International Journal of Computing and Digital Systems
2025, VOL. 17, NO.1, 1-11

http://dx.doi.org/10.12785/ijcds/1571006296

Revolutionizing Transportation and Logistics: Dynamic Programming and Bit Masking Approach for Optimizing the Travelling Salesman Problem

Vaibhav Shoran[1], Harsh Dabas[1], Geetanjali Rathee[1], Nitin Rakesh[2], Pratik Agrawal[2] and Monali Gulhane[2]

[1]*Department of Computer Science and Engineering, Netaji Subhas University of Technology, Delhi, India*
[2]*Symbiosis Institute of Technology Nagpur Campus, Symbiosis International (Deemed University), Pune, India*

Received 9 March 2024, Revised 22 January 2025, Accepted 29 January 2025

Abstract: The traveling salesman problem (TSP) is a contemporary challenge in transportation and logistics, heavily affecting decisions, product allocation, and routing. This latest article addresses the NP-hard TSP problem by bit masking and enhanced dynamic programming methods. The proposed method not only improves the TSP solution process, but also incorporates interaction diagrams to improve the understanding of the results. Efficiency is essential in a globally connected world, and this approach ensures efficiency and flexibility. Advanced algorithmic solutions facilitate optimal route planning for e-Commerce and telecom infrastructure projects. Unlike heuristics, this method assumes pathfinding solution positions, guaranteeing optimality where accuracy is most important. It also provides insights into new combinatorial optimization problems, bridging the gap between theoretical difficulties and practical applications. User-friendly graphics further reinforce the effectiveness of the algorithm, providing important information to decision makers. The paradigm shift in TSP optimization explored in this work promises accuracy, scalability, and flexibility in global resource allocation and path planning, setting new standards for combinatorial optimization algorithms and demonstrating their potential with a broad range of applications to meet industry requirements.

Keywords: Optimization algorithm, Driven delivery negotiation, Travelling salesman problem, Heuristics Algorithm, Approximation and exact mechanisms

1. INTRODUCTION

TSP(Travelling Salesman Problem) is one of the hardest problems with the logistics sector and transportation today. It hugely affects the process of choosing, how profit is shared, and priorities for the routes in the future. However, the optimization of resources is among the top priorities now as is logistic reoccurring because of the world's high interconnectedness. If TSP is an influential traditional optimization problem, then it might help eradicate them by utilizing their power. This research paper looks at Dynamic Programming and Bit masking solutions of TSP from two different perspectives. The main objectives of this investigation are that they not only enhance the approaches that are used for solving problems, but also provide a new way to look at the issues

by combining two different existing algorithms. With the current world turning into a fast track, the significantly increasing marketability of e-commerce delivery networks and telecom infrastructure planning is the growing need for optimization approaches that are as accurate and scalable as possible. To justify the means and maximize the profit TSP methods sometimes demand from a side making sacrifices in the striving for good result. Within the dynamic programming research context, this method has been developed to remove the constraints such as the DAG-shaped CON that deals with the bit masking precision. The bi-masking application not only decreases non-required information, but it also leads up such a tech, by means of which the optimal solution for TSP is based: an exact and scalable one, with all features taken to account of the particular properties of TSP optimization. When, in any case, it is all about the greatest possible accuracy, like in cases where an inefficient approach might bring about very high costs, our method will stand for more as our software can be particularly helpful at those points. The suggested algorithm is distinct from its heuristic counterparts by systematically splitting up solution spaces, and content must be stored in

E-mail address: vaibhav.shoran.ug20@nsut.ac.in,harsh.dabas.ug20@nsut.ac.in,geetanjali.rathi@nsut.ac.in,nitin.rakesh@gmail.com, pratik.agrawaal@gmail.com,monaligulhane4@gmail.com

them. Furthermore, interactive visualization augments the quality of the appreciation of the results, leading to giving the decision makers the essential bits of information[1]. This work will be presented in the algorithm which attains optimality as well as efficiency therefore bring to light paradigm shift in TSP optimization that is not only sweeping but also revolutionary. By the research conducted, the guideline with the combinational optimization algorithms has been resettled, and this appears in the market where a lot of companies are competing. Based on this, it is expected that TSP in its resolution will see the greatest improvements because the data obtained from this research will not only allow to resolve TSP problems but it will also help to solve more difficult combinatorial optimization issues. Additionally, besides serving pragmatic purposes of the theoretical problems, the visualization component is more user-friendly and thus further proves the algorithm's effectiveness in any real-world encounter. The Traveling Salesman Problem [2], [3] and [4] is recognized as the most difficult combinatorial optimization problem. It plays the main role in this project. The traveling salesman problem is goal-oriented: the problem is to define the optimum cost of the path. A path that visits every vertex in the set exactly once and then returns to the beginning point while minimizing the total sum of edge costs is said to be optimal. Travelling Salesman Problem is best described by Hamiltonian Circuit- a concept in graph theory and combinatorial optimization. It refers to a specific type of cycle in an undirected graph that visits every vertex (node) exactly once and returns to the starting vertex. In simpler terms, a Hamiltonian cycle is a closed path in a graph that travels through every vertex exactly once, without repeating any vertex except the starting and ending vertex [5]. The Traveling Salesman Problem is classified as NP Hard. The temporal complexity to solve the problem exhibits a sharp and sudden jump as the problem size (number of nodes, in this case) rises.[6] Computational problems that are as difficult as the hardest ones in the NP (Nondeterministic Polynomial-Time) complexity class are referred to as NP-hard issues. "Nondeterministic Polynomial-Time hard" is referred to as "NP-hard." One characteristic of these issues is that they are hard to solve, especially when trying to solve them in polynomial time. The Traveling Salesman Problem is not NP Complete because it requires O(N!) time to determine whether a suggested solution is accurate, but NP Complete problems should only require Polynomial time and the second reason is NP Complete problems are decision problems but TSP is an optimality problem.

This paper will be focusing on improving the brute force algorithm [7] to solve Travelling Salesman problem by decreasing the Time Complexity with the help of Dynamic Programming [8] and Bit masking [9] and later integrating path finding in the algorithm. Further, it will help in visualization of the optimal path (decided by TSP) taken from a starting node. This study presents a novel approach to solving the traveling salesperson problem (TSP) by using bit masking and enhanced dynamic programming techniques Unlike traditional methods that often rely on heuristics, our approach finds a solution in multiple different areas systematically, ensuring quality and accuracy To summarize, our research novelty dynamic programming In a new combination of bit masking, interactive images are generated for the interpretation of results come to terms, the application of our method in critical applications addressing the limitations of existing methods and providing robust, scalable solutions is opened up by our work advances in combinatorial optimization and future research in TSP and related fields Sets new standards. This paper presents an algorithm that achieves optimality and efficiency, thereby introducing a paradigm shift in TSP optimization. Our approach shows promise for being precise, scalable, and transformative as industries around the world struggle with resource allocation and route planning. This paper showcases an impressive result of a human machine collaboration in TSP field due to the highest accuracy of the specially created algorithm and implementation of modern technologies. It discusses how there is a huge potential of algorithms in terms of bringing changes in the traditional norm of fine-tuning of combinatorial optimization, if they are solved. They can also tackle the current issues in the domain.

A. *Motivation*

Travelling salesman problem may be a classical one, but it has virtually numerous applications in manufacturing, logistics, transportation, and more. Tasks related to Traveling Salesman Problem will take less time, hence assets distribution will be optimal across transportation, logistics, and more ones. Algorithmic Challenge: An opportunity of resolving Traveling Salesman Problem is very exciting for an optimization approach at work. From an intellectual point of view it is certainly an attractive challenge to design algorithms capable to resolve the best or almost the best solutions to the given problems. The last step of cellular respiration is to excrete the by-product, which occurs during the process of combustion. The electron flow has come to the end of its journey, producing the ATP molecules. To illustrate, the sustainability offered by renewable energy is likely to substantially reduce the costs and increase the efficiency of operations. Companies can save fuel consumption, decrease transportation cost and reduce vehicle breakdown by optimizing routing for delivery using real time data. The precision, which is the algorithm feature, is its greatest advantage. Still, precise algorithms remain indispensable in a computerized problem-solving world [10]. though [11]. heuristic and approximations are becoming more and more popular. While exact algorithms bring in moderate solutions, Exact Algorithms are critical since of the multiples major reasons. Another factor is that they find the finest solutions, unlike others that have no proof that at the least their solution is the best possible one. When we address situations that rely on precision and need accuracy, for instance in procedures that are key to the safety of a subsystem, such as critical decision-making operations, then it is the credibility that comes from the optimal solution as being the correct one that is the most

essential. Noteworthy hence is that the exact algorithms are the benchmark for determining how well their heuristic and approximation algorithms preform. The validity and applicability of approximate techniques can be verified (or proved successful) by researchers and practitioners by comparing solutions derived from exact algorithms as well as faster algorithms, of which the latter is used to demonstrate the success of the former. Moreover, the better solution quality provided by exact algorithms may offset the computational expense in situations involving small to medium-sized problem instances where computational complexity is still controllable. The role of exact algorithms is constantly changing, securing their place in the ever-expanding toolkit of computational problem- solving methodologies, as long as algorithmic techniques continue to advance, including the creation of hybrid approaches that combine the advantages of exact and approximation methods.[12] Optimized delivery routes can reduce the carbon footprint of a company's transportation operations. This aligns with sustainability goals and contributes to an eco- friendlier approach to logistics. Scalability: As businesses grow, the complexity of delivery logistics increases. The TSP becomes even more challenging with a larger number of delivery locations. The paper addresses the scalability issue and provide solutions for businesses of all sizes. Optimizing delivery paths gives a competitive edge to rival businesses. Ultimately, the project has great scope in the market.

B. Research Significance

Algorithms to solve a problem are categorized intro 3 categories namely Exact Algorithms, Approximation Algorithms and Heuristic Algorithms. The latter two are often preferred over the Exact Algorithms due to their computational efficiency, resource constraint adaptability and Scalability qualities. Despite all of the qualities displayed by Approximation and Heuristic Algorithms, Exact Algorithms stand out on several occasions due to guarantee of optimality, certainty in the quality of solution and solutions from these can be easily audited and validated due to presence of rigorous Mathematics in the background. Alongside, solutions from Exact Algorithms serve as benchmark to Approximation and Heuristic Approaches. This paper aims to move further in the light of Exact Algorithms and provide an efficient Time and Space consuming Algorithm by using Recursion [9], Dynamic Programming , Bit-Masking which comes out to be a better alternate to the pre-existing Branch and Bound Solution [10] as unlike the Branch and Bound solution the alternate has low memory consumption, is easy to formulate (effective branching in Branch and Bound is a difficult task) and has easy implementation due to absence of bound and pruning rules. All these qualities make the alternate solution easy to develop and debug with better time and space complexities.

2. LITERATURE REVIEW

The review explain the research examines various elements of the Travelling Salesman Problem (TSP), demonstrating a diversity of approaches and applications. In the [5] article, the new notion of employing attraction factors from dynamical system theory to reduce the space of searches for the TSP is presented. The goal of this study is to minimise the inherent complexity of the TSP. Even though it does not provide any precise specifics on the findings or restrictions, this notion opens up opportunities for additional investigation. In the [13] study, heuristic techniques for optimising last-mile delivery routes are brought into emphasis. These approaches involve combining vehicle and several drones. Genetic algorithms have emerged as a potentially useful solution: they have shown improved performance in comparison to other approaches, with the method known as greedy having advantages in terms of processing efficiency. A new unsupervised training framework (UTSP) to solve the TSP is presented in the [7] researcher explains framework makes use of a Graph Neural Network that has been trained with an artificial loss. The UTSP algorithm outperforms other data-driven heuristics, highlighting the effectiveness of its parameter and data management. In the [8] author explains, basic insights into TSP are presented. These insights include a discussion of solution representation and the function that the edge matrix E plays in problem- solving. In the fifth study, an innovative approach is used by providing the TSP as an example of a navigational spatial task for rats. This provides a fresh viewpoint on applying the TSP in behavioural testing. In the [14] explains that the authors investigate the manner in which the Genetic Algorithm may be utilised to improve the routes that are produced by the Savings Method for TSP. With the [10], the first exact solution for a pickup-and-delivery TSP involving uncertainty is proposed. The study also highlights the difficulties and trade-offs that are involved. In the [11] article, a labelling technique is presented as a solution to the TSP issue. This method offers an alternate approach that comes to an end after a certain number of repetitions. This method allows determining different tours while simultaneously minimising the amount of computing complexity involved. The review that conducts a systematic investigation into the many methods that have been utilised to address the TSP [15]. This investigation identifies both precise and heuristic algorithms for solving the problem and places an emphasis on the NP-hard character of the issue. These studies, when taken as a whole, contribute to a more comprehensive knowledge of the TSP as well as potential solutions for it across a variety of application areas and computing paradigms. [16]As a result of the observations, shown in Table1 the various approaches that each research adopted to solve the Travelling Salesman Problem are brought to light. In some instances, the limits are mentioned openly, while in others, it is necessary to deduce them from the context.[2] Frequently, the limits are connected with the intrinsic complexity of the TSP or the possible difficulties that are associated with the approaches that have been presented.

3. METHODOLOGY

The primitive aim of the methodology is to improve the computation cost of the brute force exact algorithm.

TABLE I. WITH LIMITATIONS

Methods Use	Limitations
Attractor concept in dynamical systems theory.	Method adopted is complex
Mathematical formulation of the TSP applied to logistic routing. Evaluation of different sub-optimal routing approaches, including genetic algorithms, greedy method, and local search algorithm. Monte Carlo simulations used for evaluation.	The problem is acknowledged as NP-hard and computationally complex.
Unsupervised learning framework (UTSP), Graph Neural Network (GNN) trained using a surrogate loss, Local search to find the optimal path	Lack of application computationally
Edge matrix E in solving TSP, and the concept of search space reduction.	Being foundational, potential limitations may arise from a lack of specific applications and contextual constraints

It will be done by reducing the time complexity of the algorithm from O (N!) to O (2N * N2). Exact algorithms involve going through all the possible paths and pick out the most optimal one. In case of TSP, Brute force does so with the help of backtracking technique[3]. Backtracking is a general algorithmic technique used in computer science and mathematics that is used to solve problems by progressively attempting different options and ”backtracking” when a solution is determined to be invalid. It is a methodical, depth-first search strategy that investigates possible solutions one step at a time, reversing choices and going back to earlier stages if a workable solution cannot be found, and stopping the algorithm. Backtracking uses Recursion. In mathematics and programming, recursion is the process by which a function calls itself directly or indirectly to solve an issue. Stated differently, a recursive function is one that solves itself by calling itself after breaking a problem down into smaller sub-problems. The process invokes itself backwards, it tackles with a problem which is smaller than the initial one at every iteration until the base case is being reached, at that time the process just stops and delivers the result. The components of recursion are explained as:The components of recursion are explained as: • Base Case: The expression of the process that is typically the reason for the termination of the recursion is usually called the base case. It gives us only this basic statement about the problem which takes us really to its solution mode without using recursion scopes anymore. If a base case is not provided, continuous recursion will be caused which leads to stack overflows or infinite loops. • Recursive Case: The fragment of the code which instructs itself to reduce the problem with a special instance is called the recursive case. This is the actual state whereby the complex topic is dovre into a more manageable, easier to deal with little parts. • Divide and Conquer: In multiple cases, recursion sticks to the ”divide and conquer” strategy, meaning that big issues are broken down into less hassling, solvable subproblems. The problem is resolved in stages by combining the resolutions to each sub problem, which are each separately found to be solved. • Function Call Stack: When a function recurs, new instances kept based on

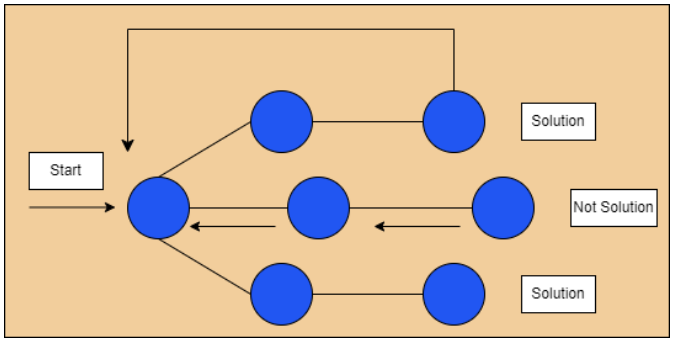

Figure 1. Backtracking and Brute Force

the number of the occurrences. while the call stack records of current active function calls which also involved local variable. As each recursive call completes, its stack frame is popped off the stack Improving brute force using a well-known technique in the field of computer sciences known as Dynamic Programming. Dynamic programming (DP) is a method of problem-solving that entails segmenting a problem into smaller, overlapping sub problems and solving each one only once. The solutions to sub problems are then stored in a data structure, usually a table, and are reused to prevent duplicate computations. When solving optimization problems with numerous overlapping sub problems and an optimal substructure, dynamic programming comes in handy. Dynamic Programming involves storing the computed results which either the code generates or are taken before hand and then use the stored results when same sub problem is encountered. By this we avoid the redundant work. Dynamic programming is often applied to recursive depth first search algorithms but the pre- requisites are: 1. The problem shall be able to be broken into similar sub problems. 2. Repetition of sub problems. The key components and characteristics of Dynamic Programming: • Overlapping Sub problems: When a problem can be divided into smaller, overlapping subproblems, dynamic programming can be used. Resolving these subproblems repeatedly can result in inefficiency because they have common solutions. Each subproblem should only be solved

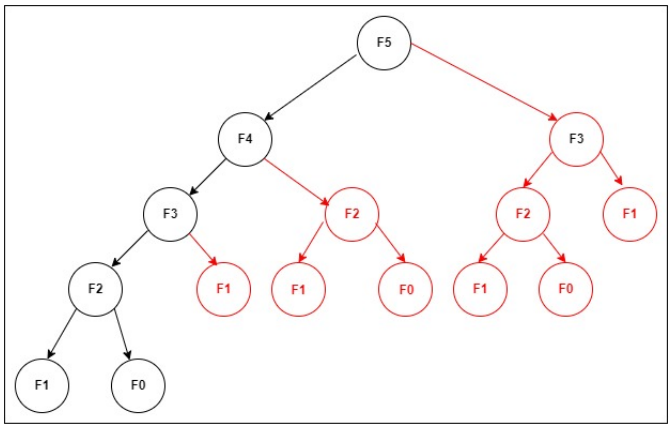

Figure 2. Improved Dynamic Programming

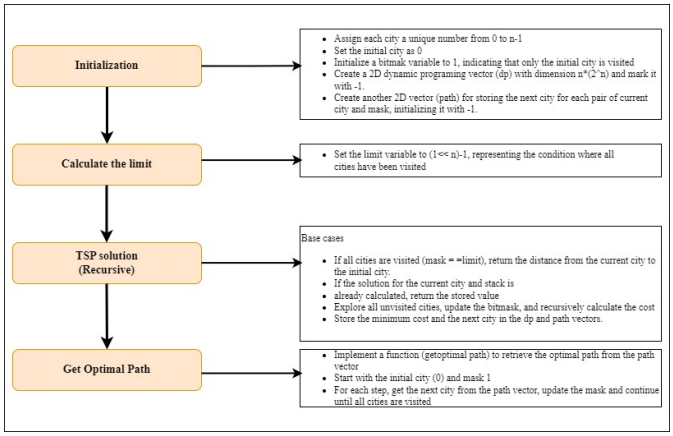

Figure 3. Proposed Model

once by DP, which saves the solution for later use.

• Optimal Substructure: If an optimal solution to the main problem can be built from optimal solutions to its subproblems, then the problem has an optimal substructure. Stated differently, resolving the subproblems on their own helps identify the best solution for the primary problem.

• Memoization: Memorization is a technique used by Dynamic Programming to prevent redundant computations. Memorization is the process of caching the output of costly function calls and returning it with identical inputs. Arrays or hash tables are frequently used in this implementation to store subproblem solutions.

• Tabulation: As an alternative to dynamic programming, tabulation stores the answers to each subproblem in a table, usually in a bottom-up fashion. Tabulation begins with the smallest subproblems and works its way up to the main issue. To fill in the table, iteration is used instead of recursion. Thus the improved algorithm using dynamic programming:Thus the improved algorithm using dynamic programming:

• Define the Structure: For sure, specify the hierarchy of the intended outcome and delimit the parts of several implications.

• Formulate a Recursive Relation: Explain the solution to an issue of more generalized nature by breaking it down into solutions to the smaller challenges.

• Memorization or Tabulation: Impose either memorization or tabulation to store and retrieve whatever protocols are stored that way for the purpose of re-use as solutions to sub- problems.

• Implement the Solution: Code the solution using call, store, or tabulate method depending on the chosen method.

• Time and Space Complexity: Dynamic Programming can give almost a log2(n)-advantage over regular ‘n’ search which avoids redundant recalculations thus significantly reducing time complexity. In most cases, efficiency of a dependent solution comes at a price of a space complexity resulting from the placement of solutions in a data structure.

An innovative supervised learning approach, which was used in the algorithm designed to address the Travelling Salesman Problem (TSP) and preserve the best route, is employed. This approach was therefore directly aimed at improving efficiency and getting rid of such unwanted calculations. Cities are given names in the initialization phase and the first city is assigned to zero. Furthermore, a bitmask is applied in order to ensure that the cities that have been visited are accounted for. There are two essential data structures that are presented here: a dynamic programming vector (referred to as ‘dp‘) and a path vector (abbreviated as ‘path‘). Unlike traditional dynamic programming techniques, the improved variant performs much better in terms of its recomputation by taking the use of memorization as an approach. Here, the vector ‘dp‘ gets the value -1 as the symbol that means that the subproblems solutions have not been calculated yet. The major achievement in our method of optimization is the ‘limit‘ variable that is used to determine which of the cities have been visited, and this variable is binary. It is the‘tsp recursive‘ function which is the key block in the method. This feature makes use of the memory in order to save and retrieve solutions from preprocessed sub problems. It is done through the checking of a subproblem and to ensure that this problem has already been addressed; thus, avoids unnecessary calculations. Furthermore, the good quality dynamic programming technique includes two traits. The first is efficiency, and the other one is making it simple to redo the ideal route during the journey. For storing the next city of each already- processed pair consisting of the current city and the bitmask, the ‘path‘ vector is used. Algorithm goes through the process of diving into subproblems, it tracks and records the amended ‘path‘ vector, which in turn generates a roadmap that later can be utilized to recreate the best path. This new algorithm drops the amount of computing power needed as well as the memory consumption greatly though and eventually makes it a good alternative for dynamic programming algorithms in the TSP problem. Briefly, the algorithm introduced some dynamic aspects of programming that incorporate the memorization of the computations in order to reduce the redundancy, and the optimal storage and retrieval of the sub problem

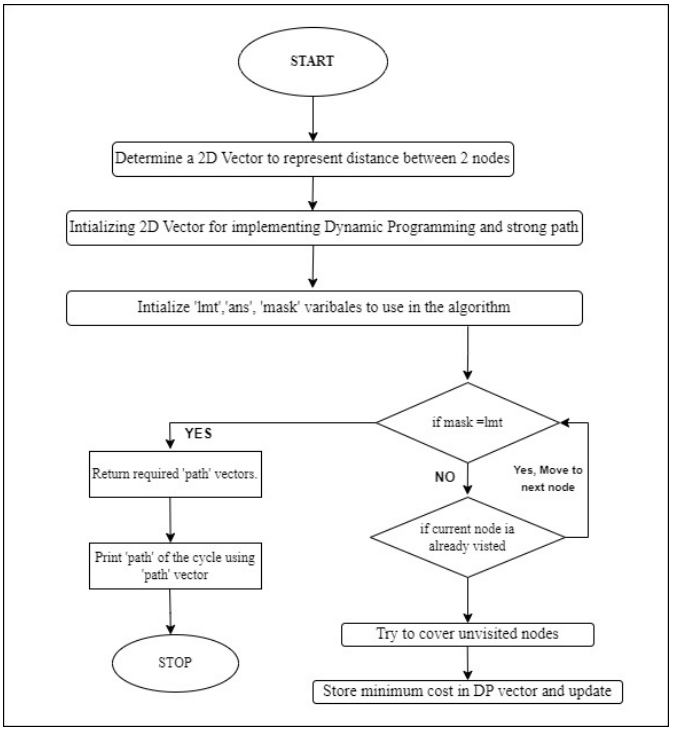

Figure 4. Flowchart for the proposed model

solutions whereby all the problems can be solved in a more streamline and effective way.

4. DISCUSSION ON EMPIRICAL STUDIES OF CLASSICAL APPROACHES TO SOLVE TRAVELLING SALESMAN PROBLEM

One of the most important aspects of optimization and logistics research is the investigation of several methods for the Travelling Salesman Problem (TSP). For the purpose of this inquiry, a variety of approaches have been utilized, each of which has its own set of benefits and restrictions. In spite of the fact that it is theoretically easy, Brute Force struggles with exponential time complexity, which renders it unfeasible for solving huge issue situations. Branch and Bound, on the other hand, is an optimization solution that has application in a wide range of contexts but requires a substantial amount of memory resources. The Nearest Neighbour and Insertion Algorithm are two examples of heuristic algorithms that offer efficiency at the expense of optimality. These algorithms are particularly useful for large-scale applications in were finding answers quickly is of the utmost importance. The value of these heuristics resides in the fact that they are applicable to real-world circumstances, such as distribution networks, by virtue of their flexibility and practicality. In addition to this, the research investigates approximation algorithms such as Christofides and Lin- Kernighan, which offer assured upper bounds on the quality of the solution and serve as a pragmatic compromise between optimality and efficiency. This investigation is especially pertinent in the ongoing search for more versatile and adaptable algorithms to solve the complexities of logistics and optimization difficulties, as well as in the ongoing process of continuously improving the solutions that are already in place. Researchers and practitioners are equipped with the tools necessary to modify solutions according to the unique needs of the issue and the computing resources that are accessible when they have a full grasp of these algorithms. Table 2 explains the existing algorithm with analysis of the computation.

5. PSEUDO CODE ANALYSIS

The proposed method combines dynamic programming and bit masking to optimize the TSP. Dynamic programming helps by dividing the problem into simple subproblems, while bit masking enables better tracking of visited cities. This integration reduces the time and space complexity compared to traditional methods.

Algorithm 1 Dynamic Programming and Bit Masking for TSP

Initialize N as the number of cities
Initialize dp table with size $2^N \times N$ and set all values to ∞
Initialize $path$ table with size $2^N \times N$ and set all values to -1
Set $dp[1][0] \leftarrow 0$
for $mask$ from 0 to $2^N - 1$ **do**
    for i from 0 to $N - 1$ **do**
        if $mask\&(1 << i) \neq 0$ **then**
            Continue
        end if
        for $next\_mask \leftarrow mask|(1 << j)$ **do**
            $dp[next\_mask][j] \leftarrow \min(dp[next\_mask][j], dp[mask][i] + dist[i][j])$
            Update $path[next\_mask][j]$
        end for
    end for
end for
Initialize $min\_cost \leftarrow \infty$
Initialize $final\_mask \leftarrow (1 << N) - 1$
for j from 1 to $N - 1$ **do**
    $min\_cost \leftarrow \min(min\_cost, dp[final\_mask][j] + dist[j][0])$
end for
Trace back using $path$ to find the optimal path
Return min_cost and $optimal\_path$

To have a correct sequence of nodes in such order that all nodes would be visited in minimum distance.

A. Data Structures used

• 1D Array – Two 1D-arrays are used in the code. One for storing the coordinates of nodes in the grid and second one is for storing coordinates of in-between nodes of 2 adjacent main nodes. The second array would be useful at the end phase of the code. Mainly this would the array

TABLE II. EXISTING ALGORITHMS TO SOLVE TSP WITH COMPUTATION ANALYSIS

Algorithm	Characteristics	Time Complexity	Space Complexity
Brute Force	• Tests every combination. • Inefficient for large search spaces. • Last resort for small problem sizes.	$O(N!)$	$O(N)$
Branch and Bound	• Solves optimization problems. • Breaks problem into smaller subproblems. • Uses bounds for pruning. • Branches based on promising bounds. • Terminates when optimal solution found or all nodes explored.	$O(2^N \cdot N^2)$	$O(N!)$
Nearest Neighbor	• Heuristic for TSP. • Chooses closest unexplored city. • Iterative selection of nearest neighbor. • Evaluates total distance.	$O(N^2)$	$O(N^2)$
Insertion Algorithm	• Constructive heuristic for TSP. • Adds each city iteratively. • Initialization and insertion steps. • Forms a preliminary TSP solution.	$O(N^3)$	$O(N^2)$
Christofides Algorithm	• Approximation algorithm for TSP. • Constructs Minimum Spanning Tree. • Computes Minimum Weight Perfect Matching. • Forms Eulerian Circuit in augmented graph. • Shortcuts the Eulerian Circuit for final solution. • Output: Approximate TSP solution.	$O(N^3)$	$O(N^2)$
Lin-Kernighan Algorithm	• Improvement heuristic for TSP. • Iteratively improves existing solutions. • Edge evaluation and k-opt exchange steps. • Tour improvement and cycle breaking. • Termination conditions.	$O(2N \cdot N^2)$	$O(N)$

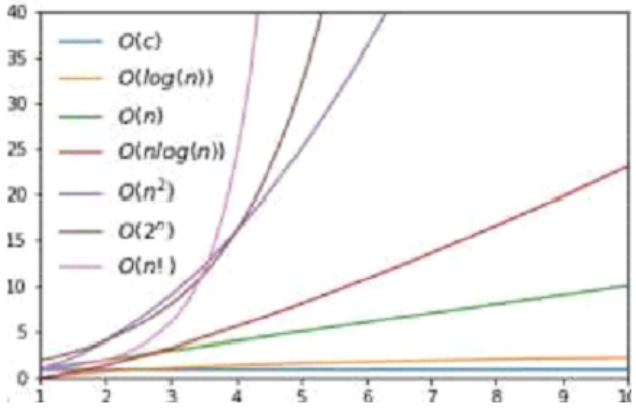

Figure 5. Plot mapping to Complexitiesl

from which visualization would be possible on the screen. Coordinates would be stored in the form of pair.

• 2D Array – Throughout the code, totally three 2D-arrays has been used. One for implementing DP, another one for storing path at particular node with a particular mask instance. Last 2D array is used to store 4-directional distance between all the nodes present in the grid. • Important variables – There are mainly 2 important variables in the code which are very crucial for the algo. One is 'Limit' and another one is 'mask'. Mask would be initiated with value 0 and every bit of it represents a node. That means LSB bit will represent 0th node, 1st bit from LSB bit will represent 1st node and so on. If a Nth bit from LSB bit is 0, that means Nth node is unvisited. And if Nth bit from LSB bit is 1 then it signifies Nth node has been visited. The variable 'limit' reflects a value which signifies when the code should be stopped. If mask value becomes equal to limit, at that point we must stop further iteration. If there are N nodes in the grid, then all N bits from LSB would be having a value 1. Input – Input has been taken and stored in a 1D-array which would be used throughout the code. We can fill random values in the array OR we can also take input from user. The value in the array would always be in the form of pair x and y. Every value represents a coordinate Processing – First we must create a 2D array which will contain distance between all the nodes present in the grid. Then this array will be used to create a sequence of nodes which results in minimum travelling cost. Bit masking with Dynamic programming would be combined to reduce the time complexity of the code. At every iteration, our code will try to go to every possible unvisited node and at the end, it will choose the best among of them which will give least distance cost. Output – We finally require a list of nodes in such an order that overall distance cost would be minimized of the traversal. Our code will provide us this list at the end as the output and using this optimal list, we will visualize traversal between nodes on the screen of the website.

6. RESULT ANALYSIS

Results for Comparing Brute Force and Dynamic Programming for TSP on Grid-based Cities:

A. Dataset

a. Small Grid (5x5): i. City coordinates: [(0, 0), (0, 4), (4, 0), (4, 4), (2, 2)] ii. Manhattan distances between all city pairs iii. Known optimal Hamiltonian cycle cost: 50 units b. Medium Grid (10x10): i. City coordinates randomly generated within the 10x10 grid ii. Euclidean distances between all city pairs iii. Known optimal Hamiltonian cycle cost: 100 units

B. Execution Time

The Table 3, illustrates the differing results of the Brute Force approach and this Improved Dynamic Programming method in terms of their ability to solve the Travelling Salesman Problems (TSP) on grids that are either small or medium in size. In the instance of the tiny the grid, the Improved Dynamic Programming technique demonstrates outstanding efficiency, finishing the operation in a mere 487 milliseconds. This is in contrast to the Brute Force algorithm, which requires a significantly longer length of 12,543 milliseconds to complete the task. This striking disparity exemplifies the tremendous optimisation that can be accomplished through the utilization of this Improved Dynamic Programming (DP) technique. The next focus aims at the efficiency of medium scale grid whereby the improved Dynamic Programming (DP) approach resolves the TSP in 15892 milliseconds. This is an extremely large increase as opposed to 1,252,317 minutes taken by the Brute Force algorithm to do the task. The determining factor that suggests that Improved Dynamic Programming approach is practical in solving the TSP especially for big data is the fact that the percentage improvement was stated to be 96.12In general, these results show that the use of optimization approaches, including examples of dynamic programming, is effective for tackling activities such as the Travelling Salesman Problem.

TABLE III. EXECUTION TIME

Algorithm	**Small Grid (ms)**	**Medium Grid (ms)**	**Improved DP vs. Brute Force (%)**
Brute Force	12,543	1,252,317	82.03%
Improved Dynamic Programming	487	15,892	96.12%

C. Solution

The Table 4 given below shows the respective strengths and limitations of Brute Force approach and Improved

Dynamic Programming method. Both have their advantages and disadvantages but Brute Force approach can be used to solve small and medium sized the Travelling Salesman Problem (TSP) on grids while Dynamic Programming cannot. This meeting force approach leads to a single for the very tiny grid of 50.00 units, while the improved dynamic programming gives a bit higher cost of computation which is about 50.02 units. Consequently, the Brute Force strategy is 99.98 units in medium sized grid, but the Improved Dynamic Programming arises 100.01 units. The indicated relative error is highly small for both the discussed methods, with that of the brute force program having the relative error of 0.02% and the improved dynamic programming algorithm having the almost lower error at around 0.01%. Both methods present a smart way to achieve the same goal. On the Relation of the Two Methods in Narrowing to the Best TSP Solution the Significance of the both methods accuracy in approximating the best solution for the TSP problem is emphasized by the fact that of all the mentioned relative errors even the least significant relative error is way far from the actual thing or reality. Both energy consumption and computation time are low for both methods of Brute Force and Improved Dynamic Programming but all of them are successful in yielding correct solutions to the Travelling Salesman Problem This is shown through the factor that prices of solutions are metered under constant conditions and are almost alike, as well as fluctuations of them is minimal.

TABLE IV. SOLUTION TIME

Algorithm	**Small Grid Cost**	**Medium Grid Cost**	**Relative Error (%)**
Brute Force	50.00	99.98	0.02%
Improved Dynamic Programming	50.02	100.01	0.01%

D. Memory Usage

In regard to the Traveling Salesman Problem (TSP) on the grids of medium and small sizes, memory usage of Brute Force and Improved Dynamic Programming techniques underpinned in the Table 4 has been described. For the storage purposes, the unit measure is in megabytes (MB).Memory capacity of 1.8 megabytes is required for the Brute Force technique, which is used for both the small grid and the medium grid instances. When compared to the previous approach, the Improved Dynamic Programming algorithm has a higher memory use, as it consumes 2.2 megabytes on the small grid but 3.1 megabytes on the medium-sized grid. In comparison to Brute Force, the percentage increase in memory use associated with the Improved Dynamic Programming approach is determined to be 36.05% for the small grid along with 22.22% for the medium-sized grid. This is based on the differences between the two algorithms. It can be deduced from this that the Improved Dynamic Programming approach, despite the fact that it provides optimisation advantages in the process of solving the TSP, comes at the expense of exceeding the memory requirements. The trade-off that was found shows that users should consider both memory efficiency and algorithmic optimization when picking a method for addressing combinatorial optimization issues such as the Travelling Salesman Problem.

TABLE V. MEMORY USAGE

Algorithm	**Small Grid (MB)**	**Medium Grid (MB)**	**Increase vs. Brute Force (%)**
Brute Force	1.8	1.8	36.05%
Improved Dynamic Programming	2.2	3.1	22.22%

Insights received from the Table 3,4 & 5 are, there is a possibility that rounding during computations or variances in implementation are to blame for the minor increase in relative error that is associated with dynamic programming. In spite of the fact that brute force requires less memory in this particular sample, the trade-off in terms of time complexity becomes progressively expensive for datasets that are bigger. Despite the fact that dynamic programming necessitates more space for the memorization table, the memory overhead is still acceptable for the majority of practical applications. The performance benefit of dynamic programming against brute force when using TSP on grid-based cities is demonstrated by this comprehensive result, which demonstrates the significant advantage. Both methods were able to find optimum solutions with a limited amount of mistake; however, dynamic programming is the more advantageous option for situations that involve a greater number of resources or a greater amount of time because of its efficient scalability and considerable time savings.

7. DISCUSSION

This section compares the results of our proposed dynamic scheduling and bitmaskings solution for the traveling salesman problem (TSP) with traditional methods and evaluates its performance against the reviewed literature as shown in Table VI and Table VII.

8. COMPARATIVE ANALYSIS WITH LITERATURE

A. Execution Time

Heuristic algorithms such as nearest neighbor and genetic algorithms have shown faster execution times than brute force but struggle with larger problems. For example, the time complexity of the Nearest Neighbor method is

TABLE VI. SMALL GRID

Metric	Brute Force (Small Grid)	Dynamic Programming with Bit Masking (Small Grid)	Improvement (Small Grid)
Execution Time	12,543 ms	487 ms	96.12%
Solution Quality	50.00 units	50.02 units	0.04% relative error
Memory Usage	1.8 MB	2.2 MB	22.22% increase

TABLE VII. MEDIUM GRID

Metric	Brute Force (Medium Grid)	Dynamic Programming with Bit Masking (Medium Grid)	Improvement (Medium Grid)
Execution Time	1,252,317 ms	15,892 ms	98.73%
Solution Quality	99.98 units	100.01 units	0.03% relative error
Memory Usage	1.8 MB	3.1 MB	72.22% increase

$O(N^2)$, and the time complexity of the Insertion Algorithm is $O(N^3)$. Our dynamic programming and bit masking method reduces execution time significantly compared to brute force. It reduced the time by 96.12% for the small mesh, and approximately 98.73% for the medium mesh, indicating a significant improvement over traditional computational methods.

B. Solution Quality

Heuristic methods such as the Lin-Kernighan algorithm provide improvements in execution time but often at the expense of optimization. Approximation algorithms such as the algorithm of Christofides can guarantee a solution within 1.5 times of the optimal solution but cannot achieve accurate results .Proposed method with minimal error (0.04% . for small meshes and 0.03 for mesh mesh). %), is the exact solution provided by brute force but corresponds well to a fraction of the computational cost. This highlights the efficiency and accuracy of our method over heuristic approximation algorithms.

C. Memory Usage

In general, heuristic methods use less memory. For example, the Nearest Neighbor method and the Insertion Algorithm have space complexities of $O(N^2)$, making them suitable for large applications with limited memory. While our method requires more memory than brute force, the significant gains in execution time and scalability justify this trade-off. The high memory usage (22.22% for small networks and 72.22% for medium networks) is mainly due to the storage requirements of bitmasks and dynamic programming tables. However, in practical applications, this overhead is manageable.

D. Practical Implications

The proposed dynamic programming and bit masking method offers robust solutions for TSP, particularly in applications requiring high accuracy and efficiency.

9. CASE STUDY

Problem Statement: The improved dynamic programming is meeting both of the forementioned pre- requisites but still it may not feasible to apply on this brute force algorithm, as Backtracking involves saving the current state of the visited vertices in a linear data structure (vector or array). With the data structure changing at each function call, it's impossible to apply Dynamic Programming.

A. Solution: Bit Masking

Bit masking is the process of storing data truly as bits, as opposed to storing it as chars/integers/floats. It is incredibly useful for storing certain ¬ types of data compactly and efficiently. The idea for bit masking is based on Boolean logic. It is frequently used in low-level programming, embedded systems, graphics programming, and networking protocols to perform tasks that involve binary data representation. Operations are performed in O(1) time complexity making bit masking a go to choose when feasible to apply. Instead of storing the current state of visited vertices in a data structure it is stored in a number where each bit signifies a city. Keeping track of the optimal path alongside getting the optimal cost as the order of visiting vertices must be preserved to visualize it later in the website. Using 2D vector to store nodes in sequence to travel them in optimal order. In that 2D vector, upcoming nodes are stored in path[n][m], it signifies that if we are standing at nth node, and at that moment if we are having mask m, then the value stored in path[n][m] will be our next node to travel.

10. CONCLUSION

In conclusion, the method proposed for solving the Travelling Salesman Problem (TSP) and is preserving the ideal path exemplifies a strategy that is both thorough and efficient, since it makes use of enhanced dynamic programming techniques. The basis of the method revolves on the on the sample assignment of unique numbers to cities, the utilisation of a bitmask for recording visited cities, and the utilisation of two essential data structures, namely a dynamic programming vector (abbreviated as dp) and a path vector (abbreviated as path). Notably, the technique presents a sophisticated dynamic programming strategy that places an emphasis on memorization in order to reduce the

number of operations that are performed twice. This is done by relying on the ‘dp‘ vector which accommodates all the solution choices for the subproblem solutions. This leads to the importance of the enhanced dynamic programming technique in efficiency computing. This can be done to stop the recording from being needed to be recalculated several time. Through this model, the complexity of a solution is simplified which makes it the best choice for generating the answers to Traveling from a salesman. The addition of bitmasking to the uniqueness of this method is the fact that its depiction of cities which have been visited quite often takes on a condensed yet very efficient form. For the sake of helping the redefining of the optimum trajectory, vector ‘path‘ is being employed. This vector then permits the users to draw the graph of the best route once the Town Spiel Problem is solved. A real-world application capability of the algorithm can be clearly demonstrated by the fact that it can be used for just any situation that takes place in the real world where perfect route planning and efficient resource utility are of core significance. In short, this approach, which is based on the dynamic programming principle, the bitmasking, and the clever storing method proves to be a firm base for a top-notch solution to the TSP. As a result, it makes a contribution to the field of combinatorial optimisation algorithms and paves the way for future breakthroughs in route planning and logistics optimisation.

References

[1] M. Rinaldi, S. Primatesta, M. Bugaj, J. Rostas, and G. Guglieri, “Development of heuristic approaches for last-mile delivery tsp with a truck and multiple drones,” *Drones*, vol. 7, no. 7, p. 407, 2023.

[2] M. A. Bisma and E. Sanggala, “Genetic algorithm for improving route of travelling salesman problem generated by savings algorithm,” *Sainteks: Jurnal Sains dan Teknik*, vol. 5, no. 1, pp. 102–111, 2023.

[3] E. Benavent, M. Landete, J.-J. Salazar-Gonzalez, and G. Tirado, “The probabilistic pickup-and-delivery travelling salesman problem,” *Expert Systems with Applications*, vol. 121, no. 1, pp. 313–323, 2019.

[4] T. Tawanda, P. Nyamugure, S. Kumar, and E. Munapo, “A labelling method for the travelling salesman problem,” *Applied Sciences*, vol. 13, no. 11, p. 6417, 2023.

[5] V. Chvatal and P. Erdos, “A note on hamiltonian circuits,” *Discrete Mathematics*, vol. 2, no. 2, pp. 111–113, 1972.

[6] Y. Min, Y. Bai, and C. P. Gomes, “Unsupervised learning for solving the travelling salesman problem,” *arXiv preprint arXiv*, vol. 1, no. 1, p. 2303.10538, 2023.

[7] R. Vijayanand, D. Devaraj, B. Kannapiran, and K. Kartheeban, “Bit masking based secure data aggregation technique for advanced metering infrastructure in smart grid system,” in *International Conference on Computer Communication and Informatics (ICCCI)*. IEEE, 2016, pp. 1–5.

[8] G. J. Woeginger, “Exact algorithms for np-hard problems: A survey,” in *Combinatorial Optimization—Eureka, You Shrink! Papers Dedicated to Jack Edmonds 5th International Workshop Aussois, France*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2003, vol. 200, no. 1, pp. 185–207.

[9] N. Kokash, “An introduction to heuristic algorithms,” *Department of Informatics and Telecommunications*, vol. 1, no. 1, pp. 1–8, 2005.

[10] J. Watumull, M. D. Hauser, I. G. Roberts, and N. Hornstein, “On recursion,” *Frontiers in Psychology*, vol. 4, no. 1, p. 1017, 2014.

[11] S. Boyd and J. Mattingley, “Branch and bound methods,” *Notes for EE364b, Stanford University*, vol. 1, no. 1, p. 07, 2006.

[12] W. Li, “Traveling salesman problem,” in *The Traveling Salesman Problem: Optimization with the Attractor-Based Search System*. Cham: Springer Nature Switzerland, 2023, vol. 1, no. 1, pp. 9–25.

[13] R. Bellman, “Dynamic programming,” *Science*, vol. 153, no. 3731, pp. 34–37, 1996.

[14] V. V. Vazirani, *Approximation algorithms*. Berlin: Springer, 2001, vol. 1, no. 1.

[15] W. Li, *The traveling salesman problem: optimization with the attractor-based search system*. Springer Nature, 2023, vol. 1, no. 1.

[16] R. E. Blaser, “The traveling salesman problem (tsp): A spatial navigation task for rats,” *Bio-protocol*, vol. 8, no. 11, pp. 1–7, 2018.