Optimally Serving Concurrent Requests on Hierarchically Well-Separated Trees

Abdolhamid Ghodselahi[1] and Fabian Kuhn[2]

1 **Department of Computer Science, University of Freiburg, Freiburg, Germany**
`hghods@cs.uni-freiburg.de`
2 **Department of Computer Science, University of Freiburg, Freiburg, Germany**
`kuhn@cs.uni-freiburg.de`

Abstract

We consider the traveling salesman problem (TSP) with $k \geq 1$ salespeople (k-TSP) on a hierarchically well-separated tree (HST). We show that the k-TSP can be optimally solved on HSTs. Based on this result, we show that the online service with delay (OSD) problem and the distributed queuing problem and even their generalized versions where there are $k \geq 1$ servers are optimally solved on HSTs if the requests arrive at the same time.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems—Sequencing and scheduling, G.2.2 Graph Theory—Network problems

Keywords and phrases k-TSP, OSD, distributed queuing, HSTs

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.

1 Introduction

The OSD problem [1] and the distributed queuing problem [6, 7, 5] deal with serving a set of requests that arrive in an online fashion at any time at the points of a metric space (for distributed queuing problem we model the given network as a graph). In each of these problems, there is a server initially located at some point of the metric space. The goal is to provide an order in which the requests must be served by moving the server to the requested points. An inherited characteristic of the two problems is that postponing serving a request is sometimes required and therefore it is generally allowed. In the distributed queuing problem, the processors of a network issue requests to mutually access a mobile shared object, say a server. In this problem, an order is defined in such a way that the processor that issues a request in the global order knows who its successor request is. *A newly arrived request is said to be enqueued whenever its predecessor in the global order learns about it.* The delay of a request in the distributed problem is the time that it takes since the request arrives until is enqueued while in the OSD problem it is the time that takes until the request is served.

The two problems are equivalent to the classic (centralized or distributed) k-server problem [8, 2] where $k = 1$ if there is a large enough time window between the releasing times of any pair of requests, say if the requests "sequentially" arrive. In other words, the two problems equivalent to the 1-server problem and thus becomes trivial if the order of serving the requests is provided in advance. Considering the situation where all the requests arrive on a metric at the same time can sometimes make our work easier to solve the given problem for the general case of arriving the requests [6, 7]. Given a parameter $\alpha > 1$, an α-HST [4] is a rooted tree where the length of any edge is smaller by a factor of α from any edge at one higher level. In this brief announcement, it is shown that the k-TSP can be optimally

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Daniel Marx, and Don Sannella;
Article No. ; pp. :1–:10


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

solved on HSTs. The k-TSP can be seen as a natural generalization of TSP [3] where there are $k \geq 1$ salespeople. We also show that the OSD problem and the distributed queuing problem and even their generalized versions for $k > 1$ servers are identical with the k-TSP when the requests arrive at the same time. Consequently, for this special pattern of arriving requests, the two problems can be optimally solved on HSTs. Further, w.r.t. [9] there is a $\Theta(\log N)$ competitive ratio for each of the two problems (for the versions of the problems with $k = 1$ server) on a graph with N nodes, for which the distance metric obeys the triangle inequality.

1.1 Further Related Work

The distributed queuing problem has been traditionally studied when there is $k = 1$ server. The OSD problem has been studied in [1] for $k \geq 1$ servers. On HSTs for $k = 1$ server, [1] and [5] provide $O(\log^3 n)$ and $O(1)$ competitive ratios for the OSD and distributed queuing problems, respectively. For $k > 1$ servers on HSTs, [1] presents a competitive ratio of $O(k \cdot \log^4 n)$ for the OSD problem. In the case when all requests arrive at the same time, [6] shows that the Arrow algorithm that solves the distributed queuing problem on tree networks simulates the Nearest-Neighbor algorithm in a distributed setting and the resulted order by Arrow is a Nearest-Neighbor TSP path w.r.t. the tree distances.

2 k-TSP on HSTs

In this section we show that the k-TSP can be optimally solved on HSTs. Given an HST T, a set of requests R, and $k \geq 1$ servers $s_1, s_2, \ldots, s_k$ initially located at some leaves of T. We assume that, w.l.o.g., at each leaf of T that hosts a server at the beginning there is a dummy request and the k dummy requests are included in R. The servers must move on T and serve the requests in R at the leaves of T. The cost of travel between each pair of requests, say the length of the edge that connects the two adjacent requests in the final TSP path, equals the length of the path travelled on T by some server between the two requests. A feasible solution of the problem is k TSP paths that span all the requests. The goal is to minimize the total length of the resulted k TSP paths.

Generalized Nearest-Neighbor (GNN) Algorithm.

The GNN algorithm is a generalization of Nearest-Neighbor algorithm [3]. The GNN algorithm sequentially and greedily moves the servers to the leaves of T that host the requests. For any server we define a serving phase. The server serves a subset of requests (at least the dummy request located at the initial location of the server) and stops at some leaf of T. The server never moves again after it stops.

1. i=1.
2. Find the shortest distance from the current location of s_i to an unserved request r (ties are arbitrarily broken).
3. **If** r is closer to some other server **then** $i = i + 1$ and go to Step 2.
4. Move s_i to the point of r and serve r.
5. **If** all the requests are served **then** terminate.
6. Go to Step 2.

▶ **Theorem 1.** *Assume that we are given an α-HST T for $\alpha > 1$, a set of requests R that are at the leaves of T from the beginning, and $k \geq 1$ servers initially located at some leaves*

of T together with dummy requests. The GNN algorithm serves all the requests in R with minimum traveling cost for the servers.

To show that the GNN algorithm correctly works, we need to show that any request in R is served by some server of GNN.

▶ **Lemma 2.** *All the requests in R are served by GNN algorithm.*

Proof. If there is only one server then GNN algorithm is the same as Nearest-Neighbor algorithm and we are done. Suppose there are more than one server. Assume that the request r is not served by GNN algorithm and let s_p denote the server that is (at least one of) the closest one to r when the algorithm terminates. The request r was not served during the serving phase of s_p because there was a closer server s_q w.r.t. the algorithm description where $q > p$. This implies that the lowest subtree T' of T that includes both r and s_q during the serving phase of s_p has smaller depth than the lowest subtree T'' of T that includes both r and any leaf that was hosting s_p in the serving phase of s_p. The assumption that the distance between s_p and r is not longer than the distance between r and s_q when the algorithm terminates, however, implies that s_q left the subtree T'' during its serving phase. This contradicts the fact that any server of GNN does not leave any subtree of T while there is an unserved request there. ◀

Consider a spanning forest $\mathcal{F}$ of a graph $G = (V, E)$ with $k \geq 1$ connected subtrees. *In this brief announcement, from now on, we mean "a spanning forest with k connected subtrees" wherever we say a spanning forest.* A spanning forest consisting of $k+1$ connected subtrees of $\mathcal{F}$ is resulted by removing the edge e of $\mathcal{F}$ from $\mathcal{F}$. Let $V_{e,1}, \ldots, V_{e,k+1}$ be the node sets of these $k+1$ connected components. We say that $(V_{e,1}, \ldots, V_{e,k+1})$ is the *k-cut induced by removing e from $\mathcal{F}$*. We provide a general minimum spanning forest (MSF) approximation result. This result holds for a family of MSFs that satisfy some property P. The property P can be nothing and therefore the approximation result holds for general MSFs. Consider a spanning forest $\mathcal{F}$ of G that satisfies the property P. The next theorem shows that the total length of $\mathcal{F}$ is within a factor λ of the length of an MSF that satisfies the property P if the same approximation factor λ is guaranteed between the edge sets of the MSF and $\mathcal{F}$ under some conditions.

▶ **Theorem 3.** *Let $\lambda \geq 1$ be some number and let $G = (V, E, w)$ be a weighted connected graph with non-negative edge weights $w(e) \geq 0$ and let $\mathcal{F}$ and $\mathcal{F}^*$ be two arbitrary spanning forests of G each with $k \geq 1$ connected subtrees that satisfies the property* P*. For every edge e of $\mathcal{F}$, consider the lightest edge e^* of $\mathcal{F}^*$ crossing the k-cut induced by removing e from $\mathcal{F}$ such that the spanning forest that is resulted by removing e from $\mathcal{F}$ and by adding e^* satisfies the property* P*. If e^* has weight $w(e^*) \geq w(e)/\lambda$, then the total weight of all edges of $\mathcal{F}$ is at most a λ-factor larger than the total weight of the edges of $\mathcal{F}^*$.*

Proof. We generalize the proof provided in the arXiv version of [5] to prove the theorem. In the following, we slightly abuse notation and we identify a spanning forest $\mathcal{F}$ with the set of edges contained in $\mathcal{F}$. For an edge set $F \subseteq E$, we also use $w(F)$ to denote the total weight of the edges in F. We prove the stronger statement that

$$w(\mathcal{F} \setminus \mathcal{F}^*) \leq \lambda \cdot w(\mathcal{F}^* \setminus \mathcal{F}). \tag{1}$$

We show (1) by induction on $|\mathcal{F} \setminus \mathcal{F}^*| = |\mathcal{F}^* \setminus \mathcal{F}|$. First note that if $|\mathcal{F} \setminus \mathcal{F}^*| = 0$, we have $\mathcal{F} = \mathcal{F}^*$ and thus (1) is clearly true. Further, if $|\mathcal{F} \setminus \mathcal{F}^*| = 1$, there is exactly one edge $e \in \mathcal{F} \setminus \mathcal{F}^*$ and exactly one edge $e^* \in \mathcal{F}^* \setminus \mathcal{F}$. Because $\mathcal{F}$ and $\mathcal{F}^*$ are spanning forests, e^*

connects the two components of the k-cut $(V_{e,1}, \ldots, V_{e,k+1})$ induced by removing e from $\mathcal{F}$ and we therefore have $w(e) \leq \lambda \cdot w(e^*)$, implying (1).

Let us therefore assume that $|\mathcal{F} \setminus \mathcal{F}^*| = \ell \geq 2$ and let e be a maximum weight edge of $\mathcal{F} \setminus \mathcal{F}^*$. Let $(V_{e,1}, \ldots, V_{e,k+1})$ be the k-cut induced by removing e from $\mathcal{F}$. Further, let $\mathcal{F}'$ be a spanning forest of G that is obtained by removing e from $\mathcal{F}$ and by adding some edge $e^* \in \mathcal{F}^* \setminus \mathcal{F}$ that connects two components $V_{e,i}$ and $V_{e,j}$ where $1 \leq i \neq j \leq k+1$ such that $\mathcal{F}'$ satisfies the property P. Note that by the assumptions of the theorem, we have $w(e) \leq \lambda \cdot w(e^*)$. To prove (1), it thus suffices to show that $w(\mathcal{F}' \setminus \mathcal{F}^*) \leq \lambda \cdot w(\mathcal{F}^* \setminus \mathcal{F}')$. We have $|\mathcal{F}' \setminus \mathcal{F}^*| = \ell - 1$ and thus, if the spanning forest $\mathcal{F}'$ satisfies the conditions of the theorem, $w(\mathcal{F}' \setminus \mathcal{F}^*) \leq \lambda \cdot w(\mathcal{F}^* \setminus \mathcal{F}')$ and (1) follows from the induction hypothesis. We therefore need to show that $\mathcal{F}'$ satisfies the conditions of the theorem.

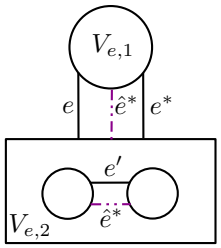


Figure 1 A spanning forest with $k = 1$ connected subtree. We can simply call it a spanning tree. The magenta edges indicate the two possibilities for $\hat{e}^*$.

Consider an arbitrary edge $e' \in \mathcal{F}' \setminus \mathcal{F}^*$ and let $(U_{e',1}, \ldots, U_{e',k+1})$ be the partition of V induced by removing e' from the forest $\mathcal{F}'$. Since e' is an edge of one of the $k+1$ subtrees of $\mathcal{F}$ resulting after removing e, e' connects two nodes in one of the component $V_{e,z}$ where $1 \leq z \leq k+1$. We need to show that for every edge $\hat{e}^* \in \mathcal{F}^* \setminus \mathcal{F}'$ crossing the k-cut $(U_{e',1}, \ldots, U_{e',k+1})$ in which the resulted spanning forest that is obtained by removing e' from $\mathcal{F}'$ and by adding the edge $\hat{e}^*$ satisfies the property P, it holds that $w(e') \leq \lambda \cdot w(\hat{e}^*)$. Such an edge $\hat{e}^*$ has to crosses the k-cut induced by removing e' from $\mathcal{F}$ or crosses the k-cut induced by removing e from $\mathcal{F}$. In the first case, we have $w(e') \leq \lambda \cdot w(\hat{e}^*)$ by the assumptions of the theorem. In the second case, we have $w(e') \leq w(e) \leq \lambda \cdot w(e^*) \leq \lambda \cdot w(\hat{e}^*)$ (recall that we chose e to be the heaviest edge from $\mathcal{F} \setminus \mathcal{F}^*$). This concludes the proof. ◀

A feasible solution for the k-TSP is a spanning forest with k connected subtrees that satisfies a property $\mathcal{P}$. *A spanning forest of requests in R satisfies the property $\mathcal{P}$ if each of the k subtrees of the spanning forest includes exactly one dummy request.* The cost of an optimal solution, thus, is lower bounded by the total length of a an MSF that satisfies the property $\mathcal{P}$. Therefore, the Theorem 1 holds if we show that the total length of the spanning forest resulted by GNN equals the total length of an MSF that satisfies the property $\mathcal{P}$.

▶ **Lemma 4.** *Let $\mathcal{F}$ be the spanning forest resulted by GNN and let $\mathcal{F}^*$ be an MSF of requests in R with $k \geq 1$ connected subtrees that satisfies the property $\mathcal{P}$. For every edge e of $\mathcal{F}$, consider the lightest edge e^* of $\mathcal{F}^*$ crossing the k-cut induced by removing e from $\mathcal{F}$ such that the spanning forest that is resulted by removing e from $\mathcal{F}$ and by adding e^* satisfies the property $\mathcal{P}$. The length of e equals the length of e^*.*

Proof. Denoted by $\tau_1, \tau_2, \ldots, \tau_k$ the k connected subtrees (TSP paths) of the spanning forest $\mathcal{F}$. Let R_i denote the request set of the subtree τ_i that are served by the server s_i for all $1 \leq i \leq k$. Assume that $(R_1, \ldots, R_{j-1}, R_{j,1}, R_{j,2}, R_{j+1}, \ldots, R_k)$ is the k-cut that is resulted

by removing any arbitrary edge $e = (r_p, r_q)$ of $\mathcal{F}$ from the subtree τ_j. W.l.o.g. assume that the dummy request of subtree τ_j is in $R_{j,1}$ and further $r_p \in R_{j,1}$ and $r_q \in R_{j,2}$. The lightest edge $e^* = (r_x, r_y)$ of $\mathcal{F}^*$ must connect one request, say r_y, in $R_{j,2}$ and one request, say r_x, in one of the k sets $R_1, \ldots, R_{j-1}, R_{j,1}, R_{j+1}, \ldots, R_k$ to guarantee that the spanning forest that is resulted by removing e from $\mathcal{F}$ and by adding e^* satisfies the property $\mathcal{P}$. Assume that the edge e^* is shorter than e. We will show some contradiction because of this assumption. Let the subtree T'' of T denote the lowest subtree of T in which it includes both r_x and r_y. Therefore, the length of the longest shortest path between any two leaves of T'', say *the diameter of* T'' is smaller than the length of e.

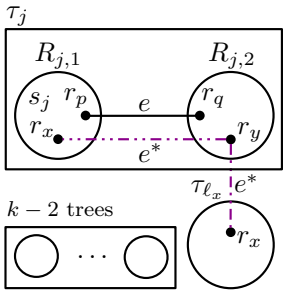


Figure 2 The spanning forest $\mathcal{F}$ with k spanning trees. The magenta edges indicate the two possibilities for the edge e^*.

Case $r_x \in R_{j,1}$: The GNN algorithm serves r_x by s_j before serving the request r_p since r_p is the last served request in $R_{j,1}$. Therefore, the order of requests served by s_j is $\ldots, r_x, \ldots, r_p, r_q, \ldots, r_y, \ldots$. This order of serving the requests together with the assumption in which the diameter of T'' is smaller than the length of e imply that s_j leaves the subtree T'' before serving r_y. However, this contradicts the fact that a server does not leave any subtree of T while there is still an unserved request there, w.r.t. the GNN algorithm description.

Case $r_x \in R_{\ell_x}$ for $\ell_x \in [1,k]$ and $\ell_x \neq j$: If $\ell_x < j$, i.e., s_{ℓ_x} moves earlier than s_j, then the fact that r_y is not served by s_{ℓ_x} implies that the server s_j during the serving phase of s_{ℓ_x} is in T'' closer to r_y than r_x to r_y (note that s_{ℓ_x} serves r_x). Hence, s_j serves the requests in the order of $\ldots, r_y, \ldots, r_p, r_q, \ldots$ using the fact that r_q is immediately served after serving the request r_p by s_j and the assumption that states the diameter of T'' is smaller than the length of e. However, the initial location of s_j is in $R_{j,1}$ and r_p is actually served before r_y and it is a contradiction.

By contrast if s_j moves earlier than s_{ℓ_x}, then the fact that r_x is not served by s_j implies that the server s_{ℓ_x} is in T'' closer to r_x than r_y to r_x during the serving phase of s_j (note that r_y is served by s_j). The requests r_p and r_q cannot both be in T'' because the diameter of T'' is shorter than e. Therefore, the server s_j must enter the subtree T'' with respect to the fact that r_p and r_q are served before r_y by s_j. However, this contradicts the fact that the GNN algorithm does not move a server into a subtree where there is another server. ◀

Let P $= \mathcal{P}$. Theorem 1 then directly follows from Theorem 3 and Lemma 4.

3 Applications

In this section it is shown that the OSD problem and the distributed queuing problem are identical to the k-TSP (TSP with $k \geq 1$ salespeople) if the requests arrive at the same time.

3.1 OSD Problem

In the OSD problem, the time that takes until a request is served count for nothing. Hence, a set of requests can all be served at some point of time when an online algorithm that solves the OSD problem moves a server. Note, however, the online algorithm must pay the total distance travelled by the servers to serve a set of requests. In the OSD problem the goal is to minimize the sum of distance traveled by the servers and the total delay of serving requests where delay of a request is the difference between the times when the request is issued and served [1].

All the requests in the OSD problem are instantaneously served when they arrive at the same time and hence the OSD problem with $k \geq 1$ servers is transformed into the k-TSP where the goal is to minimize the total movement cost of the servers.

3.2 Distributed Queuing Problem

Consider a given network. Each processor of the given network in a distributed online algorithm has to make its decision only based on the local information it has. Therefore, it must communicate with other processors to obtain necessary parts of the global picture of the configuration to make its decision. This limitation to not have a global picture of the system makes the work of an online algorithm in a distributed system harder than centralized system against an offline algorithm who suffers neither from having local information nor incomplete knowledge of the request sequence.

In the distributed queuing problem, the input network is modeled by a weighted graph. The weight of each edge is the distance between its two endpoints. Nodes are supposed to be the processors of the network and the edges are supposed to be the communication links between the processors. For the distributed queuing problem, we assume the standard message passing model where in a synchronous system the delay for sending a message over an edge is exactly the weight of the edge. In the distributed queuing problem as described in Section 1, the delay of a request is the difference between the times when the request is issued and enqueued. It was also mentioned that a newly arrived request is said to be enqueued whenever the node of its predecessor in the global order "learns" about it. This is done in such a way that the node of the new request sends a "find message" to the node of its predecessor. As soon as the predecessor request is issued and as soon as the the find message by the successor node reaches to the node of predecessor request, we say that the node of the predecessor request learns about the successor request. The goal is to minimize the total delay cost of all requests.

When the requests arrive at the same time, because all the requests are available in the system the delay of a request equals the difference between the times when the request is issued and when the find message by the new request reaches to the node of its predecessor request. In a synchronous execution, this delay equals the length of the path between the nodes of two consecutive requests in the global order. Therefore, the problem now is equivalent to the *distributed TSP* where the goal is to minimize the total distance travelled by the shared object/server. To show that the distributed queuing problem is also optimally solved on HSTs when the requests arrive at the same time, w.r.t. the Theorem 1 it remains to provide a distributed version of Nearest-Neighbor algorithm that can solve the distributed TSP.

Distributed Nearest-Neighbor Algorithm.

The Arrow algorithm is a simple and elegant link reversal distributed algorithm that solves the distributed queuing problem. The Arrow algorithm can only operate on directed tree networks. In Arrow algorithm, initially all nodes point to the location of the server, say v, that also hosts the dummy request and thus v is now the tail of the queue. A node u issues a request to access to the server. Thus, u sends a message along the arrows until it finds the current tail of the queue that is now v. Note that, at each time, there is only one outgoing edge for each node and hence there is only one unique path from each node towards the current tail of the queue. When the find message by u reaches to v, v sends the shared object to u. While the message follows the path between u and v, the directions of the arrows are reversed such that now all the nodes point to u and thus u is the current tail of the queue. It was shown that Arrow correctly works even if the requests are concurrently and dynamically issued at any time. For more details about the distributed queuing problem, the distributed model in which the problem is studied, and the Arrow algorithm description refer to [6, 7, 5].

In a synchronous execution of Arrow on a given tree, the delay of a request is always equal to the distance (d_T) between the respective nodes in the tree because Arrow always finds the predecessor on the direct path. The Arrow algorithm greedily orders the requests as follows: suppose $i-1$ requests are ordered. The request $r=(v,t)$ that is issued at time t at node v is ordered by Arrow as the i-th request if the find message by v sent at time t reaches to the current tail of the queue earlier than all find messages by the remaining requests that are not enqueued yet. This greedy property of Arrow is formally shown in [5].

▶ **Lemma 5** (Lemma 7 in [5] rephrased). *Consider a synchronous execution of* Arrow *on tree T and consider two arbitrary requests $r_i=(v_i,t_i)$ and $r_j=(v_j,t_j)$ for which r_j is ordered after r_i by* Arrow. *Then it holds that $t_i+d_T(v',v_i)\leq t_j+d_T(v',v_j)$ where $r'=(v',t')$ is the predecessor request of r_i in the resulted global order by* Arrow.

Consider two arbitrary requests $r_i=(v_i,t_i)$ and $r_j=(v_j,t_j)$ for which r_j is ordered after r_i by Arrow and $r'=(v',t')$ is the predecessor of r_i. When the requests arrive at the same time we have $t_i=t_j$ and regarding to the Lemma 5, v_i must be the closest requested node (ties are arbitrarily broken) to v' among all remaining requested nodes. This implies that the Arrow algorithm exactly implements the distributed version of Nearest-Neighbor algorithm w.r.t. the tree distances if the requests arrive at the same time. Hence, a requested node that is the closest to the current tail of the queue is ordered and be the new tail of the queue.

Distributed GNN Algorithm on Overlay Trees.

The distributed queuing problem is traditionally studied when there is one shared object/server. Also, the Arrow algorithm can solve the problem when there is only one server. In the following, we provide a new distributed queuing algorithm that is Arrow with some modification that is run on *overlay trees* to support not only $k=1$ server, but also $k>1$ servers. An overlay tree is constructed on top of the original network whose leaves are the computing nodes of the system. The modified Arrow simulates the GNN algorithm in a distributed setting when the requests are issued at the same time. The solution of modified Arrow algorithm consists of $k\geq 1$ orders each corresponds to a server. All the requests in each order therefore are served by the corresponding server.

Consider a set of k servers each with a dummy request located at the leaves of the given rooted tree T. Further, suppose a set of requests that are available at the leaves of T from the beginning. W.l.o.g. assume that the servers are at different leaves of T. In a quiescent state, for any server, all the nodes on the path from the root of T to the leaf that hosts the server, point to the server. Further, each leaf that hosts a server points to itself. Hence, we have k directed path with downward arrows from the root of T to the points of the current tails of the orders. Any other node points to its parent with upward arrows.

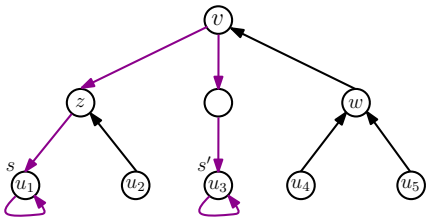


Figure 3 Modified Arrow algorithm: initial system state. The servers s and s' serve requests in orders π and π', respectively. The dummy requests at u_1 and u_3 are the tails of π and π', respectively.

- **New request r at leaf u:** If the leaf u points to itself, then r is enqueued behind the last request that has been issued by u. Otherwise, the leaf u atomically sends a "find message" to its parent through an upward arrow. The arrow from u to its parent is removed and u then points to itself.

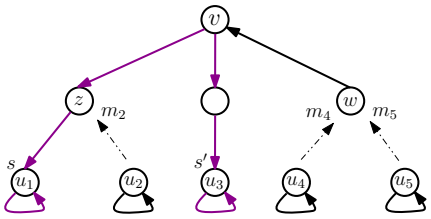
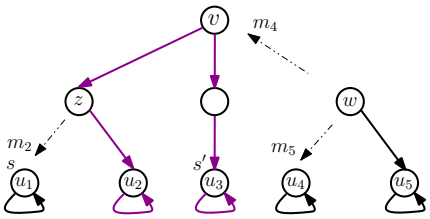
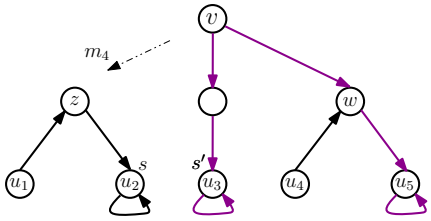


Figure 4 Modified Arrow algorithm step 1: nodes u_2, u_3, u_4, and u_5 initiate requests r_2, r_3, r_4, and r_5 and send "find messages" m_2, m_3, m_4, and m_5, respectively, along the arrows.

- **Upon w receiving the "find message by u" from node v:** If w has an upward arrow, then the "find message by u" is atomically forwarded to its parent, the upward arrow from w to its parent is removed and then w points to v using a downward arrow. Otherwise, if w points to itself then the request r issued by u is enqueued behind the last request that has been issued by w. Then, w removes the arrow that points to itself and points to v using an upward arrow. However, if w neither has an upward arrow nor points to itself, then it must have at least one downward arrow. Ties are arbitrarily broken and the "find message by u" is atomically forwarded to one child through a downward arrow. The node w then removes the downward arrow and points to v.

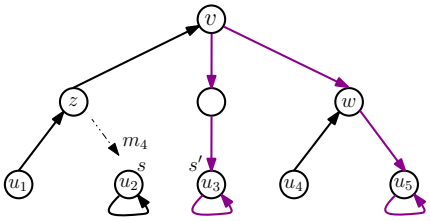


(a) Modified Arrow algorithm step 2

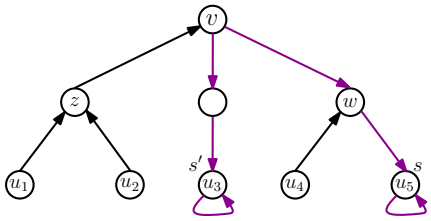


(b) Modified Arrow algorithm step 3

Figure 5 The "find messages" follow the arrows, reversing their directions along their ways. (a) The request r_3 is the current tail of π'. Both m_4 and m_5 reach to w at the same time and w arbitrarily forwards m_4 towards v and therefore m_5 has been deflected towards u_4. The request r_2 is the current tail of π. (b) The request r_2 is ordered behind the dummy request of π and s moves to u_2. The request r_5 is ordered behind r_4 while the "find message" by u_4 is (arbitrarily) forwarded to z by v. The request r_5 is the current tail of π.



(a) Modified Arrow algorithm step 4



(b) Modified Arrow algorithm step 5

Figure 6 As it is shown in Figures 3-6, there are always two connected path with magenta arrows from the root to the current tails of the orders. (b) The request r_4 is ordered behind r_2 and s moves to u_4. The server s immediately moves from u_4 to u_5 since r_5 has been already ordered behind r_4.

The Arrow algorithm with the above modification, simulates the GNN algorithm in the distributed setting as follows. While the GNN algorithm sequentially orders the requests, the Arrow with modification orders the requests in parallel. The GNN algorithm 1) does not move a server outside of a subtree of T if there is still an unserved request there and 2) does not enter a server into a subtree of T to serve a subset of unserved requests if there is another server in the subtree. The first property of GNN is guaranteed by the Lemma 5 for the Arrow with modification like the original Arrow. In other words, in any order among the k orders resulted by the Arrow with modification, a request r will be the new tail of the order if it is the closest request to the current tail of the order w.r.t. the tree distances among all other requests that appear after r in the order. The second property of GNN is guaranteed using the downward arrows from the root of T to the k current tails of the orders. Consider the current tail of the order π_i that is in the subtree T' of T and the current tail of the order π_j that is in another subtree T in which the two subtrees are the highest subtrees such that each of the two subtrees does not include both tails of π_i and π_j. Any request $r = (v, t)$ in T' is not enqueued in the order π_j since the find message by v always visits a downward arrow inside T' and therefore it cannot visit any downward arrow that points to the current tail of π_j.

▶ Remark. Although the modified Arrow algorithm as described optimally solves the distributed queuing problem with $k \geq 1$ servers on T when the requests are aissued at the same time, it cannot be competitive where there are $k > 1$ servers and when the requests are issued at any time. The reason is the same as the one for Nearest-Neighbor algorithm

that is not competitive for the classic k-server problem. Consider two different subtrees of T that are far from each other and each of them hosts a server. If two leaves of one of the subtrees issue a large enough number of requests one after each other, the server in the subtree must travel the distance between the two leaves many times while the other server is idle in the second subtree.

4 Conclusion

In [1], the authors left this question open in which whether there is an online algorithm that provides a constant competitive ratio for the OSD problem. The resemblances between the studied two problems together with the constant competitive ratio for the distributed queuing problem provided in [5] on HSTs, build our hopes up to get constant competitive ratio for the OSD problem on HSTs.

References

1 Y. Azar, A. Ganesh, R. Ge, and D. Panigrahi. Online service with delay. In *Proceedings of the 49th ACM Symposium on Theory of Computing (STOC)*, pages 551–563, 2017.

2 Y. Bartal and A. Rosen. The distributed k-server problem-a competitive distributed translator for k-server algorithms. In *Proceedings of the 33rd Symposium on Foundations of Computer Science (FOCS)*, pages 344–353, 1992.

3 M. Bellmore and G. Nemhauser. The traveling salesman problem: a survey. *Operations Research*, 16(3):538–558, 1968.

4 J. Fakcharoenphol, S. Rao, and K. Talwar. A tight bound on approximating arbitrary metrics by tree metrics. In *Proceedings of the 35th ACM Symposium on Theory of Computing (STOC)*, pages 448–455, 2003.

5 A. Ghodselahi and F. Kuhn. Dynamic analysis of the arrow distributed directory protocol in general networks. In *Proceedings of the 31st International Symposium on Distributed Computing (DISC)*, pages 22:1–22:16, 2017.

6 M. Herlihy, S. Tirthapura, and R. Wattenhofer. Competitive concurrent distributed queuing. In *Proceedings of the 20th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 127–133, 2001.

7 F. Kuhn and R. Wattenhofer. Dynamic analysis of the arrow distributed protocol. In *Proceedings of the 16th ACM Symposium on Parallelism in Algorithms and Architecture (SPAA)*, pages 294–301, 2004.

8 M. Manasse, L. McGeoch, and D. Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, 11(2):208–230, 1990.

9 D. Rosenkrantz, R. Stearns, and P. Lewis. An analysis of several heuristics for the traveling salesman problem. *SIAM Journal on Computing*, 6(3):563–581, 1977.