Modeling Metaheuristic Algorithms to Optimal Pathfinding for Vehicles

AHMAD SHARIEH
Computer Science Department,
The University of Jordan,
Queen Rania Street,
JORDAN

Abstract: - Finding optimal path (pathfinding problem) in terrain for vehicles, robots, and network routes (roads, pipes for water or gas, and network cables) is very complex and costly. Exhausted, heuristic, and meta-heuristic algorithms can be utilized to solve pathfinding problems. In this paper, we proposed a framework that finds an optimal path based on the objectives of the specifications and requirements of the pathfinding problems, terrain characteristics, and a metaheuristic algorithm. In this framework, a pathfinding problem is represented in a graph and a metaheuristic algorithm is modeled with optimal objective function *F* to find the optimal path. Thus, we present an overview of the most common metaheuristic pathfinding algorithms with heuristic objective functions. Many objective functions are modeled to find the optimal path in terms of distance, time, cost, energy, … etc., or in terms of a combination of two or more of these terms. The *F* is evaluated to find an optimal path from a starting point to a target point, subjective to constraints such as obstacles, barriers, and other constraints to satisfy the characteristics of the terrain. In this framework, the problem locations and links in terrain are represented in graph vertices and edges, respectively. The graph is implemented in adjacent matrices and the paths as vectors. We overview these algorithms with examples of their applications in vehicle scenarios. The framework will help interested readers understand how pathfinding algorithms work and pick the best fit for a particular application.

Key-Words: - Framework, Heuristic, Graph, Metaheuristic Algorithms, Model, Objective Function, Optimal Pathfinding, Vehicles.

Received: April 7, 2024. Revised: October 29, 2024. Accepted: November 28, 2024. Published: December 31, 2024.

1 Introduction

Pathfinding is one of the most concerning problems in mobile robotics and vehicles, and it is an NP-hard problem, [1], [2].

The length, safety, and smoothness of a path are optimized simultaneously in multi-objective functions. The weighted multi-objective path planning methods and Pareto-based multi-objective path planning methods share the overarching aim of finding trade-off solutions and they differ in their optimization strategies, [1].

Metaheuristic algorithms provide a set of strategies for developing optimization problems including optimal path-finding problems. There are a variety of challenges and constraints to solving optimal path-finding problems in very large and complex search spaces. The challenges include the correct specifications of the environment characteristics, graph representation of the environment, objective function subjective to a set of constraints, and suitable metaheuristic algorithms or parallel algorithms to solve the problems. Thus, exploring an overview of metaheuristic algorithms and proposing a model will help solve pathfinding problems.

Our objective is to present the specifications of heuristic or metaheuristic algorithms for pathfinding problems in a framework, which will be explained in Section 3. The framework has the following components: the aim(s) of an algorithm to optimize the path in a given terrain, problem representation, key components of the selected algorithm, objective function to optimize the total cost of the path, and the algorithm steps. These steps include initialization, deploying exploration and/or exploitation and fitness functions to reduce the number of iteration and update positions, selecting the updated agent(s) to retain better positions, and producing the best path as an optimal or near-optimal path in the given graph.

To derive a mathematical model for finding an optimal path in an environment, start by specifying the locations or points (nodes) and the links between these points. In order to compute the cost value of

the optimal path, the points and the links will be mapped into a graph. A graph is a set of collective vertices and a set of edges denoted by G = (V, E), where the number of vertices is denoted by |V| = n and the number of edges denoted by |E| = m [3]. The set of points and set of links are mapped into the set of vertices and the set of edges, respectively. Adjacent vertices v ϵV are the set of all vertices that are connected to v by an edge, {x: (v, x) ϵ E}. A subgraph of a graph is a graph where all its vertices and edges are a subset of this graph. A path in a graph is a sequence of vertices connected by edges. A search space of locations and links is represented by a graph G = (V, E). A starting location is s and a target location is t, and the objective is to find the optimal path from s to t. A cost function C (x_i, x_j) that assigns a cost (or weight) to traversing from vertex x_i to vertex x_j, and a set of constraints C_o that specifies permissible paths. Each edge in the graph is assumed to have a weight that represents the cost from a vertex to its neighbor. We need to find a path P = {x_1, x_2, ..., x_n} from s to t that optimizes the objective function F(X), similar to function (1). The F(X) can optimize multiple criteria simultaneously, such as distance, energy, and time using a weighted sum of different objective functions F_1, F_2, F_3, ..., F_m as shown in function (2). The F(P) evaluates the optimal path. The F(P) can be computed by an exhausted or an approximated algorithm on an available suitable platform.

$$\text{Optimize } F(P) = \sum_{i=1}^{n-1} C(x_i, x_{i+1}) \qquad (1)$$

Subject to C_o.

$$\text{Optimize } F(X) = \sum_{j=1}^{m} wj\, Fj(X) \qquad (2)$$

Subject to constraints C_j in each optimized F_j and $\sum_{j=1}^{m} wj = 1$.

An optimal path can be through locations known before or to be constructed, [3]. Table 1 shows examples of previously specified locations and possible links in pathfinding applications and graph representations.

In a weighted graph and edge e ∈ E, the weight C(e) represents a cost or effort to traverse the e from its start vertex to its end vertex. The weight can be distance, energy, time, elevation, flow, resource, hiker, capacity, etc., or combinations of some of these with percentage share or weighted averages. In optimal pathfinding problems, certain cost functions or objective function F assign value or cost to potential solutions that meet criteria. This cost computed by F guides the algorithm in its search for the most efficient path from the source point to the target point.

Table 1. Examples of pathfinding applications and their graph representations

Application	Graph Representation
Shortest path problem	Terrain grid points →V Paths between the points → E
Transportation networks	Locations including airports, train stations, bus stations, intersections, or ports →V Routes including flights, railways, roads, roads or railways, or ships, respectively →E
Tele-communication networks	Devices or routers →V Communication links between the devices →E
Warehouses, suppliers, and retailers	Warehouses, suppliers, and retailers →V Transportation routes → E
Urban planning and GIS	Base camp, checkpoints, buildings, or landmarks →V Roads, paths, trails, and other links → E

There are some common objective functions and algorithms used for optimizing path-finding problems. Table 2 (Appendix) shows examples of applications with possible optimization functions, [4], [5]. For example, in road networks and robotic path planning, the F is to minimize the total distance. Some functions maximize the benefits. For example, in resource allocation, the F is to maximize access to resources. In applications such as emergency response routing and transportation planning, the cost can be computed in terms of time, resource utilization, and monetary variables. In some applications like complex routing problems, trade-offs between multiple factors such as time, distance, and elevation must be considered in computing the cost by using a weighted sum of several objective functions.

Our main contributions are:

1. Presenting a framework and its utilization to solve pathfinding problems.
2. Reviewing the pathfinding problems and their representations in a graph.
3. Presenting objective functions to optimize pathfinding problems.
4. Overviewing metaheuristic algorithms to solve pathfinding problems and their classifications.

In this section, we introduce our objectives, the problem and its representation in graphs, and some pathfinding applications with its optimization functions. Section 2 reviews optimization algorithms utilized in solving the pathfinding problems and their classifications. Section 3 explains how the framework can be followed to solve the problem.

Section 4 shows examples of how to implement a few metaheuristic algorithms. Section 5 concludes the manuscript.

2 Algorithms and Pathfinding Problems

In [6], they reported that the design of the optimal path planning algorithm needs to consider its theoretical efficiency, the difficulty of reducing the computer implementation, and the resource requirements of the computer hardware system. They also reported that at present, there are many mature algorithms to solve the shortest path problem. These algorithms include the Dijkstra algorithm, Bellman-Ford's algorithm, Floyd's algorithm, and heuristic search algorithms such as the A* algorithm. There are also many improved algorithms for vehicle navigation, such as K-shortest path algorithms, genetic algorithms, and neural network-based algorithms, [6].

According to the research background and the current status of domestic and international research, it can be seen that the shortest path problems are a hot research topic in GIS, computers, and other sciences, [6], [7]. In the literature [7], a shortest path algorithm was tested in a stochastic planar network by selecting the more representative shortest path algorithms from the available ones, and the results showed that no single algorithm can be adapted to all types of networks.

There are many common algorithms utilized for optimization pathfinding problems. These use brute force, heuristic, or metaheuristic techniques, [8], [9]. In this section, we briefly review these algorithms.

2.1 Brute Force and Algebraic Algorithms

In solving pathfinding problems, the brute force or the exhausted algorithms explore all possible options to find the optimal solution, and then select the best one based on specified criteria, [10], [11]. Some common brute force algorithms are depth-first search (DFS), breadth-first search (BFS), backtracking (BT) [12], branch and bound (BB) [11], dynamic programming (DP), Dijkstra algorithm (DA) [13], and Floyd-Warshall (FW) algorithm [14]. The DFS is useful where the solution is likely to be found far from the starting point [11]. The BFS is used to find the shortest path in unweighted graphs, [3]. The BB algorithm searches by trying different paths and eliminating those that don't lead to a solution, [15]. A DP uses a grid-based approach for pathfinding to solve problems like the shortest path, [2]. The DA is a greedy algorithm, and it is used to find the shortest path between vertices in a graph with non-negative weight edges and select the vertex with the smallest distance. It is used in road networks.

The advantages of brute force algorithms are: explore all possible solutions, guarantee to find the correct or optimal path and provided it exists, do not require prior knowledge or assumptions about the problem domain, and do not rely on specific properties of the problem, [16]. Their limitations are: inefficient for large or complex problems, their time complexity is typically exponential, impractical for large-scale or real-time pathfinding problems because they consume significant amounts of memory when storing all possible paths, and are unmanageable for very large grid size, [16].

In algebraic approaches, the graph's weights reflect the cost of traversing a terrain. Vehicle pathfinding in terrains can be formulated as a linear programming problem (LP), where the objective function is to minimize the cost. Approaches like the simplex method can be used to solve such problems using matrix operations to find the shortest path, [17]. In stochastic terrain environments, the Markov decision process (MDP) can help in moving to adjacent vertices, and dynamic programming approaches can model decision-making under uncertainty, [18]. When a vehicle navigates a terrain, it can use a weighted graph to represent a terrain type and slope obstacles, a tropical semi-ring to find a path with minimum cost, an LP model to handle vehicle energy, and an MDP to handle uncertain conditions. Algebraic approaches often lead to mathematically optimal solutions in travel time, distance, and energy consumption. These approaches provide deterministic results. They can be adapted to solve pathfinding problems such as weighted, directed, or multi-constraint problems. Algebraic techniques are powerful for theoretical and structured pathfinding problems, but they have drawbacks such as: 1) may not handle applications such as dynamic traffic conditions and obstacles in the world, 2) may become computationally expensive for large or dense graphs, 3) they phase difficulty in handling non-linear applications for vehicle routing, 4) and they are often static.

2.2 Heuristic Algorithms

Heuristic algorithms make guesses to speed up the search process and find a solution very close to the optimal one and do not guarantee finding the best solution, [19]. The heuristic techniques are designed to solve problems quickly when classic methods are slow to find an exact or approximate solution, or when classic methods fail to reach an exact solution

in a search space. It finds optimal or near-optimal solutions to the optimal pathfinding problems faster and more efficiently than exhaustive search methods.

There are heuristic algorithms commonly used for pathfinding across various applications, including robotics, gaming, and logistics. Examples of heuristic algorithms for pathfinding are A* (A-star) search, Hill Climbing (HC), Greedy Best-First Search (GBFS), Simulated Annealing (SA), Beam Search (BS), Iterative Deepening A* (IDA*), and Theta* algorithm, [5], [9], [10], [16], [19]. The A* search algorithm is a popular heuristic one, [20], [21], [22]. The DAH adds a heuristic function to prioritize vertices and improve its efficiency, [8]. The GBFS focuses on the estimated cost to the target vertex *t*, it is similar to A*, but f(*v*) = h(*v*). This is achieved by trading optimality, completeness, accuracy, and precision for speed, [20].

In their analysis [22], the authors conclude that:

a. The results of the Greedy algorithm analysis with several trials can find the shortest path in a fast time, but there are some cases where the optimal solution is not found or the final state is not found at all,
b. The results of the A* algorithm analysis with several trials can find the shortest path better than the Greedy algorithm, [23].
c. The results of Dijkstra algorithm analysis with several trials can find the shortest path better than the Greedy and A* algorithms. In this case, the Dijkstra algorithm can find a solution that tends to be better than the two and always finds the optimal solution.
d. The weakness of the greedy algorithm is that it tends to make choices that do not take into account the next event, while the weakness of the A* algorithm is that the graph must have complex data such as straight-line distance to node (final state), while the weakness of Dijkstra algorithm tend to be slow in finding solutions because they have to compare the cost of one path with the cost of another path, [24].

The authors in [26] reported that the IDA* has the following characteristics: it performs DFS using the same cost function as A*, its memory usage is lower than in A*, it concentrates on exploring the most promising nodes, and does not go to the same depth everywhere in the search tree, and unlike A* it does not utilize and ends up exploring the same nodes many times. Its advantages are [26]: if an optimum solution exists, it will be discovered (completeness), it only keeps one path in memory at a time (memory efficiency), may be employed with several heuristic functions (flexibility), and sometimes outperforms other search algorithms like uniform-cost search (UCS) or BFS. These algorithms utilize heuristic functions to guide their search to estimate the cost to reach the goal vertex from a given vertex. The choice of heuristic function impacts the efficiency and accuracy of the pathfinding process. A heuristic algorithm deploys a heuristic function that depends on the specifications of the problem to be solved. The heuristic function can be as in function (3). Heuristics functions include Manhattan distance (MD) where $h(v)=|x_{target} - x_{current}|+| y_{target} - y_{current} |$, for grid-based maps, Euclidean distance (ED) $h(v)= sqrt((x_{target} - x_{current})^2+ (y_{target} - y_{current})^2)$, for maps, and Octal distance (OD) $h(v)=max(|x_{target} - x_{current}|, | y_{target} - y_{current} |)$ for diagonal moves in grids [22]. The MD is used for grids where movement is restricted to four directions (up, down, left, right), ED is used where diagonal movement is allowed, and OD which combines MD and ED is used for diagonal moves in grids, [27].

In [28], they proposed to use A* for finding the shortest path on the rough area with possible obstacles in the movement path. Compared to Dijkstra, the A* using Manhattan distance evaluated 25.16% fewer nodes, was 31.4% faster, and had a clear advantage in memory and speed. When A* used the Euclidean distance, it expanded 21.8% fewer nodes than Dijkstra, and 5.3% faster.

There are algorithms such as Ant Colony Optimization (ACO) and Simulate Annealing (SA) for Pathfinding that use heuristic functions when they compute the optimal objective functions, [29]. The heuristic function is used by ACO equals to $1/d_{ij}$, for edge e(i,j), where d_{ij} is the distance between vertex i and vertex j, [30], [31]. Simulate Annealing (SA) for pathfinding involves a heuristic function that guides the acceptance of new solutions based on a cost function equals to $\exp((E- E')/T)$, where E and E′ are the energies of the current and new solutions, respectively, and T is the temperature, [11].

In A* search algorithm, the g(*v*) function evaluates the exact cost from the start vertex *s* to the vertex *v*, and h(*v*) function evaluates the heuristic estimated cost from *v* to the target vertex *t.* Other heuristic algorithms that use function (3) are: *Beam Search (BS)* which uses f(v) to rank the vertices to determine which ones to keep [33], *Dijkstra Algorithm with a Heuristic (DAH)* [25], *Iterative Deepening A* (IDA*)* [34], [35], *Jump Point Search (JPS)*, [36], [37], [38], and *Theta* (Theta-Star)*

Algorithm, [37], [38] [39], [40]. *Greedy Best-First Search (GBFS)* considers $h(v)$ similar to A*.

$$f(v) = g(v) + h(v) \quad (3)$$

Among the advantages of heuristic algorithms are reducing the number of paths explored in large complex pathfinding problems, often find a solution much faster than brute force methods by using heuristics to estimate the cost of reaching the goal, guarantee finding the optimal solution, scale better to larger problems to handle more complex pathfinding without a significant increase in computation time, and can be adapted to different types of problems including robotics, [1], [2], [8], [16]. The drawbacks of heuristic algorithms in pathfinding are their performance depends on the quality of the heuristic function and can lead to suboptimal paths or longer computation times, the complexity can limit the applicability of heuristic algorithms if they use unsuitable heuristic, the heuristic evaluation can add computational overhead, and some heuristic algorithms may not find a best possible solution, [20].

2.3 Metaheuristic Algorithms

Metaheuristics provide a set of strategies for developing heuristic optimization algorithms and they are utilized in optimal pathfinding problems. Metaheuristic is a higher-level technique designed to find, generate, tune, or select a heuristic that may provide a sufficiently good solution to an optimization problem and machine learning problem, [41]. Metaheuristics may make relatively few assumptions about the optimization problem being solved and so may be usable for a variety of problems, [42]. They are used when exact or other approximate methods are not available. The characteristics of metaheuristics include [12]: 1) Techniques that constitute meta-heuristic algorithms range from simple local search procedures to complex learning processes, 2) The basic concepts of metaheuristics permit an abstract level description, 3) They may incorporate mechanisms to avoid getting trapped in confined areas of the search space, 4) They are approximate and usually non-deterministic, and 5) Metaheuristics may make use of domain-specific knowledge in the form of heuristics that are controlled by the upper-level strategy, and 6) More advanced metaheuristics use search experience to guide the search.

Metaheuristic algorithms are not problem-specific, generally nature-inspired, and can be used for many different problems, [35]. The goal is to efficiently explore the search space in order to find optimal or (near-) optimal solutions. Examples of metaheuristic algorithms are Ant Colony Optimization (ACO), Artificial Bee Colony (ABC), Particle Swarm Optimization (PSO), and Genetic Algorithm (GA), [34]. There are several classifications of metaheuristic algorithms, and one is reported in [42]: Nature-Swarm intelligence algorithms are a flexible and solid method that is developed inspired by animals' swarm behaviors. ACO and PSO are two of the most used swarm intelligence algorithms. ACO algorithm is mostly used in solutions of combinational optimization problems and the PSO algorithm is mostly used in continuous optimization algorithms. For example, vehicle routing can be solved using ACO, and problems that need function optimization in many different engineering fields can be solved using PSO. Swarm can be defined as discrete individuals influencing each other. Individuals can be humans or ants. In *swarms-based*, *N* individuals work together to achieve a purpose [8]. The PSO, ACO, Grey Wolf Optimization (GWO), Chickens Swarm Optimization (CSO), and Cat Swarm Optimization Algorithms (CSOA) are swarm-based, [43].

In [42], the authors reported the classification of algorithms as: *Complete (or exact*) algorithms and *approximate* methods. Approximate algorithms can be divided into *local-search* algorithms and *constructive* algorithms. *Memory-less* algorithms (keeps track of recently visited solutions (moves)) and *memory-usage* algorithms (there is a huge storage of information about the entire search process). Metaheuristics can be classified in different ways depending on the specific point of view of interest [42]:

1. *Nature-inspired algorithms* such as Genetic Algorithms (GAs), Ant Colony Algorithms, Cuckoo Search Algorithm (CSA), Beatle Antennae Search (BAS), Teaching Learning Based Optimization (TLBO), Moth Flame Optimization (MFO), … etc; and *non-nature inspired ones* (such as Tabu Search(TS) and Iterated Local Search (ILS) … etc.
2. *Population-based* such as GAs, Single point search methods (such as Tabu Search (TS), Iterated Local Search (ILS), and Simulated Annealing (SA)). These metaheuristics compute simultaneously a set of points at each time step of the search process, describing the evolution of an entire population in the search domain.
3. *Trajectory methods* because they work on a single solution at each time step describing a curve (trajectory) in the search space during the progress of the search. These include SA, TS, Guided Learning Search (GLS), etc.

Metaheuristics also can be classified according to the way they make use of the objective function: *Dynamic objective function* if during the search, the objective function is altered based on information collected during the search process or *Static objective function* if techniques keep the objective function as it is given by the problem, [42].

In [41], they classified the metaheuristic algorithms into *Evolutionary*-based, *Nature-based*, and *Trajectory*-based. The nature-based algorithms are Swarm-based such as PSO, Bio-inspired such as GWO, Physics/Chemistry such as Chemical Reaction Optimization (CRO), Human-based such as Cultural Algorithm (CA), and Plant-based such as Invasive Weed Optimization (IWO).

There are a variety of metaheuristic algorithms, [20]. Some were proposed to improve local search heuristics as in the SA and TS algorithms. Some are based on *ant colony optimization*, *evolutionary computation*, and *particle swarm optimization* algorithms. Most metaheuristics algorithms intend to employ a fitness function to evaluate the candidate solutions. The SA is used in very large and complex spaces, and its heuristic function helps to escape from local optima to the global optimum of a given function based on a probability technique, [19]. Tomar, Bansal, and Singh [10] classified the metaheuristic algorithms into:

1. *Evolution-Based* Algorithms (EBA), where the optimization techniques are inspired by natural evolution. These include GA, Differential Evolution (DE) algorithms, Flower Pollination Algorithm (FPA), etc.
2. *Particle Swarm Algorithms* (PSA) are modeled after social animals and insect's behaviors in group. Examples include BAT which was inspired by bat echolocation, Cuckoo Search (CS) algorithm which was inspired by the breeding behavior of cuckoo birds, Grasshopper Optimization Algorithm (GOA), Firefly Algorithm (FA), Dragonfly Algorithm (DA), Ant Lion Optimizer (ALO), Grey Wolf Optimizer(GWO), Flower Pollination Algorithm (FPA), and Whale Optimization Algorithm (WOA).
3. *Physics-based algorithms (PBA)* are motivated by the physical principles of nature and replicate physical rules during optimization [44]. These include SA, the Lightning Search Algorithm (LSA) which is influenced by the natural factors of lightning strikes, Gravitational Search Algorithm (GSA) is influenced by gravity and motion principles, and Electromagnetic Field Optimization (EFO). There are other PBAs including the multi-verse optimizer and the sine–cosine algorithm.
4. *Human-Related* Algorithms are driven by social interaction or behavioral patterns in people. Examples are the Brainstorm Optimization algorithm (BSO), Teaching learning optimization (TBLO), and the Gaining Sharing Knowledge-Based Algorithm (GSKA).
5. *Hybrid Metaheuristic* Algorithms: These are developed from other metaheuristic algorithms to avoid local optimization trapping, upgrade efficiency, and effectively explore the search space for better solutions.

Since sometimes it is not possible to classify an algorithm to only one of the classes [42] and some hybrid algorithms fit both classes at the same time, the Euler diagram of the different classifications of metaheuristics is used as in Fig.1, [43].

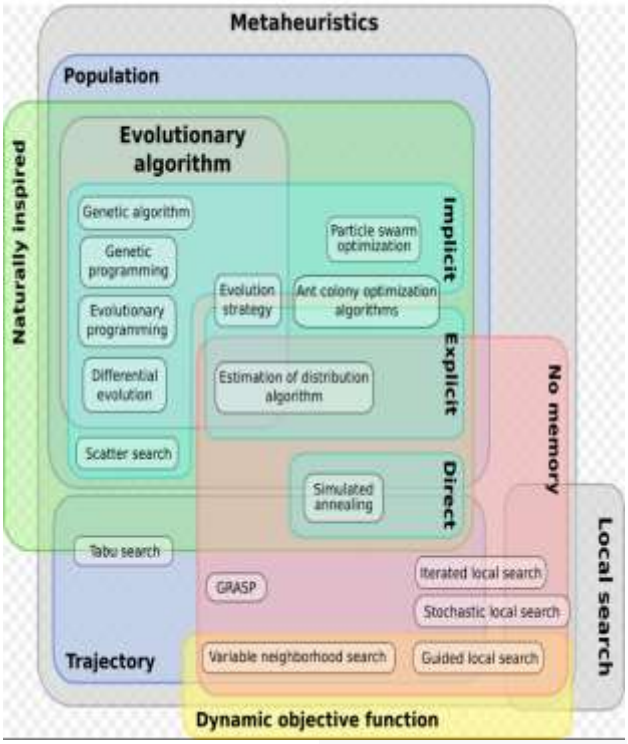

Fig. 1: The Euler diagram of different classifications of metaheuristics, [43]

The advantages of metaheuristic algorithms include: 1. *Global Search Capability:* MA are designed to explore the search space broadly and avoid getting stuck in local optima and finding a near-optimal or optimal solution, 2. *Highly Flexible and Adaptable*: to various problem domains of pathfinding problems with complex, non-linear, or dynamic environments, 3. *Handling of Complex Problems:* MA is effective in scenarios with multiple objectives, constraints, or conflicting goals,

4. Some *Balance Exploration* (searching new areas of the search space) and *Exploitation* (intensifying the search around promising areas), 5. Can *be Scaled* to larger problems by *Adjusting* parameters like population size or iteration count; which makes them suitable for large-scale pathfinding problems, and 6. Their *Stochastic* components help to diversify the search and explore different areas of the search space.

However, the MAs have drawbacks including: 1. Some MAs do not guarantee finding the optimal solution, 2. Finding the right parameter settings can be difficult and may require extensive experimentation, 3. Can be computationally expensive and slower than some heuristic methods in real-time applications, 4. Their implementations may require a deep understanding of the algorithm's mechanics, parameters tuning, and problem-specific adaptations, and 5. MA can lead to inefficiencies or failure to find a good solution, and different runs of the algorithm might yield different results because of the stochastic nature of metaheuristics even with the same initial conditions.

3 Methodology

In this section, the proposed framework is explained to find solutions to pathfinding problems using metaheuristic algorithms. Fig. 2 shows the components of the framework. Developing solutions for pathfinding problems for robots and/or vehicles can be guided by this framework. Because of space limitations in this paper, three examples will be presented showing the steps following this framework. The examples are A* algorithm- example of the heuristic algorithm, Flower Pollination Algorithm (FPA)- an example of a metaheuristic algorithm with a single objective function, and Moth Flame Optimization (MFO)- an example of a metaheuristic algorithm with an objective function with multiple factors. Other samples of metaheuristic algorithms are listed with their objective functions as in Table 2 (Appendix).

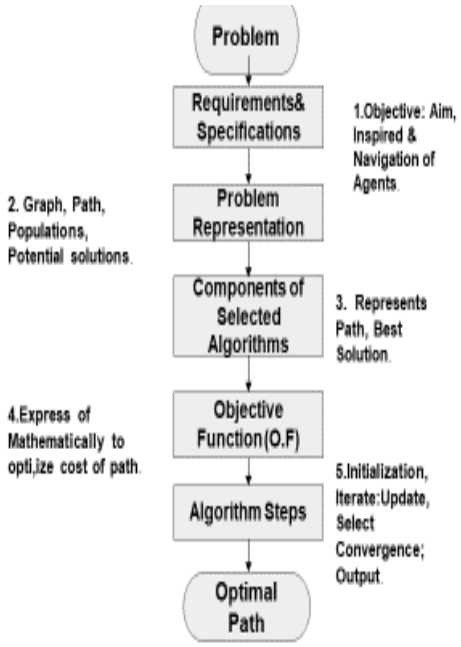

Fig. 2: Framework for developing solutions for pathfinding problems

Our objective is to present the specifications of heuristic or metaheuristic algorithms for pathfinding problems in a framework with the following components: 1. *Objective:* specify what is/are the aim(s) of an algorithm to optimize the path in a given terrain. It includes statements indicating how the algorithm is inspired and mimics the navigation behavior of the agents. 2. *Problem Representation:* specify how the pathfinding problem is represented as a graph, how the path from the start vertex to the target vertex is represented, and what are the populations and their potential solutions to choose the best solution from. 3. *Key Components of the selected algorithm:* specify what represents the candidate solutions (paths), and the best solution, and simulate the behavior. 4. *Objective Function*: express the objective function mathematically in the context of pathfinding to optimize (maximize or minimize) the total cost of the path. *Algorithm Steps: Initialization:* initialize a population of the path(s) randomly or using heuristics, evaluate the initial path(s) using a fitness function(s), and sort the solution(s) based on the fitness function**.** *Update:* deploy exploration and/or exploitation and fitness function to reduce the number of iterations, and update positions. S*election:* Compare the fitness of the updated agents with their previous fitness and retain better positions. *Convergence:* The algorithm iterates through updating and selection steps until a stopping criterion is met. *Output:* consider the founded best path as an optimal or near-optimal path in the given graph.

Example 1: Heuristic-based- A* Algorithm

1. *Objective:* The goal is to find the shortest path between a start node (represented as vertex s in the graph) and a target node (represented as vertex t in a graph) using the A* algorithm, based on function (3).
2. *Problem Representation:* The problem is represented in a graph G (V, E)**,** where the set of locations or positions is represented in the set of vertices V and the set of links between locations are represented in E the set of edges. The path begins at vertex s and ends at the target location at vertex t. A path is represented as a sequence of vertices $X=\{x_s, x_2,\ldots, x_t\}$. The moving from vertex x_i to vertex x_{i+1} is represented by the cost function $g(x_i, x_{i+1})$.
3. *Key Components of A*: Open List:* A priority queue that stores discovered vertices but not yet evaluated. The vertices in this list are prioritized based on their estimated total cost f(x)=g(x)+h(x). *Closed List:* A set that keeps track of vertices has been evaluated to avoid reprocessing. *Cost Function* g(x): Represents the cost from s to a current vertex x. *Heuristic Function h(x):* Represents the estimated cost from the current vertex x to t.
4. *Objective Function for A* Algorithm* in pathfinding: is mathematically expressed as in formula (4).

$$\text{Minimize } \boldsymbol{F(X)} = \sum_{i=1}^{n-1}(g(\boldsymbol{x_i}) + h(\boldsymbol{x_i})) \qquad (4)$$

5. *Algorithm Steps:*
   - *5.1 Initialization:* Place the start vertex *s* in the open list with g(s)= 0 and h(s) = calculated value based on the heuristic function. Set the initial vertex as the current vertex.
   - *5.2 Select* the vertex x from the open list with the lowest f(x)=g(x)+h(x). If x is the target vertex t, the algorithm terminates as the path is found. Move x from the open list to the closed list.
   - *5.3 Update:* For each neighbor y of x: If y is in the closed list, ignore it. If y is not in the open list, calculate g(y)=g(x)+g(x, y) and estimate h(y), add y to the open list with f(y)=g(y)+h(y).
     If y is already in the open list, check if the new path to y is better (lower g(y)), and update g(y) and the corresponding f(y).
   - *5.4 Convergence:* iterate through the open list until it either finds the goal vertex t or the open list becomes empty.
   - *5.5 Output:* The best path X from the start vertex *s* to the goal vertex *t* with the optimal cost (shortest path in this example).

Example 2: *Evolution-Based*- Flower Plant (FPA) Algorithm

1. *Objective:* The goal is to find the shortest or least-cost path while traversing from a start vertex s to a target vertex t in a given network or graph using the metaheuristic evolutionary FPA algorithm.
2. *Problem Representation***:** The positions or locations and links of the network or robotics are represented as a graph G (V, E), where V is the set of vertices and E is the set of edges between vertices. The Path P represented as $X=\{x_s, x_2,\ldots,x_t\}$ is a sequence of vertices starting from s to t. Distance/cost function $d(x_i,x_{i+1})$ is associated with traveling between two consecutive vertices.
3. *Key Components of FPA: 1. Global Pollination (Exploration)* generates new solutions by combining the current best solution with random pollination from the global population; and mimics the process of cross-pollination by insects. 2. *Local Pollination (Exploitation):* refine solutions by perturbing the current solutions with nearby solutions; and mimics the process of self-pollination. 3**.** *Switch a probability* p value that decides whether global or local pollination will be applied to generate new solutions.
4. *Objective Function:* It is to minimize the total path cost or distance D (single objective) and can be mathematically expressed as: Minimize the total travel distance *F(X),* as in formula (5).

$$\text{Minimize } F(X) = \sum_{i=1}^{n-1} \boldsymbol{D}(\boldsymbol{x_i}, \boldsymbol{x_{i+1}}) \qquad (5)$$

where $X = \{x_s, x_2, \ldots, x_t\}$ is a candidate solution representing a path from *s* to *t.*

5. *Steps of FPA:*
   - *5.1 Initialization:* Initialize a population of flowers (solutions), each representing a path through the graph, and assign random positions to each flower.
   - *5.2 Evaluation:* Compute the *F(X)* for each solution in the population.
   - *5.3 Iteration:* For each solution in the population, *select* and *update* as follows; With probability *p*, perform global pollination: $X_{new}=X_i+L(X_i-X_{current})$, where $X_{current}$ is the current best solution, and $L(\cdot)$ is a Lévy flight-based step size. With probability $1-p$, perform local pollination: $X_{new} = X_i + \epsilon(X_j-X_k)$, where X_j and X_k are randomly selected solutions from the population, and ϵ is a random number in [0, 1]. Evaluate the

new solution X_{new} and replace the old solution X_i if the new solution is better.

5.4 Convergence: Check the stopping criterion (maximum number of iterations or a convergence threshold). If it is satisfied, terminate, else repeat the iteration step.

5.5 Output: The best solution $\boldsymbol{X = \{x_s, x_2, \ldots, x_d\}}$ is found, representing the optimal or near-optimal path in the graph.

Example 3.3: **Moth Flame Optimization (MFO)**

The MFO is a nature-inspired metaheuristic algorithm based on the navigation behavior (transverse orientation) of moths. It is presented here because it can be adapted to explore and exploit solutions to find an optimal or near-optimal path while considering multiple objectives like distance D, energy E, obstacle avoidance O, and time T. It is effective for solving high-dimensional and complex pathfinding problems for robots and vehicles. It can be adapted to dynamic environments by adjusting the flame positions and making it suitable for real-time pathfinding. Following are the specifications of MFO adapted for solving robotic and vehicle pathfinding problems with multiple objectives following the components of the proposed framework shown in Fig. 2.

1. Objective: The goal is to find the shortest or least-cost path while traversing from a start vertex s to a target vertex t in a given graph using the metaheuristic MFO while considering: Minimization of path length D, Minimization of energy consumption E, Avoidance of obstacles O, and Minimization of travel time T.

2. Problem Representation: It is represented as G (V, E) as in the previous two examples. Each moth represents a potential path in the search space. Each moth is defined by a vector of coordinates corresponding to waypoints along the path. The flames represent the best solutions (paths) found so far; and moths are attracted to flames, with each flame representing a promising solution.

3. *Key Components of MFO: 1.* Moths represent candidate solutions (paths), where each moth is attracted to a flame, and its position is updated accordingly. 2**.** Flames represent the best solutions in the population, where flames guide the search process, and the number of flames decreases as the iterations proceed, focusing the search. 3. *Transverse Orientation:* Moths update their positions by flying around flames in a logarithmic spiral pattern, similar to formula (6) simulating their natural behavior of moving towards light sources.

$$M(t+1) = D_i * e^{-b*l*\cos(2\pi l)} + F_i(t) \quad (6)$$

Where: $M_i(t+1)$ represents the position of moth i in the next iteration. D_i represents the distance between the moth and the flame. $F_i(t)$ represents the position of flame i at iteration t and b is a constant defining the shape of the logarithmic spiral. l is a random number in the interval $[-1,1]$ that determines the direction of the movement (closer or further from the flame).

5. Objective Function: It combines multiple factors as in formula (7).

$$\text{Minimize } F(X) = w1 \sum_{i=1}^{n} (D(x_i) + w1 \sum_{i=1}^{n} O(x_i) + w3 \sum_{i=1}^{n-1} E(x_i) + w4 \sum_{i=1}^{n-1} T(x_i) \quad (7)$$

where: $D(\boldsymbol{x_i})$: Distance traveled at point $\boldsymbol{x_i}$ (path length), $O(\boldsymbol{x_i})$: Obstacle avoidance term, where proximity to obstacles or collisions leads to a higher penalty, $E(\boldsymbol{x_i})$: Energy consumption at point $\boldsymbol{x_i}$ (e.g., fuel, battery, etc.), and $T(\boldsymbol{x_i})$: Time taken to travel through point $\boldsymbol{x_i}$; and w1, w2, w3, and w4 are weight coefficients that balance the importance of each factor. The distance metric can be computed using Euclidean distance between waypoints or other appropriate distance measures depending on the environment. The obstacle avoidance term O(x) which can be defined as $O(\boldsymbol{x_i}) = \infty$ if the path intersects with an obstacle, and $O(\boldsymbol{x_i})=1$ if the path is near obstacle. A penalty is applied when the path intersects with obstacles to avoid collision. Energy consumption E(x) can be based on the vehicle's or robot's speed, acceleration, or path complexity as shown in formula (8).

$$E(\boldsymbol{x_i}) = c1 * (d(\boldsymbol{x_i}, \boldsymbol{x_{i+1}}) * v(\boldsymbol{x_i})) + c2 \cdot a(\boldsymbol{x_i}) \quad (8)$$

where: c1 and c2 are constants defining the energy cost per distance and acceleration, respectively, $v(\boldsymbol{x_i})$ denotes velocity at point $\boldsymbol{x_i}$, and $a(\boldsymbol{x_i})$ denotes acceleration. The time factor T(x) can be modeled as the time required to traverse each segment of the path as in formula (9).

$$T(\boldsymbol{x_i}) = D(\boldsymbol{x_i}) * v(\boldsymbol{x_i}) \quad (9)$$

5. MFO Algorithm Structure:

5.1 Initialize Population: Initialize the positions of moths randomly within the defined search space, set the number of flames- based on the problem's requirements, and define the search boundaries for the pathfinding space.

5.2 Initial Solutions: Evaluate the fitness of each moth based on the multi-objective criteria: path length, obstacle avoidance, energy consumption, and time as in formulas (8), (9), and (7). Rank the moths based on their fitness values, where moths with lower fitness are ranked higher. *Select* the top moths

as flames. Each moth *updates* its position by moving (movement uses the logarithmic spiral to simulate the transverse orientation mechanism as in formula (6)) toward one of the flames. The number of flames decreases as iterations progress, encouraging exploration in the early stages and exploitation in later stages. Ensure that the new positions of the moths remain within the predefined search space boundaries.

5.3 Evaluate and Update Solutions: Evaluate the new solutions (updated moth positions) based on the objective function. Update the positions of the flames if better solutions are found.

5.4 Output: When a stopping criterion is met, such as a maximum number of iterations, convergence, or achieving a threshold fitness value, output or display the best solution $\boldsymbol{X = \{x_s, x_2, \ldots, x_t\}}$ representing the optimal or near-optimal path in the graph. One example of the MFO implementation is similar to the MFO algorithm 1.

Over the last five years (2019- 2024), more than 500 metaheuristic algorithms have been developed and applied to solve pathfinding problems [44], [45]. Table 3 (Appendix) shows a list of 12 different metaheuristic algorithms with inspiration-based, solution representations, and their objective functions. These algorithms are Genetic algorithm (GA), Particle Swarm Optimization (PSO), Simulation Annealing, Ant Colony Optimization (ACO), Differential Evolution (DE), Grey Wolf Optimization (GWO), Bat Algorithm (BA), Firefly Algorithm (FA), Harmony Search (HS), Cuckoo Search, Whale Optimization Algorithm (WOA), Crow Search Algorithm (CSA). You can follow the guidelines of the proposed framework to implement each of these algorithms in solving the pathfinding problems. It is expected that each F(X) evaluates the minimum distance or cost of path X and other factors can be considered too.

In the stochastic environment, pathfinding for vehicles *faces various obstacles. These obstacles include* environmental uncertainties, vehicle constraints, stochastic variability, and driver behavior [46]. Handling such obstacles requires advanced techniques such as probabilistic models (e.g. MDP), reinforcement learning, robust planning algorithms, and metaheuristic behavior. In this paper, we consider adaptive heuristics and operators, enhanced fitness functions, and the hybrid approach of ACO and MFO. For example, pheromone levels in ACO are adjusted based on environmental conditions. In MFO, when obstacles disrupt the solution space, the MFO algorithm repositions the candidate solutions to reflect the updated environment. The ACO, MFO, and the combination of ACO and MFO algorithms include multiple criteria in the fitness function such as time, distance, obstacle avoidance, and energy consumption. These three algorithms add penalties for high-uncertainty regions or routes. We combine the ACO and MFO metaheuristics algorithms to leverage complementary strengths.

4 Implementation and Results

The algorithms A* search, ACO, MFO, and ACO+ MFO combination were implemented and tested on randomly generated sample dataset with sizes.

The algorithms A* search, ACO, MFO, and ACO+ MFO combination were implemented and tested on randomly generated sample dataset with sizes of 100 by 100, 200 by 200, …, 1000 by 1000, and a mountainous area of Jibal Al-Sharat in Jordan. The second dataset includes a digital environment model (DEM) obtained from satellite imagery, a variety of elevation levels, obstacles, and geographical features [47]. Each algorithm was run five times and the average of the metrics were registered. The evaluated metrics are run time, consumed energy, number of obstacles encountered, length of the shortest path (distance), and number of explored vertices. The consumed energy indicates the amount of energy required by an agent to traverse a path from a start position to a target position. The number of obstacles encountered indicates the number of times the agents encountered obstacles (e.g. elevation based and slope) during their search for the optimal path in the terrain. The number of explored vertices indicates the number of unique nodes that were explored. The cost of the optimal path with these or some of these factors can be computed by objective function $C(X) = \min (\sum_X w1 \cdot D(x_i, x_{i+1}) + w2 \cdot O(x_i) + w3.E(x_i) + w4.T(x_i))$.

Table 4 (Appendix) shows samples of the performance results. It is clear that when the sizes increase, the run time increases in the four algorithms. The combination of the MFO+ACO algorithm outperforms the other three algorithms in the run time in all selected data sizes. The ACO performed the worst in the majority of cases. It almost produced the same average best for different sizes and fixed starting vertex and target vertex. The three algorithms explored less number of vertices than ACO.

Table 5 shows the performance of the average run time of various algorithms (ACO+ MFO, MFO, A*, and ACO) across different grid sizes, for other start and target points differ from those tested in Table 4 (Appendix). The hybrid approach of

ACO+MFO consistently exhibited the least average runtime across all grid sizes of randomly generated datasets. Thus, combining the exploration capabilities of MFO with the exploitation strengths of ACO yields less run time for vehicle pathfinding. The A* algorithm exhibited higher average runtimes compared to both MFO and ACO. This may be attributed to the heuristic nature of A*.

Table 5. Average run time (seconds) for randomized generated datasets

Algs/ Sizes	600	700	800	900	1000
ACO+ MFO	0.70	0.73	0.886	1.29	1.43
MFO	1.34	1.58	1.10	1.22	1.69
A*	2.28	2.30	1.26	1.50	1.73
ACO	9.89	10.02	10.2	11.25	12.3

Table 6 shows the average run time of the algorithms across different grid sizes obtained from Jibal Al-Sharat datasets. The hybrid approach of ACO+MFO exhibited the least average runtime across all shown grid sizes. Thus, combining the exploration capabilities of MFO with the exploitation strengths of ACO yields less run time for vehicle pathfinding. The A* algorithm exhibited higher average runtimes compared to both MFO and ACO.

Table 6. Average Run Time (in seconds) from a sample of the Jibal Al-Sharat dataset

Alg/ Sizes	600	700	800	900	1000
ACO+ MFO	0.68	0.72	0.80	1.20	1.40
MFO	1.30	1.50	1.00	1.20	1.60
A*	2.20	2.40	1.30	1.60	1.80
ACO	9.00	9.80	10.00	11.00	12.00

Table 7. Cost functions for 1000*1000 randomize generated data, (D-Distance, O-obstacle, E-Energy, T-average run time)

Algo.	Cost 1 (D + O)	Cost 2 (D + O + E)	Cost 3 (D + O + E + T)
ACO+ MFO	414.46	649.76	793.003
MFO	493.42	782.916	951.915
A*	693.42	1076.11	1249.11
ACO	773.83	964.43	2194.43

Table 7 and Tabble 8 show the cost functions performance of the four algorithms on a dataset generated randomly, and sample from the Jibal Al-Sharat dataset, respectively, of 1000x1000 size. The hybrid algorithm has the lowest costs across all three categories, which means that it is the most efficient algorithm among the four algorithms on this grid size. The cost from the MFO is less than the costs from ACO and A* for the dataset size of 700 to 1000. The ACO has the highest costs, especially in cost 3 because of the poor run-time efficiency. The A* algorithm encounters difficulties with scaling, which results in higher costs for larger grids especially in runtime and energy consumption. The MFO algorithm performs better than ACO and A* but continues to fall behind the ACO+MFO algorithm.

Table 8. Cost functions for 1000*1000 from sample of Jibal Al-Sharat dataset

Algorithm	Cost 1 (D + O)	Cost 2 (D + O + E)	Cost 3 (D + O + E + T)
ACO+MFO	746.7	1099.6	1239.6
MFO	822.2	1256.5	1416.5
A*	1052.3	1626.3	1806.3
ACO	773.8	1869.1	2069.1

5 Conclusion

In this paper, the specifications of heuristic or metaheuristic algorithms for pathfinding problems are presented in a framework. The pathfinding problems and their representations in a graph were presented. The objective functions to optimize pathfinding problems were clearly explained. A framework and its components were proposed and explained how to be utilized to solve pathfinding problems. The metaheuristic algorithms and their classifications to solve pathfinding problems were overviewed.

Sample examples of A*, ACO, MFO, and a combination of ACO and MFO were explained how to follow the framework to solve the pathfinding problems. The MFO algorithm alone performs well but lacks the refinement provided by ACO, while the ACO by itself explores more nodes and consumes more energy, making the hybrid MFO + ACO a balanced solution for scenarios requiring both path efficiency and obstacle avoidance.

In conclusion, both ACO and MFO incorporate common strategies to handle dynamic obstacles such as: 1) incorporating real-time data to adjust paths accordingly, 2) utilizing fitness functions including penalties for collisions and encouraging exploration of safer paths, 3) ensuring exploration continues even when parts of the solution space are disrupted, and 4) using incremental learning for efficient adaptation when parts of the environment remain static.

This paper introduces how the proposed framework for metaheuristics techniques will help interested readers understand how pathfinding algorithms work and pick the best fit for a particular application.

For future research, we plan to investigate how ACO and other metaheuristic optimization can leverage deep learning (DL) and machine learning (ML) techniques to enhance their ability to handle obstacles in dynamic environments for pathfinding problems. By integrating ML/DL, we expect that these metaheuristic algorithms can make smarter decisions, adapt faster to changes, and improve pathfinding efficiency. In the stochastic environment, the hybrid of ACO and MFO and other metaheuristic approaches can be investigated when using probabilistic models (e.g. MDP), reinforcement learning, robust planning algorithms, and metaheuristic behavior. Also, we intend to refine and combine metaheuristic algorithms to solve the pathfinding problems for autonomous vehicles and unmanned aircraft vehicles (UAV).

Acknowledgement:
I appreciate and acknowledge the support of the University of Jordan for granted me a sabbatical leave during the academic year 2023-2024.

References:

[1] Peng Duan, Zhenao Yu, Kaizhou Gao, Leilei Meng, Yuyan Han, Fan Ye, Solving the multi-objective path planning problem for mobile robot using an improved NSGA-II algorithm, *Swarm and Evolutionary Computation*, Vol. 87, 2024, pp. 101576. https://doi.org/10.1016/j.swevo.2024.101576.

[2] Daniele Ferone, Paola Festa, Serena Fugaro, Tommaso Pastore, A dynamic programming algorithm for solving the k-Color Shortest Path Problem, *Optimization Letters*, Vol.15. 2021, pp. 1973–1992.https://doi.org/10.1007/s11590-020-01659-z.

[3] Shima Khoshraftar, and Aijun An, A Survey on Graph Representation Learning Methods, *ACM Transactions on Intelligent Systems and Technology*, Vol. 15, Issue 1, 2024, pp.1 – 55. https://doi.org/10.1145/3633518.

[4] Rakan Nasir H Alhwety and Nazar Elfadil, Vehicle Tracking System Approaches: A Systematic Literature, *International Journal of Computer Science and Mobile Computing*, Vol.13 Issue.8, 2024, pp. 23-31. DOI: 10.47760/ijcsmc.2024.v13i08.003.

[5] Saman M. Almufti, Ridwan Boya Marqas, Vaman Ashqi Saeed, Taxonomy of bio-inspired optimization algorithms, *Journal of Advanced Computer Science & Technology,* Vol. 8, No. 2, 2019, pp. 23-31. https://doi.org/10.14419/jacst.v8i2.29402.

[6] Yan, Ying, Research on optimal path planning technology for vehicle positioning and navigation system, *Applied Mathematics and Nonlinear Sciences*, Vol 9, No. 1, 2023, pp.1-20. DOI: 10.2478/amns.2023.1.00397.

[7] Peng Liu, Xuekui Wang, Liangfei Yin, Beng Liu. Flat random forest: a new ensemble learning method towards better training efficiency and adaptive model size to deep forest. *International Journal of Machine Learning and Cybernetics*, Vol.1, No. 2, 2024, pp. 499-513. DOI: 10.1007/s13042-020-01136-0.

[8] Chee Sheng Tan, Rosmiwati Mohd-Mokhtar, and Mohd Rizal Arshad, A Comprehensive Review of Coverage Path Planning in Robotics Using Classical and Heuristic Algorithms, *IEEE Access*, 2021, Vol. 9, 2021, pp. 119310 -119342. DOI: 10.1109/ACCESS.2021.3108177.

[9] Vinita Tomar, Mimta Bansal, and Pooja Singh, Metaheuristic Algorithms for Optimization: A Brief Review, *Engineering Proceedings,* Vol. 59, No.1, 2023, pp. 238. https://doi.org/10.3390/engproc2023059238.

[10] M. Almufti, S., Ahmad Shaban, A., Ismael Ali, R., & A. Dela Fuente, J., Overview of Metaheuristic Algorithms. *Polaris Global Journal of Scholarly Research and Trends*, Vol. 2, No.2, 2023, pp.10–32. https://doi.org/10.58429/pgjsrt.v2n2a144.

[11] Michahael T. Goodrich and RobTamassi, Algorithm Design: Foundations, Analysis and Internet Examples, 2009, John Wiley & Sons, Inc. New York, NY, US, 2009, DOI: 10.1145/992287.992293.

[12] Vicky Indriyono and Widyatmoko, Optimization of Breadth-First Search Algorithm for Path Solutions in Mazyin Games Bonifacius, International Journal of Artificial Intelligence & Robotics (IJAIR) E-ISSN: 2686-6269 Vol.3, No.2, 2021, pp.58-66. DOI:10.25139/ijair.v3i2.4256.

[13] Anton Jesuthas, Nickson Joram and Suthakar, Somaskandan, Path-finding and Planning in a 3D Environment An Analysis Using Bidirectional Versions of Dijkstra's, Weighted

A*, and Greedy Best First Search Algorithms, *2nd Asian Conference on Innovation in Technology (ASIANCON)*, Pune, India,2022. DOI: 10.1109/ASIANCON55314.2022.9909251.

[14] Ismail H. Toroslu, The Floyd-Warshall all-pairs shortest paths algorithm for disconnected and very sparse graphs, *Software: Practice and Experience*, Vol 53, Issue 6, 2023, pp. 1287-1303. https://doi.org/10.1002/spe.3188.

[15] Joshua Erlangga Sakti, Muhammad Arzaki, Gia Septiana Wulandari, A Backtracking Approach for Solving Path Puzzles, *Journal of Fundamental Mathematics and Applications(JFMA)*, Vol. 6, No.2, 2023, pp.117-135. DOI: 10.14710/jfma.v6i2.18155.

[16] W. Susanto, S. Dennis, M. Brian Aqacha Handoko and K. Margi Suryaningrum, "Compare the Path Finding Algorithms that are Applied for Route Searching in Maps," 2021 *1st International Conference on Computer Science and Artificial Intelligence (ICCSAI)*, Jakarta, Indonesia, 2021, pp. 178-183, Doi: 10.1109/ICCSAI53272.2021.9609742

[17] Kui, L.; Yu, X. A Pathfinding Algorithm for Large-Sca16le Complex Terrain Environments in the Field. *ISPRS Int. J. Geo-Inf*. Vol 13, Issue 7, 2024, pp. 251. https://doi.org/10.3390/ ijgi13070251.

[18] Chen, Y.-J.; Jhong, B.-G.; Chen, M.-Y. A Real-Time Path Planning Algorithm Based on the Markov Decision Process in a Dynamic Environment for Wheeled Mobile Robots. *Actuators*, Vol.12, Vol. 4, 2023, pp. 166. https://doi.org/10.3390/ act12040166.

[19] Gad, Ahmed G., Particle Swarm Optimization Algorithm and Its Applications: A Systematic Review, *Archives of Computational Methods in Engineering*. Vol. 29, No. 5, 2022, pp. 2531–2561. DOI: 10.1007/s11831-021-09694-4.

[20] Shubham Gupta, Hammoudi Abderazek, Betül Sultan Yıldız, Ali Riza Yildiz, Seyedali Mirjalili, Sadiq M. Sait, Comparison of metaheuristic optimization algorithms for solving constrained mechanical design optimization problems, *Expert Systems with Applications*, Vol. 183, 2021, pp. 115351, https://doi.org/10.1016/j.eswa.2021.115351.

[21] Daniel Foeada, Alifio Ghifaria, Marchel Budi Kusumaa, Novita Hanafiahb, and Eric Gunawanb, A Systematic Literature Review of A* Pathfinding, Procedia Computer Science, Vol.179, 2021, pp. 507–514. https://doi.org/10.1016/j.procs.2021.01.034.

[22]]Debora DiCaprio, Ali Ebrahimnejad, Hamidr eza Alrezaamiri, Francisco J. Santos-Arteaga, A novel Ant Colony Algorithm for Solving Shortest Path Problems with Fuzzy Arc Weights, *Alexandria Engineering Journal*, Vol. 61, Issue 5, 2022, pp 3403-3415. https://doi.org/10.1016/j.aej.2021.08.058.

[23] M. Rhifky Wayahdi, Subhan Hafiz Nanda Ginting, Dinur Syahputra, Greedy, A-Star, and Dijkstra's Algorithms in Finding Shortest Path, *International Journal of Advances in Data and Information Systems*, Vol. 2, No. 1, 2021, pp. 45-52. DOI: 10.25008/ijadis.v2i1.1206 r 45.

[24] Hong, Z.; Sun, P.; Tong, X.; Pan, H.; Zhou, R.; Zhang, Y.; Han, Y.; Wang, J.; Yang, S.; Xu, L. Improved A-Star Algorithm for Long-Distance Off-Road Path Planning Using Terrain Data Map. *ISPRS Int. J. Geo-Inf.*, Vol. 10, No. 11, 2021, pp. 785. https://doi.org/10.3390/ijgi10110785.

[25] Dian Rachmawati and Lysander Gustin, Analysis of Dijkstra's Algorithm and A* Algorithm in Shortest Path Problem, Journal of Physics: Conference Series, Vol. 1566, 2020, pp. 012061 DOI: 10.1088/1742-6596/1566/1/012061.

[26] Daniel Harbor and Alban Grastien, Improving Jump Point Search, *Proceedings of the International Conference on Automated Planning and Scheduling 2014*, 2014 pp.128-135, Australia. http://dx.doi.org/0.1609/icaps.v24i1.1363.

[27] Shrawan Kumar Sharma and Shiv Kumar, Comparative Analysis of Manhattan, and Euclidean Distance metrics using A* Algorithm, *Journal of Research in Engineering and Applied Sciences*, Vol. 1, Issue 4, 2016, pp. 196-198. DOI: 10.46565/jreas.2016.v01i04.007.

[28] K Zhigalov, D K-S Bataev, E Klochkova, O A Svirbutovich and G A Ivashchenko, Problem solution of optimal pathfinding for the movement of vehicles over rough mountainous areas, IOP Conference Series: Materials Science and Engineering, Saint-Petersburg, Russian Federation, Vol. 1111, No. 1, 2021, pp. 012033. DOI 10.1088/1757-899X/1111/1/012033.

[29] Yella Swamy, K., Gogineni, S., Gunturu, Y., Gudapati, D., & Tirumalasetti, R. (2017). Finding the shortest path using the ant colony optimization. *International Journal of*

Engineering & Technology, Vol. 7, No. 1.1, pp. 392-396. https://doi.org/10.14419/ijet.v7i1.1.9859.

[30] Russell, S., & Norvig, P., Artificial Intelligence: A Modern Approach (4th ed.). Pearson, 2021. https://doi.org/10.1109/MSP.2017.2765202.

[31] Gede Putra Kusuma, Ricky Martin Goutama and Steven Ferdianto. Comparing Pathfinding Algorithms for Indoor Positioning System, *ICIC Express Letters*, 2022, Vol. 16, No. 3, 2022, pp. 299–306. DOI: 10.24507/icicel.16.03.299.

[32] Reda Mohamed, Ahmed Onsya, Amira Y. Haikalb, and Ali Ghanbaria, Path planning algorithms in the autonomous driving system: A comprehensive review, *Robotics and Autonomous Systems*, Vol. 174, No. 10, 2024, pp. 104630. https://doi.org/10.1016/j.robot.2024.104630.

[33] Shabina Banu Mansuri, Shiv kumar, Comparative Analysis of Path Finding Algorithms, *IOSR Journal of Computer Engineering (IOSR-JCE)*, Vol. 20, Issue 5, Ver. I, 2018, pp. 38-45. doi:10.9790/0661-2005013845.Issue 12, 2019, pp. 83- 87.

[34] Ab Wahab, M. N., Nazir, A., Khalil, A., Ho, W. J., Akbar, M. F., Noor, M. H. M., & Mohamed, A. S. A., Improved genetic algorithm for mobile robot path planning in static environments. *Expert Systems With Applications*, Vol. *249*, No. 3, 2024, pp. 123762. https://doi.org/10.1016/j.eswa.2024.123762.

[35] Wenrong Jiang, Analysis of Iterative Deepening A* Algorithm, 8th Annual International Conference on Geo-Spatial Knowledge and Intelligence IOP Conf. Series: Earth and Environmental Science Vol. 693, 2021, pp. 012028 IOP Publishing, Shaanxi, China. DOI: 10.1088/1755-1315/693/1/012028.

[36] Luo, Y.; Lu, J.; Zhang, Y.; Qin, Q.; Liu, Y. 3D JPS Path Optimization Algorithm and Dynamic-Obstacle Avoidance Design Based on Near-Ground Search Drone, *Applied Science*. Vol.12, No. 14, 2022, pp. 7333. https:// doi.org/10.3390/app12147333.

[37] C. Van Dang, H. Ahn, D. S. Lee and S. C. Lee, A Path Planning Method Based on Theta-star Search for Non-Holonomic Robots, Joint 12th International Conference on Soft Computing and Intelligent Systems and 23rd International Symposium on Advanced Intelligent Systems (SCIS&ISIS), Ise, Japan, 2022, pp. 1-6, DOI: 10.1109/SCISISIS55246.2022.10002141.

[38] Diego Carlos Luna M´arquez ∗ Dante M´ujica-Vargas, Ana Monserrat Rojas Fern´andez, The Theta* Algorithm for Path Calculation, *Jornada de Ciencia y Tecnología Aplicada Tecnológico Nacional de México/CENIDET*, Vol. 2. 2019.

[39] Irfan Darwin and Suryadiputra Liawatimena, Dynamic Map Pathfinding Using Hierarchical Pathfinding Theta Star Algorithm, *Journal of Theoretical and Applied Information Technology*, Vol. 99, No. 20, 2019, pp. 4875-4885, [Online]. https://api.semanticscholar.org/CorpusID:245019366 (Accessed Date: October 15, 2024).

[40] Guanghui Li, Taihua Zhang, Chieh-Yuan Tsai, Liguo Yao, Yao Lu, Jiao Tang, Review of the metaheuristic algorithms in applications: Visual analysis based on bibliometrics, *Expert Systems with Applications,* Vol. 255, Part D, 2024, 124857. https://doi.org/10.1016/j.eswa.2024.124857.

[41] Franco, Dario Diaz, Jesus Hernandez-Barragan, Nancy Arana-Daniel and Michel Lopez-Franco , A Metaheuristic Optimization Approach for Trajectory Tracking of Robot Manipulators Carlos Lopez- Franco, *Mathematics*, Vol. 10, 2022, 10, pp. 1051. https://doi.org/10.3390/math10071051.

[42] Umit Can1, Bilal Alatas, Physics Based Metaheuristic Algorithms for Global Optimization, *American Journal of Information Science and Computer Engineering*, Vol. 1, No. 3, 2015, pp. 94-106. http://www.aiscience.org/journal/ajisce.

[43] S. Kavita and S. S. K., Metaheuristic Evolutionary Algorithms: Types, Applications, Future Directions, and Challenges, 2023 3rd International Conference on Intelligent Technologies (CONIT), Hubli, India, 2023, pp. 1-6, DOI: 10.1109/CONIT59222.2023.10205592.

[44] S. Consoli, Supervisor: Prof. K. Darby-Dowman, Operational Research Report: Combinatorial Optimization and Metaheuristics, School of Information Systems, Computing and Mathematics, Brunel University, January 2006, [Online]. http://hdl.handle.net/2438/503. (Accessed Date: October 15, 2024).

[45] OpenAI. (2024). *ChatGPT* [Large language model]. https://chatgpt.com/g/g-zRe3DR0KX-login-page/c/4b1fff4a-dbbd-4df3-ae98-

fed46506c97c (Accessed Date: September 20, 2024).

[46] Mahmoud Jahjouh, Uneb Gazder, Rashid Abdulrahman Ismaeel, Development of a Traffic Microsimulation Tool with the Incorporation of Variations in Driver Behaviors, *WSEAS Transactions on Computers*, Vol. 21, 2024 pp. 226-236. https://doi.org/10.37394/23205.2024.23.22.

[47] Earth Explorer. USGS, [Online]. https://earthexplorer.usgs.gov (Accessed Date: December 21, 2024).

Contribution of Individual Authors to the Creation of a Scientific Article (Ghostwriting Policy)

Conceptualization, methodology, validation investigation, resources, data curation, writing draft preparation, editing, Sharieh; Ahmad Sharieh carried out the simulation and the optimization, he organized and executed the experiments of Section 4, he implemented the Algorithms, and was responsible for the Statistics.

The author contributed to the present research, at all stages from the formulation of the problem to the final findings and solution.

Sources of Funding for Research Presented in a Scientific Article or Scientific Article Itself

No funding was received for conducting this study.

Conflict of Interest

The author has no conflicts of interest to declare.

APPENDIX

Table 2. Examples of pathfinding applications and the values to optimize [4], [5]. Those from 1 to 10, 11 to 17, and 18 minimize F(P), maximize F(P), and 21 optimizes F(P), respectively. Assume $P = X$ path and $(\boldsymbol{v_i}, \boldsymbol{v_{i+1}}) = (\boldsymbol{x_i}, \boldsymbol{x_{i+1}})$

Optimization functions F(P)=	Applications
1. Minimize the total cost of the path (P). Cost includes monetary, time, and resource usage. Minimize **F(P) =** $\sum_{i=1}^{n-1} \boldsymbol{cost}\,(\boldsymbol{v_i}, \boldsymbol{v_{i+1}})$	Logistics, delivery routing, network traffic optimization.
2.the total travel distance. $\sum_{i=1}^{n-1} \boldsymbol{distance}\,(\boldsymbol{v_i}, \boldsymbol{v_{i+1}})$	Road networks and robotic path planning
3.the total travel time, considering speed limits or constraints. $\sum_{i=1}^{n-1} \boldsymbol{travel_{time}}\,(\boldsymbol{v_i}, \boldsymbol{v_{i+1}})$; ***Subjective to: speed***$(\boldsymbol{v_i}, \boldsymbol{v_{i+1}})$	Emergency response routing, and transportation planning.
4.the risk or danger can be influenced by factors such as traffic congestion, crime rates, or environmental hazards. $\sum_{i=1}^{n-1} \boldsymbol{risk}\,(\boldsymbol{v_i}, \boldsymbol{v_{i+1}})$	Safe route planning, disaster management.
5.the energy consumption of the path relevant to electric vehicles or robots. $\sum_{i=1}^{n-1} \boldsymbol{energy}\,(\boldsymbol{v_i}, \boldsymbol{v_{i+1}})$	Electric vehicle routing, autonomous robot navigation.
6.the deviation from a predefined path or route. $\sum_{i=1}^{n-1} \boldsymbol{deviation}(\boldsymbol{v_i}, \boldsymbol{v_{i+1}})$	Autonomous vehicle navigation, and guided tours [32].
7.the number of stops or nodes visited along the path. $\sum_{i=1}^{n-1} \boldsymbol{number\_\ of\_\ stops}(\boldsymbol{v_1 \ldots v_n})$	Direct delivery routing, and express transport services.
8.the environmental impact such as emissions or disturbance to wildlife. $\sum_{i=1}^{n-1} \boldsymbol{environmental\_impact}(\boldsymbol{v_i}, \boldsymbol{v_{i+1}})$	Eco-friendly transportation planning, sustainable logistics.
9.the complexity of the path, such as the number of turns or intersections. $\sum_{i=1}^{n-1} \boldsymbol{complexity}(\boldsymbol{v_i}, \boldsymbol{v_{i+1}})$	Simple route planning, novice driver navigation.
10.The detours or unnecessary deviations from the direct route. $\sum_{i=1}^{n-1} \boldsymbol{detour}\,(\boldsymbol{v_i}, \boldsymbol{v_{i+1}})$	Efficient commuting, time-sensitive deliveries.
11.Maximize the visibility or coverage of certain areas along the path. **F(P)** $= \sum_{i=1}^{n-1} \boldsymbol{visibility}\,(\boldsymbol{v_i}, \boldsymbol{v_{i+1}})$	Surveillance drone path planning, wildlife monitoring.
12.the comfort by considering factors like road quality, noise levels, or scenery. $\sum_{i=1}^{n-1} \boldsymbol{comfort}\,(\boldsymbol{v_i}, \boldsymbol{v_{i+1}})$	Tourist route planning, long-distance travel optimization
13.tthe connectivity or access to resources or points of interest along the path. $= \sum_{i=1}^{n-1} \boldsymbol{connectivity}\,(\boldsymbol{v_i}, \boldsymbol{v_{i+1}})$	Resource allocation, tourism planning.
14.the flow or throughput of the path. $\sum_{i=1}^{n-1} \boldsymbol{throughput}\,(\boldsymbol{v_i}, \boldsymbol{v_{i+1}})$	Network traffic optimization, and data routing in communication networks.
15.the capacity of the path to handle traffic or load. $\sum_{i=1}^{n-1} \boldsymbol{capacity}\,(\boldsymbol{v_i}, \boldsymbol{v_{i+1}})$	Traffic management, and load balancing in networks.
16.the smoothness of the path by reducing abrupt changes in direction or speed. $\sum_{i=1}^{n-1} \boldsymbol{smoothness}\,(\boldsymbol{v_i}, \boldsymbol{v_{i+1}})$	Autonomous vehicle navigation, and comfortable travel routes.
17. the reliability or robustness of the path against disruptions or failures. $\sum_{i=1}^{n-1} \boldsymbol{reliability}\,(\boldsymbol{v_i}, \boldsymbol{v_{i+1}})$	Critical infrastructure routing, military logistics.
18. Optimize multiple criteria simultaneously, such as distance, energy, and time. Use a weighted sum of different objective functions F1, F2, F3, ... etc., as in function (2). Example: ***F(P)=w1×F1(distance)+w2×F2(energy)+w3×F3(time)***	Complex routing problems, where trade-offs between different factors are necessary.

Table 3. list of 12 metaheuristic algorithms to optimize routes for robots and vehicles

Name	Inspired By	Solution Representation and Objective Function
GA	Natural evolution	The population of chromosomes represents different possible sets of routes. A gene in the chromosome represents a stop on a route. $F(X)=\min(\sum_X w1.D(x_i,x_{i+1})+ w2.T(x_i)+w3.O(x_i))$
PSO	Bird flocking, fish schooling.	A particle represents a set of routes for all vehicles. A particle's position corresponds to a complete vehicle routing solution. $F(X)=\min(\sum_X w1.D(x_i,x_{i+1})+ w2.T(x_i)+w3.E(x_i))$
SA	Annealing process in metallurgy	A single solution (a complete set of vehicle routes) is gradually improved by perturbing the current solution based on temperature parameters. $F(X)=\min(\sum_X w1.D(x_i,x_{i+1})+ w2.E(x_i)+w3.O(x_i))$
ACO	Behaviour of ant colonies in searching for food	The sequence of customer stops makes up a vehicle route. The solution emerges from the collective behavior of ants finding optimal paths. $F(X)=\min(\sum_X 1/\tau(x_i,x_{i+1}).D(x_i,x_{i+1})$
DE	Evolutionary strategy	A vector represents a solution. Each element of the vector corresponds to an agent and its assigned vehicle route. $F(X)=\min(\sum_X \text{mutation}(x_i,x_{i+1}) + cr\ (x_i,x_{i+1})+ D(x_i,x_{i+1}))$
GWO	Leadership hierarchy and hunting mechanism of wolves	A grey wolf represents a set of vehicle routes. Different wolves represent different solutions. $F(X)=\min(\sum_X \alpha.w1(x_i,x_{i+1})\cdot D(x_i,x_{i+1}))$
BA	Echolocation of bats	A bat represents a vehicle route. The position of the bat corresponds to the configuration of the routes, and it adjusts based on echolocation principles. $F(X)=\min(\sum_X \text{loudness}(x_i,x_{i+1}) + w1.D(x_i,x_{i+1})+w2.E(x_i))$
FA	Attraction of fireflies	A firefly represents a set of routes, where the brightness of the firefly corresponds to how good the solution is. $F(X)=\min(\sum_X I/r(x_i,x_{i+1})+ D(x_i,x_{i+1}))$
HS	Jazz improvisation process	A harmony vector represents a candidate solution, where each element corresponds to a specific vehicle route. $F(X)=\min(\sum_X \text{pitch}(x_i,x_{i+1})+ w1.D(x_i,x_{i+1})+w2.T(x_i)+w3.O(x_i))$
CS	Brood parasitism of some cuckoo species	A cuckoo egg represents a set of vehicle routes. It replaces less optimal nests with new ones as it searches for better routes. $F(X)=\min(\sum_X w1.L(\lambda) + w2.D(x_i,x_{i+1})+w3.O(x_i)+ w4.T(x_i))$
WOA	Bubble-net hunting strategy of humpback whales	A whale represents a set of routes. The solution is adjusted by the whale's movements in the search space. $F(X)=\min(\sum_X w1\cdot D(x_i,x_{i+1})+ w2\cdot O(xi) +w3.T(x_i)+w4.E(x_i))$
CS	Intelligent caching behavior of crows	A crow represents a set of routes. Each crow hides its solution in its memory. It updates the solution based on interactions between crows and their stored routes. $F(X)=\min(\sum w1\cdot D(x_i\ ,\ x_{i+1})+w2\cdot O(xi))$

Table 4. Samples of results on data sizes 500 by 500 to 1000 by1000. Runtime in seconds, and Length or distance in meters

Algorithm	Average of runtime	Best Path Length	Number of vertices Explored
500*500			
MFO + ACO	0.546	49	15
MFO	1.123	55	14
A*	1.24	57	40
ACO	8.19	112	106
600*600			
MFO + ACO	0.096	89	16
MFO	1.245	59	14
A*	22.76	57	40
ACO	9.0	113	122
700*700			
MFO + ACO	0.231	90	14
MFO	1.085	136	15
A*	12.0	56	40
ACO	9.90	113	104
800*800			
MFO + ACO	0.886	136	15
MFO	1.104	57	17
A*	1.26	57	40
ACO	10.2	111	104
900*900			
MFO + ACO	0.193	59	14
MFO	1.22	65	16
A*	1.50	49	85
ACO	11.25	113	128
1000*1000			
MFO + ACO	1.23	48	13
MFO	1.29	49	15
A*	1.73	56	40
ACO	12.3	64	157