Studying the Practicality of Theoretically Proven Online Routing/Scheduling Algorithms for Autonomous Logistics Systems

Su Dong, Parimala Thulasiraman, Pak Ching Li
Department of Computer Science
University of Manitoba
Winnipeg, Canada
e-mail: dongs@myumanitoba.ca, parimala.thulasiraman@umanitoba.ca, ben.li@cs.umanitoba.ca

***Abstract*—Online routing is a complex problem in autonomous logistics systems such as online delivery applications. The problem data is not known in advance. Therefore, finding efficient solutions to schedule and route for online problems is difficult. Practitioners rely on historical data to propose data-driven machine learning solutions, while theoreticians propose strong, theoretically bound, non-data-driven methods using competitive analysis. The question this paper attempts to answer is whether we can implement theoretical algorithms for logistics systems in an uncertain environment. In this paper, we do this by studying the Online Travelling Salesperson Problem (OLTSP). We examine the gap between the theoretical performance guarantees of the proposed algorithms for OLTSP and their observed performance in practice. We perform various experiments and show that competitive algorithms perform significantly better in practice than their worst-case guarantees suggest.**

Keywords-Online Travelling Salesperson Problem; Intelligent Transportation System; Competitive Algorithms.

I. INTRODUCTION

Recent advances in automation have made connected autonomous vehicles and aerial drones a promising solution for modern logistics systems. One of the key components of logistics systems is an Intelligent Transportation System (ITS). An ITS comprises two main components: a road network and a vehicular network or Internet of Vehicles (IoV). An IoV consists of vehicles interconnected as a distributed network. An IoV performs tasks such as delivering and collecting goods (e.g., online food delivery) within a city without human intervention. Such urban logistics is more complex and time-sensitive. Designing efficient online real-time scheduling strategies for IoV applications is a difficult problem.

In a typical logistics scenario, a service provider operates autonomous vehicles or drones to perform delivery and pickup tasks within a fixed service region. The region consists of predefined locations such as warehouses, customer sites, and pickup points. These locations are connected by routes with known costs that represent travel time or distance. Service requests are released over time. Each request specifies a location and a release time. A request cannot be served before its release time. At any moment, vehicles or drones only know the requests that have already been released. Information about future requests is unavailable. As a result, routing decisions must be made based on incomplete information and may need to be updated when new requests appear. The objective is to serve all requests while keeping the total travel cost as low as possible. This setting can be naturally modeled by the OLTSP that captures the key challenges of online logistics systems, where requests arrive over time and routing decisions must be made without knowledge of future demands.

Ausiello et al. [1] proposed the OLTSP. In this problem, a single salesperson is assumed. In our work, we interpret the salesperson as a single autonomous vehicle. The OLTSP, in general, can be extended or adapted to multi-vehicle systems through decomposition or assignment strategies.

There are two versions of OLTSP, Homing OLTSP (H-OLTSP) and Nomadic OLTSP (N-OLTSP) [1]. Although in both versions the goal is to minimize the total completion time to serve all the presented requests, in H-OLTSP, the server must return to the origin after completing all requests. In this paper, we focus on the H-OLTSP on general metric spaces. In this model, the server starts from a designated depot and must return to the same depot after completing all requests. This setting is practical for real-world applications.

In recent years, data-driven methods have been key in addressing real-world routing and logistics problems, making machine learning and other heuristic approaches attractive in practice. However, these methods usually lack proven performance guarantees. They are also not theoretically proven. In automated logistics systems, service regions may change over time, and historical data collected from previous regions may no longer be accurate or useful. In such settings, data-driven approaches may perform poorly.

Online algorithms for H-OLTSP have been theoretically analyzed using competitive analysis, and are therefore often referred to as competitive algorithms, but have not been experimentally explored. *This method compares the performance of the online algorithm against an offline optimal algorithm* in which the input sequence is known in advance. The competitive analysis of an algorithm provides guarantees on solution quality under all possible input sequences in the worst-case. This property is important and reliable when there is insufficient data. As a result, competitive algorithms offer a robust alternative when data is limited, outdated, or unavailable.

Although the performance guarantees of competitive algorithms may appear pessimistic, they are derived from worst-case analysis rather than typical behavior. It is unclear how often such worst-case scenarios occur in practical applications. This raises an important question about the practical perfor-

mance of competitive algorithms.

Ausiello et al. [1] proposed online routing algorithms for scheduling requests: the *Plan At Home* (PAH) algorithm for the H-OLTSP on general metric spaces and the *Greedily Travelling between Requests* (GTR) algorithm for the N-OLTSP. Based on these competitive algorithms, our contribution is as follows:

- Provide a formal definition and detailed description of the GTR algorithm for H-OLTSP, following the adaptation stated by the authors [1].
- Experimentally evaluate scheduling strategies for routing requests in PAH and GTR.
- Study the gap between theoretical worst-case guarantees and observed empirical performance.

To the best of our knowledge, PAH and GTR are the only algorithms designed for H-OLTSP on general metric spaces, without additional assumptions or restrictions.

The remainder of this paper is organized as follows. Section II describes the problem model of the H-OLTSP. Section III presents the competitive algorithms studied in this paper. Section IV details the experimental data generation. Section V reports the experimental results. Section VI discusses these results. Section VII explains limitations. Section VIII concludes the paper and provides directions for future work.

II. Problem Model

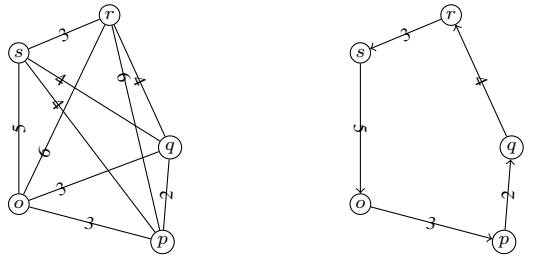
In the classical Travelling Salesperson Problem (TSP), given a set of locations, the goal is to find the shortest Hamiltonian cycle that visits each location exactly once. This is an offline problem where all the locations to be visited are given as input. Computing such an optimal tour is NP-hard on general metric spaces. As a result, computing an optimal solution for the offline version of H-OLTSP on general metric spaces is computationally intractable unless $P = NP$. The formal definition of H-OLTSP is given in Definition 1 [1]. It models a moving agent, called the server, that travels in a metric space to serve online requests.

Definition 1. *H-OLTSP*

The input to the H-OLTSP consists of the following:

- *A metric space (M, d), where M is a set of locations or points and $d : M \times M \to \mathbb{R}_{\geq 0}$ is a distance function satisfying the standard metric properties: non-negativity, identity, symmetry, and the triangle inequality.*
- *A designated starting point $o \in M$, called the* origin*, where the server is initially located at time zero.*
- *An online sequence of requests $Q = \{(t_1, p_1), (t_2, p_2), \ldots, (t_n, p_n)\}$, where each $p_i \in M$ is a requested location and each $t_i \in \mathbb{R}_{\geq 0}$ is the release time of the request. The sequence is ordered such that $t_i \leq t_{i+1}$ for all $1 \leq i < n$, and request (t_i, p_i) is revealed to the algorithm only at time t_i.*

The server moves through the metric space at unit speed, so traveling between any two locations x and $y \in M$ requires exactly $d(x, y)$ units of time. A request (t_i, p_i) is considered served if the server visits the location p_i at some time $t \geq t_i$.



(a) Metric Space (b) Offline Optimal Solution

Figure 1. An Example OLTSP Instance

The objective is to guide the server, in an online fashion, starting from the origin, to serve all requests in Q and return to the origin after all requests are completed. The goal is to minimize the ***completion time****, defined as the total time that elapses from the start (time zero) until all requests have been served and the server has returned to the origin. This includes both the time spent traveling and any time the server remains idle.*

III. Competitive Online Algorithms

To illustrate these algorithms, we consider an example of H-OLTSP: $[(M, d), o, Q]$. The request sequence Q is given by $\{(0, r), (2, p), (4, q), (10, s)\}$. The metric space (M, d) is defined by the graph shown in Figure 1 (a). In this graph, M represents the set of nodes, o is the origin node, and d is a distance function mapping pairs of nodes to their distances, with the corresponding values specified by the edge labels. An offline optimal solution for this instance is shown in Figure 1 (b).

A. PAH Algorithm

In the PAH algorithm, the server always schedules its route at the origin. Several special cases may arise during the algorithm's execution. These cases are described in Definition 2 [1].

Definition 2. *Main Logic of PAH*

- ***Case 1*** *The server is at the origin:*
  - *The server schedules or reschedules a route that starts at the origin, visits all remaining released but unserved request locations exactly once, and returns to the origin at the end. It immediately starts moving along this route.*
- ***Case 2*** *If the server is not at the origin and one or more new requests are released, the server decides whether or not to return to the origin to reschedule based on its location and the locations of new requests.*
  - ***Case 2.1*** *All newly released requests are located at positions whose distances from the origin are no greater than the distance from the origin to the server's current location at the moment they are released:*
    - *These new requests are temporarily ignored, and the server continues its current route. They are*

scheduled when the server next arrives at the origin (re-entering Case 1).

- ***Case 2.2*** *Otherwise, the server returns to the origin by a shortest route (re-enter Case 1).*

We illustrate the PAH algorithm on the example instance in Figure 1. At time 0, the server is at the origin and the request $(0, r)$ is released. Since the server is at the origin (**case 1**), it computes a tour $o \to r \to o$ and moves from o toward r.

At time 2, the request $(2, p)$ is released. The server is not at the origin, so **case 2** applies. Since $d(p, o) > R_d(server, o)$, **case 2.2** applies, and the server returns to the origin to recompute a tour where $R_d(server, o)$ is the distance of the shortest route from the server back to the origin.

The server arrives at the origin at $t = 4$, when the request $(4, q)$ is released. As the server is at the origin (**case 1**), it computes the tour $o \to p \to q \to r \to o$. The server moves from o to p from time 4 to 7, and from p to q from time 7 to 9. Thus, p is served at time 7 and q at time 9.

At time 10, the request $(10, s)$ is released. Since $d(s, o) > R_d(server, o)$, **case 2.2** applies and the server returns to the origin via server's location $\to q \to o$, arriving at $t = 14$. At time 14, the server computes the tour $o \to s \to r \to o$. It moves from o to s from time 14 to 19, from s to r from time 19 to 22, and from r back to o from time 22 to 28. Hence, the completion time is 28.

Scheduling Strategies: The PAH algorithm requires a scheduling algorithm to compute routes for serving requests, where each route is nothing but a TSP tour. In this paper, we consider two scheduling methods. The first method is dynamic programming [2], which can compute optimal TSP tours. The second is the Christofides heuristic [3]. Since a TSP tour is a cycle, the origin has two adjacent locations in the tour. When starting from the origin, the server can move toward either one. In this work, we let the server choose the one that is closer to the origin, as a simple and reasonable modeling choice consistent with routing decisions commonly used in autonomous logistics systems.

B. GTR Algorithm

The GTR algorithm guides the server to schedule or reschedule a route every time new requests are released. Ausiello et al. [1] proposed the GTR algorithm for N-OLTSP, and stated that the GTR can also be applied to the H-OLTSP by modifying the route so that it ends at the origin. However, the authors [1] did not provide a formal definition or pseudocode for the H-OLTSP variant. To make the algorithm well defined in our setting, we present an explicit formulation of the core logic of GTR for H-OLTSP, based on the descriptions and proofs given in [1]. While our formulation is intended to closely follow the original design philosophy, minor differences in interpretation or implementation details may arise, due to the absence of an explicit H-OLTSP specification in [1]. For completeness and reproducibility, the exact version of GTR used in our experiments is stated in Definition 3.

Definition 3. *Main logic of GTR for H-OLTSP*
At any time instant, when one or more new requests are released, the server immediately enters Case 2; if no new requests are released, Case 1 applies.
Case 1: There are no new requests.

- ***Case 1.1:*** *A route is currently scheduled and being followed (The route contains all released but unserved request locations):*
  - *The server continues moving along the route.*
- ***Case 1.2:*** *No route is currently being followed (i.e., Case 1.1 does not apply):*
  - *The server remains idle.*

Case 2: One or more new requests are released.

- ***Case 2.1:*** *The server is at a location (point).*
  - *The server computes a route starting from its current location that visits all remaining released but unserved request locations exactly once and returns to the origin at the end. It immediately begins moving along the route.*
- ***Case 2.2:*** *The server's current position lies on the shortest path (with respect to the underlying metric) between two locations, denoted by x and y.*
  - ***Case 2.2-a:*** *Each of x and y is either the origin or a request location that has been released at or before the current time.*
    ⇒ The server computes two candidate routes that begin at its current location, one that first visits x and one that first visits y, each then visiting all other released but unserved request locations exactly once and returning to the origin at the end. It immediately starts moving along the shorter route, breaking ties arbitrarily.
  - ***Case 2.2-b:*** *Exactly one of x or y is the origin or a request location that has been released at or before the current time.*
    ⇒ Identify the location $l_p \in \{x, y\}$ that is either the origin or a request location that has been released at or before the current time. The server computes a route that begins at its current location, first visits l_p, then visits all other released but unserved request locations exactly once and returns to the origin at the end. It immediately starts moving along this route.

To illustrate Definition 3, we apply GTR to the example instance of Figure 1. At time 0, the server is at the origin o. The request $(0, r)$ is released, so the server computes the route $o \to r \to o$ and starts moving along it (**case 2.1**).

At time 2, the request $(2, p)$ is released while the server is between the nodes o and r (**case 2.2**). The server computes two candidate routes, $P(s) \to r \to p \to o$ and $P(s) \to o \to p \to r \to o$, and starts moving along the shorter route $P(s) \to r \to p \to o$, where $P(s)$ is the position of the server.

At time 4, the request $(4, q)$ is released. The server again compares two possible routes and follows the shorter one, which is $P(s) \to r \to q \to p \to o$. The server arrives at r at time 6 and arrives at q at time 10.

At time 10, the request $(10, s)$ is released. The server recomputes a route $q \to s \to p \to o$ and moves along this route (**case 2.1**). The server completes s at time 14, completes p at time 18, and arrives back at the origin at time 21. Hence, the completion time is 21.

Scheduling Strategies The GTR algorithm also requires a scheduling algorithm to compute routes. Since GTR schedules immediately, each computed route starts from the server's current location, serves all currently active requested locations, and returns to the origin at the end. In this paper, we consider two scheduling methods for GTR. The first is an exact scheduling algorithm based on dynamic programming [2], which can compute optimal routes. The second is the Minimum Spanning Tree (MST) heuristic [1].

IV. Experimental Setting

In experiments, our objective is to evaluate the practical behavior of competitive algorithms in a way that is representative and broadly applicable. Rather than targeting a specific application or dataset, we aim to capture typical characteristics of automated logistics systems while retaining full control over instance difficulty and structure.

To model the spatial component of the problem, we use benchmark instances from TSPLIB [4] to induce discrete metric spaces. Each TSPLIB instance defines a finite set of locations with pairwise distances, which can be naturally interpreted as a service region in an automated logistics setting, such as an area served by autonomous delivery vehicles or drones. In many practical applications, service requests are typically modeled as originating from a finite set of predefined locations within the service region, and any such location may issue requests. To reflect this assumption without introducing spatial bias, requested locations are selected uniformly at random from the set of locations in the induced metric space.

The temporal component of request generation is handled differently. In real automated logistics systems, requests are rarely issued uniformly over time. Instead, demand often exhibits temporal concentration and peak periods. To approximate this behavior in a controlled manner, we generate request release times using a normal distribution. By varying the standard deviation, we control how closely release times are clustered, allowing us to model different levels of temporal congestion and overlap among active requests.

A key design choice is the use of a fixed release time interval. If request times were sampled over an unbounded or excessively long horizon, the gaps between successive requests could become large. In such cases, the server might complete one request before the next is released. This would weaken the need for online decision-making and reduce performance differences among algorithms, potentially causing their behavior to converge. Fixing the time interval prevents this effect and ensures that increasing the number of requests leads to greater scheduling difficulty.

This choice also aligns with real-world intuition. A fixed interval can be interpreted as a realistic operational window, such as several hours within a day. In practice, different time windows may exhibit different demand patterns, which corresponds to sampling from distributions with different parameters over comparable time spans. Our approach captures this structure while keeping the experimental setting controlled and comparable across instances.

Algorithm 1 H-OLTSP Instance Generation

Require: (I_{TSP}, σ, N)
- A symmetric TSPLIB instance I_{TSP}.
- Standard deviation σ.
- Total number of requests N.

$M \leftarrow$ the set of all locations in I_{TSP}
$d \leftarrow$ the distance function induced by I_{TSP}
$o \leftarrow$ a location chosen uniformly at random from M
$Q \leftarrow [\,]$ ▷ empty list
$d_{\text{median}} \leftarrow$ the median distance over all distinct pairs of locations in M
$T_{\max} \leftarrow 10 \cdot d_{\text{median}}$
$\mu \leftarrow 5 \cdot d_{\text{median}}$
$T \leftarrow$ a list of N integers sampled from a truncated normal distribution $\mathcal{N}(\mu, \sigma^2)$ restricted to $[0, T_{\max}]$ (duplicates allowed)
$P \leftarrow$ a list of N locations sampled uniformly at random from M (duplicates allowed)
Sort the list T in increasing order
for $i \in \{1, \ldots, N\}$ **do**
    Append request $(T[i], P[i])$ to Q
end for
return $\{(M, d), o, Q\}$

The instance generation procedure is described in detail in Algorithm 1. Request release times are generated within a fixed interval $[0, T_{max}]$, where $T_{max} = 10 \times d_{median}$ and d_{median} denotes the median distance between locations in the metric space M. The median distance provides a representative travel scale that is less sensitive to extreme values and thus offers a stable reference for relating spatial distances to temporal release patterns.

In our experiments, we consider four normal distributions with the same mean μ, which is fixed at the center of the interval $[0, T_{max}]$, and different values of the standard deviation σ. Specifically, we use $\sigma = T_{max}$, $\frac{T_{max}}{3}$, $\frac{T_{max}}{6}$, and $\frac{T_{max}}{10}$ to model varying degrees of temporal concentration. The total number of requests is set to $N = 5, 7, 9, 11, 13, 15$, and 17. Due to space limitations, we use two symmetric TSPLIB instances, berlin52 and gr202, obtained from a publicly available Kaggle dataset [5], to induce the discrete metric spaces.

For each configuration of the input parameters (I_{TSP}, σ, N), five independent instances are generated for experimental evaluation. The experiments are implemented in Python 3.11 and executed on a machine equipped with an Intel i5-13500H processor. To mitigate variability arising from system and hardware conditions, the execution time of each algorithm on a given instance is reported as the average over five independent runs.

TABLE I. AVERAGE COMPLETION TIME.

berlin52							
N	5	7	9	11	13	15	17
PAH-DP	6201.45	6662.00	7400.25	7526.50	7837.05	8296.20	8638.90
PAH-Chris	6216.30	6713.95	7516.80	7719.50	8022.45	8505.85	8804.20
GTR-DP	5439.30	5863.20	6352.30	6346.10	6569.80	6773.50	7121.45
GTR-MST	5457.15	5908.40	6427.60	6498.80	6770.95	7172.95	7392.25
Offline-Optimal	4838.10	5119.80	5626.55	5686.75	5798.10	6069.05	6224.10
gr202							
N	5	7	9	11	13	15	17
PAH-DP	14192.90	16580.90	17164.55	18010.15	18600.35	20351.60	19455.75
PAH-Chris	14272.40	16755.00	17503.75	18531.80	19247.70	21081.65	19968.05
GTR-DP	12226.50	14325.25	14377.55	15331.10	15403.90	17548.65	16237.70
GTR-MST	12314.30	14525.90	14650.65	15823.35	15735.60	18478.30	16885.60
Offline-Optimal	10535.15	12293.15	12593.55	13421.50	13735.55	15302.20	14539.00

TABLE II. RATIO.

	Min.	Mean	Median	Max.	Std.	Competitive Ratio
PAH-DP	1.0585	1.3405	1.3296	1.7692	0.1327	2(tight)
PAH-Chris	1.0585	1.3676	1.3522	1.7740	0.1399	3
GTR-DP	1.0042	1.1357	1.1247	1.3841	0.0615	2.5 †
GTR-MST	1.0226	1.1649	1.1544	1.4655	0.0725	unknown

† The competitive ratio is stated by Ausiello et al. [1], but a formal proof is not provided.

V. EXPERIMENTAL RESULTS

We refer to the PAH algorithm using dynamic programming as PAH-DP, and to the PAH algorithm using the Christofides heuristic as PAH-Chris. We refer to the GTR algorithm using dynamic programming as GTR-DP, and to the GTR algorithm using the MST heuristic as GTR-MST.

Table I presents average completion times for varying numbers of requests N, with separate sub-tables for instances generated from berlin52 and gr202; all values are rounded for presentation. From Table I, we observe that, across different values of N, the average completion time of all algorithms is higher than the corresponding offline optimal average completion time, and remains below twice the offline optimal average completion time for all evaluated cases. In comparisons between dynamic programming and heuristic variants within the same algorithm family (PAH-DP vs. PAH-Chris and GTR-DP vs. GTR-MST), we observe that the dynamic programming variants generally achieve a lower average completion time for most values of N in the table. Among online decision making methods, GTR yields lower average completion times than PAH when comparing corresponding variants (GTR-DP vs. PAH-DP and GTR-MST vs. PAH-Chris).

Table II reports the ratio between the completion time of each algorithm and the offline optimal completion time; all reported values are rounded for presentation. Lower values indicate solutions closer to the optimal. For all algorithms with known competitive ratios, the observed ratios are well below their competitive ratio.

Table III presents average execution times for varying N, with four sub-tables corresponding to instances whose release times are generated from a normal distribution with different σ values; all values are rounded for presentation. Here, execution time refers to the simulated time required by an algorithm to complete a single H-OLTSP instance. This measure provides an approximate indication of the total computational effort incurred over the course of the instance. Although execution time does not directly measure per-decision latency, it can offer insight into an algorithm's overall responsiveness. In particular, a longer execution time suggests that scheduling and rescheduling decisions may require greater computational effort as requests arrive, while shorter execution times indicate more efficient decision-making in dynamic settings.

Overall, for algorithms that rely on dynamic programming, instances with smaller σ and larger N tend to have higher average execution times in most cases. This is because more concentrated release times typically lead to a larger number of simultaneously active requests, which increases scheduling difficulty. For polynomial time scheduling methods (Christofides and MST heuristics), this trend is less apparent in our experiments, partly due to the relatively small instance sizes considered. However, in online scheduling, execution time is influenced not only by scheduling complexity but also by the frequency of rescheduling. A notable special case is GTR-DP, whose average execution time increases sharply at $N = 17$ (third sub-table of Table III). In some instances, newly released requests repeatedly interrupt the current route, forcing frequent rescheduling, as GTR recomputes the entire routes whenever one or more requests are released. This behavior can have a particularly strong impact when an exact scheduler, such as dynamic programming, is used. Despite this, GTR-DP still exhibits a lower average execution time than PAH-DP across the evaluated cases.

Graph operations are implemented using NetworkX [6]. The Christofides and MST heuristics rely more on graph operations, which introduce a small data conversion overhead. This overhead is more noticeable when the number of requests N is small, since only a few requests are scheduled at each decision point and complexity differences between methods are less pronounced. As a result, heuristic methods may exhibit longer average execution times than dynamic programming for some small values of N in Table III.

Table III also shows that GTR-DP consistently achieves lower average execution times than PAH-DP, and that GTR-MST consistently achieves lower average execution times than PAH-Chris in the reported results. A key reason lies in how the two algorithms handle scheduling and rescheduling. GTR reschedules immediately when new requests are released, so each scheduling decision is based on the set of requests that are active at that moment. In contrast, PAH only schedules or reschedules routes when the server is at the origin. When new requests arrive, and PAH decides that rescheduling is needed, it must first return to the origin before computing a new route. During this return period, additional requests may be released. As a result, when PAH eventually performs rescheduling, it may need to consider a larger set of active requests than GTR. This can increase the effort required for each tour computation and tends to result in higher overall execution time.

VI. DISCUSSION

Based on the experimental results, we observe that the evaluated competitive algorithms perform significantly better

TABLE III. AVERAGE EXECUTION TIME(MS).

$\sigma = T_{max}$							
	5	7	9	11	13	15	17
PAH-DP	21.61	35.67	40.41	148.75	292.94	237.89	818.47
PAH-Chris	22.66	39.77	43.34	169.73	51.32	121.31	96.75
GTR-DP	9.01	12.15	12.97	39.99	28.25	42.59	68.92
GTR-MST	9.50	13.80	14.75	26.99	25.65	31.64	36.61
$\sigma = \frac{T_{max}}{3}$							
N	5	7	9	11	13	15	17
PAH-DP	12.08	36.85	30.40	91.01	208.88	748.49	4010.96
PAH-Chris	12.53	37.67	27.45	79.65	45.68	66.71	77.40
GTR-DP	8.42	12.47	16.94	21.92	36.40	525.76	324.34
GTR-MST	9.22	13.88	17.61	20.04	24.51	36.87	34.42
$\sigma = \frac{T_{max}}{6}$							
N	5	7	9	11	13	15	17
PAH-DP	31.89	46.06	91.19	77.09	439.09	627.62	4118.77
PAH-Chris	33.33	47.10	74.14	36.32	39.48	51.43	88.51
GTR-DP	8.97	11.40	22.33	32.54	150.78	143.70	3063.06
GTR-MST	9.60	13.36	20.63	22.15	27.72	33.43	42.31
$\sigma = \frac{T_{max}}{10}$							
N	5	7	9	11	13	15	17
PAH-DP	13.26	32.89	40.72	86.19	664.54	1821.45	6171.44
PAH-Chris	14.82	32.10	19.51	37.61	43.57	90.51	65.95
GTR-DP	9.05	14.18	35.36	62.78	433.57	1764.86	1913.66
GTR-MST	10.25	14.98	18.75	22.79	31.03	42.59	44.01

in practice than their competitive ratios suggest. In all tested settings, the gap between solutions of these algorithms and the offline optimal solutions remains small. This indicates that competitive analysis provides a conservative bound, and that these algorithms can achieve much stronger performance under realistic conditions.

Among the evaluated methods, GTR consistently demonstrates strong practical performance when evaluated through paired comparisons against the corresponding PAH methods (GTR-DP vs. PAH-DP and GTR-MST vs. PAH-Chris). Its strategy of immediately scheduling or rescheduling routes allows it to incorporate new requests quickly, which typically results in lower completion times in these paired comparisons. This behavior is particularly advantageous in delivery scenarios where a timely response to new orders is important. At the same time, our experimental results indicate that PAH performs better in practice than its competitive ratio suggests for the evaluated cases. However, its design requires the server to return to the origin before recomputing a route, which introduces additional travel costs in practical settings, especially when requests arrive frequently.

VII. LIMITATIONS

The experimental evaluation is subject to several limitations. First, the experiments are conducted on benchmark-based metric spaces induced from TSPLIB instances, while request sequences are synthetically generated. This design enables precise control over instance characteristics, such as request volume and temporal distribution, and facilitates systematic comparison across algorithms. However, it may not capture all aspects of demand patterns encountered in specific real-world logistics deployments, where request behavior can be influenced by application-specific factors.

Second, the implementation relies on external libraries for graph-related operations. These libraries are used to support correctness and reproducibility. At the same time, they introduce a small overhead due to data structure conversion between library representations and the internal data structures used by our algorithms. As a result, the reported execution times may include minor implementation-related overhead that is not intrinsic to the algorithms themselves.

Finally, the number of requests is limited to $N \leq 17$, since computing offline optimal solutions is computationally expensive. This enables comparison with the offline optimal objective value but restricts evaluation to small instances.

VIII. CONCLUSION AND FUTURE WORK

In this paper, we evaluated competitive algorithms in practical settings to assess their performance in autonomous logistics systems. The experimental results show that competitive algorithms perform well in practice, despite their conservative theoretical performance guarantees. In particular, GTR demonstrates promising practical applicability for autonomous logistics systems such as autonomous vehicles and drone delivery in IoV applications. Its ability to make effective online decisions without relying on historical data makes it well-suited for deployment in dynamic and uncertain environments.

Several directions remain for future work. One important direction is to evaluate these competitive algorithms on real-world data from automated logistics systems, such as electric vehicle or drone delivery, enabling direct comparison with data-driven approaches. Another direction is to extend the problem model with additional practical constraints arising in autonomous logistics systems.

REFERENCES

[1] G. Ausiello, E. Feuerstein, S. Leonardi, L. Stougie, and M. Talamo, "Algorithms for the on-line travelling salesman," *Algorithmica*, vol. 29, no. 4, pp. 560–581, 2001. DOI: 10.1007/s004530010071. Accessed: Jan. 21, 2026. [Online]. Available: https://link-springer-com.uml.idm.oclc.org/article/10.1007/s004530010071.

[2] R. Bellman, "Dynamic programming treatment of the travelling salesman problem," *Journal of the ACM (JACM)*, vol. 9, no. 1, pp. 61–63, Jan. 1962. DOI: 10.1145/321105.321111.

[3] N. Christofides, "Worst-case analysis of a new heuristic for the travelling salesman problem," *Operational Research Forum*, vol. 3, no. 1, p. 20, Mar. 2022. DOI: 10.1007/s43069-021-00101-z.

[4] G. Reinelt, "Tsplib—a traveling salesman problem library," *ORSA Journal on Computing*, vol. 3, no. 4, pp. 376–384, 1991. DOI: 10.1287/ijoc.3.4.376. Accessed: Jan. 21, 2026. [Online]. Available: https://doi.org/10.1287/ijoc.3.4.376.

[5] "Tsplib symmetric dataset," Accessed: Jan. 21, 2026. [Online]. Available: https://www.kaggle.com/datasets/hiimhoanglam/tsplib-symmetric.

[6] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using networkx," in *Proceedings of the 7th Python in Science Conference (SciPy2008)*, G. Varoquaux, T. Vaught, and J. Millman, Eds., Pasadena, CA, USA, 2008, pp. 11–15.