Efficient Text Formatting: A Comparative Analysis of Algorithmic Approaches

Bandaru Jaya Nandini, Mamidi Leha Sahithi, Siddareddy Gari Harshika, Apurvanand Sahay
Department of Computer Science and Engineering
Amrita School of Computing, Bangalore
Amrita Vishwa Vidyapeetham, India
bl.en.u4aie22006@bl.students.amrita.edu, bl.en.u4aie22035@bl.students.amrita.edu,
bl.en.u4aie22053@bl.students.amrita.edu, a_sahay@blr.amrita.edu

2024 First International Conference for Women in Computing (InCoWoCo) | 979-8-3315-1894-3/24/$31.00 ©2024 IEEE | DOI: 10.1109/INCOWOCO64194.2024.10863158

Abstract—**Efficient text formatting, especially when printing with mono spaced fonts, is a very important factor. In an attempt to address the problem of printing clean paragraphs, this work has been presented in the form of implementing and trying to demonstrate the efficiency of a large variety of algorithms. The algorithms include Binary Search, Shortest Path, Greedy Approach, Branch and Bound, Divide and Conquer, and Brute Force. This problem tries to format a set of texts into lines of fixed given width, where at a time it looks appropriate and readable, trying to minimize the number of vacant places and blank spaces. Hence, this dissertation dwells on a text formatting technique that possesses the least number of vacant places and blank spaces yet manages to look nice. In general, a comparison of the methods in the number of vacant places on a line raised to the cube, as a cost metric and the runtime complexity. Performance testing on a very broad spectrum of test scenarios is also there, which is low word count, random content, fixed character length, and fixed word count. The implementation of this problem will help us understand what method is effective and where so that it can be used in a real-world scenario. Text formatting jobs in domains like publishing, academia, and web development are crucial processes for efficient communication.Among these, the Dynamic Programming and Divide and Conquer algorithms achieved the highest performance, with a minimal total cost of 2529 and optimal line breaks in fixed character-length paragraphs. The results demonstrate that these algorithms strike an ideal balance between runtime efficiency and formatting quality, making them well-suited for real-world applications.**

Index Terms—**Text formatting,Dynamic Programming, Greedy Approach, Branch and Bound, Divide and Conquer, Binary Search, Shortest Path, Brute Force.**

I. Introduction

Good text formatting, which is efficiently done, is paramount for several applications that range from document processing to printing. More so, with respect to printers, mono-space fonts have each character sharing the same width, and neatly printing paragraphs becomes difficult. This study addresses the problem of text formatting, considering not only the implementation but also the analysis of different algorithms regarding different alignments: left, center, and right.

Now the main objective is, given a paragraph of text, represented by a sequence of n words, all with different lengths, that has to be formatted into lines that fit into a maximum width M of characters for each line. The objective is to minimize the number of blank space characters used in each line, maintaining the paragraph readable and good-looking at the same time. Seven different algorithms are applied to obtain this such as dynamic programming, greedy approach, branch and bound, divide and conquer, binary search, shortest path, and brute-force.

The challenge lies in effectively balancing readability and space efficiency while formatting text within constrained widths, especially when using mono-spaced fonts on limited-width mediums like printers. This study aims to provide a comprehensive analysis of various text formatting algorithms, identifying the most efficient approaches for different alignment types and constraints. By evaluating the runtime, cost, and quality of solutions, the research will offer insights into optimizing text formatting processes in real-world applications.

The study would be diving into the details of each algorithm, giving an elaborative analysis of the performance. Each section, after the introduction, would be serving the purposes of the theoretical background and practical use of algorithms: Dynamic Programming, Greedy Approach, Branch and Bound, Divide and Conquer, Binary Search, Shortest Path, Brute Force. The study will then compare, contrast according to many metrics, followed by discussions on the overall implications text formatting brings to many domains.

Going deeper into the details of the algorithms used, each of them provides a distinctive way to solve the problem of text formatting. For instance, Dynamic Programming provides the theory of optimal substructure to efficiently solve sub problems and develop them towards an optimal solution. In this dimension, the algorithm is the best as it has overlapping sub problems and which leads to simpler computation processes. The Greedy Approach, on the other hand, makes a choice at each step prioritizing immediate gain, thereby looking for a locally optimal solution that will hopefully lead to a globally optimal one. In real life, though, this approach does not always ensure the most optimal solution, but the simplicity and efficiency of the process make it convenient for some scenarios of text formatting, especially when real-time processing is important.

Branch and Bound has systematic exploration of the solution space, with intelligent pruning of branches that do not promise the optimal solution. The algorithm, therefore, can go towards the optimal or near-optimal solution by systematic

Authorized licensed use limited to: Amrita School of Engineering. Downloaded on August 27,2025 at 18:04:23 UTC from IEEE Xplore. Restrictions apply.

exploration into different possibilities and the elimination of less promising options. This is the reason that it becomes greatly effective when the text formatting problems are of a large scale.

The technique of Divide and Conquer breaks down a text formatting problem into smaller sub-problems. Every sub-problem is then solved in a recursive manner, after which the solutions of these smaller sub-problems are combined to obtain a solution to the text formatting problem in question. Basically, it leverages the power of recursion to reduce the computational complexity of the problem, which is very beneficial when applied to a big paragraph of text. It is a paradigm that breaks a bigger problem into smaller problems that can easily be solved. In this regard, Binary Search, Shortest Path, and Brute Force are quite diversified in their own accord to the textual formatting problem.

Some evaluation of various metrics is used to keep at least some level of efficiency in each of the algorithms. The runtime complexity of the algorithms translates directly to computational efficiency. A derived cost metric is produced by a cubic measurement of the number of empty spaces in a line as a means of quality for the formatted output. The performance of the algorithms is also tested with several paragraphs under testing conditions spelled out as low word count, random content, fixed character count, and fixed word count.

It will enlighten the working of the text formatting algorithms and how effective they are for practical use. It is the intention of this study to find important insights that will be helpful for optimization of practical problems of text formatting with respect to runtime comparison, cost, and quality of solution provided by different algorithms. In this study, beside running time, cost, and quality evaluation of solutions, wider aspects of text formatting in many domains will be highlighted.

II. LITERATURE SURVEY

Ayoola et al. [1] conducted research on the use of the branch-and-bound method in solving combinatorial optimization problems; their study was based on the use of the method on the Travelling Salesman Problem of moderate size.

In their study, Wang et al. [2] suggested a greedy-based algorithm for resource scheduling in order to improve task completion times, resource utilization, and energy efficiency in cloud computing environments. The research has used a greedy-based methodology to calculate total task execution time on virtual resources and its impact on task scheduling and resource utilization in cloud computing.

The two articles of Iskandar et al. [3] and Jin et al. [4] solve optimization problems of computational algorithms. The former focuses on making efficient network planning by performing optimization of the Travelling Salesman Problem through a fast heuristic algorithm.

Yang et al. [5] discuss the classical Shortest Path Problem (SPP) and its general relevance within applications of network technology, computer science, communications, operations research, and geographic information science. They outline a Genetic Algorithm approach for application as a solution technique and comparatively assess its effectiveness with that of the Simulated Annealing Algorithm. Their comparison results in the Genetic Algorithm being found more efficient than the Simulated Annealing Algorithm in the solution of the SPP.

Naghibzadeh et al. [6] tackle the challenge of aligning long genomic sequences for mutation discovery and the diagnosis of genome-related diseases. They highlight the limitations of existing alignment algorithms, such as Needleman-Wunsch and Smith-Waterman, particularly when dealing with longer sequences. This approach involves breaking sequences into smaller parts based on the longest common subsequence, aiming to reduce runtime, main memory utilization, and better utilize multiprocessors and GPGPUs. The method demonstrates efficiency in time and memory utilization, making it suitable for aligning long genomic sequences.

Wang et al. [7] introduced an enhanced adaptive dynamic programming (ADP) method for cooperative dynamic games that is both data-driven and iterative. This approach uses the Hamilton-Jacobi-Bellman (HJB) equation alongside basis function approximation to refine control strategies progressively. By doing so, it aims to achieve both global optimality and stability, ensuring that the control solutions are robust and effective.

Singhar et al. [8] discuss a study related to the optimization of chip placement in VLSI design, focusing particularly on the wire length of the interconnects. They propose a greedy algorithm that removes overlaps among cells after the global placement stage. An experimental result shows an improvement to a promising 5 percent reduction in overlap when compared to the conventional methods, which hints at quality improvement in placement. The algorithm also claims to ease the subsequent legal and detail placement stages in chip design.

Posypkin et al. [9] performed a comparative performance study of shared and distributed memory dynamic programming algorithms for application cases to solve combinatorial optimization problems, such as the Knapsack Problem.

As pointed out by Zhihua et al. [11], based on the improved greedy algorithm, an algorithm for dynamic adaptive slicing of additive manufacturing, particularly 3D printing, is presented. The paper is targeted to overcome the staircase effect and long printing time, which are general problems in 3D printing. In further demonstration, Wang et al. [12] presented a hybrid optimization approach known as pLGA-AIP in solving the problem of nurse rostering. This approach jointly integrates both integer programming and local optimization search in a time-constrained greedy search in treating the nurse scheduling problem, which is balancing preferences and trade-offs.

III. METHODOLOGY

A. Objective

The main objective is to implement algorithms for solving text formatting problems (Printing Neatly) and to provide an

Authorized licensed use limited to: Amrita School of Engineering. Downloaded on August 27,2025 at 18:04:23 UTC from IEEE Xplore. Restrictions apply.

experimental study of their running time and the quality of the solutions found. The problem concerns the neatly printing a paragraph of text with a mono-spaced font (all the characters have the same width) from a printer. The input is a sequence of n words of lengths l1, l2, ., ln, measured in characters. The goal is to print this paragraph on a number of lines, each of which holds at most M characters. The study goes on to explore three kinds of alignments, namely left, center, and right.

B. Input

Inputs: A string of words provided by the user.

Threshold: The maximum number of characters allowed in one line, say M.

Alignment Option: The type of alignment of the character(s) within the line, which can be left, right, or center.

C. Processes and Algorithms

Dynamic Programming: The dynamic algorithm is implemented based on a cost matrix that presents potential breaking points and their evaluation to meet the optimal solution. The algorithm in one of the matrices fills out every cell in a way that the cost reflected in that cell is a result of placing the words from one index to another in one line. The evaluated matrix provides the minimum cost of breaking text into lines. In this regard, dynamic programming is more efficient compared to recursive algorithms, for it does not have to evaluate the same results more than once. The results are presented in terms of the constructed lines, total cost, and time of execution.

Greedy Approach: The greedy approach of this problem selects words to be added to a line one after the other until the threshold is surpassed. It evaluates the cost based on the remaining space in lines. The algorithm evaluates starting with an empty line and starts adding words one after the other. If adding one more word exceeds the threshold, then it starts a new line. The method is efficient, but it does not always have an optimal solution. The output is in terms of the constructed lines, total cost, and execution time.

Branch and Bound: The branch and bound approach uses a bounding function to prune the search space and then investigates the possible breaks through recursive methods. This method uses a systematic process of taking partial solutions and dropping those that cannot present an optimal solution in order to decrease the number of combinations to be checked. Generally, the algorithm will subdivide the problem into sub problems, calculate costs, and prune sub optimal solutions. The output is in terms of the constructed lines, total cost, and execution time.

Divide and Conquer: It reduces the problem to the smallest sub problems, solves each of these sub problems recursively, and then combines the solutions. It searches for text through a binary search and solves a ”Printing Neatly” problem for every half of the given text. Then it merges. Large inputs can be efficiently managed with this strategy because they will be divided into more accessible segments. The number of constructed lines, the total cost, and the execution time will form the output.

Binary Search and Shortest Path: A binary search is used to determine the line breaks, which include an optimal set of spaces in each of the possible space subsets for construction. Shortest path applies a binary search strategy to ascertain the cheapest way to break the text. In the output, constructed lines will first be written and finally the total cost, followed by the execution time showing how efficient the algorithms are at finding almost optimal solutions.

Brute Force Approach: The brute force approach is the method of formulating all possible ways to break text into lines and calculating the cost for each of those combinations. The cost is calculated by the deviation from the threshold in a way that costs are minimized. The algorithm builds lines out of the combination that resulted in the lowest cost. It works by going through successive combinations of word indices and calculating line lengths and the sum of squared differences from the threshold. The approach is exhaustive; it ensures that it finds an optimal solution. The output that can be expected consists of the constructed lines, the total cost, and the execution time.

D. Experimental Study

The cost to which each of the algorithms is tested is the cube of empty places that a line carries. The algorithms are experimented with several text paragraphs to evaluate their performance under different cases:

Case Study 1: Less Number of Words in the Paragraphs: Tests the algorithms with paragraphs containing a small number of words.

Case Study 2: Random Paragraphs: Uses a number of randomly generated paragraphs to evaluate the algorithms.

Case Study 3: Fixed Character Paragraph: Tests the algorithms with paragraphs that have a fixed number of characters within a line.

Case Study 4: Fixed Word Paragraph: Evaluates the algorithms with paragraphs that have a fixed number of words per line.

E. Output and Display

It is run as a graphical user interface (GUI), in which the user inputs the given text and threshold and selects whether to perform an alignment. The buttons available on the interface allow the user to run each algorithm, and the results include the constructed line, total cost, and runtime. Interaction of the user is facilitated because the user has a visual presentation of the solutions presented as well as making it easier to compare different algorithms.

F. Analysis

The runtime, cost, and quality of each algorithm are recorded for analysis in the final study. This then leads to the understanding of the trade-offs between approaches, as regards the computational efficiency and quality of the solutions. The study is carried out to come up with the best algorithm to be

Authorized licensed use limited to: Amrita School of Engineering. Downloaded on August 27,2025 at 18:04:23 UTC from IEEE Xplore. Restrictions apply.

used in neatly printing a paragraph, and it is guided by the performance and quality of the solutions. .

IV. IMPLEMENTATION

The Problem implements a comprehensive framework for solving the text formatting problem, featuring a graphical user interface (GUI) developed using the tkinter library. The GUI facilitates user interaction by including a text input area for users to provide the text, input fields to set the page width, and radio buttons to select alignment options such as left, center, and right. Additionally, the interface offers buttons to run various algorithms, including Greedy Approach, Dynamic Programming, Branch and Bound, Brute Force, Divide and Conquer, Binary Search, and Shortest Path.

The Brute Force Approach generates all possible ways to break the text into lines and calculates the cost for each combination. The cost is determined by the deviation from the threshold, aiming to minimize this deviation. The Branch and Bound method employs a bounding function to prune the search space and recursively explores possible breaks. It systematically considers partial solutions and abandons those that cannot lead to an optimal solution. By dividing the problem into sub problems, calculating costs, and pruning sub-optimal solutions, this method effectively reduces the number of combinations to evaluate.

In the Greedy Approach, the algorithm iteratively constructs lines by adding words until the threshold is exceeded. It calculates the cost based on the remaining space in each line and starts a new line once adding another word would exceed the threshold. The Dynamic Programming approach calculates a cost matrix for possible breaking points and uses it to find the optimal solution. By filling a matrix where each entry represents the cost of placing words from one index to another in a single line, the algorithm computes the minimum cost for breaking the text into lines. It stores intermediate results to avoid redundant calculations, making it more efficient than recursive algorithms. The outputs include the constructed lines, total cost, and execution time.

The Divide and Conquer tactic divides the problem into smaller sub problems, combines the solutions to each, and solves each recursively. The text is divided into two sections, each with a problem that is solved, and the solutions are combined. The outputs include the constructed lines, total cost, and execution time.

Binary Search and Shortest Path are distinct approaches used to find the optimal line breaks and minimize the cost. Binary search iteratively narrows down the range of possible solutions by comparing the middle value with the target cost. On the other hand, the shortest path algorithm identifies the least costly way to break the text into lines. Each approach outputs the constructed lines, total cost, and execution time, demonstrating their efficiency in finding near-optimal solutions independently.

The GUI displays the results of the algorithms, including the constructed lines, total cost, and execution time, allowing users to interact with the interface to compare the performance and quality of different algorithms. The analysis involves recording and comparing the runtime, cost, and quality of solutions produced by each algorithm. The study aims to identify the most effective algorithm for neatly printing a paragraph, balancing computational efficiency and solution quality.

V. RESULTS AND ANALYSIS

The objective of the study is to implement and compare the performance of various algorithms to solve the text formatting problem, in particular, looking at a clean paragraphed output with mono spaced out fonts. The algorithms were tested for their efficiency, measured in terms of runtime, cost, and the quality of the formatted output. The main aim was to minimize the cost, defined as the cube of the number of empty line but still maintains readability and aesthetic beauty of the text. Subsection summaries of the findings are given below from the experiments performed for the various case studies. The output for each of the algorithms was collected and ana- Examined to get their effectiveness for diverse situations. This was done by applying the algorithms to the document's paragraphs with the various features such as low word count, randomness content, fixed length of characters of a character and fixed number of characters of the word. The algorithm performance was evaluated using a test on its print the well formatted output within the given limits. The runtime of each algorithm has been measured quite accurately. to get an idea of its computational efficiency. This is essential For all practical applications where speed constitutes a major criterion. Runtime as well as the overall cost was calculated, giving insights to how good each of the algorithms minimized space wastage this way ensuring that space is used correctly and efficiently. Another typical characteristic of good formatting is that, in addition to numerical measures, it provides a likeness of text was checked visual. This comprised the text alignment (left, center or right) was preserved and making sure that the readable. These qualitative features are required since they affect the usability of the formatted text in the real world applications.

A. Research questions addressed:

1) ***How do different algorithms perform in terms of runtime and solution quality when formatting paragraphs with a mono-spaced font?:*** The performance of text alignment algorithms varies significantly in terms of runtime and solution quality when formatting paragraphs with a mono-spaced font. Dynamic Programming is highly efficient for larger inputs due to its ability to exploit optimal substructure and overlapping subproblems, operating with a runtime of O(n2), where n is the number of words. It produces high-quality solutions by examining all possible breaks and choosing the optimal one, thus minimizing whitespace and ensuring neatly formatted text.

The Greedy Approach is very fast, with a runtime of O(n), as it makes decisions in a single pass through the text. However, it may not always provide the optimal solution since it only considers local optimality, potentially leading

Authorized licensed use limited to: Amrita School of Engineering. Downloaded on August 27,2025 at 18:04:23 UTC from IEEE Xplore. Restrictions apply.

to suboptimal results in the overall formatting. Branch and Bound is more computationally intensive, especially for larger inputs, as it systematically explores all possible breaks but uses bounds to prune the search space, balancing exhaustive search and efficiency to provide near-optimal solutions.

Divide and Conquer efficiently handles larger inputs by breaking the problem into smaller subproblems and combining their solutions, operating with a runtime of O(nlogn) and producing high-quality solutions through recursive solving. Binary Search leverages binary search to optimize line breaks, with O(nlogn) complexity, balancing runtime and solution quality effectively. Shortest Path dynamically updates minimum costs and breaks to provide high-quality solutions with a runtime of O(n^2). Brute Force guarantees the optimal solution by evaluating all possible breaks, but it is computationally prohibitive for large inputs due to its O(2^n)complexity, making it impractical for large texts despite ensuring the best solution.

*2) **How does the alignment type (left, center, right) impact the performance and quality of the formatted text?:*** The alignment type plays a crucial role in shaping both the performance and the quality of the formatted text. Left alignment is generally the easiest to implement and often delivers a balanced outcome across various algorithms. It usually ensures consistent performance in terms of both runtime and solution quality, as it naturally aligns text and minimizes unnecessary whitespace on the left side. On the other hand, center alignment can be more demanding computationally, especially for algorithms that need to calculate the exact midpoint for alignment. While it often results in a more visually appealing format, it can introduce more whitespace, which impacts the overall cost. Algorithms such as Dynamic Programming, Divide and Conquer, and Shortest Path handle center alignment effectively, maintaining high-quality results with minimal additional computational load.

Right alignment tends to be more complex to calculate because it requires precise adjustments to ensure the text lines up correctly on the right side. This alignment can increase the total cost due to additional whitespace on the left. However, algorithms like Binary Search and Dynamic Programming manage right alignment efficiently, producing high-quality solutions without significantly affecting runtime. In summary, while left alignment is generally the most efficient, center and right alignments offer better visual appeal but at the expense of potentially higher computational complexity and increased whitespace.

*3) **How do specific parameters, such as threshold values and text length, affect the performance and results of each algorithm?:*** Threshold values and text length are critical in shaping how well each algorithm performs. The threshold, or the maximum number of characters allowed per line, has a direct impact on an algorithm's efficiency. For example, a higher threshold usually means fewer lines and less computational complexity, whereas a lower threshold increases the number of lines, potentially boosting runtime and total cost. Algorithms like Greedy and Brute Force are particularly sensitive to these threshold values because they directly influence the number of iterations and combinations to be evaluated. The length of the input text also plays a significant role in an algorithm's performance. Longer texts add to the computational load, especially for algorithms like Brute Force, which have exponential complexity. In contrast, algorithms like Dynamic Programming and Shortest Path manage longer texts more efficiently thanks to their optimized methods for breaking down and solving problems. In essence, higher thresholds and shorter texts generally lead to better performance, while lower thresholds and longer texts challenge the algorithms' efficiency and effectiveness. This underscores the importance of selecting the right parameters to achieve optimal results.

*4) **What are the trade-offs between different algorithms when applied to text formatting?:*** Each text formatting algorithm comes with its own set of trade-offs between runtime, solution quality, and computational complexity. Dynamic Programming strikes a balance between speed and quality, delivering high-quality solutions without overly taxing computational resources. It's a great choice for scenarios where you need optimal results without overwhelming your system. The Greedy Approach, on the other hand, focuses on speed, making quick decisions that are perfect for real-time applications. However, this can sometimes lead to less-than-perfect formatting since it only looks at the immediate gain.

Branch and Bound is more thorough, exploring many possible solutions and pruning the less promising ones, which leads to near-optimal results. This thoroughness, though, comes at the cost of increased computational time, making it better for situations where quality is more critical than speed. Divide and Conquer excels at handling large texts efficiently, breaking them into manageable pieces and solving them for high-quality outcomes. It's ideal for applications that need to process extensive texts efficiently. Binary Search offers a good middle ground, effectively balancing speed and solution quality by leveraging binary search to find optimal line breaks.

It's suitable for applications where you need efficient and reasonably good results. Shortest Path also ensures high-quality solutions by dynamically finding the best path through the text, balancing runtime and complexity well, making it perfect for scenarios where both speed and quality matter. Lastly, Brute Force guarantees the best possible solution by considering all possible combinations. However, this comes with the downside of impractically long runtimes for larger texts, making it best for small inputs or situations where you have plenty of time and computational resources.

B. Experimental Study:

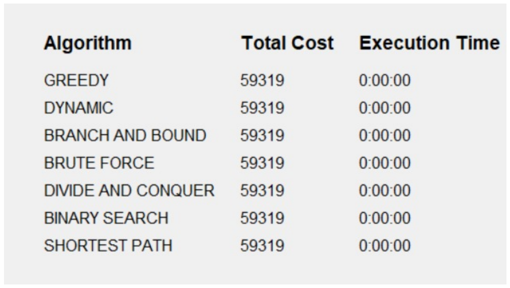
The algorithms were tested on various text paragraphs to assess their performance under different scenarios:

Case Study 1: Less Number of Words in the Paragraphs:

For an input text "Hello World" and page width of 50, The total cost and execution time are fixed for all algorithms across all three justifications (left, center, right). The overall cost is 59319 for all algorithms and alignments. The runtime is 00:00:00 for all algorithms. The uniform total cost across all the algorithms present the actual input text to be ("Hello

Authorized licensed use limited to: Amrita School of Engineering. Downloaded on August 27,2025 at 18:04:23 UTC from IEEE Xplore. Restrictions apply.

World") and page width (50) have the same formatting impact for each algorithm.

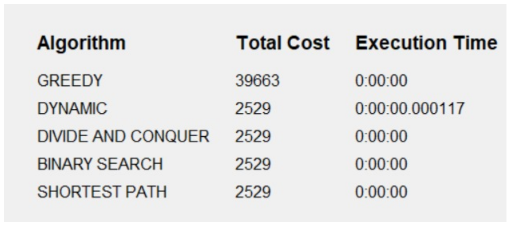


Algorithm	Total Cost	Execution Time
GREEDY	59319	0:00:00
DYNAMIC	59319	0:00:00
BRANCH AND BOUND	59319	0:00:00
BRUTE FORCE	59319	0:00:00
DIVIDE AND CONQUER	59319	0:00:00
BINARY SEARCH	59319	0:00:00
SHORTEST PATH	59319	0:00:00

Fig. 1. Case 1 results

Case Study 2: Random Paragraphs:

For the input text "Nature is the beautiful and diverse world around us, including plants, animals, mountains, rivers, and the sky. It provides us with fresh air to breathe, water to drink, and food to eat. Nature changes with the seasons, showing different colors and landscapes throughout the year. It offers a peaceful escape from our busy lives, where we can relax and enjoy the serenity of a forest, the sound of waves at the beach, or the sight of a colorful sunset." with a page width of 50, the results vary between the algorithms for all three alignments (left, center, right).



Algorithm	Total Cost	Execution Time
GREEDY	39663	0:00:00
DYNAMIC	2529	0:00:00.000117
DIVIDE AND CONQUER	2529	0:00:00
BINARY SEARCH	2529	0:00:00
SHORTEST PATH	2529	0:00:00

Fig. 2. Case 2 results for left alignment

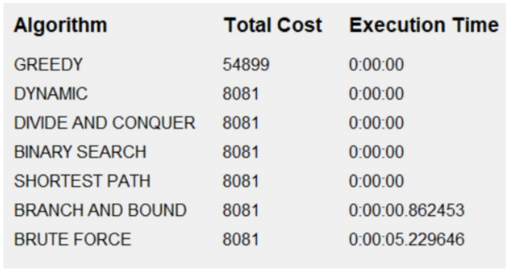
While the Dynamic, Divide and Conquer, Binary Search, and Shortest Path algorithms all reached a total cost of 2529 with low execution times, the Greedy algorithm produced a result for left alignment with a total cost of 39663 and an execution time of 0:00:00. The outcomes for center alignment displayed the same total costs and execution times for each method as those for left alignment. Similarly, for right alignment, the other algorithms all had a total cost of 2529, with the Divide and Conquer algorithm having a slightly longer execution time of 0:00:00.001. In contrast, the Greedy algorithm had a total cost of 39663 and an execution time of 0:00:00.

For the given input text and page width of 50, the results demonstrate that the Greedy algorithm performs worse in terms of total cost compared to the Dynamic, Divide and Conquer, Binary Search, and Shortest Path algorithms. The execution times are very similar across all algorithms, indicating that the performance differences are primarily in cost optimization.

Setting up limit values in the page width for branch and bound , brute force algorithms is the main idea for making computation efficient and effective. They play a major role in managing the complexity and resource utilization of the algorithm and, in this process, may affect the user experience to a large extent. Limits are also controls on resources like memory and processing power, the lack of which therefore does not lead to performance hits or system crashes. Also, limiting the output size helps to make the generated output readable and usable. All in all, limit value setting in algorithms enables scalability through a compromise in algorithmic complexity and performance to assure optimal operation throughout a range of scenarios.

Case Study 3: Fixed Character Paragraph:

The input text used for the alignment algorithms is: "The quick brown fox jumps over the lazy dog. Every good boy does fine. Pack my box with five dozen liquor jugs." The page width specified for the alignment is 50 characters. The performance metrics for left, center, and right alignments are evaluated.



Algorithm	Total Cost	Execution Time
GREEDY	54899	0:00:00
DYNAMIC	8081	0:00:00
DIVIDE AND CONQUER	8081	0:00:00
BINARY SEARCH	8081	0:00:00
SHORTEST PATH	8081	0:00:00
BRANCH AND BOUND	8081	0:00:00.862453
BRUTE FORCE	8081	0:00:05.229646

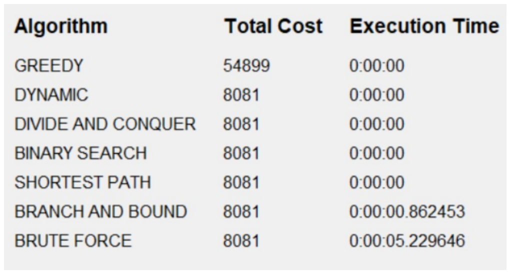
Fig. 3. Case 3 results for left alignment

As shown in the Figure 5 across all alignments, the Greedy algorithm shows the highest total cost of 54899, with execution times nearly instantaneous, ranging from 0 to 0.0007 seconds. This indicates that while the Greedy algorithm is very fast, it is less efficient in minimizing the cost of line breaks compared to other methods. In contrast, the Dynamic Programming, Divide and Conquer, Binary Search, and Shortest Path algorithms all achieve the same optimal total cost of 8081, with execution times almost instantaneous, in the order of microseconds to milliseconds. This suggests that these algorithms are highly effective in finding the optimal solution for text alignment.

The Branch and Bound algorithm also achieves the optimal total cost of 8081, but its execution time ranges from approximately 0.72 to 0.95 seconds. This indicates that while it finds the optimal solution, it takes significantly longer to execute compared to the fastest algorithms. The Brute Force algorithm, which also finds the optimal solution, has the longest execution time among all methods, ranging from approximately 4.695219 to 5.47 seconds. This highlights its inefficiency in terms of execution time, especially for larger inputs.

In conclusion, the Greedy method is the fastest but least efficient in minimizing the cost for the given text and page width. Algorithms like Binary Search, Dynamic Programming, Divide and Conquer, and Shortest Path may find the best answer fast and effectively, which makes them appropriate for real-time applications. Although it works well, the Branch

Authorized licensed use limited to: Amrita School of Engineering. Downloaded on August 27,2025 at 18:04:23 UTC from IEEE Xplore. Restrictions apply.



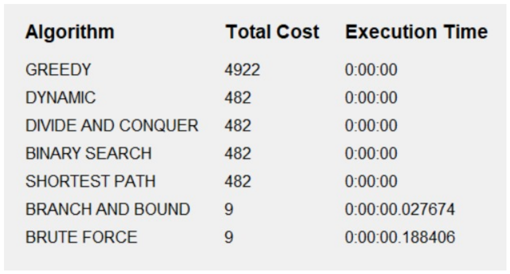
Algorithm	Total Cost	Execution Time
GREEDY	54899	0:00:00
DYNAMIC	8081	0:00:00
DIVIDE AND CONQUER	8081	0:00:00
BINARY SEARCH	8081	0:00:00
SHORTEST PATH	8081	0:00:00
BRANCH AND BOUND	8081	0:00:00.862453
BRUTE FORCE	8081	0:00:05.229646

Fig. 4. Case 3 results for center alignment

and Bound approach is slower and better suited for situations when execution time is not as important. The Brute Force algorithm's much longer execution time makes it unfeasible as input size increases, despite its dependability for tiny inputs. These findings highlight how crucial it is to select the best algorithm by weighing the trade-off between solution quality and execution time.

Case Study 4: Fixed Word Paragraph: The input text for the alignment algorithms in this case is: "The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog." The page width specified for the alignment is 30 characters, with a specific constraint set for Branch and Bound and Brute Force algorithms: the limit for words per line is 25. The performance metrics for left, center, and right alignments are evaluated.

The algorithm selection consistently affects the performance metrics in terms of cost and execution time, rather than the alignment itself, even though each alignment has a different visual style.



Algorithm	Total Cost	Execution Time
GREEDY	4922	0:00:00
DYNAMIC	482	0:00:00
DIVIDE AND CONQUER	482	0:00:00
BINARY SEARCH	482	0:00:00
SHORTEST PATH	482	0:00:00
BRANCH AND BOUND	9	0:00:00.027674
BRUTE FORCE	9	0:00:00.188406

Fig. 5. Case 4 results for right alignment

The Greedy method is still the fastest in this case study with a set word paragraph and a page width of 30 characters, but it shows a significant cost because there are fewer ideal line breaks. An effective trade-off between speed of execution and spacing quality can be found in the algorithms Divide and Conquer, Shortest Path, Dynamic Programming, and Binary Search. With a word limit of 25, the Branch and Bound and Brute Force algorithms guarantee the least amount of line break cost; the former provides a better trade-off between speed and efficiency. Because of this, Branch and Bound are especially well suited for applications where some delay is tolerable but output quality is essential. Despite being extremely accurate, the Brute Force the Brute Force method's poor performance makes it less practical for larger or time-sensitive applications.

TABLE I
ANALYSIS OF TIME COMPLEXITY FOR TEXT FORMATTING ALGORITHMS

Algorithm	Best Case	Average Case	Worst Case
Greedy	O(n)	O(n)	O(n)
Dynamic Programming	O(n^2)	O(n^2)	O(n^2)
Brute Force	O(2^n)	O(2^n)	O(2^n)
Branch and Bound	O(n) in best case	Varies, often between O(n) and O(2^n)	O(2^n)
Binary Search	O(log n $\times$ check time)	O(log n $\times$ check time)	O(log n $\times$ check time)
Divide and Conquer	O(n log n)	O(n log n)	O(n^2)
Shortest Path	O(n^2)	O(n^2)	O(n^2)

The table I presents a comparative analysis of several algorithms used in text formatting tasks. For each algorithm, the table provides the best case, average case, and worst case complexities. These metrics are crucial for understanding how efficiently an algorithm can process data under varying conditions. The complexities are expressed using Big O notation, which describes the upper limit of the algorithm's running time or space requirement in terms of the size of the input data.

When choosing the right algorithm for a certain application, it is crucial to analyze these complexity. This is especially true for applications like word processors, programming environments, and content management systems, where efficient text formatting is necessary.

C. Discussion:

By Analyzing the results of the four case studies, The observation was distinct performance patterns among the algorithms. In Case Study 1, with a simple input like "Hello World" and a page width of 50, all algorithms performed uniformly, resulting in the same total cost of 59319 and negligible execution times. This indicates that for very small inputs, the choice of algorithm has little impact on performance or quality. In Case Study 2, which involved random paragraphs, the Greedy algorithm consistently showed higher total costs (39663) compared to other algorithms, which maintained similar execution times and achieved a total cost of 2529. This highlights that while Greedy is fast, it is less effective in cost optimization.

In Case Study 3, involving a fixed character paragraph, again saw the Greedy algorithm exhibit the highest total cost of 54899 with minimal execution times. In contrast, Dynamic Programming, Divide and Conquer, Binary Search, and Shortest Path algorithms consistently achieved optimal costs of 8081 with very fast execution times. Although Branch and Bound and Brute Force also achieved optimal costs, their significantly longer execution times demonstrated their impracticality for larger inputs.

In Case Study 4, with a fixed word paragraph, the Greedy algorithm remained the fastest but incurred high costs due to fewer ideal line breaks. Algorithms like Divide and Conquer,

Authorized licensed use limited to: Amrita School of Engineering. Downloaded on August 27,2025 at 18:04:23 UTC from IEEE Xplore. Restrictions apply.

Shortest Path, Dynamic Programming, and Binary Search balanced speed and cost efficiency effectively. Branch and Bound, while slower, guaranteed minimal line break costs, making it suitable for applications prioritizing quality over speed. The Brute Force algorithm, despite being accurate, proved impractical for larger inputs due to its long execution times. Overall, the results underscore the importance of choosing the right algorithm based on specific needs, balancing execution time and solution quality.

VI. CONCLUSION

The study focused on comparing different algorithms to determine the best method for formatting text using monospaced fonts. The goal was to find an algorithm that could efficiently format paragraphs while minimizing the cost of unused space in each line. The performance of each algorithm was tested using various types of text, such as paragraphs with random content or fixed character lengths. These tests were designed to assess each algorithm's effectiveness in different scenarios and to evaluate their runtime, cost, and the overall quality of the output. The study revealed that while some algorithms were quick, they did not necessarily offer the best value in terms of space efficiency, which was a key consideration. Algorithms like Dynamic Programming, Divide and Conquer, Binary Search, and Shortest Path were identified as optimal for real-time applications due to their ability to balance speed and formatting quality effectively. On the other hand, the Brute Force and Branch and Bound algorithms, despite their accuracy, were less practical for larger or time-sensitive tasks due to their longer execution times. The research highlighted the importance of selecting the right algorithm based on specific needs and constraints of the text formatting task. For applications where speed is less critical, more thorough algorithms that can deliver high-quality formatting are preferable, whereas for quick processing with moderate data sizes, faster algorithms may be more suitable. This understanding helps in choosing the appropriate algorithm for various applications in word processing, programming environments, and content management systems, ensuring efficient and effective text formatting.

REFERENCES

[1] J. A. Ayoola, E. O. Asani, A. E. Okeyinka and P. O. Ayegba, "Towards Comparative Analysis Of Branch - And - Bound And Nearest Neighbour Algorithms," 2020 International Conference in Mathematics, Computer Engineering and Computer Science (ICMCECS), Ayobo, Nigeria, 2020, pp. 1-5, doi: 10.1109/ICMCECS47690.2020.240901

[2] Z. Wang, Y. Han and B. Xiao, "Research on Resource Scheduling Algorithm Based on Greedy Strategy," 2022 International Conference on Computer Network, Electronic and Automation (ICCNEA), Xi'an, China, 2022, pp. 76-81, doi: 10.1109/ICCNEA57056.2022.00027.

[3] A. F. Iskandar, A. F. Sani, R. Riyadi, S. Febriani and N. R. Syambas, "Fast Heuristic Algorithm Optimization for Travelling Salesman Problem," 2021 7th International Conference on Wireless and Telematics (ICWT), Bandung, Indonesia, 2021, pp. 1-6, doi: 10.1109/ICWT52862.2021.9678454.

[4] Z. Jin and H. Finkel, "Performance Evaluation of the Vectorizable Binary Search Algorithms on an FPGA Platform," 2020 IEEE/ACM 10th Workshop on Irregular Applications: Architectures and Algorithms (IA3), GA, USA, 2020, pp. 63-67, doi: 10.1109/IA351965.2020.00014.

[5] Z. Yang, H. Xia, F. Su, J. Zhao and F. Feng, "Application of Genetic Algorithm in modelling of Shortest Path problem," 2020 Chinese Automation Congress (CAC), Shanghai, China, 2020, pp. 3447-3450, doi: 10.1109/CAC51589.2020.9327269.

[6] M. Naghibzadeh, S. Babaei, B. Behkmal and M. Hatami, "Divide and Conquer Approach to Long Genomic Sequence Alignment," 2021 11th International Conference on Computer Engineering and Knowledge (ICCKE), Mashhad, Iran, Islamic Republic of, 2021, pp. 399-405, doi: 10.1109/ICCKE54056.2021.9721501.

[7] Y.-M. Wang, T. Sun, S. Ma, and L.-M. He, "An Improved Data-Driven Adaptive Dynamic Programming Approach for Cooperative Games," in 2023 42nd Chinese Control Conference (CCC), Tianjin, China, 2023, pp. 8109-8114. DOI: 10.23919/CCC58697.2023.10240369.

[8] S. Samanta Singhar, B. N. B. Ray, A. K. Dash and A. Malla, "Optimizing Mixed Size and Large Scale Block Placement Using Greedy Approach," 2019 International Conference on Information Technology (ICIT), Bhubaneswar, India, 2019, pp. 442-447, doi: 10.1109/ICIT48102.2019.00084.

[9] M. Posypkin and S. T. Thant Sin, "Comparative Performance Study of Shared and Distributed Memory Dynamic Programming Algorithms," 2020 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus), St. Petersburg and Moscow, Russia, 2020, pp. 2010-2013, doi: 10.1109/EIConRus49466.2020.9039470.

[10] X. Shen and W. Lam, "Improved Divide-and-Conquer Approach to Abstractive Summarization of Scientific Papers," 2022 4th International Conference on Natural Language Processing (ICNLP), Xi'an, China, 2022, pp. 395-398, doi: 10.1109/ICNLP55136.2022.00073.

[11] Z. Tang, G. Chen, Y. Han and Y. Zhu, "An Dynamic Adaptive Slicing Algorithm Based on Improved Greedy Algorithm," 2022 3rd International Conference on Computer Vision, Image and Deep Learning & International Conference on Computer Engineering and Applications (CVIDL & ICCEA), Changchun, China, 2022, pp. 1060-1063, doi: 10.1109/CVIDLICCEA56201.2022.9825027.

[12] T. Wang, S. Huang, Z. Zhang and D. Shen, "Hybrid Optimization for Nurse Rostering Problem : Integrating Greedy Solution and Local Genetic Algorithm with Dynamic Encoding," 2023 9th International Conference on Big Data and Information Analytics (BigDIA), Haikou, China, 2023, pp. 118-123, doi: 10.1109/BigDIA60676.2023.10429601.

[13] D. Wu, J. Lang and G. Liu, "Research on the application of algorithms based on the greedy algorithm and optimal model," 2022 International Conference on Applied Physics and Computing (ICAPC), Ottawa, ON, Canada, 2022, pp. 100-103, doi: 10.1109/ICAPC57304.2022.00025.

[14] P. Chaitanya, "Binary Search Sort Algorithm -Yet Another Sorting Algorithm with Binary Search having O(nlogn) and O(n) time complexity," 2023 1st International Conference on Optimization Techniques for Learning (ICOTL), Bengaluru, India, 2023, pp. 1-6, doi: 10.1109/ICOTL59758.2023.10435020.

[15] S. Mi, Z. Ji, H. Zheng and Y. Gao, "A shortest path algorithm of long-period cyclic fully connected layer graph based on Dijkstra algorithm," 2020 International Conference on Advanced Mechatronic Systems (ICAMechS), Hanoi, Vietnam, 2020, pp. 323-327, doi: 10.1109/ICAMechS49982.2020.9310152.

[16] Wang, Yizhun. (2023). Review on greedy algorithm. Theoretical and Natural Science. 14. 233-239. 10.54254/2753-8818/14/20241041.

[17] A. Aziz, S. Tasfia and M. Akhtaruzzaman, "A Comparative Analysis among Three Different Shortest Path-finding Algorithms," 2022 3rd International Conference for Emerging Technology (INCET), Belgaum, India, 2022, pp. 1-4, doi: 10.1109/INCET54531.2022.9824074.

[18] Souza, Flávio & Couto, Bráulio & Fernandes, Gustavo. (2023). PARALLELIZATION OF SHORTEST PATH CLASS ALGORITHMS: A COMPARATIVE ANALYSIS. Pesquisa Operacional. 43. 10.1590/0101-7438.2023.043.00272130.

[19] Verma, Rajat & Dhanda, Namrata & Nagar, Vishal. (2022).Enhancing Security with In-Depth Analysis of Brute-Force Attack on Secure Hashing Algorithms. 10.1007/978-981-16-8826-3_44.

[20] 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," 2023 International Conference on Information Management and Technology (ICIMTech), Malang, Indonesia, 2023, pp. 340-345, doi: 10.1109/ICIMTech59029.2023.10277955.

Authorized licensed use limited to: Amrita School of Engineering. Downloaded on August 27,2025 at 18:04:23 UTC from IEEE Xplore. Restrictions apply.