

Interpolation Extrapolation Search: An Optimized Hybrid Algorithm for Data Retrieval in Management Applications

Gabriella Youzanna Rorong¹, Syafril Fachri Pane^{2*}, Roni Habibi³

^{1,2,3}Applied Bachelor of Informatics Engineering, Universitas Logistik dan Bisnis Internasional, Bandung, West Java, Indonesia

E-mail: ¹gabriellayouzanna@gmail.com, ^{2*}syafril.fachri@ulbi.ac.id, ³roni.habibi@ulbi.ac.id

(Received: 25 Jun 2025, revised: 29 Jul 2025, accepted: 5 Aug 2025)

Abstract

The escalating volume and often irregular structure of social assistance data pose significant challenges for efficient data retrieval in management systems. Traditional search algorithms, such as linear and binary search, frequently encounter limitations when handling these large-scale datasets. This research conducts a comparative study between two hybrid algorithms, Jump Binary Search (JBS) and Interpolation Extrapolation Search (IES), aiming to identify the most effective method for a web-based social assistance data management system. Evaluations were performed on a dataset comprising 480 names of social assistance recipients, measuring the number of iterations, execution time, and search accuracy. The results demonstrate IES's superiority over JBS in both iteration efficiency and execution speed. IES exhibited an execution time ranging from 0.002 to 0.006 ms, whereas JBS had an execution time ranging from 0.015 to 0.039 ms. Based on these findings, IES was successfully implemented into a Laravel-based application utilizing a MySQL database. This system is capable of executing searches in less than one second per request. This implementation significantly enhances the system's adaptability and provides an effective search solution for dynamic, large-scale data environments, offering rapid and efficient access to data.

Keywords: Hybrid Algorithm, Interpolation Extrapolation Search, Web Application

I. INTRODUCTION

The current digital era has witnessed rapid advancements in information technology, fundamentally transforming data management practices [1]. However, the ever-increasing volume and dynamic nature of data, particularly within social assistance systems, present significant challenges for efficient management and access [2]. Effective social assistance data management, encompassing recipient names, assistance types, and eligibility, is crucial for targeted aid, minimizing misuse and duplication, and facilitating accurate decision-making [3]. A primary challenge in these applications lies in implementing efficient data retrieval processes.

Despite their everyday use, traditional search algorithms, such as linear, binary, ternary, and interpolation search, have drawbacks [4]. On large datasets, linear search is sluggish, although it can be used on unsorted data [5]. Binary search is effective on sorted data but suboptimal for non-uniform datasets [6]. Consequently, a hybrid algorithm that combines the advantages of various algorithms is necessary to optimize data search in social assistance databases, significantly reducing access time and improving search efficiency by adapting to data characteristics [7]. This study addresses

explicitly the underexplored area of applying hybrid search algorithms in real-world, web-based social assistance data management systems, offering a practical solution for large and irregular datasets.

This research proposes two hybrid algorithms, namely Jump Binary Search (JBS) and Interpolation Extrapolation Search (IES). JBS combines jump search and binary search algorithms to minimize comparisons by first narrowing down the search block and then precisely locating the target element within it [8]. Meanwhile, IES is designed to overcome the limitations of interpolation search on non-uniform data by integrating interpolation and extrapolation to balance search boundaries and accelerate convergence [9].

Several previous studies have proposed improvements to traditional search methods, including hybrid algorithms such as Interpolated Binary Search (IBS), which combine interpolation and binary strategies to reduce iteration counts [10]. However, these studies are often limited to theoretical or simulation-based evaluation. However, the application of hybrid algorithms like JBS and IES in real-world, web-based systems, especially for irregular and large-scale datasets such as social assistance data, remains underexplored [8]. The implementation of search algorithms in social assistance data



management applications is still rarely researched specifically [11]. The research conducts a comparative test between the JBS and IES algorithms to determine the most optimal search algorithm. This research is novel because its comparative testing results will inform the implementation of the most effective algorithm in a web-based social assistance data management application, thereby accelerating data retrieval and enhancing data management accuracy.

This research aims to conduct a comparative study between Jump Binary Search (JBS) and Interpolation Extrapolation Search (IES), evaluating their performance based on the number of iterations, execution time, and data accuracy. The most optimal algorithm identified from this comparison will then be implemented in a web-based social assistance data management application. This study is expected to provide a practical and efficient search algorithm solution for data retrieval in large-scale social assistance data management applications.

A. Paper Structure

This research is structured into several sections, namely, Section I discusses the background of the problem, objectives, and scope of the study. Section II explains the system architecture design and the working mechanism of the proposed JBS and IES algorithms. Section III presents the comparative testing results between JBS and IES, as well as the implementation of the selected algorithms within a Laravel-based application. Finally, Section IV concludes the findings and provides recommendations for future system development.

II. RESEARCH METHODOLOGY

A. System Architecture Design

The social assistance data management application developed in this research is designed as a web-based system consisting of a frontend, backend, and database [12]. This system is designed using the Laravel 11 framework as the backend and MySQL as the database management system. The frontend serves as the medium of interaction between the user and the system. Users can search for social assistance recipient data by entering the name parameter. After the user submits the request, the backend will process the request by running the search algorithm that will be used.

Database data undergoes preparation before the search, which may involve sorting, eliminating duplicates, or value transformation to meet algorithm requirements. Once the data is ready, the backend will run the search algorithm. This process generates search results that are returned to the frontend in the form of structured information.

The architectural design in Figure 1 allows the application to handle search processes with large-scale data and diverse value distributions [13]. This can provide execution time efficiency in data searches within the database [14]. Thus, this architecture supports the achievement of the main research objective, which is to implement efficient search algorithms in social assistance data management applications [15].

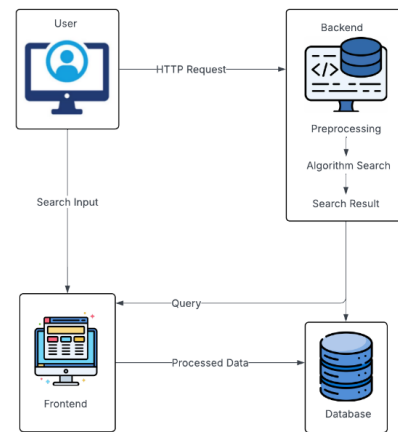


Figure 1. Architecture of The Search Process System

B. Jump Binary Search Algorithm

A hybrid search algorithm, the Jump Binary Search (JBS) algorithm, combines the benefits of binary and jump search methods [16]. JBS was developed to enhance data search efficiency on sorted datasets by minimizing the number of data accesses required during the search process, with a time complexity of $O(\log n)$ [8]. JBS performs the search by executing a jump search to determine jumps with a predetermined step size [17]. The number of items in the dataset, n determines the step size in jump search. Finding the block that could contain the target element is the goal of this stage.

$$\text{Step} = \lceil \sqrt{n} \rceil \quad (1)$$

In Formula 1, step indicates the length of each jump in the array and n is the total number of elements in the dataset. This value will be used to estimate the block where the target is likely to be located. Binary search looks into the target block after it has been located to determine the precise location of the target component [18]. To find the target element in that block, binary search looks for the middle index of the block [19]. The target value and the middle element of the interval are compared in binary search. If the target equals the middle element, the search is complete. For smaller targets, the search moves to the left; for bigger targets, it moves to the right [8].

$$\text{Mid} = \text{left} + \left\lceil \frac{\text{right} - \text{left}}{2} \right\rceil \quad (2)$$

In Formula 2, the mid value is used to compare the middle element with the target. The values of right and left are the index values of each block based on the number of steps. If the target value is smaller, the search continues to the left. If the target value is larger, it continues to the right. In large datasets, JBS outperforms binary search. This is because jump search in JBS can minimize the number of accesses to elements. Although efficient, JBS is highly dependent on sorted data, which is a limitation [20]. JBS does not work optimally on unordered datasets, as it introduces minor sorting errors that compromise the block identification process [21]. In addition,

JBS can experience overhead when used on small-sized datasets. Here is the JBS implementation code to efficiently find the target in sorted elements [8].

Algorithm 1. Jump Binary Search

```
def jump_binary_search(arr, target):
    n = len(arr) # Elements in the array
    step = int(math.sqrt(n)) # Jump size
    prev = 0
    # Perform jump search to find the block
    while prev < n and arr[prev] < target:
        prev = step
        step += int(math.sqrt(n))
    if prev >= n:
        return -1
    # Perform binary search identified block
    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
```

C. Interpolation Extrapolation Search Algorithm

An enhancement of the search for interpolation method is the Interpolation Extrapolation Search (IES) algorithm. Uneven data distribution leads to poor performance, and IES is designed to address this issue. Its time complexity is $O(\log \log n)$ [9]. IES integrates the principles of interpolation search with extrapolation search, thereby accelerating the convergence point of the search even when the data is not equally dispersed [22].

To find the target element's position, IES does an interpolation search based on the input value distribution. Unlike interpolation search, IES will adjust the lower or upper bounds using extrapolation search [9].

$$X_s = X_l - \left[\frac{f(X_l) - s}{f(X_u) - f(X_l)} \right] [X_u - X_l] \quad (3)$$

In Formula 3, x_s the estimated index of the interpolation result. x_l and x_u are the lower and upper bounds of the array. $f(x_l)$ and $f(x_u)$ are the data values at those indices, where T is the target value being sought. When interpolation search fails to make significant progress, IES initializes the lower bound x_l and the upper bound x_u . It then estimates the target element's location x_s by performing an initial interpolation. If stagnation occurs, and the target is close to the lower or upper bound, then extrapolation search is performed in the opposite direction [9]. If stagnation is detected and the target is near the lower limit, extrapolation is performed from the lower direction as per Formula 4.

$$x_l^e = x_l + \left[\frac{s - f(x_l)}{f(x_u) - f(x_s)} \right] [x_s - x_l] \quad (4)$$

In Formula 4, x_l^e the result of an estimation using extrapolation, where p is the step control constant ranging from 0.1 to 0.5. If stagnation occurs near the upper limit, the formula, as in Formula 5 is used.

$$x_u^e = x_u + \left[\frac{f(x_l) - s}{f(x_u) - f(x_s)} \right] [x_u - x_s] \quad (5)$$

In Formula 5, x_u^e the search for the upper bound, with q as a constant like p , but in the upward direction. Once the element has been located or the search period has been used up, adjust the boundary values and continue searching. This is the formula used to choose the new index based on condition flag F [9].

$$x_s := \begin{cases} \min(x_l^e, x_s + p(x_u - x_l)), & \text{if } F = 1 \\ \min(x_u^e, x_s - q(x_u - x_l)), & \text{if } F = -1 \\ x_l - \left[\frac{f(x_s) - s}{f(x_u) - f(x_l)} \right] [f(x_u) - f(x_l)], & \text{otherwise} \end{cases} \quad (6)$$

In Formula 6, to determine the following search index, the algorithm evaluates the value of flag F , which indicates the search direction. If $F = 1$ for stagnation at the lower bound, $F = -1$ for stagnation at the upper bound, and $F = 0$ if no stagnation is found.

Algorithm 2. Interpolation Extrapolation Search

```
def ies_search_string(encoded_data,
    target_encoded, p=0.3, q=0.3):
    xl, xu = 0, len(encoded_data) - 1 # Bounds arr
    F = 0 # Direction flag
    xs = -1 # Estimated position target
    iterations = 0 # Iteration counter
    # Continue search lower bound <= upper bound
    while xl <= xu:
        iterations += 1
        f_xl, f_xu = encoded_data[xl],
            encoded_data[xu]
        # Handle stagnation near lower bound
        # Perform extrapolation
        if F == 1 and encoded_data[xs] != f_xl:
            xel = int(xl + ((target_encoded -
                f_xl) / (encoded_data[xs] - f_xl)) * (xs -
                    xl))
            xs = min(xel, xs + int(p * (xu - xl)))
        # Handle stagnation near upper bound
        elif F == -1 and f_xu !=
            encoded_data[xs]:
            xeu = int(xu - ((f_xl -
                target_encoded) / (f_xu -
                    encoded_data[xs])) * (xu - xs))
            xs = min(xeu, xs - int(q * (xu -
                xl)))
        # Step interpolation search
        else:
            if f_xu == f_xl:
                break
            xs = xl + int(((target_encoded - f_xl) *
                (xu - xl)) / (f_xu - f_xl))
        # Check estimated position out of bound
        if xs < xl or xs > xu:
            break
        f_xs = encoded_data[xs]
```

```

# Compare with target and adjust bound
if f_xs < target_encoded:
    xl = xs + 1
    F = 1
elif f_xs > target_encoded:
    xu = xs - 1
    F = -1
else:
    return xs, iterations

return -1, iterations

```

Based on previous research, IES was tested on six datasets with non-uniform distribution, showing faster execution time compared to binary search and interpolation search [9]. The number of iterations is fewer, especially for non-linear data such as square root, exponential, and trigonometric distributions. Here is the IES implementation code for the test [9].

However, IES also has limitations that require dynamic flag conditions and updates (F), as well as interpolation and extrapolation calculations that are more complex than binary search [23]. The performance of the algorithm can depend on the values of the constants p and q ; incorrect values can slow down the search process [24]. If the data distribution is very chaotic or contains noise, the performance of IES becomes less optimal.

III. RESULTS AND DISCUSSION

A. Comparison of JBS And IES

Initial testing assessed the effectiveness of Jump Binary Search (JBS) and Interpolation Extrapolation Search (IES) in locating social assistance recipient data. Tests were conducted using Google Colab with sorted data [25]. The dataset used is social assistance recipient data, containing 480 elements. The search algorithm code in Python is run on Google Colab. The test measures parameters such as the number of iterations and execution time.

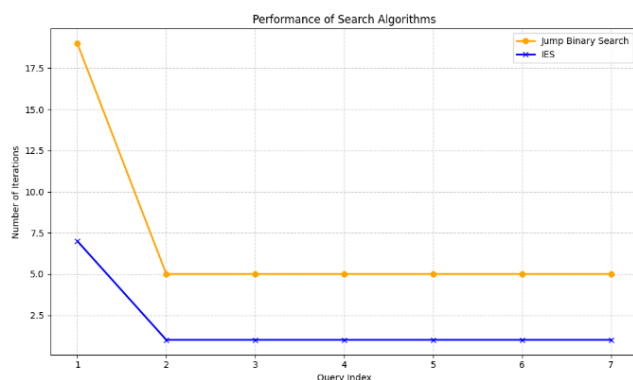


Figure 2. The Number of Iterations of JBS and IES

Figure 2 presents a comparative graph illustrating the number of iterations performed by JBS and IES across seven randomly selected recipient names. The JBS algorithm exhibits a significant decrease from 19 iterations in the first search, stabilizing at 5 iterations from the second to the seventh search. In contrast, IES consistently maintains a lower number

of iterations compared to JBS, with 7 iterations in the first search and stabilizing at 1 iteration in each search from the second to the seventh. In terms of the number of iterations used, it is evident that IES is superior in achieving convergence towards the target, even when the data distribution is not uniform [26].

Figure 3 displays a comparative graph of the Execution Time for the JBS and IES algorithms. This test was conducted with varying array sizes from 100 to 400 elements. Both algorithms search for data using the name parameter, each looking for one input name. The test showed that the execution time of JBS in searching for the desired data required about 0.039 ms at a size of 100 elements, decreasing to about 0.015 ms at a size of 400. Meanwhile, IES demonstrated stable and consistently lower execution times compared to JBS at all array sizes, requiring execution times of 0.002 to 0.006 ms. The test indicated that the IES algorithm still outperformed JBS in data searching based on the execution time parameter [27].

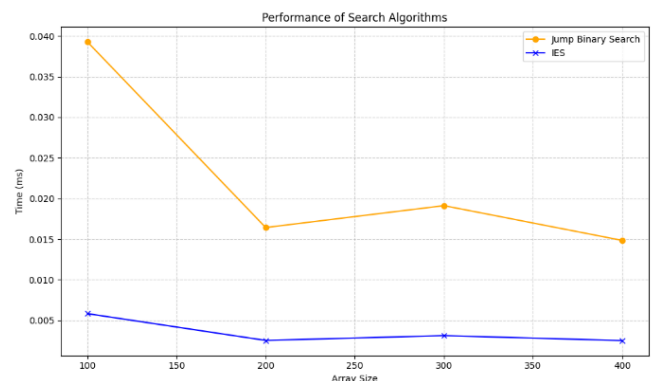


Figure 3. Execution Time of JBS and IES

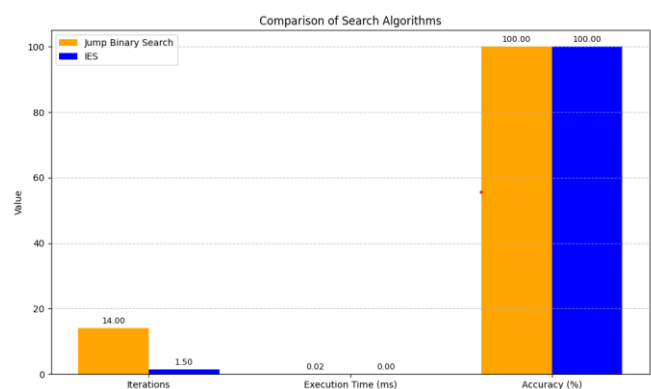


Figure 4. Comparison of JBS and IES with Four Targets

Further testing was conducted by searching for four names of social assistance recipients as search parameters, and by comparing the final performance of both algorithms in terms of the number of iterations, execution time, and search result accuracy. Figure 4 presents a bar graph comparing JBS and IES performance metrics when searching for four names of social assistance recipients. Based on the iteration count, JBS averaged 14 iterations to complete data searches for the four targets. Meanwhile, IES only requires an average of 1.5

iterations to find the target being searched for. This difference indicates that IES exhibits a significantly faster convergence speed, particularly with non-uniform data distributions [28].

Considering the execution time, JBS requires a search time of approximately 0.02 ms. Meanwhile, IES can complete faster than JBS. Although this difference may change, in real systems that perform large-scale searches, millisecond efficiency can significantly impact an application's overall performance [29]. Both algorithms achieved 100% accuracy in every test against the four target elements. This shows that JBS and IES are capable of returning accurate search results.

Overall, the initial testing results indicate that the IES algorithm outperforms other algorithms in terms of execution time and iteration efficiency. IES demonstrates more stable performance and efficiency compared to JBS. With these results, the IES algorithm will be implemented in the social assistance data management application in this research.

B. Implementation of IES in The Application

Based on previous test results, the Interpolation Extrapolation Search algorithm was selected as the most optimal and efficient. The IES algorithm will be implemented in the social assistance data management application. This application is built using the Laravel 11 framework as the backend and MySQL as the database management system.

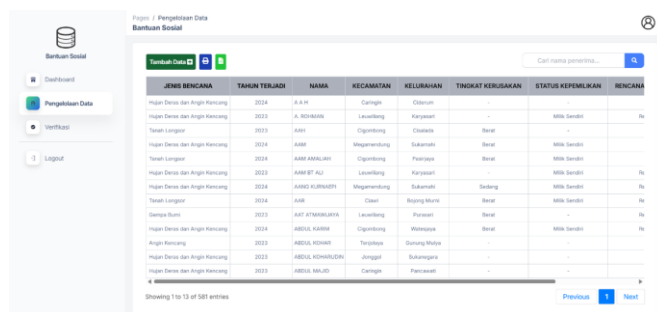


Figure 5. Data Management Page

The implementation is carried out on the data management page, as shown in Figure 5, where users can search for social assistance recipient data based on the recipient's name that has been input. The names in the database are sorted alphabetically to support the efficiency of the IES algorithm. The search process begins when the user inputs the target element through the search frontend. The system then processes the input and executes the IES logic that has been integrated into the Laravel controller.

Algorithm 3. Data Search Using IES on a Application

- 1: Start timing the execution of the search
- 2: Take the input name of the user data (searchTerm)
- 3: Convert all names from the database to numerical form using positional encoding
- 4: Initialization:
 $xl = 0$
 $xu = \text{Data length} - 1$
 $F = 0$ (search direction flag)
 $xs = -1$ (initial estimate position)
- 5: Perform iterations for $xl \leq xu$

```

a: Calculate the estimated position (xs) using
   interpolation/extrapolation
b: If  $xs < xl$  or  $xs > xu \rightarrow$  exit the loop
c: Compare the names in the  $xs$  index with the
   target name
d: If it matches  $\rightarrow$  return the result, set flag
   = true
e: If the name is smaller than the target  $\rightarrow xl$ 
   =  $xs + 1$ ;  $F = 1$ 
f: If the name is greater than the target  $\rightarrow xu$ 
   =  $xs - 1$ ;  $F = -1$ 
6: If no suitable results are found
   a: Perform a fallback using linear search
   b: If found, display the fallback result
7: Save the total execution time and log it
8: Return the search results in the form of a web
   application display
  
```

The search is executed on the server side using a numerical index approach based on position estimation, which involves interpolation and extrapolation according to the characteristics of the IES algorithm.

Integrated into the Laravel application, the IES algorithm speeds up searches for aid recipient names in the database. The system converts name data into numbers using positional encoding techniques, then estimates index positions through interpolation or extrapolation based on the search direction flag. If the primary search fails or the estimation is invalid, the system defaults to a linear search. This algorithm enables the system to operate efficiently, even on large datasets with uneven distributions. The entire process is recorded through Laravel Log for performance evaluation purposes.



Figure 6. Data Page Found

The search results will be displayed in a real-time table. If the data is found, all information related to the recipient will be displayed. In Figure 6, the system successfully returned the search result for the name "SAANI," indicating that the name is present in the database.

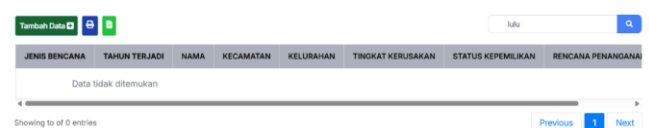


Figure 7. Data Page Not Found

Conversely, if the data is not in the database or does not match the database's contents, the system will return a message stating "Data not found." In Figure 7, the search for the name "LULU" returned no results, indicating that the name is not in the database. This suggests that the system has been operating effectively in terms of validating search results while maintaining the integrity of user responses [30].

To ensure the speed of data retrieval in the application, it can be observed in the local debug [31]. With the implementation of the IES algorithm, the application shows

improved efficiency and search stability even when handling data entry errors [32]. The search speed remains stable at under one second per request, and the accuracy of the results remains consistent. The application requires 0.04 seconds to search for the data "SAANI" in the database. This demonstrates that IES is a suitable solution for fast searching in large-scale and dynamic data systems, as seen in the social assistance data collection case study [33].

IV. CONCLUSION

This research successfully compared two hybrid search algorithms, Jump Binary Search (JBS) and Interpolation Extrapolation Search (IES), in terms of the number of iterations, execution time, and search accuracy. These findings show that IES consistently outperforms JBS in terms of speed and iteration efficiency, especially for datasets with non-uniform distributions. IES achieves execution times that are significantly faster, ranging from 0.002 ms to 0.006 ms, and requires fewer iterations, averaging only 1.5 iterations. A key scientific contribution of this study is the successful practical implementation of the IES algorithm within a Laravel-based web application with a MySQL database. This implementation validated IES's responsive and stable search performance, achieving an average search time of under one second and specifically locating data in 0.04 seconds. These results directly fulfill the research objective of providing an efficient and effective data search solution for large-scale and dynamic data systems, demonstrating the feasibility of IES as a robust algorithm for real-world management applications such as social aid data collection.

ACKNOWLEDGEMENT

The researcher would like to express gratitude to DPKPP Kabupaten Bogor as the internship site, as well as to Mr. Dian Sutardi as the field supervisor who provided guidance, direction, and valuable opportunities throughout the internship process. The researcher also extends gratitude to the Universitas Logistik dan Bisnis Internasional, specifically the Applied Bachelor of Informatics Engineering, for the academic support and facilities provided during the course of this research.

REFERENCES

- [1] A. Aoun, A. Ilinca, M. Ghandour, and H. Ibrahim, "A review of Industry 4.0 characteristics and challenges, with potential improvements using blockchain technology," *Comput. & Ind. Eng.*, vol. 162, p. 107746, 2021.
- [2] I. A. Machado, C. Costa, and M. Y. Santos, "Data mesh: concepts and principles of a paradigm shift in data architectures," *Procedia Comput. Sci.*, vol. 196, pp. 263–271, 2022.
- [3] K. Ahmad, M. Maabreh, M. Ghaly, K. Khan, J. Qadir, and A. Al-Fuqaha, "Developing future human-centered smart cities: Critical analysis of smart city security, Data management, and Ethical challenges," *Comput. Sci. Rev.*, vol. 43, p. 100452, 2022.
- [4] A. D. Trigilio, Y. W. Marien, M. Edeleva, P. H. M. Van Steenberge, and D. R. D'hooge, "Optimal search methods for selecting distributed species in Gillespie-based kinetic Monte Carlo," *Comput. & Chem. Eng.*, vol. 158, p. 107580, 2022.
- [5] N. A. Zinnia and E. Hanada, "Optimizing Search Strategies: A Study of Two-Pointer Linear Search Implementation," *arXiv Prepr. arXiv2406.16729*, 2024.
- [6] B. A. Saputra, S. C. Candra, F. B. Wijaya, K. M. Suryaningrum, and H. A. Saputri, "Comparative Analysis of Binary and Interpolation Search Algorithms on Integer Data Using C Programming Language," in *2023 International Conference on Information Management and Technology (ICIMTech)*, 2023, pp. 340–345.
- [7] F. Qin, A. M. Zain, and K.-Q. Zhou, "Harmony search algorithm and related variants: A systematic review," *Swarm Evol. Comput.*, vol. 74, p. 101126, 2022.
- [8] G. Y. Rorong, S. F. Pane, and M. A. H. Siregar, "Optimizing Search Efficiency in Ordered Data: A Hybrid Approach Using Jump Binary Search," *J. Dinda Data Sci. Inf. Technol. Data Anal.*, vol. 5, no. 1, pp. 36–44, 2025.
- [9] M. N. Kabir, Y. M. Alginahi, and J. Ali, "Optimal search algorithm in a big database using interpolation – extrapolation method," vol. 55, no. 21, pp. 21–23, 2019, doi: 10.1049/el.2019.1965.
- [10] 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.
- [11] N. S. Gupta and P. Kumar, "Perspective of artificial intelligence in healthcare data management: A journey towards precision medicine," *Comput. Biol. Med.*, vol. 162, p. 107051, 2023.
- [12] H. Ma, "Research and Design of a B/S Platform Information Management System Based on Django and Vue Frameworks," in *2024 4th Asia-Pacific Conference on Communications Technology and Computer Science (ACCTCS)*, 2024, pp. 103–107.
- [13] S. Ait Errami, H. Hajji, K. Ait El Kadi, and H. Badir, "Spatial big data architecture: from data warehouses and data lakes to the Lakehouse," *J. Parallel Distrib. Comput.*, vol. 176, pp. 70–79, 2023.
- [14] B. Zaabar, O. Cheikhrouhou, F. Jamil, M. Ammi, and M. Abid, "HealthBlock: A secure blockchain-based healthcare data management system," *Comput. Networks*, vol. 200, p. 108500, 2021.
- [15] S. Aminzadeh *et al.*, "The applications of machine learning techniques in medical data processing based on distributed computing and the Internet of Things," *Comput. Methods Programs Biomed.*, vol. 241, p. 107745, 2023.
- [16] T.-V. Dang and P. X. Tan, "Hybrid Mobile Robot Path

- Planning Using Safe JBS-A* B Algorithm and Improved DWA Based on Monocular Camera,” *J. Intell. & Robot. Syst.*, vol. 110, no. 4, p. 151, 2024.
- [17] P. D. Kusuma, “Group Better-Worse Algorithm: A Superior Swarm-based Metaheuristic Embedded with Jump Search,” *IAENG Int. J. Appl. Math.*, vol. 54, no. 4, pp. 614–622, 2024.
- [18] H. Wang *et al.*, “Enhancing dnn-based binary code function search with low-cost equivalence checking,” *IEEE Trans. Softw. Eng.*, vol. 49, no. 1, pp. 226–250, 2022.
- [19] A. Shabbir *et al.*, “A review of algorithms’s complexities on different valued sorted and unsorted data,” in *2023 International Conference on IT and Industrial Technologies (ICIT)*, 2023, pp. 1–6.
- [20] A.-D. Li, B. Xue, and M. Zhang, “Improved binary particle swarm optimization for feature selection with new initialization and search space reduction strategies,” *Appl. Soft Comput.*, vol. 106, p. 107302, 2021.
- [21] P. Zhang, J. Wang, Z. Tian, S. Sun, J. Li, and J. Yang, “A genetic algorithm with jumping gene and heuristic operators for traveling salesman problem,” *Appl. Soft Comput.*, vol. 127, p. 109339, 2022.
- [22] Z. Yang, Z. Song, X. Zhao, and X. Zhou, “Time-domain extrapolation method for tractor drive shaft loads in stationary operating conditions,” *Biosyst. Eng.*, vol. 210, pp. 143–155, 2021.
- [23] A. H. Alzraiee, J. T. White, M. J. Knowling, R. J. Hunt, and M. N. Fienen, “A scalable model-independent iterative data assimilation tool for sequential and batch estimation of high dimensional model parameters and states,” *Environ. Model. & Softw.*, vol. 150, p. 105284, 2022.
- [24] Y. Sun, S. Ding, Z. Zhang, and W. Jia, “An improved grid search algorithm to optimize SVR for prediction,” *Soft Comput.*, vol. 25, pp. 5633–5644, 2021.
- [25] M. A. Bin Zuraimi and F. H. K. Zaman, “Vehicle detection and tracking using YOLO and DeepSORT,” in *2021 IEEE 11th IEEE Symposium on Computer Applications & Industrial Electronics (ISCAIE)*, 2021, pp. 23–29.
- [26] J. Liang, X. Sui, C. Yue, Y. Bi, and M. Yu, “Differential evolution using interpolation strategy for multimodal multiobjective optimization,” in *2023 3rd international symposium on computer technology and information science (ISCTIS)*, 2023, pp. 764–769.
- [27] A. M. Sharafaddini and N. Mansouri, “A binary chaotic transient search optimization algorithm for enhancing feature selection,” *Arab. J. Sci. Eng.*, vol. 50, no. 2, pp. 679–702, 2025.
- [28] M. Kim, D. Ki, S.-W. Shim, and B.-J. Lee, “Adaptive Non-Uniform Timestep Sampling for Diffusion Model Training,” *arXiv Prepr. arXiv2411.09998*, 2024.
- [29] R. Rua and J. Saraiva, “A large-scale empirical study on mobile performance: energy, run-time and memory,” *Empir. Softw. Eng.*, vol. 29, no. 1, p. 31, 2024.
- [30] M. Griffin *et al.*, “Ensuring survey research data integrity in the era of internet bots,” *Qual. & Quant.*, vol. 56, no. 4, pp. 2841–2852, 2022.
- [31] A. R. Chen, T.-H. Chen, and S. Wang, “Pathidea: Improving information retrieval-based bug localization by re-constructing execution paths using logs,” *IEEE Trans. Softw. Eng.*, vol. 48, no. 8, pp. 2905–2919, 2021.
- [32] Y. Li, C. Yu, M. Shahidehpour, T. Yang, Z. Zeng, and T. Chai, “Deep reinforcement learning for smart grid operations: Algorithms, applications, and prospects,” *Proc. IEEE*, vol. 111, no. 9, pp. 1055–1096, 2023.
- [33] L. Zhang *et al.*, “A data-driven approach to anomaly detection and vulnerability dynamic analysis for large-scale integrated energy systems,” *Energy Convers. Manag.*, vol. 234, p. 113926, 2021.