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

Su Dong [1] Parimala Thulasiraman Pak Ching Li

Presenter: Su Dong[1]

[1] Computer Science, University of Manitoba(Canada)

Contact Email: dongs@myumanitoba.ca

University of Manitoba

IARIA

Su Dong

Su Dong received the master's degree in Computer Science from the University of Manitoba, Canada in 2025. He is currently a doctoral student majoring in Computer Science at the University of Manitoba.

His research investigates combinatorial optimization under enhanced structural and informational complexity. This includes online routing models, where decisions must be made without full knowledge of the entire problem input, and connectivity-constrained graph problems, where feasible solutions are required to satisfy additional structural properties. He studies these problems through theoretical analysis and systematic experimental evaluation to characterize formal performance guarantees and practical algorithmic behavior.

Table of Contents

Introduction

Online routing is a fundamental challenge in autonomous logistics systems such as online delivery platforms. Future requests are not known in advance, which makes scheduling and routing decisions inherently difficult.

- In practice, data driven and machine learning approaches are commonly used, relying on historical data.
- In theory, competitive analysis provides strong performance guarantees through non data driven competitive algorithms.

This work studies whether theoretical competitive algorithms can be effective in uncertain, practical environments.

- We focus on the Online Travelling Salesperson Problem (OLTSP) and compare theoretical guarantees of competitive algorithms with their empirical performance.

Problem Model - OLTSP

Autonomous Logistics Systems and the OLTSP:

- Recent advances in automation have made connected autonomous vehicles and aerial drones a promising solution for modern logistics systems. Such systems are supported by Intelligent Transportation Systems (ITS), which integrate road networks with vehicular networks (Internet of Vehicles, IoV).
- In logistics, autonomous vehicles perform delivery tasks within a fixed service region consisting of predefined locations connected by routes with known travel costs. Service requests are released over time, and future requests are unknown.
- Routing decisions must therefore be made in real time under incomplete information and may need to be revised as new requests appear. The objective is to serve all requests while minimizing total travel cost.
- This setting is naturally modeled by the OLTSP, which captures the core challenges of autonomous logistics systems.

We therefore study competitive algorithms for OLTSP to assess their potential in designing robust online routing strategies for autonomous logistics systems. In this paper, we focus on the Homing OLTSP (H-OLTSP).

Problem Model - Definition of H-OLTSP

The input for an instance of H-OLTSP consists 3 things:

1. A metric space (M, d):
   - M is a set of locations;
   - $d : M \times M \to \mathbb{R}_{\geq 0}$ is a distance function satisfying the standard metric properties: non-negativity, identity, symmetry, and the triangle inequality;
2. A designated starting point: $o \in M$, called the *origin*;
3. An online sequence of requests $Q = \{(t_1, p_1), (t_2, p_2), \ldots, (t_n, p_n)\}$
   - each $p_i \in M$ is a requested location;
   - 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 the request (t_i, p_i) is revealed to the algorithm only at time t_i;

The **objective** of the H-OLTSP is to guide a moving server(or agent) start from the origin, serve all requests in Q and return to origin so as to minimize the **completion time**, which consists of

- the total amount of time that costs from the start (time zero) until all requests have been served and return to origin.
- includes both the time spent traveling and any time the server remains idle.

[1]

Related Works

Ausiello et al. [1] initiated the study of the H-OLTSP using the following two algorithms that "solves" the problem for general metric spaces:

- Greedily Travelling between Request (GTR) algorithm [1];
- Plan at Home (PAH) algorithm;

The main theoretical results were:

- A 2-competitive (non-polynomial time) algorithm (PAH) for H-OLTSP [1];
- A 3-competitive (polynomial time) algorithm (PAH) for H-OLTSP [1];
- A 2.5-competitive (non-polynomial time) algorithm (GTR) for H-OLTSP [2];

[1] The GTR algorithm was originally proposed for the Nomadic OLTSP by the authors [1], they stated that it can be applied to the H-OLTSP by modifying the route to end at the origin.

[2] The competitive ratio is stated by the authors [1], but a formal proof is not provided.

PAH - Introduction

We now present the Plan-at-Home (PAH) algorithm for solving the H-OLTSP.

- It is a very simple algorithm that uses a TSP solver construct a schedule for visiting requested locations.

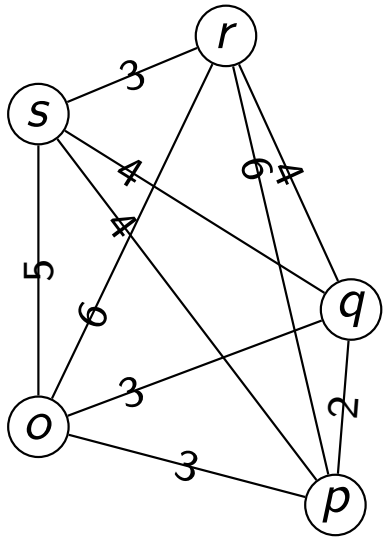
PAH - Main Logic

- **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).

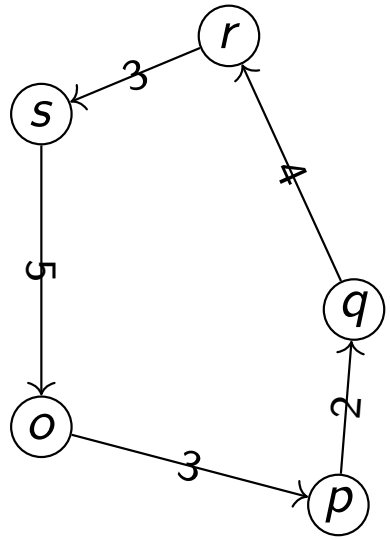
[1]

PAH - An Example of PAH Execution

List of Requests $\{(r, 0), (p, 2), (q, 4), (s, 10)\}$



(a) Metric Space



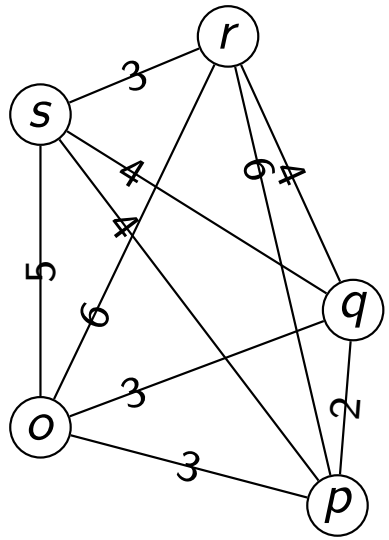
(b) Offline Optimal Solution

The metric space (M, d) defines all locations (M) that can be requested and the distance function (see (a)).

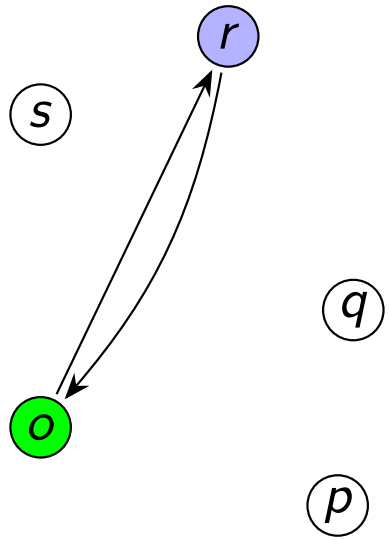
The offline optimal solution have completion time: $3 + 2 + 4 + 3 + 5 = 17$ (see (b))

PAH - An Example of PAH Execution

O: origin ● unserved request ● server's location



(a) Metric Space



(b) $t = 0$

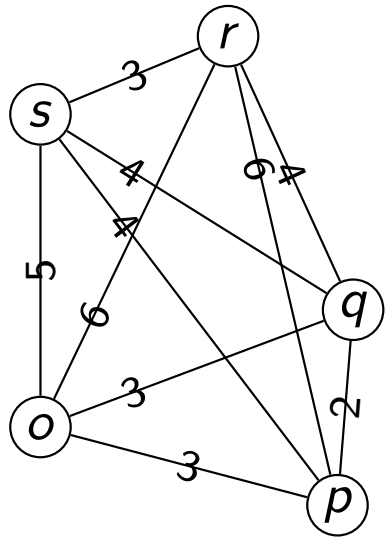
At time 0, server is located at origin and the request (r, 0) is released.

As the server is located at location (**case 1**), server schedule a route that visits all unserved requests and returns to origin (see (b)).

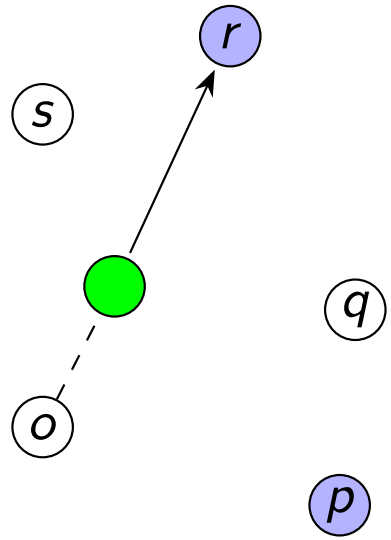
Hence, server moves from o towards r.

PAH - An Example of PAH Execution

O: origin ● unserved request ● server's location



(a) Metric Space

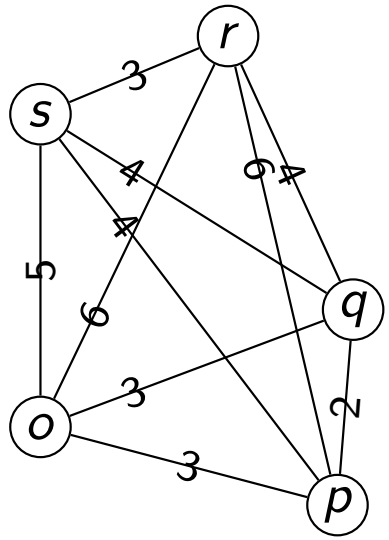


(b) $t = 2$

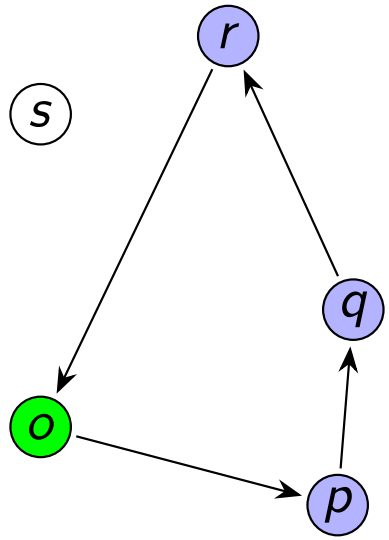
At time 2, the request $(p, 2)$ is released. As the server is not at the origin and a new request is released, **case 2** applies.
In case 2, since $d(p, o) > R_d(o, server)$, the **case 2.2** applied, where $R_d(o, server)$ is the distance of the shortest route from the server's position to the origin.
The server needs to return to the origin and compute a new route. The server starts to return to the origin.

PAH - An Example of PAH Execution

O: origin ● unserved request ● server's location



(a) Metric Space



(b) $t = 4$

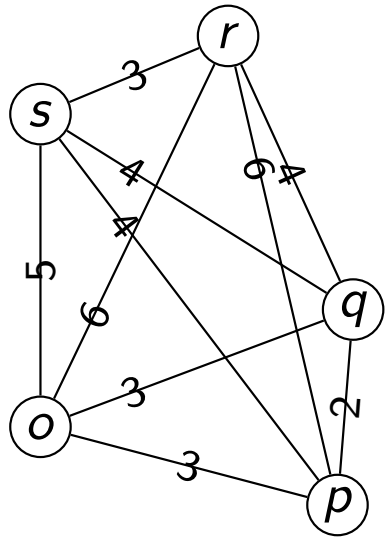
When the server arrive origin, $t = 4$, the request $(q, 4)$ is released, the server is already at origin (**case 1**), so it computes tour $o \to p \to q \to r \to o$ (see (b)).

From time 4 to 7, the server moves from o to p. And, from 7 to 9, the server moves from p to q.

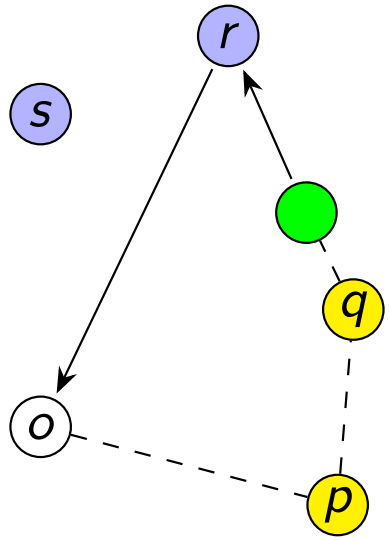
Hence, p is severed at time 7, and q is at time 9.

PAH - An Example of PAH Execution

O: origin unserved request served request server's location



(a) Metric Space



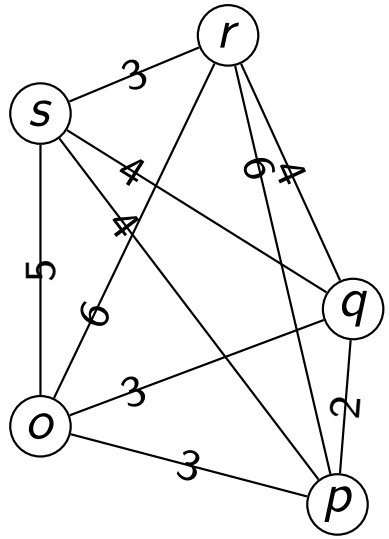
(b) $t = 10$

At time 10, request $(s, 10)$ is released.

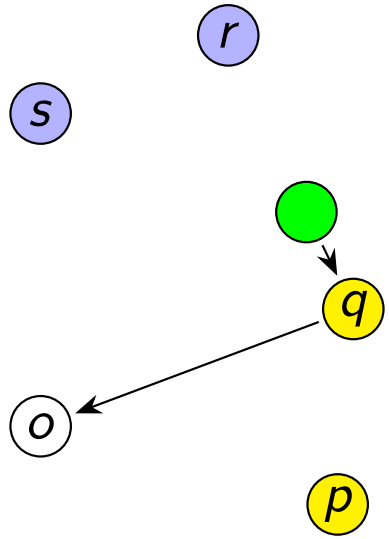
As $d(s, o) > R_t(o, server)$, Case 2.2 applies and the server returns to the origin.

PAH - An Example of PAH Execution

O: origin ● unserved request ● served request ● server's location



(a) Metric Space



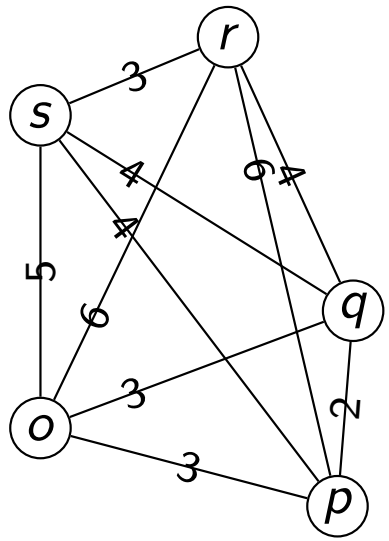
(b) $t = 10$

Server moves by route: server's location $\rightarrow q \rightarrow o$.

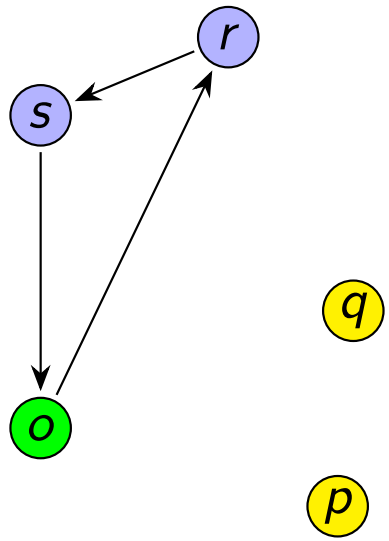
As the route costs 4 time units, the server arrives origin at $t = 14$.

PAH - An Example of PAH Execution

O: origin unserved request served request server's location



(a) Metric Space



(b) $t = 14$

At time 14, the server arrives at origin, and computes a new TSP tour $o \to r \to s \to o$ that costs 14 time units.

- From time 14 to 20, server moves from o to r.
- From time 20 to 23, the server moves from r to s.
- From time 23 to 28, server moves from s to o.

Hence, the completion time is 28 (less than double the offline optimal cost of 34).

PAH - Scheduling/Routing Algorithms

- Scheduling algorithms are used by the PAH algorithm to schedule a list of requested but not visited locations.
- The scheduling algorithms compute a TSP tour of the locations to be visited.
- Technically, any TSP algorithm can be used.
- However, theoretical results for PAH are only known for several TSP algorithms:
  - When PAH uses an exact scheduling algorithm (such as Dynamic Programming[2]), its completion time is at most 2 times the offline optimal completion time [1].
  - When PAH uses an approximation algorithm that guarantees a cost within 1.5 times the optimal cost (such as Christofides' Heuristic [3]), its completion time is at most 3 times the offline optimal completion time [1].

GTR for H-OLTSP - Introduction

Ausiello et al. [1] proposed GTR for the Nomadic OLTSP and stated that it 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.

Our Formulation.

- 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 in [1].
- While our formulation closely follows 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].

GTR for H-OLTSP - Main Logic (Part I)

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.

[1]

GTR for H-OLTSP-Main Logic (Part II)

- **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.

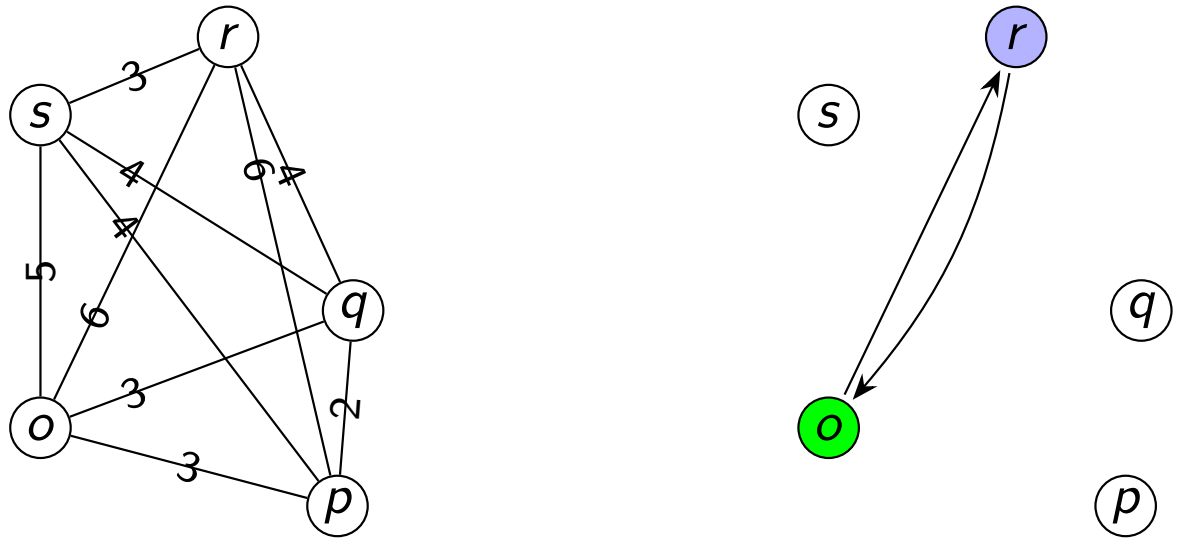
    $\Rightarrow$ The server computes two candidate routes that begin at its current location, one that first visits x and one that first visits y, each then visiting all other released but unserved request locations exactly once and returning to the origin at the end. It immediately starts moving along 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.

    $\Rightarrow$ 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.

[1]

GTR for H-OLTSP - An Example of Execution

O: origin ● unserved request ● server's location



(a) Metric Space (b) $t = 0$

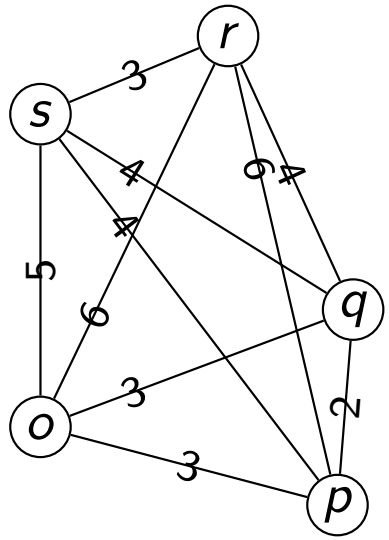
At time 0, the server is located at the origin, and the request (0, r) is released.

As the server is located at location (**case 2.1**), the server schedules a route that serves all uncompleted requests and returns to the origin (see (b)).

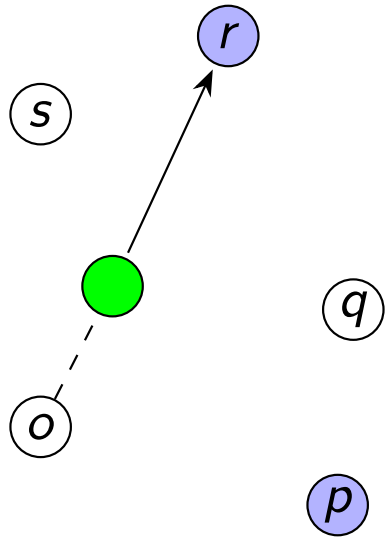
Hence, the server moves along the route from $o \rightarrow r \rightarrow o$.

GTR for H-OLTSP - An Example of Execution

O: origin unserved request server's location ($P(s)$)



(a) Metric Space



(b) $t = 2$

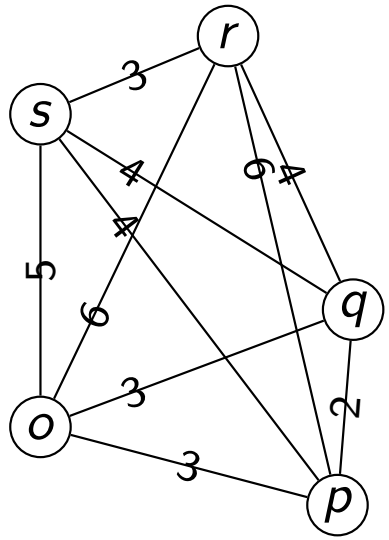
At time 2, the request $(p, 2)$ is released. The server lies on the shortest path between two locations (o and r), the **case 2.2-a** applied.
The server computes two candidate routes:

- $P(s) \to r \to p \to o$;
- $P(s) \to o \to p \to r \to o$;

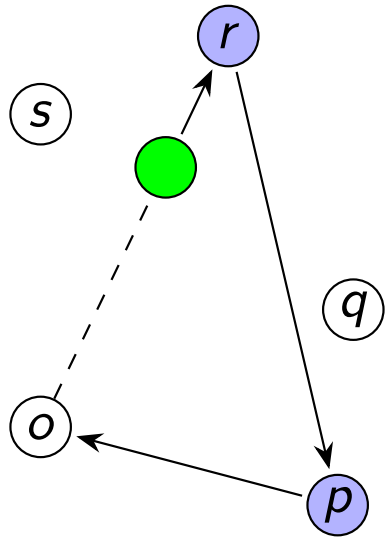
The server moves along the shorter one.

GTR for H-OLTSP - An Example of Execution

O: origin unserved request server's location ($P(s)$)



(a) Metric Space

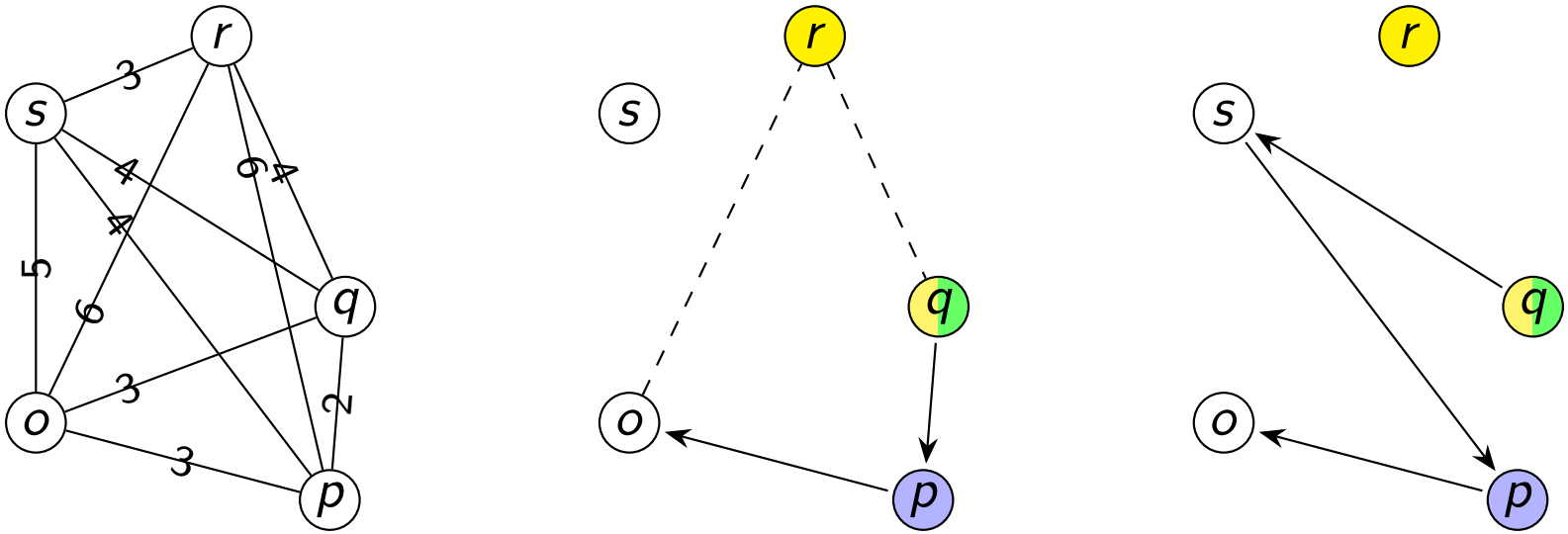


(b) $t = 4$

At time 4, the request (4, q) is released. The server lies on the shortest path between two locations (o and r), the **case 2.2-a** applied. The server computes two candidate routes and moves along the shorter one.
New route: $P(s) \to r \to q \to p \to o$

GTR for H-OLTSP - An Example of Execution

O: origin ● unserved request ● served request ● server's location



(a) Metric Space (b) $t = 10$ (c) $t = 10$

At time 10, the request (10, s) is released. As the server is at a location (q), **case 2.1** applies.
The server computes a new route and moves along it (see (c)).
Completion time: $10 + 4 + 4 + 3 = 21$

GTR - Scheduling/Routing Algorithms

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.
- The scheduling algorithms compute a Hamiltonian path with fixed endpoints.
- In this paper, we consider two scheduling methods for GTR.
  - an exact scheduling algorithm based on dynamic programming [2], which can compute optimal routes;
  - the Minimum Spanning Tree (MST) heuristic [1];

Experimental Setup - Introduction

Our experiments aim to evaluate the practical behavior of competitive algorithms under controlled yet representative conditions. Rather than targeting a specific application, we model typical characteristics of automated logistics systems while maintaining full control over instance structure and difficulty.

- **Spatial model:** We use benchmark instances from TSPLIB [4] to induce discrete metric spaces. Locations correspond to service points in an autonomous logistics region, and request locations are sampled uniformly at random from these predefined sites to avoid spatial bias.
- **Temporal model:** Request release times are generated using a normal distribution to approximate demand concentration and peak periods. By varying the standard deviation, we control the level of temporal clustering and congestion.
- To possible release times are restricted to a fixed time interval. This prevents idle gaps and ensures that increasing the number of requests leads to greater scheduling difficulty.

Experimental Setup - Problem Instance Generation

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.

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

Experimental Setup - Problem Instance Generation

- **More realistic:** In real systems, requests often arrive in busy periods (like rush hours) rather than being evenly spread.
- **Easy to control:** The mean sets the typical time, and the spread (σ) controls how close or far apart the requests appear. Truncating to $[0, T_{max}]$ keeps all release times within a fixed range.
  - a small σ makes many requests appear close together (high request load, scheduling complexity is high)
  - a large σ makes them more spread out (low request load, scheduling complexity is lower).

Experimental Configuration

TSPLIB instance I_{TSP}:

- Two symmetric TSPLIB instances: *berlin52* and *gr202* are obtained from a publicly available Kaggle dataset [5].

Temporal distributions σ: We generate release times from four normal distributions with mean $\mu = \frac{T_{\max}}{2}$ and varying standard deviations:

$$\sigma \in \left\{ T_{\max}, \frac{T_{\max}}{3}, \frac{T_{\max}}{6}, \frac{T_{\max}}{10} \right\},$$

modeling different levels of temporal concentration.

Total Number of Requests N:

$$N \in \{5, 7, 9, 11, 13, 15, 17\}.$$

For each parameter configuration (I_{TSP}, σ, N), five independent instances are generated.
Experiments are implemented in Python 3.11 and executed on an Intel i5-13500H processor.
Reported execution times are averaged over five independent runs to reduce system variability.

Experimental Results

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: Average completion time.

Experimental Results

	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 [3]
GTR-MST	1.0226	1.1649	1.1544	1.4655	0.0725	unknown

Table: Ratio.

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

Experiment Results

$\sigma = T_{max}$							
	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

Table: Average Execution Time(ms).

Discussion

The evaluated competitive algorithms perform significantly better in practice than their competitive ratios suggest based our experimental results.

- 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.

Limitations

- The experiments are conducted on benchmark-based metric spaces induced from TSPLIB instances, while request sequences are synthetically generated. 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.
- The implementation relies on external libraries for graph-related operations (such as NetworkX [6]). As a result, the reported execution times may include minor implementation-related overhead that is not intrinsic to the algorithms themselves.
- 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.

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:

- competitive algorithms perform well in practice, despite their conservative theoretical performance guarantees.
- In particular, GTR demonstrates promising practical applicability for autonomous logistics systems. 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.

[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.

[5] "TSPLIB symmetric dataset," Accessed: Jan. 21, 2026. 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 (SciPy 2008)*, G. Varoquaux, T. Vaught, and J. Millman, Eds., Pasadena, CA, USA, 2008, pp. 11–15.

Thank you very much!

Questions?