E0 318: Topics in Geometric Algorithms October 28, 2024

Learning-Augmented Algorithms for Online TSP

Instructor: Rahul Adhikari *Scribe: Rahul Adhikari*

1 Introduction

This lecture will discuss the Online version of the Travelling Salesman Problem (OLTSP). We'll provide online algorithms for different versions of OLTSP and also provide lower bounds for the competitive ratios. Finally, we'll give online algorithms based on different prediction models, which may perform better than the worst-case ratios in online setting.

2 Problem Setup

The OLTSP takes a continuous metric space M as input and a distinguished point (the origin) o of M. The metric space is available offline, while a set of requests $x = (x_1, ..., x_n)$, where $x_i = (t_i, p_i)$ are shared in an online fashion. t_i denotes the arrival time of the i-th request and p_i is the position of the request. The t_i's form an ordered sequence i.e. $t_i \leq t_j$ whenever $i < j$.

We call the salesman as the *server*. The server starts at origin o at time $t = 0$ and serves the request x_i by visiting position p_i at time no earlier than t_i. The server can move with at most unit speed. Based on the end objective, we have two variants of this problem:

- **Nomadic-OLTSP (N-OLTSP):** The server has to minimize the completion time to serve all presented requests.
- **Homing-OLTSP (H-OLTSP):** The server has to minimize the completion time to serve all the requests and return to origin o.

The offline algorithm (call it the *adversary*) has access to all the requests in the beginning and it can plan accordingly. But it also can serve a request x_i only after its release time t_i.
Denote the position of the server at time t by $p(t)$ and that of the offline adversary as $p^*(t)$. Also denote the set of released but unserved requests at time t by U_t and optimal tour on a set of unserved requests S as T_S. Define the completion time of an online algorithm as Z^{ALG} and that of the optimal offline solution by Z^*. We call the online algorithm ρ-competitive if $Z^{ALG} \leq \rho \cdot Z^*$ for any instance of the problem.

3 Online Algorithm for H-OLTSP

3.1 Algorithm

For H-OLTSP, we design the algorithm in a greedy framework based on the fact that the optimal also has to return to the origin in the end. Whenever we are at the origin, we plan an optimal tour based on the set of released unserved requests and start serving them. When a request comes while the server is on a tour, we check if it is relatively near or far from origin. If it's near, we ignore it until we're back at origin again and if it's far, we postpone everything, directly return to o and replan the tour.
The planning of an optimal tour is done in an oracle fashion and it's computation time is not considered in our algorithm. Notice that computation of an optimal tour itself may take exponential time, so we can devise polynomial time algorithms by computing the approximate tour instead (for example, using Christofide's heuristic) This worsens the competitive ratio by the approximation factor.

We call this algorithm PAH (Plan-AT-Home). The following is the pseudocode for the algorithm:

Algorithm 1: Plan-At-Home (PAH)

```
Input: Current time $t$ and set of current released unserved requests $U_t$
while $U_t \neq \phi$ do
    if $p(t) = o$ then
        Start to follow an optimal route $T_{U_t}$, passing through each request in $U_t$;
    end
    if The server is on a route $T_{U_{t'}}$ for some $t' < t$ then
        if a request $x_i = (t_i, p_i)$ arrives then
            if $d(p_i, o) > d(p(t), o)$ then
                Go back to the origin $o$;
            end
            else
                Move ahead on the current route $T_{U_{t'}}$;
            end
        end
    end
end
```

We claim that the PAH algorithm is 2-competitive and it is also the best that we can do for any H-OLTSP algorithm designed for general metric spaces.

3.2 Competitive Ratio for PAH

Theorem 1. *PAH is a 2-competitive algorithm*

Proof. Firstly, notice that $t_n \leq Z^*$ (as adversary can end only after last request is presented) and $|T^*| \leq Z^*$ where T^* is the optimal tour starting from the origin, serves all requests and finishes at origin. Considering the position of the server at t_n and $d(p_n, o)$ w.r.t $d(p(t_n), o)$, the possible cases are:

1. $p(t_n) = o$
   Then the server starts a tour that serves all requests up to x_n and return to origin. Hence,
   $Z^{ALG} \leq t_n + |T_{U_{t_n}}| \leq t_n + |T^*| \leq 2Z^*$

2. Server is on some tour $T_{U_{t'}}$ at time t_n and $d(p_n, o) > d(p(t_n), o)$
   Then the server directly return to the origin (at time t'', say) and plans a tour which serves all requests up to x_n. Hence,
   $Z^{ALG} \leq t_n + d(p(t_n), o) + |T_{U_{t''}}| < t_n + d(p_n, o) + |T_{U_{t''}}|$
   But notice that, $t_n + d(p_n, o) \leq Z^*$ as the adversary has be at point p_n at a time no earlier than t_n and then return to origin. So,
   $Z^{ALG} \leq Z^* + |T_{U_{t''}}| \leq 2Z^*$

3. Server is on some tour $T_{U_{t'}}$ at time t_n and $d(p_n, o) \leq d(p(t_n), o)$
   Suppose, Q is the set of requests that has been ignored since server was at origin. q is the first request in Q served by the adversary and P_Q^* is the optimal tour starting at point q, visiting all points in Q and returns to o. Then, $Z^* \leq t_q + |P_Q^*|$ (by definition).
   Now, as $d(q, o) \leq d(p(t_q), o)$, at time t_q server must have travelled at least $d(q, o)$ distance on route $T_{U_{t'}}$. So, it has to travel at most $|T_{U_{t'}}| - d(q, o)$ distance before it is back to o. Then, it will plan an optimal tour T_Q that serves all Q and return to o. Clearly, $|T_Q| \leq d(q, o) + |P_Q^*|$ (as going to q from o and following P_Q^* is also a tour that starts from o, serves Q and returns to o). So,
   $Z^{ALG} \leq t_q + |T_{U_{t'}}| - d(q, o) + |T_Q| \leq t_q + |T_{U_{t'}}| - d(q, o) + d(q, o) + |P_Q^*| = t_q + |P_Q^*| + |T_{U_{t'}}| \leq 2Z^*$

□

Lemma 2. *$p(t)$ is the position of the server at time t by PAH. Then, $d(p(t), o) \leq \frac{1}{2}Z^*$*

Proof. If the server is on a route $T_{U_{t'}}$ and it is travelling between points a and b which correspong to some request or the origin o, then
$|T^*| \geq d(o, a) + d(a, b) + d(b, o) = d(a, o) + d(p(t), a) + d(p(t), b) + d(b, o)$
$d(p(t), o) \leq \min\{d(p(t), a) + d(a, o), d(p(t), b) + d(b, o)\} = \frac{1}{2}|T^*| \leq \frac{1}{2}Z^*$

Otherwise if the server is returning to o, after terminating a route $T_{U_{t'}}$, then a request $x_i = (t_i, p_i)$ can in the interval $[t', t]$. So,
$d(p(t), o) \leq d(p(t_i), o) \leq \frac{1}{2}Z^*$ (by above argument)

□

Theorem 3. *PAH with Christofides' heuristic is 3-competitive for H-OLTSP*

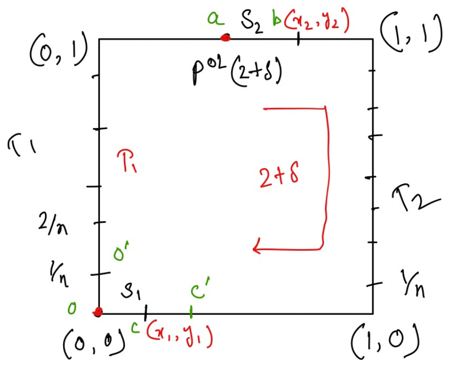
Proof. Christofides' algorithm gives an $\frac{3}{2}$- approximation of the optimal tour.
So, $Z^{ALG} \leq t_n + d(p(t_n), o) + CHR(U_{t_n} \cup \{o\}) \leq Z^* + \frac{1}{2}Z^* + \frac{3}{2}|T^*| \leq 3Z^*$ □

3.3 Lower Bounds for Competitive Ratio of H-OLTSP Algorithms

We can show that any algorithm for H-OLTSP in general metric spaces can not have competitive ratio better than 2. This does not prove that there can not be algorithms with better competitive ratio for particular metric spaces; just that such an algorithm will not have this performance in the general setup. For example, on the real line, we can devise a 1.75-competitive algorithm.

Theorem 4. *For any $\varepsilon > 0$, any ρ-competitive algorithm for H-OLTSP in general metric spaces has $\rho \geq 2-\varepsilon$*

Proof. Consider the space to be the boundary of the unit square $[0, 1]^2 \subseteq \mathbb{R}^2$. The distance between two points is defined as the length of shorter path along the boundary of the unit square. Let the $(0, 0)$ point be the origin o.



At time $t = 0$ issue requests in all points of $S = \{(0, i/n), (1, i/n), (i/n, 0), (i/n, 1) | i \in \{0, 1, , 2, ..., n\}\}$. Note that the adversary can serve all these requests with an anti-clockwise loop along the boundary, so $Z^* = 4$.
Now we claim that for some δ, $0 \leq \delta \leq 2$, at time $2 + \delta$, the server must be in one of the two points at distance $2 - \delta$ from the origin.
Say the function $f : [0, 2] \to [0, 2]$ denotes the distance of server from $(1, 1)$ at time $2 + x$. Consider, $g(x) = f(x) - x$. Note that $g(0) = f(0) \geq 0$ (as it is distance from point $(1, 1)$ at time $t = 2$) and $g(2) = f(2) - 2 \leq 0$ (as maximum distance from $(1, 1)$ on the boundary can be 2). As g is continuous, by Intermediate Value Theorem, $g(x) = 0 \implies f(x) = x$ for some $\delta \in [0, 2]$ i.e. server is at distance δ from $(1, 1)$ $(2 - \delta$ from origin) at time $2 + \delta$. Hence, our claim is proven.
Take the smallest such δ and call the point $p(2+\delta)$ as a (WLOG a is at distance $2-\delta$ from o in the clockwise sense). So, at $t = 2 + \delta$ server has served requests on segment $[o, a]$ (call it T_1) and it might also have served requests on segment $S_2 : [a, b]$ and $S_1 : [c, o]$ where $b = (x_2, y_2)$ and $c = (x_3, y_3)$. Now, server must have travelled on S_1 and S_2 twice, so $|T| + 2(|S_1| + |S_2|) \leq 2 + \delta \implies |S_1| + |S_2| \leq \delta \implies |T| + |S_1| + |S_2| \leq 2$.

Call rest of the square i.e. $[b, c]$ as T_2, then $|T_2| \geq 2$.
At time $t = 2 + \delta$, generate requests on every points of S lying in the segment T_1. This ends the sequence of requests. Now notice that the adversary can still finish it with the anti-clockwise tour, it reaches point a, at time $t = 2 + \delta$ (as a is at distance $2 - \delta$ from o moving clockwise) and after that it starts moving on T_1, so it can serve every request generated at $t = 2 + \delta$. So, $Z^* = 4$.
Whereas, the online server has to serve requests on the whole square and then return to o. It can either go to c' (the point before c) clockwise and then move anti-clockwise to finish at o or it can move anti-clockwise to reach o' (point before o) and then move clockwise finishing at o. In both cases,
$Z^{ALG} = (2 + \delta) + 2(2 - 1/n) + (2 - \delta) = 8 - 2/n$
Hence, $\rho = 2 - \frac{1}{2n}$, so for arbitrarily small $\varepsilon > 0$, we can make $\rho = 2 - \varepsilon$ by choosing sufficiently large n. $\square$

4 Prediction-based Algorithms for H-OLTSP

4.1 Prediction-based Algorithms

To beat the the theoretical bound of 2-competitiveness for H-OLTSP, we can take implement algorithms which also take a certain prediction as input. Based on the error in our prediction ε, we'll get competitive ratio $c(\varepsilon)$ as a function of it. We'll aim to maintain three qualities for our algorithms:

- **Consistency:** Our algorithm is said to be α-consistent if $c(0) = \alpha$
  Consistency denotes if our prediction is perfectly accurate, whether our algorithm performs well.

- **Robustness:** An algorithm is said to be β-robust if $c(\varepsilon) \leq \beta$ for any ε
  Robustness maintains worst-case bounds even if our prediction is terribly wrong.

- **Smoothness:** An α-consistent algorithm is $f(\varepsilon)$-smooth if $Z^{ALG} \leq \alpha \cdot Z^* + f(\varepsilon)$ for some continuous function f with $f(0) = 0$
  Smoothness denotes that the performance of the algorithm does not degrade significantly if our prediction is slightly inaccurate.

We will discuss algorithms based on three different prediction models. The first two model attempt to predict the whole sequence of requests, whereas the last model tries to predict the arrival time of the final request. The three models are following:

1. This is sequence prediction without identity i.e. we predict a sequence of arbitrary length. In this case, we design an algorithm that is $(1 + \lambda)$-consistent and $(3 + 2/\lambda)$-robust, where $\lambda \in (0, 1]$ denotes the confidence level in this prediction.

2. In the second model, length of the sequence of requests n is known beforehand and we predict a sequence with identity. We give a $\min\{3, 1 + (2\varepsilon_{time} + 4\varepsilon_{pos})/Z^{OPT}\}$-competitive algorithm, where ε_{time} is the maximum difference between issue time of a request and its corresponding prediction i.e. $\varepsilon_{time} = \max_{i \in [n]} |\hat{t}_n - t_n|$ and ε_{pos} is the total error in predicted and actual requests over all requests i.e. $\varepsilon_{pos} = \sum_i d(\hat{p}_i, p_i)$

3. In the final prediction model, we predict the arrival time of the last request and the error in the prediction is denoted by $\varepsilon_{last} = |\hat{t}_n - t_n|$. In this case, we give a $\min\{4, 2.5Z^{OPT} + \varepsilon_{last}\}$-competitive polynomial time algorithm.

4.2 Model 1

Given a prediction of requests $\hat{x}$, we find an optimal tour $\hat{T}$ on $\hat{x}$. Then we set a confidence level $\lambda \in (0, 1]$ for the prediction. Low λ denotes that we trust this prediction. Now we follow the algorithm PAH while $t < \lambda|\hat{T}|$, we also modify our PAH framework making it reach the origin exactly at time $\lambda|\hat{T}|$ if the planned route is too long. Then, we start following our prediction and complete the tour $\hat{T}$. Finally we run PAH algorithm to complete any requests in $x \setminus \hat{x}$. Following is a pseudo-code for the algorithm:

```
Algorithm 2: Learning-Augmented Routing Without Identity (LAR-NID)
Input: Current time $t$, a sequence prediction $\hat{x}$, confidence level $\lambda \in (0,1]$, set of current released
       unserved requests $U_t$
Compute an optimal tour $\hat{T}$ to serve all the requests in $\hat{x}$ and return to $o$;
while $t < \lambda|\hat{T}|$ do
    if $p(t) = o$ then
        Compute an optimal route $T_{U_t}$, to serve all requests in $U_t$ and return to $o$;
        if $t + |T_{U_t}| > \lambda|\hat{T}|$ then
            Find the moment $t_{back}$ s.t. $t_{back} + d(p(t_{back}, o)) = \lambda|\hat{T}|$;
            Modify $T_{U_t}$ into a tour $T'_{U_t}$ by sending the server back to $o$ at time $t_{back}$ along the
              shortest path;
            Start following route $T'_{U_t}$;
        end
        else
            Start following route $T_{U_t}$;
        end
    end
    if The server is on a route $T_{U_{t'}}$ for some $t' < t$ then
        if a request $x_i = (t_i, p_i)$ arrives then
            if $d(p_i, o) > d(p(t), o)$ then
                Go back to the origin $o$;
            end
            else
                Move ahead on the current route $T_{U_{t'}}$;
            end
        end
    end
end
while $t \geq \lambda|\hat{T}|$ do
    Wait until $U_t \neq \phi$;
    Follow $\hat{T}$ until the server is back to $o$;
    Follow PAH$(t, U_t)$;
end
```

Lemma 5. *The LAR-NID algorithm is $(1.5 + \lambda)$-consistent*

Proof. Say the prediction is perfect, $x = \hat{x}$ and $|\hat{T}| = Z^{OPT}$. Server follows PAH till $\lambda|\hat{T}|$ and follows the predicted route if $U_{\lambda|\hat{T}|} \neq \phi$, otherwise it waits.

- If $U_{\lambda|\hat{T}|} \neq \phi$, then
  $Z^{ALG} = \lambda|\hat{T}| + |\hat{T}| \leq (1 + \lambda)Z^{OPT}$
- Otherwise suppose it waits till t_j ($t_j = \min_t\{t > \lambda|\hat{T}| : U_t \neq \phi\}$). Suppose the adversary is moving during this time. But as some request is yet to come and adversary has to be at the requested point at that time and then return to origin, the most it can gain is $\frac{1}{2}Z^{OPT}$. So, $t_j - \lambda|\hat{T}| \leq \frac{1}{2}Z^{OPT}$
  Hence, $Z^{ALG} = \lambda|\hat{T}| + (t_j - \lambda|\hat{T}|) + |\hat{T}| \leq (1.5 + \lambda)|\hat{T}|$

□

Lemma 6. *The LAR-NID algorithm is $(3 + 2/\lambda) - robust$*

Proof. Clearly, $t_i \leq Z^{OPT}$ for any i and $|T_{U_t}| \leq Z^{OPT}$ for any t. Now, consider the cases:

1. If the server finishes before $\lambda|\hat{T}|$
   then it was following the PAH algorithm throughout, which is 2-competitive. So, $Z^{ALG} \leq 2Z^{OPT}$

2. It did not finish before $\lambda|\hat{T}|$. Then there might be some unserved request at time $\lambda|\hat{T}|$ or some request might arrive after time $\lambda|\hat{T}|$. Then, we know, $Z^{OPT} \geq \frac{\lambda|\hat{T}|}{2}$ (as it follows PAH before $\lambda|\hat{T}|$ and PAH is 2-competitive). We have 4 subcases:

   (a) $U_{\lambda|\hat{T}|} \neq \phi$, $t_n > (1+\lambda)|\hat{T}|$
   Then it follows PAH after $(1+\lambda)|\hat{T}| \implies Z^{ALG} \leq 2Z^{OPT}$

   (b) $U_{\lambda|\hat{T}|} \neq \phi$, $t_n \leq (1+\lambda)|\hat{T}|$
   $Z^{ALG} \leq \lambda|\hat{T}| + |\hat{T}| + |T_{U_{(1+\lambda)|\hat{T}|)}}| \leq 2Z^{OPT} + (2/\lambda)Z^{OPT} + Z^{OPT} = (3+2/\lambda)Z^{OPT}$

   (c) $U_{\lambda|\hat{T}|} = \phi$, $t_n > t_j + |\hat{T}|$
   Then it follows PAH after $t_j + |\hat{T}| \implies Z^{ALG} \leq 2Z^{OPT}$

   (d) $U_{\lambda|\hat{T}|} = \phi$, $t_n \leq t_j + |\hat{T}|$
   Then, we have $\lambda|\hat{T}| < t_n < Z^{OPT}$. Server starts its last route at $t_j + |\hat{T}|$.
   Hence, $Z^{ALG} \leq t_j + |\hat{T}| + |T_{U_{t_j+|\hat{T}|}}| \leq Z^{OPT} + (1/\lambda)Z^{OPT} + Z^{OPT} = (2+1/\lambda)Z^{OPT}$

□

Lemma 5 and lemma 6 implies the following theorem

Theorem 7. *The LAR-NID algorithm is* $(1.5+\lambda)$*-consistent and* $(3+2/\lambda)$*-robust but not smooth.*

4.3 Model 2

In this model, we have access to the number of requests n, so we can give a sequence prediction with identity (a prediction $\hat{x}$ of size n). We first give the LAR-TRUST algorithm and show that although it has consistency and smoothness, it is not robust. Then we can modify it to give the LAR-ID algorithm which follows LAR-TRUST till the penultimate request. When final request arrives, it compares between the predicted route and a greedy route and takes the shorter one.
In the LAR-TRUST algorithm,the server first finds an optimal tour over the predicted set of requests $\hat{T}$, then starts following it. If a request x_i arrives, the server modifies the remaining tour by inserting x_i after $\hat{x}_i$. The server also waits at the predicted i-th request $\hat{p}_i$ until the actual i-th request arrives.
The following is a pseudocode describing the LAR-TRUST algorithm:

Algorithm 3: Learning-Augmented Routing Trust (LAR-TRUST)

```
Input: Current time $t$, number of requests $n$, a sequence prediction $\hat{x}$, set of current released
       unserved requests $U_t$
1  Compute an optimal route $\hat{T} = (\hat{x}_1, ..., \hat{x}_n)$ to serve the requests in $\hat{x}$ and return to $o$, where $\hat{x}_i$ is the
     $i$-th predicted request in $\hat{T}$;
2  Start following the route $\hat{T}$;
3  while $i = 1, 2, ..., n$ do
4      if $t = t_i$ then
5          Update the route $\hat{T}$ by adding the request $x_i$ after the predicted request $\hat{x}_i$;
6      end
7      if $p(t) = \hat{p}_i$ and $t < t_i$ then
8          Wait at $\hat{p}_i$ until time $t_i$;
9      end
10 end
```

Now we modify the algorithm by setting a binary trust value F (for $F = 0$, we trust the algorithm and for $F = 1$, we do not). We follow the LAR-TRUST algorithm until the last request arrives i.e. t_n. Then we compute two routes: r_1 which takes server back to origin and serves the remaining unserved requests and r_2 which is rest of the $\hat{T}$ tour. If r_1 is shorter, we set $F = 1$ and follow r_1, otherwise we follow r_2.
The following is a pseudocode describing the LAR-ID algorithm:

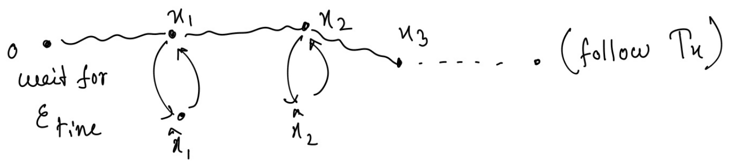
Algorithm 4: Learning-Augmented Routing With Identity (LAR-ID)

```
Input: Current time $t$, number of requests $n$, a sequence prediction $\hat{x}$, set of current released
       unserved requests $U_t$
1  Intialize $F = 0$;
2  Compute an optimal route $\hat{T} = (\hat{x}_1, ..., \hat{x}_n)$ to serve the requests in $\hat{x}$ and return to $o$, where $\hat{x}_i$ is the
     $i$-th predicted request in $\hat{T}$;
3  Start following the route $\hat{T}$;
4  while $F = 0$ do
5      Follow LAR-TRUST for $i = 1, 2, ..., n - 1$;
6      if $t = t_n$ then
7          $r_1 \leftarrow$ the remaining distance of following $\hat{T}$;
8          Compute a route $T_{U_{t_n}}$ to start, serve $U_{t_n}$ and finish at $o$;
9          $r_2 \leftarrow d(p(t), o) + |T_{U_{t_n}}|$;
10         if $r_1 > r_2$ then
11             Go back to $o$;
12             $F = 1$;
13         end
14         else
15             Continue on route $\hat{T}$;
16         end
17     end
18 end
19 while $F = 1$ do
20     Start to follow $T_{U_{t_n}}$;
21 end
```

Lemma 8. *The LAR-TRUST algorithm is $(1 + (2\varepsilon_{time} + 4\varepsilon_{pos})/Z^{OPT})$-competitive*

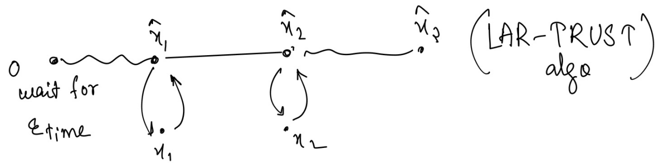
Proof. Suppose, T_x is the optimal route for serving x and $T_{\hat{x}}$ is the optimal route for serving $\hat{x}$. Notice that in the algorithm the requests in x are visited in the the order of visiting $\hat{x}$ in route $T_{\hat{x}}$. So, define:

- $|T_x| = Z^{OPT}$
- $Z^*_{\hat{x}}$ if the requests in $\hat{x}$ are visited following order of T_x.
- $|T_{\hat{x}}| = Z'_{\hat{x}}$
- Z^{ALG} is time to serve requests in x following order of $T_{\hat{x}}$.



Say the server waits at the origin for time ε_{time} then starts following T_x, but whenever the server is at x_i it takes the shortest path to $\hat{x}_i$ and returns to x_i. Notice that as T_x ensures that the server is can every

request x_i, waiting ε_{time} at origin ensures that we reach at every x_i at least ε_{time} late. Now as the service time gap between any x_i and $\hat{x}_i$ is at most ε_{time}, we can service all requests in $\hat{x}_i$ in this route. But, $Z^*_{\hat{x}}$ is the optimal route to serve $\hat{x}$ in order of T_x. Hence, $Z^*_{\hat{x}} \leq Z^{OPT} + \varepsilon_{time} + 2\varepsilon_{pos}$.
Also, $Z'_{\hat{x}} \leq Z^*_{\hat{x}}$ (As $Z'_{\hat{x}}$ is the optimal route of serve requests in $\hat{x}$ in any order)



Now, we follow $T_{\hat{x}}$ after waiting at origin for ε_{time} and visit x_i along the shortest route after $\hat{x}_i$ and return to it. Notice that, this route also visits x_i in order of $T_{\hat{x}}$.
Hence, $Z^{ALG} \leq Z'_{\hat{x}} + \varepsilon_{time} + 2\varepsilon_{pos} \leq Z^*_{\hat{x}} + \varepsilon_{time} + 2\varepsilon_{pos} \leq Z^{OPT} + 2\varepsilon_{time} + 4\varepsilon_{pos}$
So, the competitive ratio is $1 + (2\varepsilon_{time} + 4\varepsilon_{pos})/Z^{OPT}$ i.e. the LAR-TRUST algorithm is 1 consistent, $(2\varepsilon_{time} + 4\varepsilon_{pos})$-smooth but not robust. □

Theorem 9. *The LAR-ID algorithm is* $\min\{3, 1 + (2\varepsilon_{time} + 4\varepsilon_{pos})/Z^{OPT}\}$*-competitive*

Proof.
- If $r_1 \leq r_2$
  Hence, the errors are small and the server continues on $\hat{T}$. Then it follows the LAR-TRUST algorithm throughout. So, by lemma 8, $Z^{ALG} \leq Z^{OPT} + 2\varepsilon_{time} + 4\varepsilon_{pos}$

- If $r_1 > r_2$
  Server goes back to the origin and designs a new route. It reaches origin at time $t_n + d(p(t_n), o)$ and it starts to follow a route $T_{U_{t_n}}$. Hence, $Z^{ALG} = t_n + d(p(t_n), o) + |T_{U_{t_n}}| \leq t_n + t_n + |T_{U_{t_n}}| \leq 3Z^{OPT}$

The LAR-ID algorithm chooses the faster route.
Hence, the competitive ratio is $\min\{3, 1 + (2\varepsilon_{time} + 4\varepsilon_{pos})/Z^{OPT}\}$ □

4.4 Model 3

This model gives a prediction of the arrival time of the last request and we design the LAR-LAST algorithm based on it. The idea of this algorithm is to ensure that the server arrives at the origin at the predicted time $\hat{t}_n$ and follows a modified PAH algorithm for rest of the execution. We use a similar gadget as used in the LAR-NID algorithm. Whenever the server is at origin it plans an approximate route T_{U_t} based on Christofides heuristics, over the current set of released unserved requests and return to o. If the route is too long ($t < \hat{t}_n < t + |T_{U_t}|$), the server stops it at some t_{back} to return to origin and arrive exactly at time $\hat{t}_n$.

In the modified PAH framework, we return to the origin o whenever a new request arrives during a tour, irrespective of relative distance of the request. Call this algorithm REDESIGN. We can show that lemma 2 and theorem 3 holds for REDESIGN algorithm too. The proof is similar to that of PAH algorithm.

Lemma 10. *$p(t)$ is the position of a server at time t by REDESIGN. Then, $d(p(t), o) \leq \frac{1}{2}Z^{OPT}$ for all t.*

Theorem 11. *REDESIGN is a 3-competitive polynomial-time algorithm for H-OLTSP using Christofides' heuristics.*

Using this, we can show that LAR-LAST is a $\min\{4, 2.5 + \varepsilon_{last}/Z^{OPT}\}$-competitive polynomial time algorithm. The following is a pseudocode describing the LAR-LAST algorithm:

Algorithm 5: Learning-Augmented Routing with Last Arrival Time (LAR-LAST)

```
Input: Current time $t$, predicted last time $\hat{t}_n$ and set of current released unserved requests $U_t$
while $U_t \neq \phi$ do
    if $p(t) = o$ then
        Start to follow an approximate route $T_{U_t}$, to serve every request in $U_t$ and return to $o$;
        if $t < \hat{t}_n$ and $t + |T_{U_t}| > \hat{t}_n$ then
            Find the moment $t_{back}$ such that $t_{back} + d(p(t_{back}, o)) = \hat{t}_n$;
            Modify $T_{U_t}$ into $T'_{U_t}$ by sending the server back to $o$ at time $t_{back}$ along the shortest path;
            Start following $T'_{U_t}$;
        end
        else
            Start following $T_{U_t}$;
        end
    end
    if The server is on a route $T_{U_{t'}}$ for some $t' < t$ then
        if a request $x_i = (t_i, p_i)$ arrives then
            Go back to the origin $o$;
        end
    end
end
```

Theorem 12. *The LAR-LAST algorithm is a* $\min\{4, 2.5 + \varepsilon_{last}/Z^{OPT}\}$*-competitive polynomial time algorithm, where* $\varepsilon_{last} = |\hat{t}_n - t_n|$

Proof. Depending on whether the last request arrives earlier or later than our prediction, we have two cases:

1. $\hat{t}_n \leq t_n$
   Server will return to origin at time t_n and then complete a final tour.
   Hence, $Z^{ALG} = t_n + d(p(t_n), o) + |T_{U_{t_n}}| \leq Z^{OPT} + 0.5Z^{OPT} + 1.5Z^{OPT} = 3Z^{OPT}$
   also, $d(p(t_n), o) \leq |\hat{t}_n - t_n| = \varepsilon_{last}$. So, $Z^{ALG} \leq 2.5Z^{OPT} + \varepsilon_{last}$

2. $\hat{t}_n > t_n$
   In this case, server will be back to origin by $\hat{t}_n$ and the complete a final tour. So, $Z^{ALG} \leq \hat{t}_n + |T_{U_{\hat{t}_n}}|$.
   Now suppose t_L is the last time before $\hat{t}_n$, when a route T^L was planned and T'^L is the new route if it was adjusted by the gadget ($T^L = T'^L$ if T^L can return before $\hat{t}_n$). So, $t_L + |T'^L| \leq \hat{t}_n$. Now we have 3 subcases:

   (a) If $t_L + |T^L| \leq \hat{t}_n$
   Then we return to o by $\hat{t}_n$ time and there is no readjustment. So, our algorithm essentially performs like the REDESIGN algorithm. Hence, $Z^{ALG} \leq 3Z^{OPT}$.

   (b) If $t_L + |T^L| > \hat{t}_n$ and $t_n \leq t_L$
   In this case, the T^L tour contained the last request and we would have been done if we continued with it. But as the route is too long, our algorithm finds some t_{back} and using T'^L route, returns back to o at $\hat{t}_n$. In this process, it incurs a loss of at most $2d(p(t_{back}), o)$.
   Hence, $Z^{ALG} \leq 3Z^{OPT} + 2d(p(t_{back}), o) \leq 4Z^{OPT}$ (Using lemma 10)

   (c) If $t_L + |T^L| > \hat{t}_n$ and $t_n > t_L$
   In this case, our tour T^L does not contain t_n and the route is too long. So the server return at $\hat{t}_n$ and starts the last tour $T_{U_{\hat{t}_n}}$. Hence, $Z^{ALG} \leq \hat{t}_n + |T_{U_{\hat{t}_n}}| = t_L + |T'^L| + |T_{U_{\hat{t}_n}}| \leq t_n + |T'^L| + |T_{U_{\hat{t}_n}}| \leq Z^{OPT} + 1.5Z^{OPT} + 1.5Z^{OPT} = 4Z^{OPT}$
   also, $Z^{ALG} \leq \hat{t}_n + |T_{U_{\hat{t}_n}}| = t_n + \varepsilon_{last} + |T_{U_{\hat{t}_n}}| \leq Z^{OPT} + \varepsilon_{last} + 1.5Z^{OPT} = 2.5Z^{OPT} + \varepsilon_{last}$

□

References

1. Hsiao-Yu Hu, Hao-Ting Wei, Meng-Hsi Li, Kai-Min Chung, Chung-Shou Liao. Online TSP with Predictions, pages 6-12, Jun 2022.

2. Giorgio Ausiello, Esteban Feuerstein, Stefano Leonardi, Leen Stougie, and Maurizio Talamo. Algorithms for the on-line travelling salesman. Algorithmica, 29(4):560–581, Apr 2001.