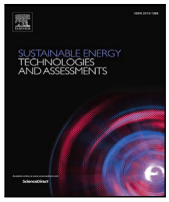
Sustainable Energy Technologies and Assessments 56 (2023) 103032

Contents lists available at ScienceDirect

Sustainable Energy Technologies and Assessments

journal homepage: www.elsevier.com/locate/seta



A collaborative and dynamic multi-source single-destination navigation algorithm for smart cities

Ziren Xiao [a], Chang Liu [a], Shan Luo [b], Kaizhu Huang [c], Honghao Gao [d], Xiaolong Xu [e], Xinheng Wang [a,*]

[a] School of Advanced Technology, Xi'an Jiaotong-Liverpool University, Suzhou, China
[b] Department of Engineering, King's College London, London, UK
[c] Data Science Research Center, Duke Kunshan University, Kunshan, China
[d] School of Computer Engineering and Science, Shanghai University, Shanghai, China
[e] Jiangsu Key Laboratory of Big Data Security and Intelligence Processing, Nanjing University of Posts and Telecommunications, Nanjing, China

ARTICLE INFO

Keywords:
Ant colony optimisation
Collaborative
Multiple sources path planning

ABSTRACT

In order to enable multiple agents to select the best gathering point dynamically, the design of a collaborative and efficient method in real-time is crucial. The dynamic path planning problem from multiple sources to a single destination (DMS-SD) without prior knowledge about the target is proposed in this paper. The modified Dijkstra's algorithm (Xiao et al., 2022) in the previous work can optimally address the DMS-SD problem, which effectively generates an optimal solution in the small map. However, it requires more than 60s to compute the result in our extensive map test, which is intolerable for real-time navigation users. Therefore, we have proposed a hybrid optimisation method to address the problem more efficiently in this paper. The proposed method integrates the Ant Colony Optimisation (ACO) with Monte Carlo Tree Search (MCTS) and modifies the heuristic function to fit the hybrid algorithm. The pure MCTS algorithm can accelerate the randomised search by only exploring unvisited nodes, instead of generating every possible solution. More importantly, benefiting from limiting the maximum exploring depth, our method can approach an optimal point and generate a sub-optimal solution without any prior training used in other neural network-based methods. Experiment results show that our proposed algorithm demonstrates competitive performance with other existing state-of-the-art methods, such as reinforcement learning-based approaches, without training the neural network model. Our method also provides up to 98% reduction in computation time while obtaining sub-optimal results, comparing with the modified Dijkstra's algorithm.

1. Introduction

The term smart city has been proposed for more than 20 years since 1998 [1]. The goal of smart cities is to provide a safe, efficient and convenient environment for human settlements. Facilities and structures, such as water, transportation and power, are designed with computerised systems, including databases and decision-making algorithms, where the data can be securely stored in the cloud [2,3]. Recently, the user interaction and dynamic route planning [4] has been quite popular in the smart city. Specifically, collaborative actions among users have been considered vital in walking and driving navigation research [5–7].

Motivated by the collaborative computation, we have solved a navigation problem that has yet to be well addressed. Specifically, there is an emergent need for a group of friends to meet at a place that fulfils most of their preferences. Based on real-time situations on the roads, such as congestion or road accidents, the preferred meeting point could be dynamically changed, and new routes to this point need to be planned for individuals. We define this problem as a dynamic Multi Sources to a Single Destination Problem (DMS-SD), which other researchers have not discussed and addressed. We emphasise 'dynamic' in our problem because users may operate real-time software, and the solver should not consume a significant amount of time on locating the target. Consequently, the results of DMS-SD can then be easily applied to the navigation recommendation system in the smart city, which helps users choose their destination. For example, a group of users, as shown in Fig. 1, in a city are planning to have a dinner, where each user may choose their specific preference for the food and area; based on the selected preference, the agent in the smart city can compute an initial goal as their first destination. However, because of different speeds of walking or driving, they may choose an alternative place to meet when

* Corresponding author.
E-mail address: xinheng.wang@xjtlu.edu.cn (X. Wang).

https://doi.org/10.1016/j.seta.2023.103032
Received 1 July 2022; Received in revised form 1 December 2022; Accepted 5 January 2023
Available online 14 January 2023

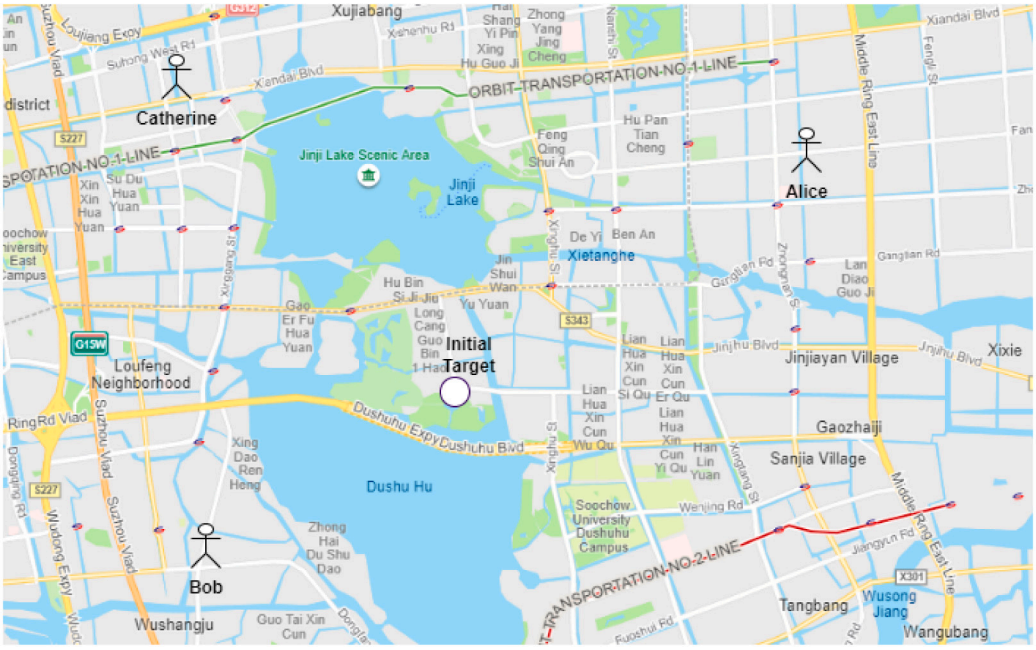


Fig. 1. An example of DMS-SD of three users.

they are on the road. In this case, the dynamic feature is essential. We also intend to reduce the load of the central server of the smart city. This means the server will be only used to allocate tasks or goals, and the computation of those tasks is performed on users' mobile devices, which should not require heavy computational resources. Moreover, this algorithm can be environmentally friendly due to efficient computing. Specifically, the cost of electricity can be saved when one of the main sources of electricity is thermal power generation. Meanwhile, because the driving distance is minimised, there will be less vehicle exhaust emission. As a result, the city contributes to protecting the global environment by reducing greenhouse gas emissions and saving non-renewable resources, which is the critical goal of the efficient smart city.

Traditionally, classical path planning algorithms often focus on choosing the path with the shortest time or the shortest distance from a starting point to a destination point, called a Single Source to a Single Destination Problem (SS-SD). Though this problem has variations, such as from a source point to multiple destinations, the classical algorithms can still be used by splitting the variant problems into multiple SS-SDs. Most solutions can successfully find an optimal or near-optimal path based on some performance indicators. Heuristic-based algorithms are generally exploited to compute the entire path, such as A* [8] and Dijkstra algorithm [9], which can guarantee to find an optimal solution. However, as the search space becomes larger, those algorithms consume much more computational resources because they visit all nodes to find the best answer. For example, Floyd's algorithm takes $O(n_v^3)$ to compute all pairs of distance, where n_v is the number of nodes. To address DMS-SD, Floyd's algorithm needs additional loops to calculate the shortest path in its Adjacent Matrix. When it is applied to a large smart city, the computation time can be even days, which real-time users cannot tolerate. The modified Dijkstra's algorithm [10] solves the DMS-SD problem using the Adjacent Matrix in Floyd's algorithm to store the shortest distance between nodes; therefore, it does not require computing the shortest distance for all pairs as in Floyd's algorithm. Then, based on the calculations and operations of the Adjacent Matrix, we can locate the temporary destination for the current users' locations. This algorithm provides an optimal solution for the current state and does not require pre-training of the model. However, it spends much time on computation when the number of nodes is big, which is intolerable for real-time users. For example, it takes approximately 56 s[1] to compute results in a 100 × 100 size map (10,000 nodes in total). This time can be even longer on a mobile device.

We also argue that many existing state-of-the-art approaches, such as Q-learning-based reinforcement learning algorithms (e.g., Deep Q Network, Policy Proximal Optimisation), are difficult to be directly applied to the DMS-SD. For example, Q-learning algorithms to address the path planning problem were deployed in [11] and [12], where one is used to address the SS-SD, and another is based on multiple users (sources). In our testing with the same parameter settings, such as action space, state space and the reward function, the training agent still assigns random movements to users because users are hard to meet at the same place when the number of users is large. This can be explained by the probability of meeting, $\frac{1}{(n_v)^{n_p}}$, where n_p is the number of users; it is obvious that the probability can be tiny if both n_v and n_p are large. Without the first meeting, the training agent does not know the goal reward and thus cannot update the Q-table (or neural networks) based on the goal, leading to aimless movements. We also test the training agent using Policy Proximal Optimisation (PPO) algorithm in which the reward is the distance between users, encouraging moving towards each other; the result shows the agent does not converge to the optimal — there is still a big gap between the current training result and the optimal solution. We will discuss the details of the results in the evaluation section.

In this paper, a hybrid method, ACO-MCTS is proposed to solve the DMS-SD problem, which integrates Ant Colony Optimisation (ACO) in the Monte Carlo Tree Search (MCTS), and costs less time to compute the result. This integration is different from the ordinary MCTS and ACO in three ways. Firstly, ordinary methods tend to expand all possible nodes to find the solution. However, those algorithms must deal with a large search space in practice. The proposed hybrid method sets a maximum exploring depth to avoid exhausting computing resources. Therefore, the new method is likely to find a local optimal solution within the exploring range instead of looking for the global optimum at the beginning. Secondly, the ACO-MCTS can perform the training process

[1] Using AMD Threadripper 3990X Desktop Processor

along with their movement because of the fast training speed and only record limited information for future usage. While the ordinary MCTS relies on the pre-trained tree which takes a long time to retrain the tree if the current state is significantly different from the existing state in the trained tree. Thirdly, we introduce a new linear distance-based reward function to calculate the score of each node, rather than a single success rate. Thus, users move towards a temporary destination using the node score as their priority.

The main contributions of this paper are listed as follows:

- We propose a hybrid algorithm: ACO-MCTS, to address DMS-SD problem that other researchers have not discussed and addressed. The proposed method can efficiently generate sub-optimal solutions based on the maximum depth without prior training, which shows faster computation speed than exhausting methods, such as the modified Dijkstra's algorithm. This can be particularly useful to users in large cities.
- A linear distance-based reward function is proposed to estimate the score of each visited node. The movement of users are guided by the node score, which tends to move towards a temporary destination. The temporary destination is the optimal solution for the current understanding of the map, which is calculated by the results of simulations.

The structure of the rest of the paper is as follows: firstly, we review related work from past studies; secondly, details of the proposed methods are described and explained; thirdly, we implement the algorithm and extensive experiments are conducted to evaluate the performance of the new methods; finally, a brief conclusion and possible future directions are given.

2. Related work

Existing navigation algorithms are broadly classified into three categories: search-based, learning-based and model-based.

Search-based planning methods construct a feasible path from the source to the destination that focuses on addressing problems based on known surroundings and obstacles on the map. Visibility Graph (VG) [13], Breadth First Search (BFS) [14], Depth First Search (DFS) [15], Greedy Best-First Search (GBFS) [16], Dijkstra's algorithm [9], A* [17], Reduced A* [18] and [19] algorithms are examples of graph traversal algorithms designed to find the optimal path via visiting or updating each vertex in a graph or state space. Specifically, GBFS [16], Dijkstra's [9], A* [17] and Reduced A* [18] search algorithms are heuristic-based search planning that typically use a predefined heuristic function and a classical computing algorithm to address the fundamental planning problems (SS-SD). Other variants of A* algorithm such as Lifelong Planning A* (LPA*) [20], Anytime Repairing A* (ARA*) [21], Real-Time Adaptive A* (RTAA*) [22], Fringe Saving A* (FSA*) [23], Fringe-Saving A* (FSA*) [23] and Generalised Adaptive A* (GAA*) [24] are classified as incremental heuristic search algorithms, that reduce time of computing the optimal path to a series of similar search problems based on reused information from previous searches. D* [25], D* Lite [26] and Anytime Dynamic A* (AD*) [27] algorithms search the map by destination-started. Overall, these search-based methods attempt to address the SS-SD path planning problem with a known goal. We note that exhausting search methods, such as Dijkstra's algorithm [9], can potentially address DMS-SD with mathematical calculations [10]. However, since those methods try to visit every vertex in the graph, the time complexity is based on the number of vertices (nodes), such as $O(n_v + n_e \log(n_v))$ for the standard Dijkstra's algorithm, where n_v and n_e present the number of vertices and the number of edges, respectively. This means that the computing time can be significantly long when the size of the graph increases, which provides a delayed response to real-time users. Other partial search methods, such as variations of A* [18], can be trapped into a local optimal and produce a bad solution because it relies on a fixed heuristic function, which cannot list all environment conditions. Notably, random search methods, such as Ant Colony Optimisation (ACO) [28] and Particle Swarm Optimisation (PSO) [29], are able to address DMS-SD with lower computation time compared with exhausting methods. However, these can be potentially stuck in the local optimal due to 'bad luck'; for example, the random search always selects the worst or sub-optimal choice.

Learning-based approaches learn control from past experience or through simulations. Algorithms such as Rapidly Exploring Random Tree (RRT) [30] and its derivatives including RRT* [31], RRT*-smart [32], RRT*-AR [33], Theta*-RRT [34] and RRT* FND [35] are designed to learn previous experience according to a built space-filling tree that implements random samples from the search space. Other deep reinforcement learning (DRL) based algorithms include RLGWO [36], and actor–critic experience replay (ACER) [37], which can generate with a minimised distance at less time. Some learning-based approaches can solve the static SS-SD problem, such as [38] using Double DQN (DDQN) and dynamic SS-SD problems, such as Real-Time RRT* (RT-RRT*) [39]. There are many multi-agents (multi-sources) path planning methods in existing studies, such as MAPPER [12] and GA3CCADRL [40], which attempt to solve MS-SD problem that the obstacles dynamically move in the environment, but it is different from the concept of DMS-SD we proposed in this work. Other DRL-based methods such as Multi-Agent Deep Deterministic Policy Gradient (MADDPG) [41] address the static MS-SD problem, where the target is fixed. Furthermore, learning-based approaches usually require a large amount of experience or data to train the model well, especially for DRL methods, which are time-consuming. More importantly, existing state-of-the-art DRL approaches cannot offer a good choice of the reward function that can best fit our goal. Existing reward functions can be static [12] or function-based [42]. Users are difficult to meet at the same point because of randomly moving using static reward function, whereas existing function-based rewards cannot converge to optimal in our problem. This can be the main reason why those methods fail to address our DMS-SD problem.

Model-based methods need a defined model for the problem, such as actions, initial state, goal state and sensors, then a solver computes the controller automatically. For example, a probabilistic model-based method [43] is proposed to design path planners based on transition probabilistic matrices. Other methods such as Partially Observable Markov Decision Processes (POMDPs) and Hermite interpolation are applied in [44] and [45], respectively, to plan paths in dynamic multi-obstacle environments. Although this work focuses on solving SS-SD under a dynamic environment, model learning is difficult and computationally expensive since an iterative learning process is always required for model-based methods.

In summary, there are a variety of multi-source path planning algorithms in past studies. Notably, existing studies focus on three main path planning areas: (i) users from multiple sources move to a fixed destination with fixed obstacles; (ii) users move to a fixed destination and try to pass dynamically moving obstacles; (iii) users move to multiple destinations. However, in our problem, the destination can be dynamically changed based on the current state of users, which requires high computation speed to provide a fast response to users. Those existing algorithms are difficult to be directly applied to solve our problem with high efficiency because of (i) time-consuming, (ii) not converging to optimal, and (iii) no suitable reward function for the DMS-SD problem.

3. Methodology

Assume there are n_p users locating at different positions (x_1, y_1), (x_2, y_2).. (x_{n_p}, y_{n_p}). Our goal is to locate a target point where all agents will arrive there with a similar time and travelling distance. Additionally, the target point can be updated dynamically and periodically while users are moving, because users may behave non-ideally such as moving slowly or stopping at a point for a while. We also assume that the environment is known, including successors of a node, score of a node, the goal state, and coordinate positions of other agents. We introduce a hybrid method, namely ACO-MCTS, in our work to address DMS-SD.

3.1. ACO

The ordinary ACO was able to solve DMS-SD problems with modifications. Thus, we apply several modifications to fit DMS-SD. When each object moves to a new node, it sends out n_{ant} ants to find the target point, where n_{ant} is fixed and customisable. At some point c_t, n_o or more ants from different agents stick together and this can be one of the best possible solutions. The path distance d_{path} of each ant at the point c_t is calculated and summed up as $d_{shortest}$ (choosing the shortest path if there are multiple ants from the same object), where d_{path} is the sum of path distances from the start point to c_t. After that, $d_{shortest}$ is compared with the best solution and overrides the best one if the new one is better. Other configurations are the same as in the ordinary ACO [28], we list two critical equations here for using in the next method. The ant moves based on the probability of successors and this is calculated by

$$p_a^s = \frac{\tau_a^\alpha \times \eta_a^\beta}{\sum_{z \in allowed\_actions}(\tau_z^\alpha \times \eta_z^\beta)} \tag{1}$$

where the probability p_a^s of the next action a at the current state s equals to the product of the pheromone τ to the action a and the priori knowledge η of applying action a to s over the sum of those. Constants α and β are parameters to control the influence of τ and η, respectively. Typically, the ACO uses the reciprocal of the distance from s to the state applied a in η and the pheromone τ decays with time. The equation therefore can be expressed as

$$\tau_a = \rho\tau_a + \sum_k^m \frac{Q}{d_{total}^k} \tag{2}$$

where ρ is the decaying coefficient of previous τ, m presents the total number of ants, Q is a multiplier of the new pheromone, and d_{total}^k is the distance moved by the ant k from its source point to the destination. The initial value of the pheromone is zero and accumulated by the reciprocal of d_{total}. Overall, the pseudo code of moving to the next vertex is shown in Algorithm 2.

Algorithm 1 Probability Based Selection

```
a ← sort(array_of_p_a)
for each item i in a do
  p ← a random value between 0 and 1
  if p ≤ i then
    i is selected
    return i
  end if
end for
return the last item in a
```

Additionally, we limit the maximum moving steps of each ant. Specifically, the ant tends to move towards to a node that has not been visited before, and the maximum distance should not exceed half of the map size, assuming the distance between each neighbour node is 1.

3.2. ACO-MCTS

The goal of our hybrid method is to improve the computing speed of the ACO algorithm, though accuracy may be slightly sacrificed in some cases. MCTS takes long time to learn, which violates our goal — fast response. Therefore, we discard partial features from ordinary MCTS to speed up learning. In specific, we apply three main steps in the ACO-MCTS algorithm. In the first step, we perform Pre-Actions to calculate some constants used in the later simulation. Secondly, the simulation is performed in a given times n_s and a maximum exploring depth max_d. During this step, node scores are updated. Finally, the object moves to the next state based on nodes around it. Fig. 2 shows the overall procedure of ACO-MCTS algorithm. Details of this process are described in following subsections.

Algorithm 2 ACO

```
bestPath ← null
for each simulation do
  g ← simulation group of ants
  for each ant i in g do
    next ← choose a successor from i based on pheromone using
    Algorithm 1
    i move to next
    update pheromone to next
    next is added to i's path
  end for
  decay pheromone using Eq. (2)
  g′ ← ants from different users at a node
  for each g′ do
    if size(g′) ≥ size(users) then
      sum ← sum length of g′ path
      if size(bestPath) < sum then
        bestPath ← g′ path
      end if
      stop current simulation
    end if
  end for
end for
apply path to corresponding users
```

3.2.1. Centroid

We estimate a destination that fulfils all users in this stage. Therefore, users tend to move towards this point. The most outer nodes can form vertices of polygon and the centre of gravity c_g can be expressed as

$$\begin{aligned} x_{pg} &= \frac{\sum_{i \in vertices}^{n_{outer\_node}} x_i}{n_{outer\_node}} \\ y_{pg} &= \frac{\sum_{i \in vertices}^{n_{outer\_node}} y_i}{n_{outer\_node}} \end{aligned} \tag{3}$$

where x_{pg} and y_{pg} are x and y coordinates of the centre of gravity point and n_{outer_node} is the number of vertices to form the polygon. In most scenarios, there may not be a node just at (x_{pg}, y_{pg}), we then select the closest node as the c_g. c_g will be used in calculate the node score in the later stage.

3.2.2. Play-out

Similar to the ordinary MCTS, the play-out stage creates a virtual environment to perform simulations in a specific times n_s and selects the best action after all n_s simulations are performed. The next action selection is based on the node score:

$$a_{next} = action(\max(ns_0, ns_1, \ldots, ns_n)) \tag{4}$$

where a_{next} is the next action, ns_i are node scores from valid successors from the current state and $action$ is a function finding the corresponding action using a node score value. The algorithm will keep choosing the next action until all agents are at the same place. Each play-out contains four stages: Selection, Expansion, Simulation, and Back-Propagation.

Selection. In the selection stage, all next possible actions are selected and compared. Specifically, we successively select valid actions in the action space and apply the remaining stages on it. In the final stage, there will be a comparison procedure based on generated scores to assign the success state. This is different from the ordinary MCTS: (1). the action space is small, whereas in other games, such as The Game of Go, the successor space can be huge and therefore time and space will be largely occupied. (2). the criteria of success are limited. The ordinary method accumulates success times only when the selected action wins the simulation.

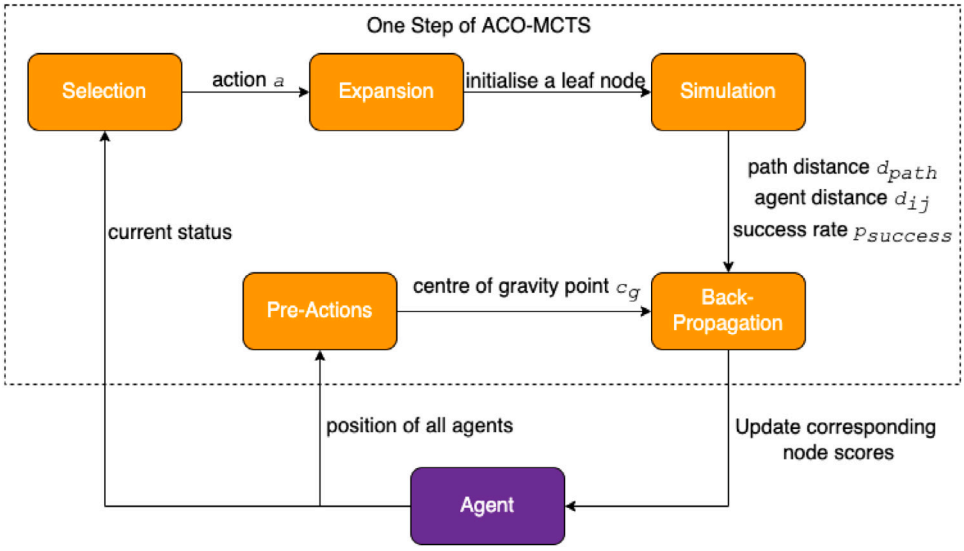


Fig. 2. Overall Process of ACO-MCTS.

Expansion. In this stage, the algorithm expands a leaf node of the selected node in Selection stage. Similar to the ordinary MCTS, the expansion action will initialise the node, which assigns the node score to zero. Instead of performing a brand new simulation in the ACO, the simulation starts from the leaf node, which is able to explore more unvisited states and avoids duplicated explorations.

Simulation. This is the key stage integrating features from ACO and MCTS, also called sampling, which is different from the simulation concept of play-out and only a part of the entire play-out simulation. We achieve this integration in the following steps. Firstly, we introduce a maximum exploring depth to limit the computational resource consuming. While the current exploring depth is less than the maximum explore threshold d_{max}, the algorithm moves to the Back-Propagation stage. Otherwise, it goes to the next step. We apply this feature because in practice, we are not likely to explore paths to the gathering point in each simulation if the search space is huge; instead, the algorithm can update and find the optimal solution within the maximum depth and leads to locate the DMS-SD solution based on it. More importantly, the computational speed is much faster, as the ACO method needs more simulations, though this sacrifices some accuracy. In the second step, all agents move one step like ACO algorithm. That is, each object selects the best next action based on the probability transformed from the node score, as similar to Eq. (1), but η_a is defined as the node score in this algorithm. τ for an ant a is calculated by the total distance travelled by the ant to the destination or the deepest point, and this can be expressed as in Eq. (2) instead of the reciprocal of the travelling distance. It should be noted that the action with the higher probability will have higher priority to be selected, but this does not guarantee selection and lower chance actions are also able to be chosen with lower priority. The specific selection algorithm is shown in Algorithm 1. During the second step, if all agents are at the same place, the current stage will be terminated and move to the next stage. Algorithm 3 presents the overall procedure of this stage.

Back-propagation. This stage updates the node score after each simulation. The node score acts as the heuristic function in the entire algorithm and navigates each object to move to the best direction. This score consists of three factors: (i) the distance travelled from the current node to the best solution point, (ii) the sum distance from the current object to others, (iii) the distance to the centre point of gravity and (iv) the simulation success rate. For each node i, the node score ns_i can be summarised as:

$$ns_i = \frac{\iota}{d_{path}} + \frac{\delta}{\sum_{j \in agents} d_{ij}} + \frac{\omega}{d_{ig}} + \gamma \times p_{success} \tag{5}$$

where ι, δ, ω and γ are constants controlling the influence rate of corresponding variables, d_{path} is the distance travelled to the solution point if there is any, d_{ij} is the travelling distance transmitting from node i to node j, d_{ig} is the distance from node i to the centre point of gravity c_g computed in the Pre-Actions stage, and $p_{success}$ is calculated based on the number of simulations:

$$p_{success} = \frac{success\_times}{total\_times} \tag{6}$$

which is the same item as in the ordinary MCTS. d_{path} is updated only if a target has been located, which is achievable by all agents and with a smaller sum of their path distance. Specifically, we maintain the best solution, but we only use its distance to nodes that ants visited as a term in the node score function. Because we selected all possible actions in the Selection stage, we are able to evaluate simulation results using the node score of the leaf node and mark the best one as the success simulation, whereas others will be marked as failed. The algorithm will update all nodes where the node has been visited in the simulation and override the old node score if the new score is better in the latest simulation. However, the movement of each object in the simulation will not be recorded and applied to those real agents. Additionally, the agent marks the node and the path to the node as 'dead end' nodes if there is only one way out of the node, which can escape from those dead-end roads. After the update is completed, the current MCTS simulation is ended and the next new simulation will be started again from the Selection stage.

Algorithm 3 Simulation Stage

```
d ← 0
while d < d_max do
  if users meet at the same place then
    move to Back-Propagation
  end if
  all users move one step
  d ← d + 1
end while
move to Back-Propagation
```

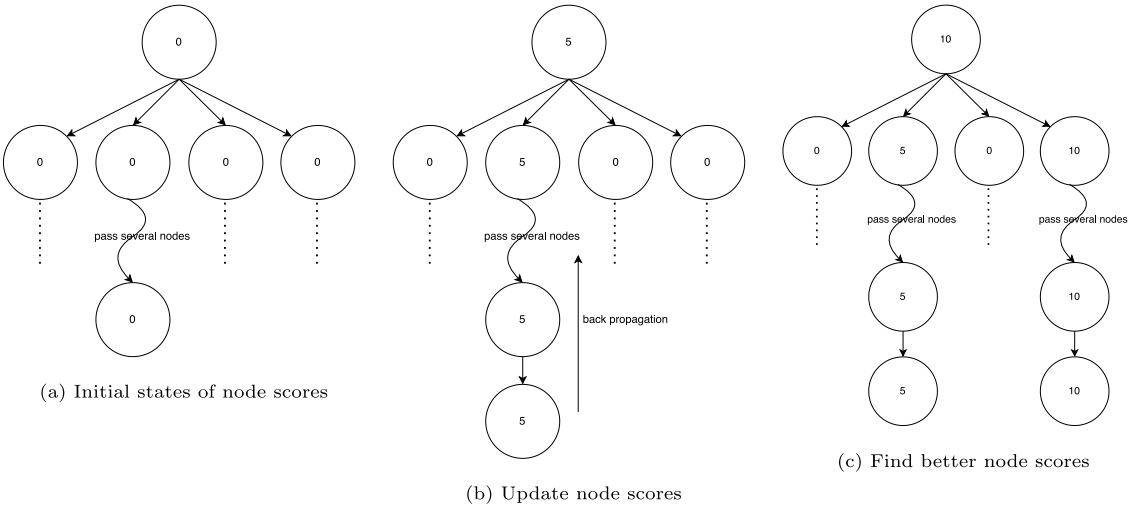


Fig. 3. Example of updating node scores.

We keep the same idea as in ACO algorithm that ants initially move randomly and explore the shorter path if a path has been found. Specifically, the agent randomly explores the nodes at the initial stage and expand the tree, because users do not yet meet at the same location (node) and not reach c_g; as shown in Fig. 3(a), all node scores are 0 and ants therefore select random movements. Once one of goals is achieved (either meeting at the same point or reach c_g), the leaf node score is not 0 anymore and node scores of its parents are updated, as shown in Fig. 3(b). The node score acts like a path and navigates ants or users to the destination, where the navigation result is based on the current best choice. For example, in Fig. 3(b), all nodes in the second path shows 5 score and all other node scores are zero, thus nodes in this path have the highest probability to be selected. Since there are many other ants exploring different path in different directions, the new node score from another leaf may be better than the current choice e.g., shorter distance to travel or closer to other users; in this case, the node score is updated again and overwrite the old scores if the current score is better, like Fig. 3(c). Because of the expanding strategy of the MCTS, ACO-MCTS can explore more states and therefore can reach further nodes. For example, in 1000 simulations and 20 exploring depth, a user can explore at most 20,000 further from the current node, if the agent always tries to expand the new child in the most depth direction.

3.2.3. Complexity analysis

The time complexity of ACO during each move can be expressed as

$$O(n_p n_s n_{ant} n_v) \tag{7}$$

where n_p is the number of users, n_s is the times of virtual simulations that will be performed by ants, n_{ant} is the number of ants sent out from each object, and n_v is the number of vertices. The simulation time n_s is recommended to be greater than $(\frac{n_v}{n_{ant}})^{n_o}$ or otherwise c_t is difficult to be located during the simulation, since the probability of locating the target point c_t is

$$p_{c_t} = (n_{ant})^{n_p}(n_v)^{n_p} \tag{8}$$

Therefore, as the number of agents grows, the time complexity will be increased significantly.

In another method ACO-MCTS, during each play-out action, the algorithm is expected to consume time with the complexity

$$O(n_p n_s n_{ant} d_{max}) \tag{9}$$

where n_o is number of agents, n_s is number of play-out or simulations, and d_{max} is the maximum exploring depth. Apparently, the expected time consumed is less than the ACO, because n_{action} and d_{max} are generally small. We will analyse this specifically in the next section. (see Table 1).

4. Implementation & evaluation

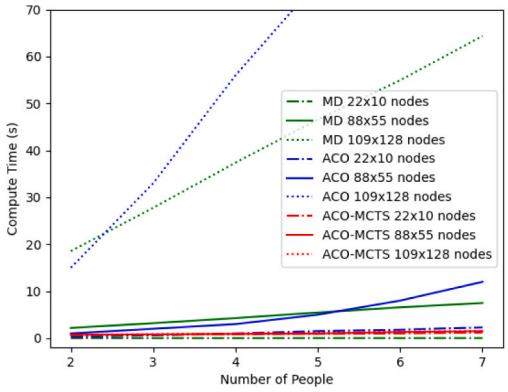
In this section, we mainly evaluate and compare the performance of the modified Dijkstra's algorithm [10], ACO and ACO-MCTS. We also implement and modify state-of-the-art deep reinforcement learning (DRL) approaches from literature, such as Q-learning and DQN, to show their disadvantages and challenges in addressing DMS-SD.

We use OpenAI Gym to build the simulation environment. More specifically, the environment simulates a grid world map containing accessible routes and inaccessible walls. At each time step t, the training agent sends the next locations of users (action a_t) to the environment and receives a new map (state s_{t+1}) after applying the action, involving all information in the map, such as users' new locations. The reward r_{t+1} is used to evaluate the new action passed to the environment; for the modified Dijkstra's algorithm, ACO and ACO-MCTS, the environment does not send the reward back to the agent because these algorithms do not use the reward to calculate the next step. When all users meet at the same node, the environment also returns a $done = True$ signal, indicating that the current $trajectory = \{(s_0), (a_0, s_1, r_1), (a_1, s_2, r_2)...(a_{n-1}, s_n, r_n)\}$ is ended, then the environment is reset to the original status and the initial positions of users are randomised. In training DRL algorithms [12,42], the environment involves two more functionalities: (i) the corresponding rewards in the paper are returned to the agent to evaluate the sent action; (ii) for each episode (= 400-time steps), the partial $trajectory$ is also sent to the agent to perform training.

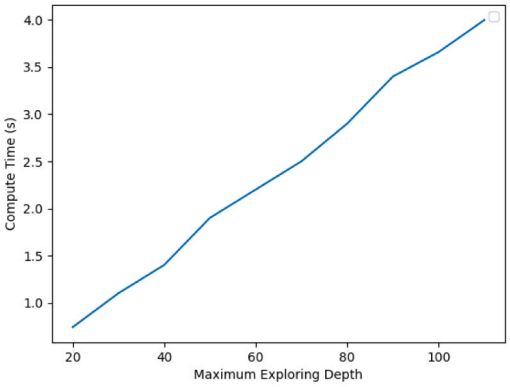
Computational complexity. We have discussed the time complexity in Eqs. (8) and (9). Regarding Big-O notation, the ACO costs more time to calculate the next possible movement because it potentially visits all vertices and tends to compute the whole path to the destination. In another word, n_s should be greater than $(\frac{n_v}{n_{ant}})^{n_o}$ as discussed before, which could be significantly large if either the search space or agents are big. Typically in practice, the search space can be a map of an area that includes millions of nodes, and this could largely affect the performance of ACO. Whilst the number of simulations n_s in ACO-MCTS is independent of n_o and n_{action}, it is only related to the accuracy of the target. Moreover, terms n_{action} and d are usually small numbers comparing to n_v. From the perspective of space complexity, ACO-MCTS also spends less memory to store the information, according to the

Table 1
Constants in ACO-MCTS and ACO Algorithm.

Constant	Description	Value	Range
α	Control the influence of τ	1	$0 \sim 1$
β	Control the influence of η	0.9	$0 \sim 1$
ρ	The decaying coefficient of previous τ	0.9	$0 \sim 1$
n_s	The number of simulations	1000	any
d_{max}	The maximum exploring depth	20	any
ι	Control the influence of the distance travelled to the solution	10	any
δ	Control the influence of the distance transmitting from i to j	2	any
ω	Control the influence of the distance from i to c_p	1	any
γ	Control the influence of the success rate	1	any



(a) Computation Time v.s. Map Size



(b) Computation Time v.s. Exploring Depth in a 109x120 nodes map using ACO-MCTS

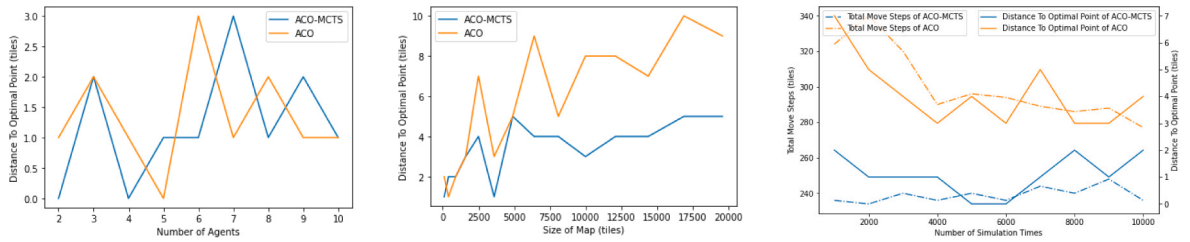
Fig. 4. Test of computation time. The lower computation time means a shorter time consumed to calculate the result.

table. Therefore, in theory, we expect ACO-MCTS to compute much faster than ACO. While the modified Dijkstra's algorithm exhausts all possibilities of paths to achieve the optimal, it spends much more time on the computation, compared with ACO-MCTS, which is highly depended on the number of nodes: $O(n_v + n_e \log(n_v))$. In our environment, n_e is approximate n_v because each node uses up to 4 edges (up, down, left, right) to connect to its adjacent nodes. When the number of nodes becomes large, the actual computation time can even be days.

Average computation time. Fig. 4 shows the average computation time of the modified Dijkstra's algorithm, ACO and ACO-MCTS to reach the next node. Results are tested over three different size map environments, showcasing the model's sensitivity to changes in the number of agents and the size of the map. In Fig. 4(a), it is evident that the average computation time is proportional to the number of users in all three algorithms. We observe that the modified Dijkstra's algorithm only spends less than 5 s on smaller maps, which is acceptable by real-time software users. However, it costs more than 60 s to compute the result in the 109 × 128 nodes with seven users, though the result is optimal. Additionally, the time of ACO to reach the next node consumes more than 80 times than in ACO-MCTS when the number of users increases; the gap between the two algorithms is significantly large when the size of the map increases, whereas the time costs of ACO-MCTS in all three environments are overlapped as a single line. This situation is because we limit the maximum search depth of ACO-MCTS; thus, the time cost is mainly related to the number of simulations instead of the size of the map. Since the increase in the number of agents is not significant as the map size does, the time taken by ACO-MCTS is slightly affected. However, the increase in ACO's performance is not linear overall. Although we set limitations in the ACO to avoid exploring the repeat path, the average computation time depends on both parameters.

Distance to the optimal. The modified Dijkstra's algorithm computes an optimal result with the shortest and equivalent travelling distance to all users because our calculations are based on the Adjacent Matrix, which stores the shortest (optimal) distance from a user to all other nodes. We use the modified Dijkstra's algorithm as the baseline (distance to optimal = 0) to evaluate the other two methods. ACO and ACO-MCTS do not guarantee finding either the optimal point or the optimal distance. In experiments, both algorithms are close to the optimal point, the distance is acceptable ($\leq$ 10 tiles), and the results are shown in Fig. 5. Nevertheless, ACO-MCTS is closer to the optimal point while the map size increases, though both methods share a similar performance in changing the number of agents. Because of the expansion process, ACO-MCTS can access more nodes precisely based on previous results, whereas ACO explores a fresh new path from the beginning. Therefore, ACO-MCTS typically costs less the number of simulations than ACO. In this case, adjusting the ACO's number of simulations is necessary, and we will discuss this in the next paragraph.

The number of simulations. The number of simulations n_s is one of the critical factors affecting the result of ACO-based algorithms. The total move steps and the accuracy can be improved by adjusting the number of simulations. We set up an experiment by changing n_s under a fixed map size 50 × 50 with 4 agents. Fig. 5(c) shows the relationship between the number of simulations and the incremental move steps of two algorithms. For ACO, a more considerable value of n_s can help the algorithm find a better path to approach the optimal solution. Therefore, it needs more computations to avoid unnecessary movements and reach optimal. However, increasing n_s will cost more computation time. By contrast, $n_s = 1000$ is sufficient for ACO-MCTS in those tests due to the expansion process, and the increase in the number of simulations does not contribute to significant improvements.



(a) Distance to optimal point with change in number of users

(b) Distance to optimal point with change in size of map

(c) Distance to optimal point and total steps taken with change in number of simulations

Fig. 5. Tests over the maze environment, showing the performance in terms of the distance to the optimal point and the number of simulations by changing the number of agents and size of the map. In all graphs, lower is better.

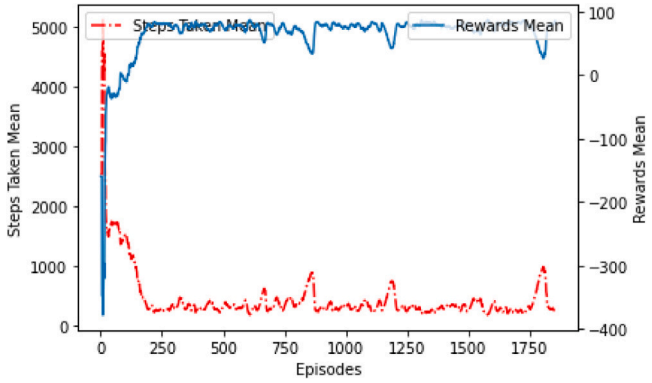


Fig. 6. DRL training statistics of [42] in solving DMS-SD for 22 × 10 nodes.

Choice of maximum exploring depth d_{max}. The purpose of d_{max} is to limit the exploring and the simulation part acts like pure ACO when $d_{max} \to \infty$. Thus, the simulation is able to be early terminated without achieving the goal. Overall, d_{max} can be a trade-off between the performance and the computation speed. Specifically, this feature can potentially limit the performance in some special cases. For example, when the agent faces a complicated environment (e.g., needs 40 steps to bypass the wall) and is only with a small exploring depth (e.g., 20), it is hard to escape from the current state and can be potentially stuck in the local optimal. Therefore, we usually select a small value for a familiar environment (e.g., in city navigation), but it could be larger if the environment is complex, such as a maze. However, increasing d_{max} consumes more computational resources and requires longer computation time, as shown in Fig. 4(b).

DRL convergence problems. [12] are two approaches to solve the multiple users navigation problem in a dynamic environment. [12] uses fixed rewards; for example, the training agent receives a −5 reward on collisions and a −0.1 reward on movements, whereas [42] deploys dynamic reward, such as calculating the distance from the user to the target. We changed the goal of those two methods to fit our problem, where the new goal is defined as users meeting at the same node. In our tests, [12] is difficult to find the destination when the number of users is big because the reward does not navigate the agent to the meeting point. Specifically, the probability of users' meeting at the same point is $\frac{1}{(n_v)^{n_p}}$, which increases exponentially when n_p increases. This means users can only meet if they are 'lucky' enough. Consequently, the training agent cannot learn the reward of achieving the goal because it rarely reaches the meeting point. Nevertheless, the agent can learn to approach the optimal in our tests when only two users are on the small map. In this case, the modified Dijkstra's algorithm is even faster as it does not require training. Another method [42] does not converge to the optimal solution if the environment is too complicated. For example, our environment involves 'walls' to simulate buildings in the real world, which are inaccessible to users. The agent fails to learn the process when the user has to travel a long distance to bypass the wall. This is because the reward is calculated by straight distance instead of the actual path length to the destination and does not consider the environment, which can be a complicated maze. However, apart from the two disadvantages above, the final training result does not converge to the optimal in both algorithms. They both generate sub-optimal solutions that can reach the goal but are still far away from the optimal (e.g., taking more steps to reach the goal). For example, as shown in Fig. 6, the average step taken for an episode, the Steps Taken Mean (STM), of [42] in solving DMS-SD decreases to around 200 steps as more training is performed, where each step means users move once. It shows that the DRL agent is trying to learn the optimal policy, and the agent reaches its best solution at around 500 episodes. However, the optimal STM should be approximately ten steps, indicating that there is still a significant gap between the training results and the optimal solution.

5. Conclusion

In this paper, we propose an emerging collaborative and dynamic path planning problem: DMS-SD, which can be useful in the smart city navigation. We then propose our method in this work, which sacrifices the accuracy to achieve better computation speed. Different from the ordinary MCTS, a maximum exploring depth is applied preventing over-exploration of the map. In the sampling stage, we have used ACO as the randomisation strategy to navigate movement and avoid completely random. We have also introduced a new heuristic function in calculating the node score, instead of a single success rate term. ACO-MCTS may not be accurate enough if the training environment is complicated and the maximum exploring depth sets to a small value. In this case, a higher depth value should be considered. However, increasing the depth leads to slow down the computation, which may delay the response to users. As a result, the emission of CO_2 and the congestion can be potentially reduced using our efficient navigation method.

However, our method still costs more computational resources than a fixed model, such as DRL based methods, because it does not record any computed results. In the evaluation, we show the difficulties and challenges of two state-of-the-art DRL methods to be directly deployed on solving DMS-SD. For future directions, we believe that the bad performance in our evaluation is caused by an inappropriate reward function. Therefore, a proper reward function and simulation environment settings that perfectly matches our goal for DRL algorithms can be carefully considered in the future work.

CRediT authorship contribution statement

Ziren Xiao: Conceptualization, Methodology, Software. **Chang Liu:** Writing – original draft, Visualization Data curation. **Shan Luo:** Supervision, Investigation. **Kaizhu Huang:** Reviewing, Editing. **Honghao Gao:** Resources, Reviewing, Editing. **Xiaolong Xu:** Investigation, Reviewing, Editing. **Xinheng Wang:** Supervision, Writing – review & editing, Project administration.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

This research was funded by: the National Natural Science Foundation of China (NSFC) under grant 52175030; Jiangsu Science and Technology Programme under No. BE2020006-4; Key Program Special Fund in XJTLU under project KSF-E-64 and KSF-T-06; XJTLU Research Development Funding under projects RDF-19-01-14 and RDF-20-01-15.

References

[1] Van Bastelaer B. Digital cities and transferability of results. In: 4th EDC Conference on Digital Cities. Salzburg; 1998, p. 61–70.
[2] Seth B, Dalal S, Jaglan V, Le D-N, Mohan S, Srivastava G. Integrating encryption techniques for secure data storage in the cloud. Trans Emerg Telecommun Technol 2022;33(4):e4108.
[3] Fang K, Wang T, Yuan X, Miao C, Pan Y, Li J. Detection of weak electromagnetic interference attacks based on fingerprint in IIoT systems. Future Gener Comput Syst 2022;126:295–304. http://dx.doi.org/10.1016/j.future.2021.08.020, URL https://www.sciencedirect.com/science/article/pii/S0167739X21003289.
[4] Li P, Wang X, Gao H, Xu X, Iqbal M, Dahal K. A dynamic and scalable user-centric route planning algorithm based on polychromatic sets theory. IEEE Trans Intell Transp Syst 2022;23(3):2762–72. http://dx.doi.org/10.1109/TITS.2021.3085026.
[5] Wang B, Chen C, Zhang T. Commercial vehicle road collaborative system based on 5G-V2X and satellite navigation technologies. In: China satellite navigation conference proceedings. Springer; 2021, p. 274–82.
[6] Zhou Z, Lyu P, Lai J, Zhu X, Huang K. Multi-vehicle collaborative navigation method based on datalink relative distance in GNSS denied environment. In: Advances in guidance, navigation and control. Springer; 2022, p. 3875–88.
[7] Liu H, Hyodo A, Suzuki S. Reinforcement learning based indoor, collaborative autonomous mobility. In: The 6th international conference on control engineering and artificial intelligence. 2022, p. 53–6.
[8] Nosrati M, Karimi R, Hasanvand HA. Investigation of the*(star) search algorithms: Characteristics, methods and approaches. World Appl Program 2012;2(4):251–6.
[9] Dijkstra EW, et al. A note on two problems in connexion with graphs. Numer Math 1959;1(1):269–71.
[10] Xiao Z, Xiao R, Liu C, Luo S, Wang X. An efficient dynamic multi-sources to single-destination (DMS-SD) algorithm in smart city navigation using adjacent matrix. In: 2022 international conference on human-centered cognitive systems. HCCS, IEEE; 2022.
[11] Abdi A, Adhikari D, Park JH. A novel hybrid path planning method based on q-learning and neural network for robot arm. Appl Sci 2021;11(15):6770.
[12] Liu Z, Chen B, Zhou H, Koushik G, Hebert M, Zhao D. Mapper: Multi-agent path planning with evolutionary reinforcement learning in mixed dynamic environments. In: 2020 IEEE/RSJ international conference on intelligent robots and systems. IROS, IEEE; 2020, p. 11748–54.
[13] Lozano-Pérez T, Wesley MA. An algorithm for planning collision-free paths among polyhedral obstacles. Commun ACM 1979;22(10):560–70.
[14] Lee CY. An algorithm for path connections and its applications. IRE Trans Electron Comput 1961;(3):346–65.
[15] Even S. Graph algorithms. Cambridge University Press; 2011.
[16] Korf RE. Artificial intelligence search algorithms. Citeseer; 1999.
[17] Delling D, Sanders P, Schultes D, Wagner D. Engineering route planning algorithms. In: Algorithmics of large and complex networks. Springer; 2009, p. 117–39.
[18] Fareh R, Baziyad M, Rahman MH, Rabie T, Bettayeb M. Investigating reduced path planning strategy for differential wheeled mobile robot. Robotica 2020;38(2):235–55.
[19] Dang T, Mascarich F, Khattak S, Papachristos C, Alexis K. Graph-based path planning for autonomous robotic exploration in subterranean environments. In: 2019 IEEE/RSJ international conference on intelligent robots and systems. IROS, IEEE; 2019, p. 3105–12.
[20] Koenig S, Likhachev M, Furcy D. Lifelong planning A*. Artificial Intelligence 2004;155(1–2):93–146.
[21] Likhachev M, Gordon GJ, Thrun S. ARA*: Anytime A* with provable bounds on sub-optimality. Adv Neural Inf Process Syst 2003;16.
[22] Koenig S, Likhachev M. Real-time adaptive a. In: Proceedings of the fifth international joint conference on autonomous agents and multiagent systems. 2006, p. 281–8.
[23] Sun X, Koenig S, et al. The fringe-saving A* search algorithm-A feasibility study. In: IJCAI, vol. 7, 2007, p. 2391–7.
[24] Sun X, Koenig S, Yeoh W. Generalized adaptive a. In: Proceedings of the 7th international joint conference on autonomous agents and multiagent systems-volume 1. 2008, p. 469–76.
[25] Stentz A. Optimal and efficient path planning for partially known environments. In: Intelligent unmanned ground vehicles. Springer; 1997, p. 203–20.
[26] Koenig S, Likhachev M. D^* lite. Aaai/Iaai 2002;15:476–83.
[27] Likhachev M, Ferguson DI, Gordon GJ, Stentz A, Thrun S. Anytime dynamic A*: An anytime, replanning algorithm.. In: ICAPS, vol. 5, 2005, p. 262–71.
[28] Dorigo M, Di Caro G. Ant colony optimization: A new meta-heuristic. In: Proceedings of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406), 2, IEEE; 1999, p. 1470–7.
[29] Poli R, Kennedy J, Blackwell T. Particle swarm optimization. Swarm Intell 2007;1(1):33–57.
[30] LaValle SM, et al. Rapidly-exploring random trees: A new tool for path planning. IA, USA: Ames; 1998.
[31] Karaman S, Frazzoli E. Sampling-based algorithms for optimal motion planning. Int J Robot Res 2011;30(7):846–94.
[32] Islam F, Nasir J, Malik U, Ayaz Y, Hasan O. RRT*-smart: Rapid convergence implementation of RRT* towards optimal solution. In: 2012 IEEE international conference on mechatronics and automation. IEEE; 2012, p. 1651–6.
[33] Choudhury S, Scherer S, Singh S. RRT*-AR: Sampling-based alternate routes planning with applications to autonomous emergency landing of a helicopter. In: 2013 IEEE international conference on robotics and automation. IEEE; 2013, p. 3947–52.
[34] Palmieri L, Koenig S, Arras KO. RRT-based nonholonomic motion planning using any-angle path biasing. In: 2016 IEEE international conference on robotics and automation. ICRA, IEEE; 2016, p. 2775–81.
[35] Adiyatov O, Varol HA. A novel RRT*-based algorithm for motion planning in dynamic environments. In: 2017 IEEE international conference on mechatronics and automation. ICMA, IEEE; 2017, p. 1416–21.
[36] Qu C, Gai W, Zhong M, Zhang J. A novel reinforcement learning based grey wolf optimizer algorithm for unmanned aerial vehicles (UAVs) path planning. Appl Soft Comput 2020;89:106099.
[37] Apuroop KGS, Le AV, Elara MR, Sheu BJ. Reinforcement learning-based complete area coverage path planning for a modified hTrihex robot. Sensors 2021;21(4):1067.
[38] Wang Y, Fang Y, Lou P, Yan J, Liu N. Deep reinforcement learning based path planning for mobile robot in unknown environment. 1576, (1):IOP Publishing; 2020, 012009,
[39] Naderi K, Rajamäki J, Hämäläinen P. RT-RRT* a real-time path planning algorithm based on RRT. In: Proceedings of the 8th ACM SIGGRAPH Conference on Motion in Games. 2015, p. 113–8.
[40] Semnani SH, Liu H, Everett M, De Ruiter A, How JP. Multi-agent motion planning for dense and dynamic environments via deep reinforcement learning. IEEE Robot Autom Lett 2020;5(2):3221–6.
[41] Shang Z, Mao Z, Zhang H, Xu M. Collaborative path planning of multiple carrier-based aircraft based on multi-agent reinforcement learning. In: 2022 23rd IEEE international conference on mobile data management. MDM, IEEE; 2022, p. 512–7.
[42] Qie H, Shi D, Shen T, Xu X, Li Y, Wang L. Joint optimization of multi-UAV target assignment and path planning based on multi-agent reinforcement learning. IEEE Access 2019;7:146264–72.
[43] Gong W. Probabilistic model based path planning. Physica A 2021;568:125718.
[44] Ragi S, Chong EKP. UAV path planning in a dynamic environment via partially observable Markov decision process. IEEE Trans Aerosp Electron Syst 2013;49(4):2397–412.
[45] Li C, Wang J, Wang X, Zhang Y. A model based path planning algorithm for self-driving cars in dynamic environment. In: 2015 Chinese Automation Congress. CAC, IEEE; 2015, p. 1123–8.

Ziren Xiao is currently a Ph.D. candidate at Xi'an Jiaotong-Liverpool University. He received a B.Sc degree in Computing and Software Systems and an M.Sc in Computer Science degree from The University of Melbourne. His past work focused on outlier detection using machine learning algorithms, and he is currently working on developing reinforcement learning approaches for transportation systems.

Chang Liu received the B.Sc. degree in Applied Mathematics from Xi'an Jiaotong-Liverpool University in 2021. She is currently an M.Sc candidate in Applied Computational Science and Engineering at Imperial College, London. Her past work focused on statistics and mathematical analysis, and she is now working on numerical analysis and machine learning.

Shan Luo Here is hte biography of Shan Luo. I am currently a Senior Lecturer (Associate Professor) in Robotics at the Department of Engineering, King's College London. I am an enthusiastic and dedicated researcher doing research on machine learning and computer vision algorithms with applications in robotics, e.g., robot perception, object recognition & localisation, and multimodal sensing fusion. I was a Lecturer (Assistant Professor) at the Department of Computer Science, the University of Liverpool, from 2018 to 2021, and was Director for the smARTLab at the Department. Prior to that, I was a Postdoctoral Research Fellow at Harvard University from 2017 to 2018, and at the University of Leeds from 2016 to 2017. I received the PhD degree in Robotics from King's College London, in 2016. I was a Visiting Scientist at the Computer Science and Artificial Intelligence Laboratory (CSAIL), MIT, in 2016. I am a Fellow of the Higher Education Academy (FHEA).

Kaizhu Huang Kaizhu Huang is now Full Professor of ECE at Data Science Research Center, Duke Kunshan University. Prof. Huang obtained his PhD degree from Chinese University of Hong Kong (CUHK) in 2004. He has published more than 200 international referred papers including 100+ international journals. His research interests include adversarial learning, computer vision, and pattern recognition.

Honghao Gao Honghao Gao (SM'18) is currently with the School of Computer Engineering and Science, Shanghai University, China. He is also a Professor at the College of Future Industry, Gachon University, Korea. Prior to that, he was a Research Fellow with the Software Engineering Information Technology Institute at Central Michigan University, USA, and was an Adjunct Professor at Hangzhou Dianzi University, China. His research interests include Industrial IoT Software Security, Cloud-Edge-Terminal Computing, Intelligent Data Processing. He has publications in IEEE TII, IEEE T-ITS, IEEE TNNLS, IEEE TSC, IEEE TFS, IEEE TNSE, IEEE TNSM, IEEE TCCN, IEEE TGCN, IEEE TCSS, IEEE TETCI, IEEE/ACM TCBB, IEEE IoT-J, IEEE JSAC, IEEE JBHI, IEEE Network, ACM TOIT, ACM TOMM, etc.

Xiaolong Xu Xiaolong Xu is currently a professor in the School of Computer Science, Nanjing University of Posts & Telecommunications, Nanjing, China. He is also working for the Jiangsu Key Laboratory of Big Data Security & Intelligent Processing. He received the B.E. in computer and its applications, M.E. in computer software and theories and Ph.D. degree in communications and information systems, in 1999, 2002 and 2008, respectively.

Xinheng Wang Xinheng Wang received the B.E. and MSc. degrees in electrical engineering from Xian Jiaotong University Xi'an, China, in 1991 and 1994, respectively, and the Ph.D. degree in electronics and computer engineering from Brunel University, Uxbridge, U.K. in 2001. He is currently a professor with the Department of Mechatronics and Robotics, School of Advanced Technology, Xi'an Jiaotong-Liverpool University (XJTLU). He was the founding head of the department. His current research interests lie in industrial Internet of Things, Edge AI, Simultaneous Localisation and Mapping (SLAM) and navigation for robots, and intelligent manufacturing. He has published 200+ referred papers and is a recipient of 16 granted patents.