Inf. Process. Syst.*, vol. 36, pp. 26563–26584, 2023.

[8] S. Morshtein, R. Ettinger, and S. Tyszberowicz, "Verifying time complexity of binary search using Dafny," *arXiv Prepr. arXiv2108.02966*, 2021.

[9] A. S. Mohammed, \cSahin Emrah Amrahov, and F. V Çelebi, "Interpolated binary search: An efficient hybrid search algorithm on ordered datasets," *Eng. Sci. Technol. an Int. J.*, vol. 24, no. 5, pp. 1072–1079, 2021.

[10] L. Liu *et al.*, "Global dynamic path planning fusion algorithm combining jump-A* algorithm and dynamic window approach," *IEEE Access*, vol. 9, pp. 19632–19638, 2021.

[11] X. Hao and B. Chandramouli, "Bf-tree: A modern read-write-optimized concurrent larger-than-memory range index," *Proc. VLDB Endow.*, vol. 17, no. 11, pp. 3442–3455, 2024.

[12] M. Braik, A. Sheta, and H. Al-Hiary, "A novel meta-heuristic search algorithm for solving optimization problems: capuchin search algorithm," *Neural Comput. Appl.*, vol. 33, no. 7, pp. 2515–2547, 2021.

[13] S. Kumar, A. Shailu, A. Jain, and N. R. Moparthi, "Enhanced method of object tracing using extended Kalman filter via binary search algorithm," *J. Inf. Technol. Manag.*, vol. 14, no. Special Issue: Security and Resource Management challenges for Internet of Things, pp. 180–199, 2022.

[14] R. Gomez-Merchan *et al.*, "Binary search based flexible power point tracking algorithm for photovoltaic systems," *IEEE Trans. Ind. Electron.*, vol. 68, no. 7, pp. 5909–5920, 2020.

[15] Y. Du, "Multi-UAV Search and Rescue with Enhanced A∗ Algorithm Path Planning in 3D Environment," *Int. J. Aerosp. Eng.*, vol. 2023, no. 1, p. 8614117, 2023.

[16] L. Zhou, X. Bai, X. Liu, J. Zhou, and E. R. Hancock, "Learning binary code for fast nearest subspace search," *Pattern Recognit.*, vol. 98, p. 107040, 2020.

[17] H. Qin, R. Gong, X. Liu, X. Bai, J. Song, and N. Sebe, "Binary neural networks: A survey," *Pattern Recognit.*, vol. 105, p. 107281, 2020.

[18] S. F. Pane, R. M. Awan, M. A. H. Siregar, and D. Majesty, "Mapping log data activity using heuristic miner algorithm in manufacture and logistics company," *TELKOMNIKA (Telecommunication Comput. Electron. Control.*, vol. 19, no. 3, pp. 781–791, 2021.

[19] A. G. Gad, K. M. Sallam, R. K. Chakrabortty, M. J. Ryan, and A. A. Abohany, "An improved binary sparrow search algorithm for feature selection in data classification," *Neural Comput. Appl.*, vol. 34, no. 18, pp. 15705–15752, 2022.

[20] D. Antipov, B. Doerr, and V. Karavaev, "A rigorous runtime analysis of the (1+($λ$, $λ$)) GA on jump functions," *Algorithmica*, vol. 84, no. 6, pp. 1573–1602, 2022.

[21] D. Harvey and J. Van Der Hoeven, "Integer multiplication in time O(nlog$\backslash$,n)," *Ann. Math.*, vol. 193, no. 2, pp. 563–617, 2021.

[22] Y. Huang, L. Lai, W. Li, and H. Wang, "A

differential evolution algorithm with ternary search tree for solving the three-dimensional packing problem," *Inf. Sci. (Ny).*, vol. 606, pp. 440–452, 2022.

[23] T. Kumar, J. Park, M. S. Ali, A. F. M. S. Uddin, J. H. Ko, and S.-H. Bae, "Binary-classifiers-enabled filters for semi-supervised learning," *IEEE Access*, vol. 9, pp. 167663–167673, 2021.

[24] S. F. Pane, A. G. Putrada, N. Alamsyah, and M. N. Fauzan, "A PSO-GBR solution for association rule optimization on supermarket sales," in *2022 seventh international conference on informatics and computing (ICIC)*, 2022, pp. 1–6.

[25] J. Kepner *et al.*, "Fast mapping onto census blocks," in *2020 IEEE High Performance Extreme Computing Conference (HPEC)*, 2020, pp. 1–8.

[26] H. Alibrahim and S. A. Ludwig, "Hyperparameter optimization: Comparing genetic algorithm against grid search and bayesian optimization," in *2021 IEEE Congress on Evolutionary Computation (CEC)*, 2021, pp. 1551–1559.

[27] J. Zhang, R. Xie, Y. Hou, X. Zhao, L. Lin, and J.-R. Wen, "Recommendation as instruction following: A large language model empowered recommendation approach," *ACM Trans. Inf. Syst.*, 2023.

[28] E. Sopov and others, "A Novel Binary DE Based on the Binary Search Space Topology," *Eur. Proc. Comput. Technol.*, 2022.

[29] K. K. Gola, S. Kumar, T. Jain, N. Jee, S. Kushwaha, and N. Jain, "Odd even: A hybrid search technique based on bi-linear and jump search," in *AIP Conference Proceedings*, 2023, vol. 2917, no. 1.

[30] T. Chen, S. Chen, K. Zhang, G. Qiu, Q. Li, and X. Chen, "A jump point search improved ant colony hybrid optimization algorithm for path planning of mobile robot," *Int. J. Adv. Robot. Syst.*, vol. 19, no. 5, p. 17298806221127952, 2022.

[31] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2022.

[32] S. F. Pane, J. Ramdan, A. G. Putrada, M. N. Fauzan, R. M. Awangga, and N. Alamsyah, "A hybrid cnn-lstm model with word-emoji embedding for improving the twitter sentiment analysis on indonesia's ppkm policy," in *2022 6th International Conference on Information Technology, Information Systems and Electrical Engineering (ICITISEE)*, 2022, pp. 51–56.

[33] J. Erickson, *Algorithms*. 2023.

[34] G. B. Balogun, "A Comparative Analysis of the Efficiencies of Binary and Linear Search Algorithms.," *African J. Comput. \& ICT*, vol. 13, no. 1, 2020.

[35] A. G. Putrada, N. Alamsyah, S. F. Pane, M. N. Fauzan, and D. Perdana, "AUC Maximization for Flood Attack Detection on MQTT with Imbalanced Dataset," in *2023 International Conference on Information Technology Research and Innovation (ICITRI)*, 2023, pp. 133–138.

[36] A. G. Putrada, N. Alamsyah, S. F. Pane, and M. N. Fauzan, "Xgboost for ids on wsn cyber attacks with imbalanced data," in *2022 International Symposium on Electronics and Smart Devices (ISESD)*, 2022, pp. 1–7.

[37] A. K. Mishra, S. Mohapatra, and P. K. Sahu, "Adaptive Tasmanian Devil Optimization algorithm based efficient task scheduling for big data application in a cloud computing environment," *Multimed. Tools Appl.*, pp. 1–20, 2024.

[38] T. M. Ghazal, "Performances of k-means clustering algorithm with different distance metrics," *Intell. Autom. \& Soft Comput.*, vol. 30, no. 2, pp. 735–742, 2021.