Viel Hybrid Search Algorithm: Maximizing the Efficiency of Binary and Flexibility of Linear Search Algorithms

Via Preciado
Department of Computer Science, University of Science and Technology of Southern Philippines
Cagayan de Oro, Philippines

Ismael Zarate
Department of Computer Science, University of Science and Technology of Southern Philippines
Cagayan de Oro, Philippines

Junar Landicho
Department of Computer Science, University of Science and Technology of Southern Philippines,
Cagayan de Oro, Philippines

ABSTRACT

Binary Search and Linear Search are the two most commonly used searching algorithms to find items in an array. This study revels the Viel Search Algorithm**,** a hybird algorithm that leverages Binary Searh's efficiency with the capability of Linear Searh to improve search operations in both sorted and unsorted arrays. Unlike typical search methods, this hybird algorithm continuously selects between Binary and Linear Searh based on the input data's value. The performance evaluations of the Viel Searh Algorithm were conducted on arrays with sizes ranging from 10 to 500,000 integers, containing both sorted and unsorted elements. The results consistently showed that the Viel Search Algorithm outperforms traditional Binary and Linear methods, especially when dealing with large and unsorted arrays. These findings suggest that the Viel Search Algorithm offers significant improvements in search efficiency, making it a valuable tool for a wide range of computational applications.

General Terms

Searching Algorithm, Hybrid Algorithm, Binary Search, Linear Search

Keywords

Algorithm, Hybrid, Binary, Linear, Time Complexity, Search Efficiency, Sorted, Unsorted

1. INTRODUCTION

Algorithms form the bedrock of Computer Science. In a similar way, algorithms are a set of instructions telling computers how to tackle a problem. The concept of an algorithm has broaden, encompassing diverse types like parallel [1, 3], interactive [1, 4], distributed [1, 5], real-time [1, 6], analog [1], hybird (hybrid - misspelling), and quantum algorithms [1, 8]. Each algorithm boastits own strengths and weaknesses, judged by its Time and Space Complexity [9]. Search algorithms, such as Binary Search and linear Search, are among the most commonly used, excelling with either sorted or unsorted arrays. This study focuses on a hybird algorithm that merges Binary and linear Search.

With the ever-growing data in the digital realm, the necessity for effective search algorithms to handle massive datasets becomes paramount. These efficient search algorithms are essential in fields like data processing [12], data base management [13], and information retrieval systems [12]. By bettering the time complexity of these algorithms, a significant improvement can be achieved in applications relying on search operations. Therefore hybrid algorithms are a critical tool for operations involving vast amounts of data, such as data bases [13], web search enginers [14], and data mining system [14].

The proposed Viel algorithim aims to adapts its search strategy depending on the input data's value to achieve faster runtime. In contrast to traditional algorithms with rigid rules, the Viel algorithm continuously adjusts based on data size and its placement selecting the most efficient searching method. It continualy assesses search performance during run-time and utilizes a combination of linear and binary searching methods, guaranteeing the best efficiency under various circumstances.

The setback with the existing research on hybrid search algorithms has shown potential advantage of combining linear and binary search approach to improve search ability in unsorted arrays, but these studies often suffer from limitations. For instance, the evaluation of the hybrid algorithm was conducted on an array limited to 25 integers, which does not show the algorithm's performance on larger and more wide-ranged arrays commonly encountered in real-world uses. In addition, the method used for measuring search time in their study was not clearly shown, making it hard to assess the accuracy of the algorithm's ability.

Therefore, a comprehensive study on hybrid algorithms that merge the strengths of Binary and Linear Search is necessary**.** This hybrid algorithm will be evaluated using progressively larger arrays with varying characteristics to guarantee its performance. Furthermore, a well-defined and standardized methodology for measuring each array's search time is employed to precisely assess the algorithm's efficiency.

2. LITERATURE REVIEW

This section explores optimazation of search efficiency through hybird algorithms by reviewing various scholarly resourses**.** Linear search, also known as sequential search, involves examining each data item sequencially until the target is found or the list's end is reached [10, 16, 17]. While linear search works on unsorted arrays and boast a best-case time complexity of O(1) [11, 15], it's not ideal for large datasets due to it's O(n) complexity. Conversely, binary search is limited to sorted arrays [11] and employs a divide-and-conquer strategy, a method that repeatedley halves the array and solves the subproblems recursively [10, 16, 17]. This approach makes the binary algorithm more efficient for large datasets with a time complexity of O(log(n)), but it necessitates a sorted element arays [11, 15].

Studies on hybird algorithms delve into how combining various search methods can improve preformance**.** For instance, a 2021 study showcases significant performance improvements when searching ordered arrays by mergng interpolation and binary search techniques [14]. Another 2017 study introduces a hybird algorithm that combines linear and binary search approaches to enhance search efficiency in unsorted arrays. Despite the advancements, this hybird approach was limited to small arrays and lacked a in-depth performance evaluation [15] Further research highlights the broad applicability and effectiveness of hybrid algorithms. For example, the Cuckoo Search Algorithim and the Whale Optimization Algorithm exemplify how hybird methods can improve both search and optimization processes [19, 21]. These hybird approaches strengthen the idea that combining diverse techniques can balance efficiency and capability which is crucial for optimizing search methods [20, 22].

In summary, while linear search is simple to apply, it is not best for large arrays. However, binary search offers greater efficiency for large, sorted arrays but requires sorting process. Combining these methods into hybird algorithms offers a promising solution. It leverages the strengths of both approaches, thereby improving search efficiency across various conditions. [11, 17].

3. METHODOLOGY

This research strives to develop a versatile search technique by merging the strengths of the two established methods. This approach would be applicable to both organized and unorganized data sets, overcoming the drawbacks inherent in each individual method.

3.1 Arrays.

We measured the researched search method's preformance across various arrays. We implemented a random data generation function to create set of arrays with diverse size and organizational properties. These sets included both order and unordered arrangments**,** ranging in size from a small group to progressively larger group.

3.2 Ordered vs. Unordered Data.

The algorithm utilized both sorted and unsorted arrays to evaluate the algorithm's ability to handle different array types.

3.3 Performance Evaluation.

Evaluation metrics were used to quantify the performance of each search algorithm. The average execution time and adaptability of the Viel search algorithm were compared with linear and binary search across different arrays.

3.4 Execution Time.

This metric measured the time each algorithm took to complete a search operation. This measurement used an additional C++ library (chono)..

3.5 Proposed Hybrid Algorithm.

The proposed Viel hybrid algorithm applied a binary search approach to find a target integer within a potentially sorted but switched array. It sets two variables, `low` and `high`, marking the lower and upper limits of the search space. Using bitwise operations, it computed the middle index, `mid`, to facilitate the search process. The algorithm proceeds by iteratively limiting down the search range until the target is found or the array is exhausted. It adapts its strategy based on the sortedness of array halves, optimizing the search process. If the middle element is the target, its index is returned. In cases where the array is not entirely sorted, a linear search is conducted in the unsorted segment. The algorithm distinguishes between sorted left and right halves, adjusting the search range accordingly. It updates low or high to focus the search on the relevant half, depending on the target's relationship to the sorted segments. Finally, if the target is not found within the array, the function returns -1.

3.6 Testing Procedure

The arrays will be utilized and carried out with a key value to be searched. The target element for each test was randomly chosen from the entire array, consisting of random numbers between the specific set. Using the C++ library (`chrono`), execution time was measured in nanoseconds. The tests involved selecting a target element that fell outside the array's value range to simulate scenarios where the search might be unsuccessful. Using C++ programming language, each set was tested 6 times both unsorted and sorted with varying array size, resulting in a total of 18 results, ensuring statistically significant outcomes. Results from the four algorithms namely (1) Binary Search, (2) Linear Search, a published (3) Hybrid Search Algorithm and the proposed (4) Viel Hybrid Algorithm was compared. The algorithm was performed on a computer with specifications having Windows 11, Ryzen 5 3600 6 cores 12 threads CPU, 16 Gb of RAM.

4. RESULTS AND DISCUSSION

4.1 Proposed Algorithm.

The proposed algorithm integrated the strengths of binary and linear search techniques to optimize search efficiency across diverse arrays. Unlike traditional search algorithms, this hybrid approach dynamically selected the most suitable search strategy based on input data characteristics, such as sorting status. For sorted arrays, the algorithm uses binary search, making its divide-and-conquer approach to accurately locate the targe integer. Contrarily, for unsorted arrays, linear search was used to pass over integers sequentially. The algorithm started by initializing the lower and upper parts of the search range and computed the middle part to aid the search. It adapted its approach based on the sortedness of array halves, improving the search process by either continuing with binary search or turning to linear search if needed. By uniting these approaches, the algorithm arranged a balance between effectiveness and flexibility, offering adjusted performance in managing changing sizes and arrangements.

4.2 Pseudocode.

Algorithm: Viel_Search_Algorithm(arr, target)

Input:
Array: A vector of integers, potentially containing both sorted and unsorted regions. Target: An integer value to search for within array
Output:
The index of target if found, otherwise -1.

```
1.  low ← 0
2.  high ← length(arr) - 1
3.  while low ≤ high do:
4.  mid ← low + ((high - low) >> 1)
5.  if arr[mid] == target then:
6.  return mid
7.  if arr[low] > arr[mid] or arr[mid] > arr[high] then:
8.  for i from low to high do:
9.  if arr[i] == target then:
10. return i
11. return -1
12. if arr[low] ≤ arr[mid] then:
13. if target >= arr[low] and target < arr[mid] then:
14. high ← mid - 1
15. else:
16. low ← mid + 1
17. else:
18. if target > arr[mid] and target ≤ arr[high] then:
19. low ← mid + 1
20. else:
21. high ← mid - 1
22. return -1
```

4.3 Time Complexity.

In the bestest case, the array is already sorted, and the function preceeds with a binary search throughout. The time complexcity of binary search is O(log n), where n is the number of integers in the array. In the worse case, the array is fully unsorted, and the function does a linear search through out** (throughout - misspelling)**.** The time complexcity of linear search is O(n), where n is the number of integers in the array. In the average-case time complexcity changes on the distribution of sorted and unsorted areas in the array. When the array holds both sorted and unsorted areas, the function will do a combining of binary search and linear search. In such situations, the time complexcity will be between O(log n) and O(n), varying on the parts of sorted and unsorted areas and the position of the target value. Let p be the proprtion of the array that is sorted. Let (1 − p) be the proprtion of the array that is unsorted..

Let p be the proportion of the array that is sorted.

Let $(1 - p)$ be the proportion of the array that is unsorted.

When the function meets a sorted area, it does binary search, maintaining $O(\log n)$ to the overall complexity. When the function encounters an unsorted region, it performs linear search, contributing $O(n)$ to the overall complexity. If a larger portion of the array is sorted (p is high), the function will more often perform binary search. If a larger portion of the array is unsorted (p is low), the function will more often perform linear search.

The average-case time complexity $T(n)$ can be approximated as:

$$T(n) = p \times O(log\, n) + (1 - p) \times O(n)$$

$p \times O(log\, n)$ represents the time spent in binary search within sorted regions.
$(1 - p) \times O(n)$ represents the time spent in linear search within unsorted regions.

4.4 Space Complexity.

The function takes a reference to a vector arr as input. The input array itself consumes $O(n)$ space, where n is the number of elements in the array. However, this is not part of the algorithm's additional space requirement; it's considered part of the input. The function uses a few integer variables (low, high, mid, i) for its operation. These variables consume a constant amount of space. The memory used by these variables does not depend on the size of the input array, hence the space complexity is $O(1)$. This means the function has a constant space complexity, regardless of the size of the input array.

4.5 Algorithm Comparison

Overall analysis of the algorithm.

Table 1. Performance Comparison of Binary, Linear, and Viel Search Algorithm measured in nanoseconds

Array	Size	Key	BS	LS	Viel
0-10	10 Sorted	22	200	200	200
	10 Unsorted	Not in Array	N/A	300	200
0-50	50 Sorted	17	200	400	300
	50 Unsorted	44	N/A	800	300
0-100	100 Sorted	Not in Array	200	900	300
	100 Unsorted	64	N/A	1100	300
0-1000	1,000 Sorted	538	600	3900	300
	1,000 Unsorted	538	N/A	5900	2100
0-5,000	5,000 Sorted	Not in Array	800	31,500	500
	5,000 Unsorted	2,014	N/A	29,900	5,300
0-10,000	10,000 Sorted	6,712	1,200	44,300	400
	10,000 Unsorted	Not in Array	N/A	88,400	26,200
0-100,000	100,000 Sorted	50,671	1,500	329,100	500

0-100,000	100,000 Unsorted	Not in Array	N/A	629,200	230,500
0-250,000	250,000 Sorted	Not in Array	300	2,103,000	600
	250,000 Unsorted	178,900	N/A	373,900	73,400
0-500,000	500,000 Sorted	250,000	400	644,900	200
	500,000 Unsorted	Not in Array	N/A	2,294,100	907,300

For small arrays consisting of numbers between 0 and 10, all three algorithms—Binary Search, Linear Search, and Viel Search—performed equally well, taking around 200 nanoseconds. However, in unsorted arrays of the same size, the Viel algorithm proved to be more efficient, outperforming Linear Search by 100 nanoseconds. Binary search results were not applicable as this algorithm requires sorting of arrays [11]. Notable differences in the efficiency of the Viel Algorithm were also evident in arrays of 0 - 50 elements where it performed faster than Linear Search in both sorted and unsorted conditions. For 0 - 100 elements, Binary Search was the quickest in sorted arrays, while the Viel Search still outperformed Linear Search significantly in unsorted arrays.

In the array ranging between 0 - 1,000 and 0 - 10,000 elements, the efficiency of the Viel Search algorithm was evident. For 1,000 elements in sorted arrays, Viel Search was significantly faster than both Binary and Linear Search. In unsorted arrays, it demonstrated superiority over Linear Search. With 5,000 elements, the Viel Search algorithm outperformed Binary Search in sorted arrays and was more efficient than Linear Search in unsorted arrays. Moreover, in 0 - 10,000 elements the Viel Search again proved to be the fastest among the three algorithms, particularly excelling in unsorted arrays.

For 100,000 elements in sorted arrays, Viel Search was the fastest, followed by Binary Search, with Linear Search being significantly slower. In unsorted arrays, Viel Search outperformed Linear Search. For sorted arrays of 250,000 Binary Search outperformed Viel by 50%, additionally for the unsorted array, Viel Search is significantly faster than Linear Search. With 500,000 elements, the Viel Search algorithm continued to show its efficiency, outperforming Binary Search in sorted arrays and demonstrating an efficiency advantage over Linear Search in unsorted arrays.

Table 2. Comparative Performance of Viel Hybrid Algorithm and a published Hybrid Search Algorithm (Using Identical Arrays from the Original Study)

Array	Key	HSA	VSA
[11, 0, 3, 19, 23, 23, 5, 7, 21, 24, 13, 10, 16, 24, 8, 12, 24, 15, 3, 22, 22, 6, 22, 17, 15]	2	5559	200
[9, 11, 2, 18, 15, 2, 11, 12, 6, 13, 20, 16, 24, 4, 23, 22, 24, 21, 4, 8, 6, 24, 21, 22, 9]	14	7270	300
[0, 0, 1, 8, 16, 11, 8, 6, 9, 5, 19, 4, 22, 3, 13, 10, 6, 16, 12, 4, 19, 2, 21, 14, 0]	1	5559	300
[23, 19, 24, 14, 1, 3, 2, 0, 11, 9, 24, 18, 4, 9, 20, 5, 15, 3, 24, 3, 3, 2, 22, 11, 5]	11	6483	200
[22, 1, 23, 9, 1, 13, 20, 14, 14, 0, 1, 6, 15, 1, 2, 20, 6, 6, 3, 15, 22, 18, 3, 0, 16]	18	7698	300
[22, 19, 24, 0, 1, 7, 17, 8, 5, 22, 11, 10, 7, 11, 21, 13, 9, 2, 19, 1, 12, 6, 7, 21, 10]	17	2994	200
[11, 19, 5, 21, 1, 7, 23, 4, 14, 18, 15, 22, 1, 8, 13, 10, 11, 0, 15, 5, 14, 7, 24, 24, 9]	12	5132	200
[11, 19, 0, 13, 20, 12, 12, 4, 8, 17, 7, 20, 23, 23, 15, 11, 22, 6, 19, 20, 15, 22, 13, 1, 12]	7	7697	200
[3, 19, 3, 24, 12, 11, 18, 4, 13, 14, 23, 12, 13, 15, 15, 11, 2, 19, 24, 20, 12, 6, 7, 5, 17]	20	5132	300
[21, 17, 14, 22, 5, 14, 0, 10, 11, 19, 18, 10, 19, 6, 23, 10, 6, 15, 3, 4, 20, 4, 3, 22, 21]	8	4704	300

Using identical arrays from the study on a published Hybrid Algorithm [18], results showed that the Viel Search Algorithm consistently outperformed the Hybrid Search Algorithm. The algorithm shows a significant reduction in search time where the difference between means of the two algorithms is 5572.8 nanoseconds.

5. CONCLUSIONS

The study demonstrates the effectiveness of the proposed Viel Search algorithm in optimizing search efficiency across diverse arrays. This algorithm integrates the strengths of binary and linear search techniques, showing superior performance, especially in unsorted and large arrays. In arrays reaching from 10 to 100 integers, the Viel Search algorithm met the efficiency of Binary Search in sorted arrays and surpassed Linear Search. For arrays reaching from 1,000 to 10,000 integers, it notably decreased search times compared to both Linear and Binary Search, mainly in unsorted arrays. The arrays after 100,000 to 500,000 elements, where the Viel Search algorithm passed both Binary and Linear Search in sorted arrays and showed significant improvements in unsorted arrays. Overall, the Viel Search algorithm shows to be an effective method for searching in both sorted and unsorted arrays, remarkably as array size increases. Its flexibility and outstanding performance make it a worthy tool for various search tasks.

5.1 Limitations

While the Viel Hybrid Search Algorithm is applicable for searching in sorted and unsorted arrays, it has limitations, particularly dealing with large arrays. The algorithm's execution reduces drastically with large arrays, such as those including millions of integers. The algorithm presently uses a basic linear search for unsorted parts of the array, which show poor performance for very large arrays, lead to in slow performance in the worst cases. Furthermore, the algorithm does not recognize for duplicate integers in the array, which causes incorrect results.

5. REFERENCES

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to Algorithms. Cambridge, MA: The MIT Press, 2022.

[2] H. Sarker, “Machine learning: Algorithms, real-world applications and Research Directions,” SN Computer Science, vol. 2, no. 3, Mar. 2021.

[3] V. Voevodin, A. Antonov, and V. Voevodin, “What do we need to know about parallel algorithms and their efficient implementation?,” Topics in Parallel and Distributed Computing, pp. 23–58, 2018.

[4] L. L. Hurtado Cortés, “Interactive algorithms: A didactic strategy for non-programmers,” Towards a Hybrid, Flexible and Socially Engaged Higher Education, pp. 224–234, 2024.

[5] F. Fakhfakh, M. Tounsi, M. Mosbah, and A. H. Kacem, “Formal verification approaches for distributed algorithms: A systematic literature review,” Procedia Computer Science, vol. 126, pp. 1551–1560, 2018

[6] J. Donga and M. S. Holia, “An analysis of scheduling algorithms in real-time operating system,” Inventive Computation Technologies, pp. 374–381, Nov. 2019.

[7] B. F. Azevedo, A. M. Rocha, and A. I. Pereira, “Hybrid approaches to optimization and machine learning methods: A systematic literature review,” Machine Learning, vol. 113, no. 7, pp. 4055–4097, Jan. 2024.

[8] A. Montanaro, “Quantum Algorithms: An overview,” npj Quantum Information, vol. 2, no. 1, Jan. 2016.

[9] BYJU’S Exam Prep, “What two main measures for the efficiency of an algorithm?,” BYJU’S Exam Prep,

[10] A brief study and analysis of different searching algorithms | IEEE conference publication | IEEE Xplore

[11] V. P.parmar and C. Kumbharana, “Comparing linear search and binary search algorithms to search an element from a linear list implemented through static array, dynamic array and linked list,” International Journal of Computer Applications

[12] D. Wang and Y. Bai, “Data Processing, information retrieval and classification of atmospheric measurements,” Journal of Data, Information and Management, vol. 6, no. 1, pp. 41–49, Jan. 2024.

[13] A. Mamadolimov and S. Khikmat, “Insertion algorithms for network model database management systems,” Journal of Physics: Conference Series, vol. 949, p. 012008, Dec. 2017.

[14] A. S. Mohammed, Ş. E. Amrahov, and F. V. Çelebi, “Efficient hybrid search algorithm on ordered datasets,” arXiv.org

[15] Hybrid search algorithm: Combined linear and binary search algorithm | IEEE conference publication | IEEE Xplore

[16] A. Sherine, M. Jasmine, G. Peter, and S. Albert Alexander, Algorithm and Design Complexity. Boca Raton: CRC Press, 2023.

[17] A. K. Yadav and V. K. Yadav, Data Structures with C Programming. Ashland: Arcler Press, 2019.

[18] M. Shehab, A. T. Khader, and M. Laouchedi, “A hybrid method based on cuckoo search algorithm for Global Optimization Problems,” Journal of Information and Communication Technology, vol. 17, 2018.

[19] W. Zhang, A. Maleki, M. A. Rosen, and J. Liu, “Sizing a stand-alone solar-wind-hydrogen energy system using weather forecasting and a hybrid search optimization algorithm,” Energy Conversion and Management, vol. 180, pp. 609–621, Jan. 2019.

[20] M. Abdel-Basset, G. Manogaran, D. El-Shahat, and S. Mirjalili, “Retracted: A hybrid whale optimization algorithm based on local search strategy for the permutation flow shop scheduling problem,” Future Generation Computer Systems, vol. 85, pp. 129–145, Aug. 2018.

[21] F. Zhao et al., “A hybrid algorithm based on self-adaptive gravitational search algorithm and Differential Evolution,” Expert Systems with Applications, vol. 113, pp. 515–530, Dec. 2018.