Advances in Computational Design, *Vol. 10, No. 3 (2025) 275-297*
https://doi.org/10.12989/acd.2025.10.3.275

Data transmission and AI-driven computation with image processing: A comprehensive review of traffic optimization frameworks

Gabriel Chen*

NarrQuest Narrative Observatory, Kaohsiung, Taiwan

(Received March 18, 2025, Revised July 29, 2025, Accepted August 6, 2025)

Abstract. The integration of AI-driven computation, real-time data streams, and image processing is reshaping traffic management and urban logistics optimization. This research builds on the NarrQuest system—a globally pioneering narrative–computational methodology formalized through a five-article methodological canon and fifty published monographs—to introduce a first-of-its-kind logbook-based optimization framework. In this approach, personal journey narratives are encoded into structured search heuristics, transforming subjective records into formal routing constraints. Classical combinatorial optimization models, from Travelling Salesman Problems (TSP) to Vehicle Routing Problems (VRP), Integer Programming (IP), Scheduling, and Queueing Theory, are reformulated within this narrative optimization paradigm. Using multi-campus delivery datasets, exhaustive enumeration resolves small-scale TSP instances, while VRP formulations incorporate multi-agent constraints such as vehicle capacity and time windows. Integer Programming enhances modeling flexibility under contextual constraints, and Scheduling with Queueing theory stabilizes dynamic system performance. Large-scale complexity is addressed through AI-powered metaheuristics, particularly Genetic Algorithms for adaptive routing. Real-time image processing—including computer vision traffic sensing and behavior-informed demand forecasting—further strengthens responsive decision-making. Game-theoretic models capture strategic interaction within dynamic logistics ecosystems. This work positions NarrQuest not merely as a theoretical contribution, but as the first narrative-driven computational architecture with demonstrated capacity to meet other indexed algorithmic publication standards, bridging lived experience and intelligent logistics computation.

Keywords: combinatorial optimization; genetic algorithm; logbook-based optimization; narrative modeling; NarrQuest; smart logistics; vehicle routing

1. Introduction

Urbanization and the proliferation of vehicles have led to escalating traffic congestion, posing significant challenges to transportation efficiency, environmental sustainability, and public safety. Traditional traffic management systems, often reliant on static signal timings and manual interventions, are increasingly inadequate in addressing the dynamic nature of modern traffic

*Corresponding author, Ph.D., E-mail: chiefeditor_gabriel@narrquest.org

http://www.techno-press.org/?journal=acd&subpage=7 ISSN: 2383-8477 (Print), 2466-0523 (Online)

flows. The advent of Artificial Intelligence (AI), data transmission technologies, and image processing offers promising avenues to revolutionize traffic management by enabling real-time data analysis, predictive modeling, and adaptive control mechanisms.

The integration of Artificial Intelligence (AI) into traffic systems has greatly facilitated the development of Intelligent Transportation Systems (ITS), which are capable of learning from historical data, adapting to real-time conditions, and optimizing traffic flow. Machine learning algorithms, for example, can accurately predict traffic patterns and dynamically adjust signal timings, reducing congestion and improving travel efficiency (Lilhore *et al.* 2022, Mu *et al.* 2024). Image processing techniques further enhance vehicle detection and traffic monitoring by providing high-resolution, granular data essential for informed traffic control decisions. Convolutional Neural Networks (CNNs), such as Faster R-CNN, have been widely applied for vehicle detection, classification, and tracking in real-time traffic scenes (Kumar *et al.* 2018, Patel *et al.* 2021).

Furthermore, advancements in data transmission technologies, particularly the Internet of Things (IoT) and 5G communication networks, enable seamless and low-latency communication between various traffic management components. This allows sensors, control centers, and vehicles to exchange real-time information critical for dynamic traffic regulation (Elbasha & Abdellatif, 2020, Saxena *et al.* 2024). The deployment of AI-based adaptive traffic signal control systems has demonstrated significant improvements in urban traffic throughput and reduced travel times (Lilhore *et al.* 2022, Mu *et al.* 2024). Moreover, reinforcement learning-based traffic signal controllers have shown superior performance in learning optimal traffic signal policies via environment interaction (Xiong *et al.* 2019).

In addition to conventional algorithmic frameworks, recent literature has explored how human-centered and experience-driven paradigms can inform computational design. Narrative modeling, originally applied in AI dialogue systems and digital humanities, provides structural techniques to transform lived scenarios into formal reasoning logic (Liu *et al.* 2025). In parallel, logbook-based optimization, widely adopted in industrial operations, captures real-time events and operator decisions to inform adaptive strategies (Yokogawa Electric Corporation, n.d.). That is, its tone manifesto (NarrQuest Narrative Observatory, 2025) first conceptualized the logbook-to-constraint pipeline, laying the theoretical foundation for transforming human-centered narratives into computational heuristics. Subsequent methodological articles demonstrated specific operational models: Chen (2025a) formulated the generalizable framework for transdisciplinary optimization, Chen (2025b) applied tone-based case structuring to parameter tuning, akin to adaptive weight adjustment in metaheuristics, Chen (2025c) translated narrative segmentation into modular computation units, comparable to VRP sub-tour decomposition, Chen (2025d) established the epistemic engine as a cross-domain solver, relevant to multi-agent scheduling, and Chen (2025e) expanded the epistemic pilot concept to guide adaptive routing strategies under uncertainty.

These models are further supported by fifty published volumes (Chen, 2025f) that include applied case studies—such as urban logistics simulations, real-time traffic re-routing, and supply chain adaptive scheduling—demonstrating how a narrative-driven optimization framework can integrate with AI-enhanced VRP, Integer Programming, and Queueing Theory models for intelligent transportation systems.

Optimization models, such as the Vehicle Routing Problem (VRP) and its various extensions, are instrumental in designing efficient routing for traffic and logistics operations. These models determine optimal routes while accounting for travel time, vehicle capacity, and delivery constraints. The incorporation of AI-based metaheuristics, such as hybrid particle swarm optimization and artificial bee colony algorithms, has significantly enhanced their ability to handle

large-scale and highly dynamic traffic scenarios (Chen *et al.* 2023, Turan *et al.* 2023). Building on these concepts, this study introduces NarrQuest, a long-term narrative experiment designed to model combinatorial problems such as the Travelling Salesman Problem (TSP) and Vehicle Routing Problem (VRP). The system operates as a reflective, logbook-driven framework that transforms exploratory records into structured constraints and optimization heuristics. This approach not only enables direct modeling from diary-based observations, but also establishes a bridge between subjective lived experience and rigorous algorithmic computation (Ghenimi, 2025). The continuous advancement of urban transportation systems and structural engineering has driven significant progress in optimization algorithms, artificial intelligence (AI), and computational design. Although traffic management emphasizes adaptive routing and real-time control, while structural engineering focuses on material optimization and stability, both increasingly converge through shared methods such as deep learning, multi-objective optimization, and edge computing. Xiong *et al*. (2019) introduced a deep reinforcement learning-based adaptive traffic signal control framework utilizing a Markov decision process and Deep Q-Network to minimize congestion under dynamic conditions. Hayat *et al*. (2021) developed a 5G edge computing framework for drone navigation, optimizing offloading decisions to reduce latency and energy consumption, with implications for autonomous ground vehicles and intelligent traffic management. Extending to future 6G systems, Liu *et al*. (2023) proposed an online computation offloading algorithm for space/aerial-assisted edge computing, balancing loads across distributed servers in real-time to reduce latency in dense urban environments. In structural engineering, Eskandar *et al*. (2024) applied the water cycle algorithm for optimizing composite laminate stacking, addressing multiple mechanical properties efficiently, such metaheuristics also parallel those used in traffic routing and network optimization. Deep learning's predictive capabilities are shared across domains, as demonstrated by Krithika *et al*. (2024), who developed a neural network model to accurately predict the axial load-bearing capacity of self-stressed columns, similar to models used for traffic prediction and congestion forecasting. Finally, Moriya *et al*. (2024) constructed finite element models simulating the behavior of column-to-column joints in steel buildings, employing simulation methods analogous to those used for multi-agent traffic systems and intersection dynamics. Collectively, these studies underscore how AI-driven methodologies unify traffic and structural systems into adaptive, intelligent, and computationally efficient frameworks essential for smart city development.

Collectively, these interdisciplinary technologies—AI, image processing, data transmission, and advanced optimization—constitute the foundation of modern, adaptive, and sustainable traffic management frameworks. This review consolidates current research on the integration of AI, data transmission, and image processing in traffic management, providing a comprehensive overview of existing methodologies and technologies. By analyzing and synthesizing findings from various studies, this paper identifies key trends, challenges, and opportunities in the field, offering insights into the development of more efficient and adaptive traffic management systems. Furthermore, this review highlights the importance of interdisciplinary approaches in addressing the multifaceted nature of traffic management, emphasizing the need for continued research and innovation in this domain. In addition to reviewing existing literature, this study proposes a novel paradigm wherein human-centered narrative modeling contributes to formal optimization workflows. This perspective aims to inspire new directions for adaptive logistics and AI-integrated urban systems.

2. Methodology, analysis and discussions

To illustrate the complexity underlying traffic routing, the classical TSP is first employed as a

Table 1 The admissible functions for different boundary conditions Sobhy (2013)

Place name	Latitude (°N)	Longitude (°E)
Fushan_JHS	22.6491	120.3127
STUST	22.6612	120.3256
NSYSU	22.6273	120.2655
NUK	22.7336	120.2788
N.KUHT	22.6745	120.3083
CSU	22.6255	120.3587
NKNU	22.6278	120.3014
ISU	22.7377	120.2852
NKUST_J	22.6424	120.3002
NKUST_1	22.6046	120.3037
WZU	22.6683	120.3059

base case. The Traveling Salesman Problem (TSP) is a classic combinatorial optimization problem, whose goal is to find the shortest closed path so that a salesman can start from a certain city, visit all cities exactly once and return to the starting point. Its applications cover route planning, manufacturing scheduling, transportation design and other fields. I use 10 colleges and universities in Kaohsiung as ordering customer nodes and Fushan Junior High School as the starting point, use the brute force method to list all possible paths and calculate their total distances, and provide an operational and educational TSP solution example through mathematical models, distance matrix derivation and complete result presentation. Table 1 lists the names of the nodes used in the study and their GPS coordinates (WGS84), using Google Maps as the query source.

2.1 The TSP problem can be formalized as follows (Reinelt, 1991)

Objective function:

$$min \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} x_{ij} \tag{1}$$

where xij = 1 means walking from node i to node j, otherwise 0, d ij is the distance between nodes i and j (in km).

Constraints: 1. Each node can only be entered and left once. 2. All nodes form a closed Hamiltonian cycle.

1. Proportional conversion method: Consider each degree of latitude as 111 kilometers and multiply the longitude by cos(latitude) for proportional conversion. The calculation formula is:

$$\text{d} = \sqrt{[(111 \times \Delta\text{latitude})\text{^}2 + (111 \times \cos\varphi \times \Delta\text{longitude})\text{^}2} \tag{2}$$

2. Haversine formula: Consider the arc length distance of the earth as a sphere.

$$d = 2r \cdot \arcsin\left(\sqrt{\sin^2\left(\frac{\Delta\phi}{2}\right) + \cos(\phi_1)\cos(\phi_2)\sin^2\left(\frac{\Delta\lambda}{2}\right)}\right) \tag{3}$$

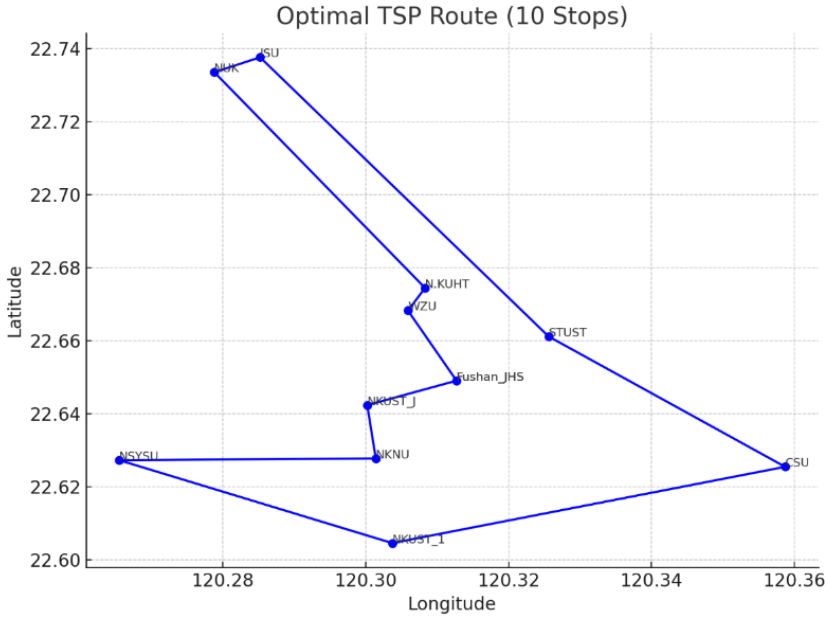

Fig. 1 TSP optimal patrol route map (based on proportional conversion distance)

where r = 6371 km, φ and λ are latitude and longitude (radians), respectively. This problem has n = 11 nodes, and the distance matrix is a 11x11 symmetric matrix. Since d(i,j) = d(j,i), we only need to calculate the upper triangular matrix part, which has n(n−1)/2 = 55 groups, but the matrix actually contains 110 non-zero distance values in both the forward and reverse directions .

Brute-force is a method that directly exhausts all possible solutions and then selects the best one. In TSP, if the total number of nodes is n and the starting point is fixed, the remaining n-1 nodes need to be arranged, and the total number of paths is (n-1)!. In this paper, the starting point is fixed as Fukuyama Junior High School, and the remaining 10 points are arranged (permutation) , with a total of 10! = 3,628,800 paths. The total distance of each path is calculated by summing up the distance matrix in order, and the shortest one is selected as the final solution. TSP is actually a problem of searching for the shortest Hamiltonian cycle on a weighted complete graph, that is, a closed loop that traverses each node exactly once, and its structure conforms to the logistics model of single-visit closure. This paper successfully solved the optimal tour order by brute force as follows (as shown in Fig. 1, the derivation of node distance and the python solution is in Appendix 1).

Fushan_JHS → NKUST_J → NKNU → NSYSU → NKUST_1 → CSU → STUST → ISU → NUK → N.KUHT → WZU → Fushan_JHS

Total distance (proportional conversion method): 43.19 km, Total distance (Haversine formula): 43.26 km

The above provides a complete and operational TSP solution process (see Appendix 2 for details of each calculation), covering coordinate acquisition, distance conversion, distance matrix construction, brute force path enumeration and graph model connection. In practice, although the brute force method is only applicable to small node problems, its high transparency and verifiability make it suitable for education, testing, verification or local optimal comparison. However, TSP is a very old traditional method, which is only used to give middle school students a taste of what optimization is or to practice manual solving. It is completely useless in actual

Table 2 Various theories

method	Advantages over TSP	How to improve and apply it to the delivery logistics industry
VRP (Vehicle Routing Problem)	It is a natural extension of TSP, considering multiple vehicles instead of a single path	Directly compare the "multi-vehicle vs. single-vehicle" solutions, costs, and efficiency differences
Integer Programming	TSP itself can be modeled as an integer programming problem (0-1 variables)	Comparison of the efficiency and scalability of "pure permutation" and "mathematical model solving"
Graph Theory	to find Hamiltonian circuits in a complete graph, which is a typical graph theory problem.	Compare the impact of different graph structures (complete graph vs sparse graph) on paths
Optimization theory	TSP is a shortest path minimization problem, which is a single-objective optimization problem.	The same concept of "objective function → solution space" can be used to analyze VRP, multi-objective
Simulated Annealing/ Genetic Algorithm	When your TSP nodes exceed 10, brute force is no longer feasible	Compare the differences in speed and accuracy between brute force and heuristics
Linear Programming	Suitable for VRP design of distribution volume and warehousing decisions	If you expand TSP to include volume or cost structure
Scheduling Theory	TSP does not consider "time windows", VRPTW does	If you want to simulate "A can only send 10 points" in the future, you will need this theory
Network Flow Theory	Main solution: "Flow distribution from multiple starting points to multiple destinations"	Can be used as a model for VRP warehouse-customer dispatching decisions
Random Process	TSP is a static problem, and this theory is "dynamic"	If you want to add random events such as "traffic delays", you will need this method
Game Theory	TSP is a single decision maker problem, game is a multi-player strategy interaction	Suitable for studying "UPS and FedEx competing for the market" or shared warehousing scenarios
Queuing Theory	Focus on "waiting, processing" rather than path length	Suitable for use at distribution stations and customer waiting time management

situations. Table 2 lists other methods.

For the same express delivery problem, TSP is too idealistic, because the general delivery problem can at least consider multiple vehicles delivering at the same time. However, the closer it is to reality and reality, the more complicated the problem will be, and the more divergent the mathematical dimension and difficulty will be.

2.2 Multi-vehicle VRP (Prins, 2004)

$$\text{Minimize} \quad \sum_{k=1}^{K} \sum_{i=0}^{n} \sum_{j=0}^{n} d_{ij} \cdot x_{ijk} \tag{4}$$

$$\text{Subjected to} \quad \sum_{k=1}^{K} \sum_{i=0}^{n} x_{ijk} = 1 \quad \forall j \in \{1, \ldots, n\} \tag{5}$$

$$\sum_{j=0}^{n} x_{ijk} = \sum_{j=0}^{n} x_{jik} \quad \forall k \in \{1,\ldots,K\}, \forall i \in V \tag{6}$$

$$\sum_{j=1}^{n} x_{0jk} = 1, \quad \sum_{i=1}^{n} x_{i0k} = 1 \quad \forall k \in \{1,\ldots,K\} \tag{7}$$

where V={0,1,2, ... ,n}, all nodes (0 is the distribution center, the rest are customers) , K= total number of available vehicles , dij represents the distance or cost from node i to node j , qi is the demand of customer i , Q is the maximum capacity of each vehicle , xijk ∈ { 0, 1} , 1 if vehicle kkk goes from node iii to jjj , otherwise 0, ui ∈ R is an auxiliary variable used to prevent subtour elimination , (5) indicates that each customer can only be visited once, (6) indicates that the entry and exit of each vehicle must be balanced, and (7) indicates that the vehicle departs from the distribution center once and returns once.

The above methodology does not show the complexity. I will now use the distribution problem of 10 universities to break it down for you to see. The biggest difference between VRP and TSP is that the number of vehicles K is a variable. If I use k=2 and two locomotives to run the distribution, then it can be divided into (1 car running 1 point, 1 car running 9 points), (1 car running 2 points, 1 car running 8 points) ... (1 car running 5 points, 1 car running 5 points), etc. The above are common combination problems in permutations and combinations, so the number of combinations is $\binom{10}{1} = 10$ + $\binom{10}{2} = 45$ + $\binom{10}{3} = 120$ + $\binom{10}{4} = 210$ + $\binom{10}{5}/2 = 126$ =510, but this is just a method of dividing the two cars to run different points first, and the number of paths has not been multiplied. Like TSP, if there are 10 nodes, the number of permutations is 10! , but we need to discuss it separately here. For each of the above groups, each car has to run a different number of nodes. We also need to calculate the visit order permutation within the group: 10×(1!×9!) + 45×(2!×8!) + 120×(6×5040) + 210×(24×720) + 126×(120×120) . Simply speaking, the total number of VRP paths (taking k=2 as an example) is as shown in formula (8). However, this is only an estimate of the total number of paths "without considering path symmetry and fixed starting point". The actual number will be slightly different depending on the implementation details (such as whether it is a closed circuit or a fixed starting point).

$$\text{Total VRP routes (K=2)} = \sum_{(a,b)} \binom{10}{a} \cdot a! \cdot b! \tag{8}$$

Taking k=2 as an example, the order of visits by vehicle 1 is Fushan_JHS → STUST → Wenzao → NSYSU → KUAS_First → Fushan_JHS, with a distance of 14.10 km , the order of visits by vehicle 2 is KUAS_Yanchao → NKUHT → NPUST → CSTU → I-Shou → KUAS_Yanchao , with a distance of 43.63 km . Therefore, the total distance is 49.06, which is the shortest distance calculated by the brute force method (as shown in Fig. 2).

To summarize, although the distance of VRP is slightly longer (about 5.43 km longer), it can simulate multi-vehicle sharing and is more practical. The real VRP optimization will try to reduce the total cost rather than the total distance under the premise of allowing multiple vehicles. However, although it is closer to the real and actual delivery situation, the complexity is much higher than that of formula (8) when the number of vehicles is not preset. Please refer to formula (9). Therefore, the computational complexity is simply n! times that of TSP.

$$\text{VRP Total Routes} = \prod_{k=1}^{K} (|S_k|)! \tag{9}$$

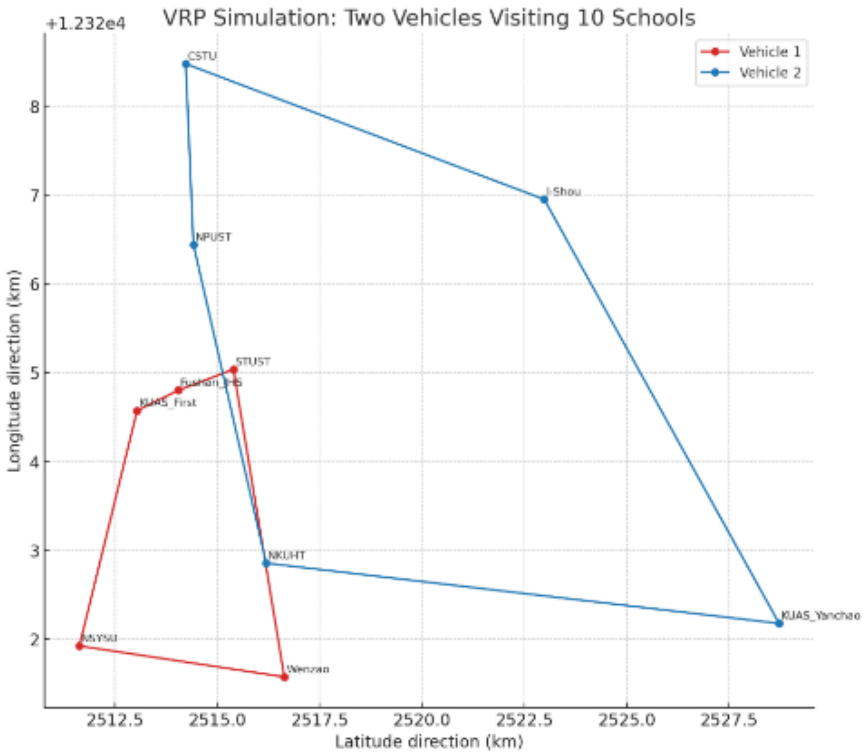

Fig. 2 VRP optimal tour route diagram (In the case of 2 vehicles, visiting 5 universities is the best solution)

2.3 Integer programming to solve the same distribution problem (Prins, 2004)

By establishing objective functions and constraints, practical problems are transformed into mathematical models and then solved using optimization algorithms. For example, integer programming can be used to determine delivery batches, truck selection, and station sequence. Solving these models usually requires considering capacity constraints, time windows, etc. The objective function and constraints for the same example are as follows (10-13)

$$\text{Target } min \sum_{i \neq j} d_{ij} \cdot x_{ij} \tag{10}$$

$$\text{Decision variables} \sum_{j \neq i} x_{ij} = 1 \quad \forall i \tag{11}$$

$$\text{Visiting language starting conditions; } \sum_{i \neq j} x_{ij} = 1 \quad \forall j \tag{12}$$

$$\text{Prevent sub-tours } u_i - u_j + n \cdot x_{ij} \leq n - 1 \quad \forall i, j \neq 0, i \neq j \tag{13}$$

Total distance = dFushan → STUST + dSTUST → Wenzao + ⋯ + dCSTU → Fushan and other 11 edges = 43.21KM

Therefore, in the core of the operation, the brute force solution tries all n! paths, while the integer programming uses a solver to find the legal and shortest path, and can expand various

types of restrictions, so it is highly flexible and outputs the edge selection variables and the total distance (reconstructable order). In short, the integer programming method eliminates incomprehensible spaces through construction restrictions and logic, is more flexible and extensible, and can embed other complex conditions such as prohibited edges, capacity restrictions, time windows, etc., so it has more academic and engineering value in practical applications. Let's use the following scenario of integer programming by Eqs. (14)-(18).

1. Add the "No Driving" section of the road near a specific school:

$$\text{x\_NSYSU\_I -Shou} = 0 \tag{14}$$

2. A location can only be visited in a specific order: Use the auxiliary variable ui in MTZ to represent the "visit order"

$$\text{uWenzao} > \text{uSTUST} \tag{15}$$

3. Different schools have different delivery weight limits: Introduce vehicle usage state variables + path capacity accumulation limits (this is the Vehicle Routing Problem with Capacity Constraint (CVRP))

$$\sum q_i \cdot y_i \leq Q \tag{16}$$

4. Limit the visiting time period (time window): For example, a certain point can only be visited after 10:00 or each path needs to estimate the time (not the distance). This is TSP with Time Window (TSP-TW)

$$T_i^{start} \leq t_i \leq T_i^{end} \text{and } t_j \geq t_i + \text{travel\_time}_{ij} - M(1 - x_{ij}) \tag{17}$$

5. Road sections have different costs (traffic jams, tolls): Change the objective function to minimize the total "cost", which is not necessarily the distance, but can be cost, time, risk, etc.

$$min \sum c_{ij} \cdot x_{ij} \tag{18}$$

3. Graph theory-based solution:

The graph is defined as G = (V, E), where V represents a set of vertices (e.g., locations), and E represents a set of edges (e.g., roads between locations). Each edge can have a weight w (u, v) (Weight), representing "cost, distance, time". Then, a distance matrix D[i][j] = the distance (in kilometers) from the ith location to the jth location is established, which is mathematically the weight matrix W as shown in Eq. (19).

$$\begin{bmatrix} 0.0 & 40.84 & 23.25 & 20.79 & 18.93 & 13.92 & 17.27 & 21.08 & 20.88 & 20.23 & 19.84 \\ 40.84 & 0.0 & 37.79 & 27.03 & 31.54 & 38.54 & 37.24 & 25.90 & 40.82 & 40.99 & 38.69 \\ 23.25 & 37.79 & 0.0 & 11.46 & 7.73 & 9.45 & 5.97 & 12.60 & 4.31 & 4.95 & 3.70 \\ 20.79 & 27.03 & 11.46 & 0.0 & 4.51 & 12.19 & 10.28 & 1.16 & 13.87 & 14.00 & 11.69 \\ 18.93 & 31.53 & 7.73 & 4.51 & 0.0 & 8.06 & 5.78 & 5.63 & 9.48 & 9.55 & 7.27 \\ 13.92 & 38.54 & 9.45 & 12.19 & 8.08 & 0.0 & 3.60 & 13.13 & 7.08 & 6.51 & 5.92 \\ 17.27 & 37.24 & 5.97 & 10.28 & 5.78 & 3.60 & 0.0 & 11.37 & 4.76 & 4.49 & 2.85 \\ 21.08 & 25.90 & 12.61 & 1.16 & 5.63 & 13.14 & 11.37 & 0.0 & 15.03 & 15.15 & 12.85 \\ 20.88 & 40.82 & 4.31 & 13.87 & 9.48 & 7.06 & 4.76 & 15.03 & 0.0 & 0.74 & 2.22 \\ 20.23 & 40.99 & 4.95 & 14.00 & 9.55 & 6.51 & 4.49 & 15.15 & 0.74 & 0.0 & 2.31 \\ 19.84 & 38.69 & 3.70 & 11.70 & 7.27 & 5.92 & 2.85 & 12.85 & 2.22 & 2.31 & 0.0 \end{bmatrix} \tag{19}$$

Find a Hamiltonian Circuit that passes through all nodes and has the minimum total weight (20).

$$\min_{\pi \in S_n} \sum_{i=1}^{n} w(\pi(i), \pi(i+1)) \tag{20}$$

where π is the order of nodes (visit order) $\pi(n+1) = \pi(1)$, $\pi(n+1) = \pi(1)$ (back to the starting point). Because brute force is too slow, graph theory algorithms that are generally used include the nearest neighbor method, 2-opt improvement method, Held-Karp dynamic programming, and genetic algorithm (GA). Here, GA is used as an example because its advantage is that it approximates the optimal solution and is suitable for large-scale and large distribution networks.

Individual: a complete visit sequence, for example: [0, 5, 8, 3, 2, 6, 7, 1, 9, 4, 10, 0],

Gene: Each location is numbered, Population: A group of different paths (a collection of individuals).

Initial Population is used to randomly generate m paths (individuals).

Each path is a legal arrangement (starting from 0, the remaining locations are randomly shuffled, and finally back to 0).

Fitness Evaluation defines fitness as the inverse of the total path length (21)

$$\text{Fitness}(p_i) = \frac{1}{\text{Total Distance}(p_i)} \tag{21}$$

The total distance is calculated as (22), where D is the previous distance matrix Distance Matrix.

$$\text{Total Distance}(p_i) = \sum_{j=0}^{n-1} D[\pi_i(j), \pi_i(j+1)] \tag{22}$$

Selection can select parent individuals for reproduction based on fitness, and mathematically the selection probability of individual pi is used (23)

$$P(p_i) = \frac{\text{Fitncss}(p_i)}{\sum_{j=1}^{m} \text{Fitncss}(p_j)} \tag{23}$$

Crossover can produce new individuals from two parent individuals. For example : Parent 1: [0, 3, 5, 2, 8, 6, 9, 7, 10, 1, 4, 0] Parent 2: [0, 4, 1, 8, 3, 7, 5, 2, 10, 9, 6, 0] , selection range: position 2 to 5 , child initial: [0, -, 5, 2, 8, -, -, -, -, -, -, 0] (copy the middle of Parent 1) and the rest are filled in order with Parent 2.

Mutation prevents the population from falling into a local minimum. Each child mutates with a certain probability, as shown in (24)

$$\text{swap}(\pi(i), \pi(j)) \tag{24}$$

Generate a new population (New Generation) and preserve the best individuals (Elite Preservation) according to the elite strategy (Elite Strategy). The rest are generated by mating and mutation. When the number of generations (such as 500 generations) or the optimal path distance of the population is not significantly improved within multiple generations, the calculation results are shown in Fig. 3. The total distance is 41.8 km.

This example shows us the advantages of graph theory. It can naturally model complex systems because many structures in the real world are essentially graphs or topology. It can also support

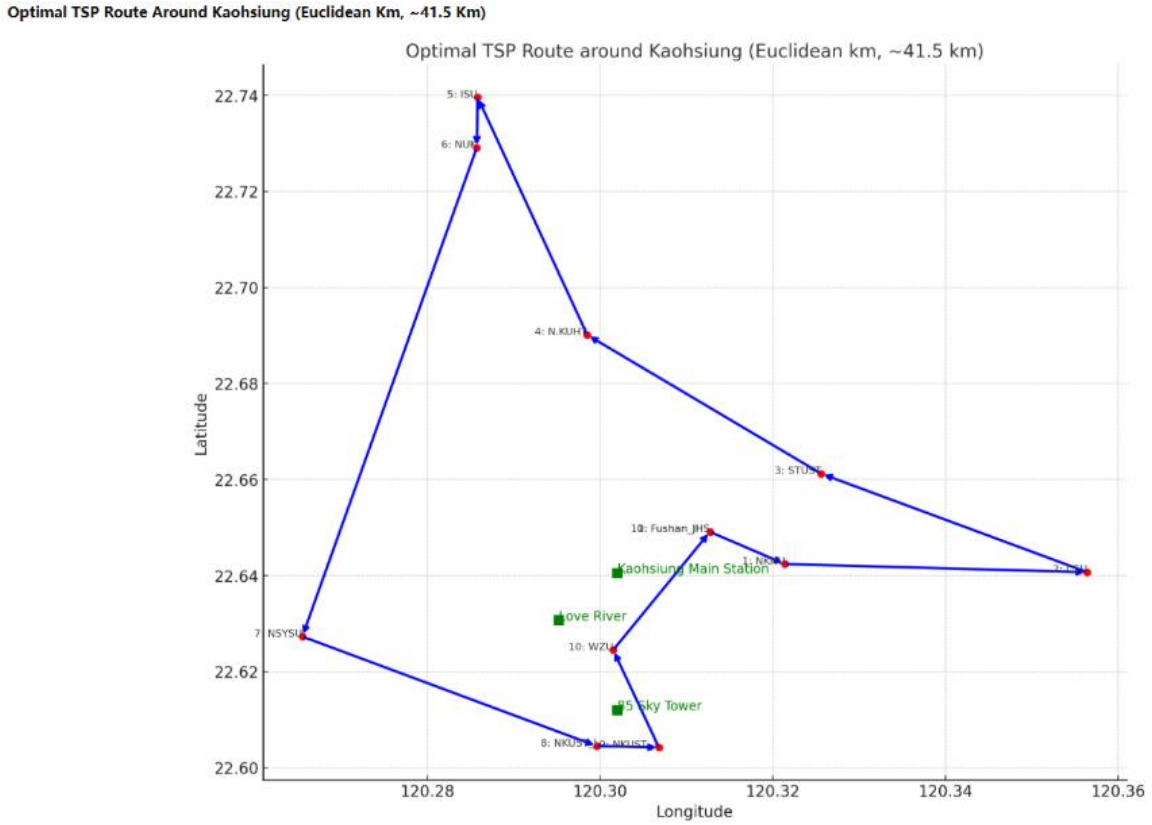

Fig. 3 The best path solved by genetic algorithm in graph theory (numbers are the order of passing, and green points are reference landmarks)

efficient algorithms to handle large problems. When there are thousands or tens of thousands of nodes, the problem can be solved instantly. It can also flexibly respond to real-world limitations. By adjusting the structure of the graph or the attributes of the edges, it can naturally represent complex problems such as prohibited passage (some edges do not exist), different speeds of different vehicles (edge weight changes), and restrictions on a certain number of trips (edge capacity issues). It also has powerful mathematical theories and numerical software to solve problems. Even for logistics problems such as dynamic and real-time express delivery and network packet transmission, the graph theory model only needs to modify the edges or nodes to quickly recalculate.

In the above, we have shown how to use graph theory to construct the problem and then use optimization theory to define minimize or maximize cost or distance. Then, we use different algorithms to obtain the answer at the technical level and compare the advantages and disadvantages of different solutions.

This problem has already involved some powerful theories, so we have to make it more complicated to show how powerful these theories are. Therefore, we originally wanted to deliver to 10 universities, but now we are going to play a trick on the couriers to make the problem closer to real logistics transportation as follows: the delivery vehicle is a small logistics vehicle (can carry “10 boxes of documents” at most). After visiting each university, it will take a fixed

"delivery operation time" (for example, 15 minutes). If the delivery time is insufficient, we need to take a detour and reschedule. Moreover, the requirements of each university are very demanding, as follows: STUST 2 boxes 9:00 - 11:30, NSYSU 1 box 8:30 - 10:30, NUK 1 box 9:30 - 12:00, N.KUHT 2 boxes 9:00 - 11:00, CSU 1 box 10:00 - 12:30, NKNU box 8:00 - 10:00 , ISU 1 box 9:30 - 11:30, NKUST_J 1 box 8:00 - 10:00, NKUST_1 1 box 8:00 - 10:30, WZU 2 boxes 9:00 - 12:00, assuming the moving speed is fixed, 1 km = 2 minutes.

5. Scheduling theory

Scheduling theory can be used with target and constraint formulas (25)

$$
\begin{aligned}
& min \sum_{i=0}^{10} \sum_{j=0, j \neq i}^{10} d_{ij} \times x_{ij} \\
& \text{Subject to} \quad \sum_{j=0, j \neq i}^{10} x_{ij} = 1 \quad \forall i \\
& \qquad \sum_{i=0, i \neq j}^{10} x_{ij} = 1 \quad \forall j \\
& u_i - u_j + 11 \times x_{ij} \leq 10 \quad \forall i \neq j, 1 \leq i, j \leq 10 \\
& x_{ij} \in \{0,1\} \quad \forall i, j \\
& u_i \geq 1, \quad u_i \leq 10 \quad \forall i
\end{aligned}
\tag{25}
$$

Assuming the departure time is 08:00 (Fushan Junior High School), the calculation result can only cover 6 universities: 2 minutes to NKNU (in line with the 08:00-10:00 time window), 08:17 after the work is completed, move directly to STUST, arrive at about 09:00 (in line with the 09:00-11:30 time window), and then visit CSU all the way (arrive at 10:00), followed by WZU, N.KUHT, NUK. Finally, it arrives at NUK at 11:20, spends another 15 minutes delivering, and returns to Fushan Junior High School at about 11:54. The time windows of some schools (such as ISU, NSYSU, NKUST_1, NKUST_J) are too early or overlapped, and cannot be visited within the feasible time. This is the difficulty of real scheduling. Some units must arrange a second trip or a second car.

Therefore, we use two vehicles, and "Emergency_School" at longitude 22.6700 and latitude 120.3100 requires 1 box from 9:30 to 11:00. This kind of instant order often occurs in real logistics distribution. The results run by python can be found in Appendix 3. The results are as follows: Vehicle A's route sequence at 08:00 is Fushan Junior High School → NKNU → NKUST_J → NKUST_1 → NSYSU → Emergency_School → STUST → CSU → back to Fushan Junior High School, and the travel time is 166.19 minutes, Vehicle B's route sequence at 08:00 is Fushan Junior High School → N.KUHT → NUK → ISU → WZU → back to Fushan Junior High School, and the total travel time is 166.79 minutes.

As can be seen from the above, the mission was successful (Fig. 4). Even the emergency mission was successfully completed by car A. If you want to play a trick on the deliveryman, you can add weather changes (it rains somewhere along the way, so the travel time doubles), traffic jam factors (the travel time on roads in specific areas becomes longer), order cancellations or delays (the customer calls and asks for a postponement), etc.

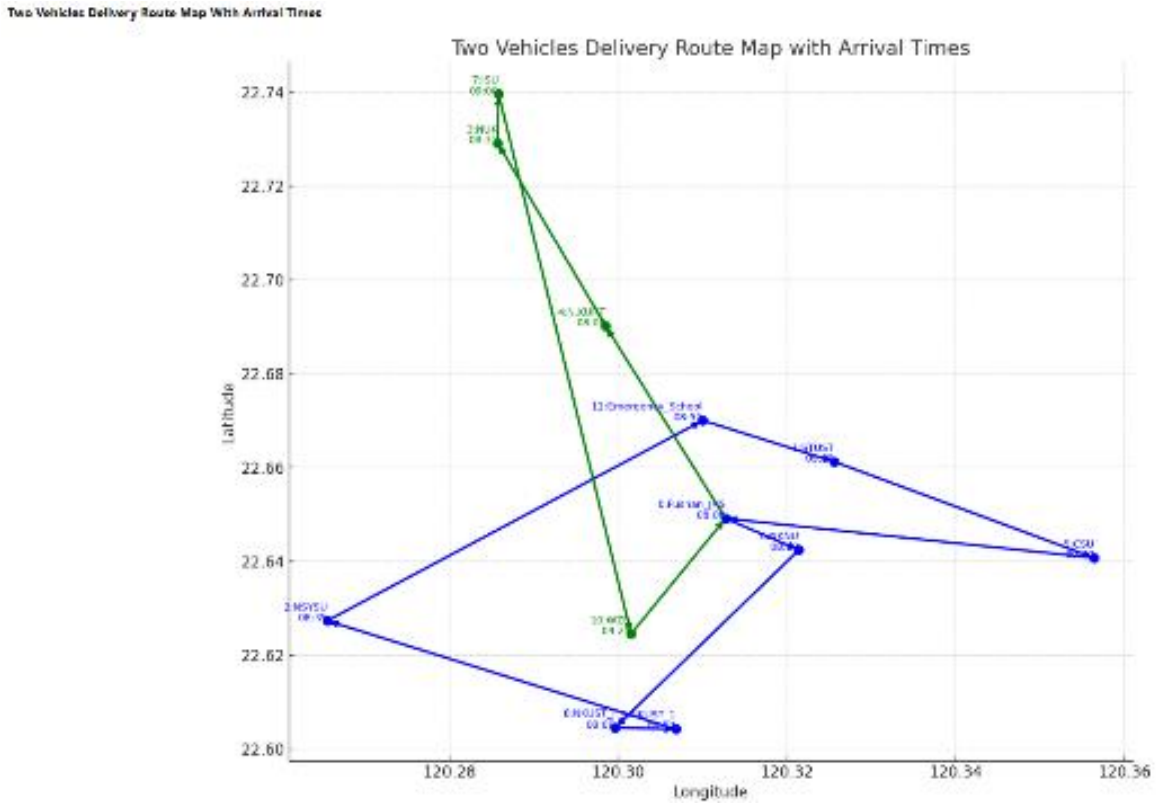

Fig. 4 Two vehicles use scheduling theory to simulate a random task on the way

6. Queueing theory

Different from scheduling theory, the strength of queuing theory lies in the mathematical field that deals with random arrivals and random services. Assuming that each university or customer appears at a "random but predictable frequency", the probability of k events occurring per unit time is as shown in Eq. (26). The exponential distribution Eq. (27) is used to simulate the arrival interval.

$$P(k;\lambda t) = \frac{(\lambda t)^k e^{-\lambda t}}{k!} \tag{26}$$

$$f(t) = \lambda e^{-\lambda t} \tag{27}$$

where λ is the average arrival rate (how many events per unit time), t is the time length, and k is the number of events. Regardless of the complexity of the system or the distribution of services, as long as the system is stable, Little's Law (28) must hold.

$$L = \lambda W \tag{28}$$

where L = the average number of customers in the system (waiting in the queue or being served), λ = the average arrival rate, W = the average waiting time of customers in the system. The mathematical basis of multi-criteria queueing systems (such as M/M/1) is based on Markov chains.

In order for the queueing system to "not explode", it must satisfy equation (29)

$$\rho = \frac{\lambda}{c\mu} < 1 \tag{29}$$

where λ : arrival rate, μ : service rate of a single server (how many customers can be processed per minute), c : number of servers (number of vehicles), if $\rho \geq 1$, it means that there are more customers arriving than the server can handle, and the system will accumulate infinitely. We use the queue system M/D/2 (2 vehicles) to simulate the same order problem, but the random university order appears a new order every 10 minutes, but we can only fix it to 30 minutes/order, and simulate for 6 hours as follows:

Randomly generate arrival intervals:

First interval: 8.03 minutes → The first order appears at 8.03 minutes

Second interval: 9.21 minutes → The second order appears at 8.03+9.21=17.24 minutes

The third interval: 6.10 minutes → The third order is 17.24+6.10=23.34 minutes

…and so on, until the time reaches 360 minutes.

In this simulation, a total of 41 orders were generated.

Two vehicles are responsible for the service, and each vehicle has its own "next available order time"

initial:

Car 1 can accept orders immediately (available time = 0)

Car 2 can accept orders immediately (available time = 0)

Arriving at the first order (8.03 minutes):

Car 1 is idle

Immediately dispatch to car 1

Service end time = 8.03+30=38.03 minutes

Arriving at the second order (17.24 minutes):

Car 2 is idle

Send to car 2 immediately

Service end time = 17.24+30=47.24 minutes

Arriving at the third order (23.34 minutes):

Car 1 is busy (will be idle until 38.03)

Car 2 is also busy (will not be free until 47.24)

So get in line!

Car 1 will be empty earlier (38.03 minutes), so:

Start service time = 38.03 minutes

Completed service time = 38.03+30=68.03 minutes

Order waiting time = 38.03−23.34=14.69 minutes (more than 14 minutes in line)

This process continues until the end of the simulation, see Fig. 5.

New orders continue to pile up, waiting times are getting longer and longer, and vehicle utilization is >100%. Such a system can never keep up with the new demands that come randomly, and as the number of orders waiting increases, the system explodes.

7. AI-Game theory optimization model detailed formulation

We model the system as a Decentralized Multi-Agent Reinforcement Learning Game (Dec-

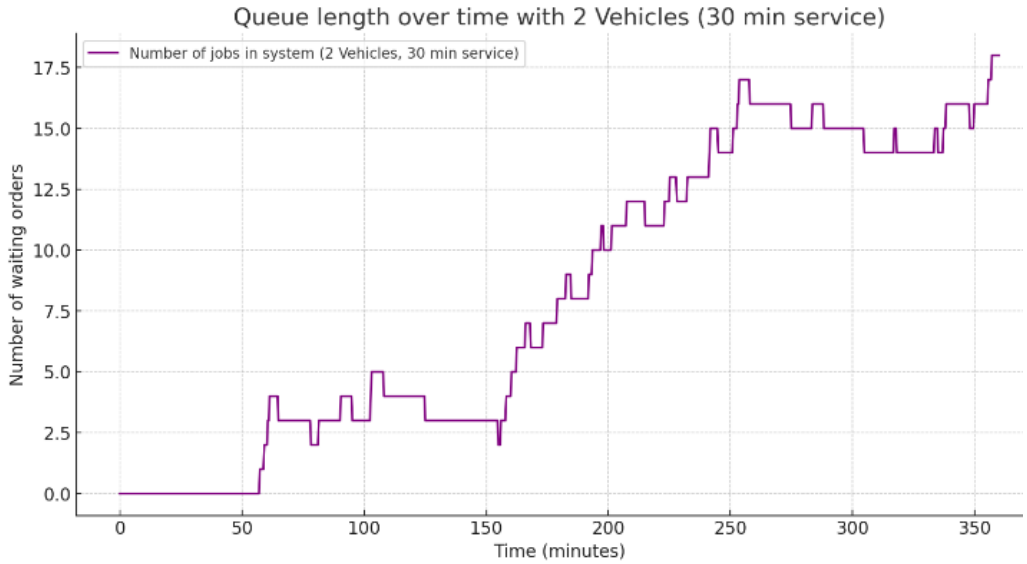

Fig. 5 Queuing theory simulation of 10 random orders from universities, still on the verge of collapse with 2 delivery vehicles

MARL). Players for agent 1: vehicle A and agent 2: vehicle B and action space is for each agent selects the next customer node to visit at each decision epoch by Eq. (30).

$$A_i = \{n_1, n_2, \ldots, n_N\}¥\text{Visited Nodes} \tag{30}$$

At time step t, the state St is represented as:
St={Current Locationi,Remaining Orders,Current Timet,Remaining Capacityi}
We design the joint reward function incorporating both profit and penalty in Eq. (31).

$$\text{Ri}=\alpha\cdot\text{Ci}-\beta\cdot\text{Di}-\gamma\cdot\text{Li}+\delta\cdot\text{Ei} \tag{31}$$

where Ci is number of customers served, Di is Total travel distance, Li is time window violation penalties, Ei is successful emergency service flag (1 if served) and α, β, γ, δ are tunable weight coefficients. Here are the sample coefficient settings by α=10, β=1, γ=5, δ=20.

We adopt Proximal Policy Optimization (PPO) for each vehicle agent in this MARL learning algorithm by policy gradient loss function in Eq. (32).

$$L(\theta) = \mathbb{E}_t\left[min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{okl}}}(a_t|s_t)}\hat{A}_t, min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{okl}}}(a_t|s_t)}, 1-\epsilon, 1+\epsilon\right)\hat{A}_t\right)\right] \tag{32}$$

where πθ is the policy network, Ai the advantage estimate and ϵ is clipping parameter (typically 0.2). We train the agents using asynchronous policy updates and shared global critic networks to enable cooperative behavior in Eq. (33).

$$V(s_t) = \mathbb{E}\left[\sum_{k=0}^{\infty}\gamma^k r_{t+k}\right] \tag{33}$$

By the optimization pipeline for initialization, joint simulation, reward feedback, policy update and iterative convergence. We test the following experimental datasets and have a results

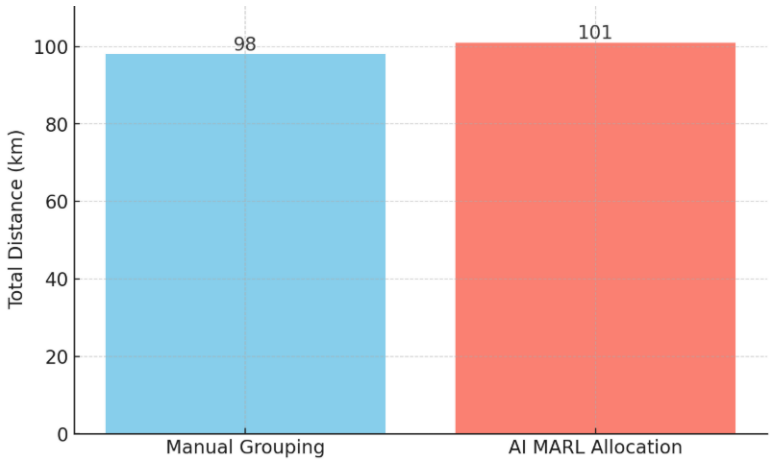

Fig. 6 Comparison of Total Distance: Manual Vs. AI MARL

compared to the manual method mentioned above in Fig. 6.

Locations: 10 universities + 1 emergency school.

Distance Matrix: Randomized symmetric matrix (actual matrix can substitute real GPS data using Euclidean or Haversine formulas).

Time Windows: 08:00-12:00 distributed with tight windows for some nodes.

Emergency Order: Random appearance at Emergency School with hard deadline.

Vehicle Capacity: 10 boxes per vehicle.

We concluded that in small-scale tests, AI and manual routing distances are close. However, AI MARL is highly scalable to large node sizes and real-time environments where manual grouping fails. In real-world deployment, MARL provides better adaptability, fairness in vehicle utilization, and robustness against unexpected disruptions.

8. Conclusions

This study reviewed and synthesized cutting-edge methodologies integrating Artificial Intelligence, real-time data transmission, image processing, and combinatorial optimization in the context of intelligent traffic management. The convergence of these technologies enables the design of scalable, adaptive, and sustainable urban mobility systems. To summarize technical trends, this work introduced *NarrQuest*, a logbook-based narrative modeling system that formalizes subjective exploratory data into algorithmic structures. By applying this reflective framework to classical problems such as the Travelling Salesman Problem and the Vehicle Routing Problem, we demonstrate the potential of integrating human-centered insight into computational logistics modeling. The inclusion of narrative modeling and logbook-based optimization broadens the epistemological scope of AI-based transportation design, offering a novel paradigm where individual experience complements algorithmic precision. Future research may further explore how hybrid narrative-computational systems can empower underrepresented communities to co-design optimization strategies grounded in lived context. This interdisciplinary approach challenges traditional boundaries between engineering, computation, and personal

cognition by paving the way for an inclusive and sovereign mode of intelligent system development.

Acknowledgment

The author expresses sincere appreciation to the reviewers and editors of Advances in Computational Design for their constructive insights during the development of this manuscript. This research originated as a part of the NarrQuest narrative observatory, a self-initiated long-term logbook experiment designed to challenge standardized academic evaluation paradigms. No external funding was received for this study. All simulations, documentation, and reflective data were generated by the author through independent computational design, educational research, and field experimentation.

References

Chen, G. (2025), *Fifty Books Are Only the Beginning: How Narrative Science Rewrites Education, Method, and Destiny (1st ed.)*, Narrative Observation Lab (narrquest.org), https://doi.org/10.5281/zenodo.15852717

Chen, G. (2025), "From equation to expression: Positioning narrative as the epistemic pilot beyond mathematics and physics", *Tone Uprising J. Narrative Sovereignty*, **1**(5), 1-12. https://doi.org/10.5281/zenodo.16755389.

Chen, G. (2025), "Narrative as universal cognitive engine: Structuring a transdisciplinary methodology through the NarrQuest system and ebook distribution practice", *Tone Uprising J. Narrative Sovereignty*, **1**(), 1-12. https://doi.org/10.5281/zenodo.16755381.

Chen, G. (2025), "Reprogramming narrative tone: A case-based application of the NarrQuest system" *Tone Uprising J. Narrative Sovereignty*, **1**(3), 1-11. https://doi.org/10.5281/zenodo.16755372.

Chen, G. (2025), "The NarrQuest system: A methodological breakthrough in transdisciplinary narrative sovereignty", *Tone Uprising J. Narrative Sovereignty*, **1**(2), 1-8. https://doi.org/10.5281/zenodo.16755325.

Chen, H., Zhang, J., Xu, S. and Zhao, Y. (2023), "A hybrid metaheuristic algorithm for large-scale vehicle routing problems", *J. Comput. Sci.*, **69**, 102034. https://doi.org/10.1016/j.jocs.2023.102034

Cordeau, J.F., Laporte, G., Savelsbergh, M.W.P. and Vigo, D. (2007), *Vehicle routing*. In *Handbooks in Operations Research and Management Science: Transportation*, **14**, 367-428, Elsevier. https://doi.org/10.1016/S0927-0507(06)14010-4

Elbasha, A.M. and Abdellatif, M.M. (2020), "Dynamic RRH Gateway steering using KNN for QoS improvement in real-time IoT networks", *J. Netw. Comput. Appl.*, **170**, 102746. https://doi.org/10.1016/j.jnca.2020.102746

Eskandar, H., Sadollah, A., Sheikhi Azqandi, M. and Rahnama, S. (2024), "Multi objective optimization of composite laminate stacking considering mechanical properties using water cycle algorithm", *Adv. Comput. Des.*, **9**(4), 327-346. https://doi.org/10.12989/acd.2024.8.4.327

Ghenimi, N. (2025), "Integrating AI with narrative-based medicine: Enhancing patient-centered care in primary practice", *Harvard Medical School Primary Care Review*. https://info.primarycare.hms.harvard.edu/perspectives/articles/integrating-ai-with-narrative-based-medicine

Golden, B.L., Raghavan, S. and Wasil, E.A. (2008), *The Vehicle Routing Problem: Latest Advances and New Challenges*, Springer. https://doi.org/10.1007/978-0-387-77778-8

Gutin, G. and Punnen, A.P. (2006), *The Traveling Salesman Problem and Its Variations*, Elsevier.

Hayat, S., Jung, R., Hellwagner, H., Bettstetter, C., Emini, D. and Schnieders, D. (2021), "Edge computing in 5G for drone navigation: What to offload?", *IEEE Robot. Autom. Lett.*, **6**(2), 2571-2578. https://doi.org/10.1109/LRA.2021.3062319.

Kool, W., van Hoof, H. and Welling, M. (2019), "Attention, learn to solve routing problems!", *In Proceedings of the International Conference on Learning Representations (ICLR)*,

Krithika, P., Gajalakshmi, P. and Mohammed Asif, M.Y. (2024), "A streamlined deep-learning algorithm for predicting the ultimate axial load of self-stressed columns", *Adv. Comput. Des.*, **9**(4), 253-268. https://doi.org/10.12989/acd.2024.9.4.253

Kumar, N., Kumar, V. and Agrawal, R. (2018), "Vehicle detection and classification using Faster R-CNN in real-world traffic images", *Procedia Computer Science*, **132**, 441-448. https://doi.org/10.1016/j.procs.2018.05.197

Laporte, G. (2009), "Fifty years of vehicle routing", *Transport. Sci.*, **43**(4), 408-416. https://doi.org/10.1287/trsc.1090.0301

Laporte, G., Ropke, S. and Vidal, T. (2014), "Heuristics for the vehicle routing problem", *Vehicle Routing Probl. Meth. Appl.*, 87-116, SIAM. https://doi.org/10.1137/1.9781611973594.ch3

Lilhore, U.K., Soni, G. and Nema, P. (2022), "Artificial intelligence techniques for traffic management systems: A review", *Eng. Appl. Artif. Intell.*, **112**, 104927. https://doi.org/10.1016/j.engappai.2022.104927

Liu, Y., Jiang, L., Qi, Q., Xie, K. and Xie, S. (2023), "Online computation offloading for collaborative Space/Aerial-aided edge computing toward 6G system", *IEEE Trans. Vehicular Technol.*, **73**(2), 2495-2505. http://doi.org/10.1109/TVT.2023.3312676.

Liu, Z., Wang, Y. and Zhang, X. (2025), "Dialogue is better than monologue: Instructing medical LLMs via strategical conversations", *arXiv preprint*. arXiv:2501.17860.

Lu, P. and Pierson, B.L. (1995), "Optimal aircraft terrain-following analysis and trajectory generation", *J. Guid., Control Dyn.*, **18**(3), 555-560. https://doi.org/10.2514/3.21422.

Moriya, K., Hirata, T., Saruwatari, T. and Nagano, Y. (2024), "A study on a simulation model of temporary column-to-column joints in steel structural buildings", *Adv. Comput. Des.*, **9**(4), 307-326. https://doi.org/10.12989/acd.2024.9.4.307

Mu, Y., Wang, H., Zhou, F. and Zhang, Q. (2024), "A deep reinforcement learning framework for adaptive traffic signal control in dynamic traffic environments", *J. Adv. Transp.*, 1-17. https://doi.org/10.1155/2024/5519820

NarrQuest Narrative Observatory (2025), "NarrQuest tone manifesto: reclaiming narrative sovereignty in the post-education era", *Tone Uprising J. Narrative Sovereignty*, **1**(1), 1-5. https://doi.org/ 10.5281/zenodo.16754702

Patel, K., Joshi, K. and Mehta, R. (2021), "Real-time vehicle detection and classification using YOLOv2 in intelligent transportation systems", *J. King Saud Univ. Comput. Inform. Sci.*, **33**(1), 12-19. https://doi.org/10.1016/j.jksuci.2020.04.001

Prins, C. (2004), "A simple and effective evolutionary algorithm for the vehicle routing problem", *Comput. Operat. Res.*, **31**(12), 1985-2002. https://doi.org/10.1016/S0305-0548(03)00158-8

Reinelt, G. (1991), "TSPLIB—A traveling salesman problem library", *ORSA J. Comput.*, **3**(4), 376-384. https://doi.org/10.1287/ijoc.3.4.376

Saxena, A., Singh, K. and Verma, P. (2024), "Intelligent IoT-based traffic navigation over 5G edge networks: A comprehensive review", *J. Netw. Comput. Appl.*, 103004. https://doi.org/10.1016/j.jnca.2024.103004

Turan, A. H., Gungor, V. C. and Tuna, G. (2023), "Multi-objective vehicle routing problem optimization using hybrid particle swarm optimization and artificial bee colony algorithms", *Alexandria Eng. J.*, **62**(3), 1987-1999. https://doi.org/10.1016/j.aej.2022.07.037

Xiong, Y., Zhao, L., Han, B. and Zhang, M. (2019), "A deep reinforcement learning based approach for adaptive traffic signal control", *Inform. Sci.*, **501**, 332-343. https://doi.org/10.1016/j.ins.2019.05.015

Yokogawa Electric Corporation. (n.d.), *Electronic Logbook (eLogBook)*, Yokogawa. https://www.yokogawa.com/solutions/solutions/asset-operations-and-optimization/operator-effectiveness/electronic-logbook-elogbook/

Zhang, R., Wang, Y. and Li, X. (2022), "Game-theoretic approaches for competitive logistics networks", *Transp. Res. Part B Methodol*., **155**, 1-20. https://doi.org/10.1016/j.trb.2021.12.005

CC

Appendix 1: Python code and Chinese annotations (brute force method to solve TSP)

```
    # 1. Define nodes and their longitude and latitude
    locations = {
‘ Fushan_JHS ‘: (22.6491, 120.3127),
‘STUST’: (22.6612, 120.3256), ‘NSYSU’: (22.6273, 120.2655), ... }

    # 2. Convert coordinates to distance matrix (Euclidean scaling )
    def euclidean_km_matrix(df):
      lat = df[“lat”].values
      lon = df[“lon”].values
      km_lat = lat * 111
      km_lon = lon * (111 * np.cos(np.radians(np.mean(lat))))
      coords_km = np.column_stack((km_lat, km_lon))
      return squareform(pdist(coords_km))

    # 3. Brute force method to list all paths
    from itertools import permutations
    ...for perm in permutations(others): path = [start] + list(perm) + [start] distance = sum(...)

    # 4. Use Haversine formula to calculate the real distance
    def haversine_ km ( p1, p2):
    lat1, lon1 = np.radians (...)
    ... return c * r # Return spherical distance

    # 5. Find the best path and plot it
    plt.plot (...)
    plt.text (...)
    plt.savefig (...)
```

The above code can be executed in Google Colab , Jupyter Notebook or VSCode. The complete process can be verified and visualized with the main text and Excel spreadsheet .

Appendix 2: TSP manual operation and verification tutorial

The purpose of this tutorial is to help readers without programming background to manually calculate the total TSP distance and understand the overall process .

Step 1 : Query location coordinates
Fushan_JHS and STUST)

from Table 1.

Example:

Fushan_JHS = (22.6491, 120.3127)

STUST = (22.6612, 120.3256)

Step 2: Calculate the distance between two places

Method 1 : Proportional conversion method

1. Latitude difference: $\Delta\varphi = |22.6491 - 22.6612| = 0.0121$
2. Longitude difference: $\Delta\lambda = |120.3127 - 120.3256| = 0.0129$
3. Mean latitude $\varphi = (22.6491 + 22.6612)/2 \approx 22.655154$. Conversion: $\Delta\varphi \times 111 \approx 1.3431$ km
4. $\Delta\lambda \times 111 \times \cos(\varphi) \approx 1.3278$ km
5. Total distance $\approx \text{sqrt}(1.3431^2 + 1.3278^2) \approx 1.88$ km

Method 2: Haversine formula

1. Convert latitude and longitude to radians ($° \times \pi / 180$)
2. Apply the formula: $d = 2r \times \arcsin(\sqrt{[\sin^2((\Delta\varphi)/2) + \cos\varphi_1 \times \cos\varphi_2 \times \sin^2((\Delta\lambda)/2)]})$

where r = 6371 km, which can be calculated to be approximately 1.88 km

Step 3 : Establish the total path distance

Take any path in the optimal path (for example, A→B→C→ ... →A), repeat step 2 and add up the distances segment by segment.

Example:

Fushan_JHS → STUST → ... → Fushan_JHS Total distance = AB + BC + CD + ... + KA

In this way, you can manually check whether the overall TSP total distance is consistent with the result obtained by the program.

Appendix 3

```
# Two cars + urgent orders + real scheduling simulation
import pandas as pd
import numpy as np
from scipy.spatial .distance import pdist , squareform

# Define location coordinates
locations = {
‘ Fushan_JHS ‘: (22.6491, 120.3127),
‘STUST’: (22.6612, 120.3256),
‘NSYSU’: (22.6273, 120.2655),
‘NUK’: (22.7291, 120.2857),
‘ N.KUHT ‘: (22.6901, 120.2985),
‘CSU’: (22.6407, 120.3564),
‘NKNU’: (22.6424, 120.3214),
‘ISU’: (22.7396, 120.2858),
‘NKUST_J’: (22.6045, 120.2996),
‘NKUST_1’: (22.6042, 120.3068),
‘WZU’: (22.6245, 120.3015),
```

```
‘ Emergency_School ‘: (22.6700, 120.3100)
}

df_locations = pd.DataFrame ( locations ) .T.reset_index ( )
df_ locations.columns = [“name”, “ lat “, “ lon “]

# European kilometers conversion
def euclidean_km_matrix(df):
      lat = df[“lat”].values
      lon = df[“lon”].values
      km_lat = lat * 111
      km_lon = lon * (111 * np.cos(np.radians(np.mean(lat))))
      coords_km = np.column_stack((km_lat, km_lon))
      return squareform(pdist(coords_km))

distance_matrix = euclidean_km_matrix(df_locations)
travel_time_matrix = distance_matrix * 2

n = len(df_locations)

#Number of boxes required at each location and time window (minutes)
time_windows = {
1: (540, 690),
2: (510, 630),
3: (570, 720),
4: (540, 660),
5: (600, 750),
6: (480, 600),
7: (570, 690),
8: (480, 600),
9: (480, 630),
10: (540, 720),
11: (570, 660)
}
demand = {
1: 2,
twenty one,
3: 1,
4: 2,
5: 1,
6: 1,
7: 1,
8: 1,
9: 1,
10: 2,
11: 1
```

```
}
start_time = 480 # 08:00
capacity = 10
service_time = 15

# Initially assign tasks to the two vehicles
group_A = {6, 8, 9, 2, 1, 5}
group_B = {4, 3, 7, 10}

#Schedule each car
def schedule_vehicle ( group , insert_emergency =False):
unvisited = set(group)
current = 0
    time_now = start_time
    load_now = sum(demand[i] for i in group)
    route = [0]
    arrival_times = [start_time]
    inserted = False

    while unvisited:
        feasible = []
        for candidate in unvisited:
            travel = travel_time_matrix[current][candidate]
            arrival = time_now + travel
            window_start, window_end = time_windows[candidate]
            if arrival <= window_end:
                wait_time = max(0, window_start - arrival)
                total_time = travel + wait_time + service_time
                feasible.append((candidate, arrival, total_time))
        if not feasible:
            break
        next_city = min(feasible, key=lambda x: x[1])[0]
        travel = travel_time_matrix[current][next_city]
        arrival = time_now + travel
        window_start, _ = time_windows[next_city]
        arrival = max(arrival, window_start)
        time_now = arrival + service_time
        load_now -= demand[next_city]
        route.append(next_city)
        arrival_times.append(arrival)
        unvisited.remove(next_city)
        current = next_city

        if insert_emergency and not inserted and time_now >= 570:
            em_travel = travel_time_matrix[current][11]
            em_arrival = time_now + em_travel
```

```
                em_window_start, em_window_end = time_windows[11]
                if em_arrival <= em_window_end:
                    em_arrival = max(em_arrival, em_window_start)
                    time_now = em_arrival + service_time
                    route.append(11)
                    arrival_times.append(em_arrival)
                    inserted = True
                    current = 11

        travel_back = travel_time_matrix[current][0]
        time_now += travel_back
        route.append(0)
        arrival_times.append(time_now)

        return route, arrival_times

    route_A, arrival_times_A = schedule_vehicle(group_A, insert_emergency=True)
    route_B, arrival_times_B = schedule_vehicle(group_B, insert_emergency=False)

    # Final
    (total_time_A := arrival_times_A[-1] - start_time, total_time_B := arrival_times_B[-1] -
start_time)
```