Delay Constrained Subtree Homeomorphism Problem with Applications

Sridhar Radhakrishnan, Senior Member, IEEE, Shankar M. Banik, Member, IEEE, Venkatesh Sarangan, and Chandra N. Sekharan, Member, IEEE

Abstract—Virtual world and other collaborative applications are increasingly becoming popular among Internet users. In such applications, users interact with each other through digital entities or avatars. In order to preserve the user experience, it is important that certain Quality of Service (QoS) requirements (e.g. delay, bandwidth) are satisfied by the interactions. These QoS requirements are usually defined by the application designer. When applications with such QoS requirements are being deployed on a network of servers, an appropriate set of servers capable of satisfying the QoS constraints of the interactions must be identified. This identification process is nothing but the subgraph homeomorphism problem. In this paper, we present polynomial-time solutions for a special case of this problem viz. subtree homeomorphism problem, wherein the guest as well as the host graphs are both trees. We also discuss generalizations of the subtree homeomorphism problem and present polynomial time solutions.

Index Terms—virtual worlds, interaction tree, overlay networks, subtree homeomorphism, homeomorphic embedding.

I. INTRODUCTION

Given two graphs Guest and Host, the subgraph homeomorphism problem is to determine if there is an injective mapping of vertices of the Guest into the vertices of the Host such that edges of the Guest are mapped into vertex-disjoint paths in the host graph.

The general subgraph homeomorphism problem is NP-complete [1] wherein both the guest and host graphs are arbitrary graphs. Recently, Lingas and Wahlén [2] provided an exponential-time exact algorithm for finding subgraph homeomorphism. Fortune et. al [3] have shown that the subgraph homeomorphism problem is polynomial-time solvable when the input graphs are directed and acyclic. Chung [4] provided the first polynomial time algorithm for the homeomorphism problem when the host and guest graphs are restricted to trees. The algorithm by Chung [4] has been used for many applications including computational evolutionary biology [5], [6].

The subtree homeomorphism problem arises when the guest and host are both trees. In this paper, we focus on a variant of this problem namely Delay-Constrained Subtree Homeomorphism Problem (DCSHP), wherein both the Guest and Host trees have edge weights. More formally, let \( G = (V_G, E_G, W_G) \) and \( H = (V_H, E_H, W_H) \) be the Guest and Host trees with corresponding vertex set \( V \) and edge set \( E \) marked with subscripts \( G \) and \( H \), respectively. We will denote a vertex (resp. edge) \( u \) (resp. \( e \)) belonging to a graph network \( G \) with its subscript \( u_G \) (resp. \( e_G \)). Let \( W_G \) (resp. \( W_H \)) be the weight function that defines the weights on the edges of the Guest (resp. Host) tree with \( W_G : E_G \to \mathbb{R}^+ \) (resp. \( W_H : E_H \to \mathbb{R}^+ \)). Let \( P(v_1 \sim v_k) = \{(v_1, v_2), (v_2, v_3), \ldots, (v_{k-1}, v_k)\} \) be a path between two vertices \( v_1 \) and \( v_k \). The weight of the path \( d(P(v_1 \sim v_k)) = \sum_{i=1}^{k-1} W_G(v_i, v_{i+1}) \), where the weight function \( W \) is any of the weight functions \( W_G \) or \( W_H \) as defined above. We note here that even though we consider delay as the edge weights of \( G \) and \( H \), the discussions in this paper are applicable to other additive metrics such as cost and reliability.

We can say that a Guest tree \( G \) can be homeomorphically embedded into a Host tree \( H \) with delay constraints, if there is a vertex mapping function \( f : V_G \to V_H \), satisfying the following conditions:

(a) the function \( f \) is a one-to-one function,
(b) for any pair of edges \( e^1_G = (u^1_G, v^1_G) \) and \( e^2_G = (u^2_G, v^2_G) \), the paths \( P_H(f(u^1_G) \sim f(v^1_G)) \) and \( P_H(f(u^2_G) \sim f(v^2_G)) \) are vertex-disjoint, and
(c) for any edge \( e_G = (u_G, v_G) \), \( d(P_H(f(u_G) \sim f(v_G))) \leq W_G(e_G) \).

The above problem of homeomorphically embedding a \( G \) into \( H \) with delay constraints is named as the Delay-Constrained Subtree Homeomorphism Problem (DCSHP). Without the condition (c) above and with \( H \) and \( G \) as general networks (resp. tree networks) the problem described above is the subgraph (resp. subtree) homeomorphism problem. When we relax condition (b) to allow paths in the host graph to be shared, we have a rather less constrained version of the problem which we term as Delay-Constrained Subtree Pseudo-Homeomorphism Problem (DCSHP).

Relaxing the one-to-one function in condition (a) to allow more guest nodes to be mapped on to a single host node will give rise to a new problem that we will term as DCSHP with load factors. When we relax conditions (a) and (b) as described above, the resulting problem will be called as DCSHP with load factors. We also note here that the pseudo-homeomorphism is closely related to the problem of minor containment \([7]\) wherein the vertices of the guest are mapped

\(^1\)The keen reader will observe that reliability is a multiplicative metric. However, it can be converted to an additive metric through the \( \log \) operator.
to disjoint connected subgraphs of the host graph. These subgraphs are contracted (the minor operation) and the edges of the guest are mapped to the edges of the host. One of the relaxations for our pseudo-homeomorphism definition is we do not require the paths in the host that are mapped to the edges in the guest to be neither vertex nor edge disjoint.

In this paper, we design polynomial time algorithms for solving the DCSHP and its generalizations. To the best of our knowledge, our work is the first to discuss the problem of delay constrained subgraph homeomorphism on trees and provide a polynomial solution to it. The rest of this paper is organized as follows. We bring out the significance of the DCSHP in section II by discussing its usefulness in designing online virtual worlds. Our proposed algorithm along with its complexity is described in section III and IV, respectively. In section V, we extend our problem to the delay-constrained pseudo-homeomorphism problem and in section VI we consider the case wherein the the host nodes can accommodate more than one guest nodes. Performance evaluation of the proposed algorithms are presented in section VII. Conclusions and future research are presented in section VIII.

II. APPLICATIONS OF DCSHP

Digital Habitats or Virtual Worlds, wherein a number of digital entities interact in a networked virtual environment, are fast becoming commonplace [8], [9]. Distributed interactive simulations [10], [11], multi-player online games [12], [13], [14], and shared virtual worlds [15], [16], [17] are well-known instantiations of digital habitats. In these habitats, the population of digital organisms (a.k.a. avatars) ranges from tens to tens of thousands. The avatars are usually involved in interactions with other avatars and/or the digital habitat. The interactions between avatars are controlled by real-world users through the Internet.

Typically, the state of a virtual world is stored in one or more servers. At times, the state is partitioned among the servers based on the geographical proximity in the virtual world – i.e., clusters of virtual locations that are closer to each other will be placed under the same partition and assigned to an individual physical server [18], [19]. In each partition, there will be several users interacting with the virtual world. If any user action is performed in a given partition, this action has to be reflected across all the partitions within the region of influence. This region of influence varies as per the virtual world application and can be as narrow as the 1 hop neighborhood or as wide as the n hop neighborhood. Further, there could be delay constraints associated with the reflections. In other words, an action done by an user at the source partition should get reflected across each of the partitions in the region of influence within appropriate time delays.

For any partition, the delay constraints associated with reflecting an user action can be expressed using a weighted rooted tree – the source partition being the root, other partitions in the region of influence being the other tree nodes, and the weights on the tree edges being the appropriate allowable delays. There will one such constraint tree for each of the partitions. However, if all such trees are to be maintained by the virtual world, scalability issues in terms of managing packet forwarding tables at the overlay routers – similar to those experienced in source based multicasting [20], could arise. A possible way to overcome this issue is to combine the individual constraint trees into a common ‘world-wide’ constraint tree that describes the forwarding constraints among the partitions in the virtual world (similar to the common group tree in multicasting)\(^2\).

It is easy to see that when the common constraint tree is obtained, determining an appropriate partition-server mapping that satisfies the delay constraints is a homeomorphic embedding problem. In other words, the constraint tree with the delay requirements acts as the guest, the server overlay network acts as the host, and the goal is to find a homeomorphic embedding of the guest on the host. In this paper, we focus on the scenario wherein the guest and host graphs are both trees with maximum tolerable delay along a guest link being the constraint to be satisfied. We present a polynomial time algorithm to find a delay constrained homeomorphic embedding of a guest tree onto a host tree.

Digressing a bit, we note that one could possibly build on our algorithm to find a delay constrained homeomorphic embedding if the host happens to be a general network. For example, one could envision a solution wherein, all spanning trees of the host network are determined and our polynomial time algorithm is run on each of the spanning tree. Clearly, the number of spanning trees in a general graph is exponential. To reduce this complexity, one might search for the embedding on the shortest path trees of the host graph. The keen reader will deduce that the unique host tree onto which the guest might be embeddable need not be a shortest path tree. Nevertheless, since we are dealing with delay-constraint embedding wherein the maximum tolerable delay is specified, searching all the shortest path trees in the host might be a good starting point.

Since the guest graph is a tree with delay constraints, one may be tempted to adopt some of the well-known QoS constrained multicasting tree discovery techniques to find a suitable partition – server mapping. However, when multicasting trees have to be found, some of the ‘host’ nodes (such as the source and destinations) that need to be a part of the mapping are already known. This knowledge is leveraged by the multicasting tree discovery algorithms. On the other hand, in our partition-server mapping problem, no such knowledge is available. Hence, multicasting tree finding solutions may not be readily applicable.

III. ALGORITHM DESCRIPTION

A. TERMINOLOGY USED IN THE ALGORITHM

Our proposed algorithm is inspired by the algorithm of Chung [4] and modified to solve DCSHP. For the sake of simplicity, we denote the host tree as \(H\), the guest tree as \(G\), and the delay function as \(d(.)\). A host tree \(H = (V_H, E_H)\) with root node \(r\) is called a rooted tree and is denoted as \(H_r = (V_H, E_H, r)\). The rooted tree specifies the direction for each edge which points away from the root. The subtree generated

\(^2\)The process of obtaining such a common constraint tree depends on the application designer and is beyond the scope of this paper.
by node $v$ in $H_r$ is denoted as $H_r(v)$. If the guest tree is rooted at $v'$, then it can be represented by $G_{v'} = (V_G, E_G, r')$. For each node $v \in V_H$ in $H_r$, we define the following two terms:

- $DS_r(v) = \{x \in V_G |$ there is a subtree of $H_r(v)$ which is delay constrained homeomorphic to $G_{r'}(x)\}$
- $P_r(v) = \{x \in V_G | x$ can be embedded on $v\}$

For each node $u \in V_G$ in $G_{r'}$, we define the following two terms:

- $TE(u) = \{x \in V_H | u$ can be embedded on $x$ and the embedding satisfies the delay constraints $\}$
- $FE(u) = \{x \in V_H | u$ is finally embedded on $x$.\}

Now if we can show that $r' \in DS_r(v)$ which implies that the subtree rooted at $v$ in $H_r$ has a delay constrained subtree homeomorphic to $G_{r'}$, then we have solved our problem. $DS_r(v)$ and $P_r(v)$ for each node $v \in V_H$ in $H_r$ is computed in bottom-up fashion. Initially $DS_r(v)$ and $P_r(v)$ is computed for all the leaf nodes in $H_r$ and the leaf nodes are marked. Note that all the leaf nodes in $G_{r'}$ will be included in $DS_r(v)$ and $P_r(v)$ if $v$ is a leaf node in $H_r$. Next, $DS_r(v)$ and $P_r(v)$ is computed for a non-leaf node $v$ in $H_r$, if $DS_r(v)$ is computed for all the children $w$ of $v$ in $H_r$. It can be easily proved that if $u \in DS_r(w)$ and $v$ is the parent of $w$, then $u \in DS_r(v)$. It can also be easily proved that if $u \in G_{r'}$ and $u$ is a leaf node, then $u \in P_r(v)\forall v \in H_r$.

Suppose node $v$ in $H_r$ has children $x_1, x_2, \ldots, x_s$, and node $u$ in $G_{r'}$ has children $y_1, y_2, \ldots, y_t$ and $x_i$ and $y_j$ are the leaf nodes of $H_r$ and $G_{r'}$, respectively for $1 \leq i \leq s$ and $1 \leq j \leq t$. Now we need to decide whether $u \in DS_r(v)$. This problem can be solved by a bipartite matching. We construct a bipartite graph $B$ with partitions $X$ and $Y$ where $X$ is the set of children of $v$ in $H_r$ and $Y$ is the set of children of $u$ in $H_r$. An edge $(x_i, y_j) \in B$ iff $y_j \in P_r(x_i)$ and $d(v, x_i) \leq d(u, y_j)$ where $d(v, x_i)$ and $d(u, y_j)$ are the delay of the edges $(v, x_i)$ and $(u, y_j)$ in $H_r$ and $G_{r'}$, respectively. We compute the matching $\mathcal{M}$ of the bipartite graph $B$. If $|\mathcal{M}| = |Y|$, then we include $u$ in $DS_r(v)$ and in $P_r(v)$ and mark $v$. This is illustrated with an example in Figure 1.

Initially, we compute the $DS_r(\cdot)$ and $P_r(\cdot)$ for all the leaf nodes in $H_r(v)$. $DS_r(x_1) = \{y_1, y_2, y_3\}$, $DS_r(x_2) = \{y_1, y_2, y_3\}$, $DS_r(x_3) = \{y_1, y_2, y_3\}$, $DS_r(x_4) = \{y_1, y_2, y_3\}$, $P_r(x_1) = \{y_1, y_2, y_3\}$, $P_r(x_2) = \{y_1, y_2, y_3\}$, $P_r(x_3) = \{y_1, y_2, y_3\}$, and $P_r(x_4) = \{y_1, y_2, y_3\}$ and mark the leaf nodes $x_1, x_2, x_3$ and $x_4$ as mapped. Next, we construct the bipartite graph $BP$ with $X = \{x_2, x_3, x_1\}$ and $Y = \{y_1, y_2, y_3\}$. Edge $(x_2, y_1)$ will be included in $BP$ as $y_1 \in P_r(x_2)$ and $d(v, x_2) \leq d(u, y_1)$. Similarly edges $(x_2, y_2)$, $(x_2, y_3)$, $(x_3, y_3)$, $(x_4, y_1)$, and $(x_4, y_3)$ will be included in $BP$. The matching $\mathcal{M}$ of $BP$ will be $\{(x_2, y_1), (x_2, y_2), (x_4, y_3)\}$ and $|\mathcal{M}| = |Y|$. So $u$ is included in $DS_r(v)$ and $P_r(v)$ and $v$ is marked as mapped.

Next, we generalize $DS_r(v)$ and $P_r(v)$ for any non-leaf node $v$ in $H_r$ based on the following condition. Let $v$ be a node in $H_r$ with children $x_1, x_2, \ldots, x_s$. Let $v$ be a node in $G_{r'}$ with children $y_1, y_2, \ldots, y_t$ and $DS_r(x_i)$ for each child $x_i$ of $v$ for $1 \leq i \leq s$ is computed. Now we construct the bipartite graph $B$ with $X$ and $Y$ where $X = \{x_1, x_2, \ldots, x_s\}$ and $Y = \{y_1, y_2, ..., y_t\}$. In $B$, an edge is created between $x_i$ and $y_j$ if

1. $y_j \in P_r(x_i)$ and $d(v, x_i) \leq d(u, y_j)$
2. $y_j \notin P_r(x_i)$ but there exists a node $x_{ik}$ along the path $(v - x_i - child(x_i) - child(child(x_i)) - \ldots - x_{ik})$ where $y_j \in P_r(x_{ik})$ and $d(v - x_i - child(x_i) - child(child(x_i)) - \ldots - x_{ik}) \leq d(u, y_j)$

IV. ALGORITHM AND ITS TIME COMPLEXITY

A. Algorithm

Based on the discussions in the previous sections, the algorithm for finding whether a given host tree has a subtree which is delay constrained homeomorphic to a given guest tree is given in Algorithm Find Delay Constrained Homeomorphism (FDCH).

Theorem IV.1. The algorithm FDCH is correct.

Proof: In the algorithm FDCH, a node $u \in G_{r'}$ is included in $DS_r(v)$ and $P_r(v)$ of a node $v \in H_r$, only when all the children of $u$ are feasibly mapped to the children of $v$. This is done with the bipartite matching. An edge between $x_i$ (a child of $v$) and $y_j$ (a child of $u$) is created in the bipartite graph, if $x_i$ is feasibly mapped to $y_j$ and the delay of $(v, x_i)$ is less than or equal to the delay of $(u, y_j)$. If $x_i$ is not feasibly mapped to $y_j$, then the algorithm traverses the tree downwards to find out the node which is feasibly mapped to $y_j$. This exhaustive traversal takes care of the possible feasible embedding of an edge in $G_{r'}$ to a path in $H_r$.

B. Illustrating the Algorithm with an example

Suppose we have the following graphs $G$ and $H$ (as shown in Figure 2) where $G$ represents the guest tree and $H$ represents the host tree. We want to find whether $G$ can be homeomorphically embedded on $H$ in such a way that the embedding satisfies the delay constraints on $G$.

We initialize $DS(\cdot)$ and $P(\cdot)$ for all the leaf nodes of $H$. $DS_A(E) = \{s, t, z\}$, $DS_A(F) = \{s, t, z\}$, $DS_A(D) = \{s, t, z\}$, $P_A(E) = \{s, t, z\}$, $P_A(F) = \{s, t, z\}$, $P_A(D) = \{s, t, z\}$. We also initialize $P_A(C) = \{s, t, z\}$, $P_A(B) = \{s, t, z\}$, and $P_A(A) = \{s, t, z\}$. We mark the nodes $E$, $F$, and $D$. Since all the children of $C$ are marked, we compute $DS_A(C)$. Initially, $DS_A(C) = \{s, t, z\}$. Now we want to see whether $y$ is included in $DS_A(C)$ and $P_A(C)$.

We construct a bipartite graph $BP$ where $X = \{E, F\}$ and $Y = \{s, t\}$. The edge $(E, s)$ belongs to $BP$ since
input: Rooted trees \( H_r = (V_H, E_H, r) \) and \( G_r = (V_G, E_G, r') \) with delay function \( d(.) \). 
output: \( FE(u) \forall u \in V_G \) if \( H_r \) has a subtree which is delay constrained homeomorphic to \( G_r \).

1. Initially all the nodes are not marked;
2. foreach leaf node \( v \) of \( H_r \) do
   3. \( DS_r(v) = \{x \mid x \text{ is a leaf node of } G_r\} \);
   4. \( P_r(v) = \{x \mid x \text{ is a leaf node of } G_r\} \);
5. endforeach
6. foreach non-leaf node \( v \) of \( H_r \) do
7. Mark all leaf nodes in \( H_r \);
8. foreach non-leaf node \( v \) of \( H_r \) do
9. if all the children of \( v \) are marked then
   10. Compute \( DS_r(v) \) \( P_r(v) \);
   11. Mark \( v \);
12. end
13. end
14. if \( r' \in DS_r(r) \) then
15. Fix Embedding \( G(r') \);
16. else
17. return "NO EMBEDDING";
18. end

Algorithm FDCH: Find Delay Constrained Homeomorphism

![Guest Tree](image1)
![Host Tree](image2)

Fig. 2. (a) A guest tree and (b) a host tree network used in the illustration of the algorithm.

\[ s \leq P_A(E) \text{ and } d(C, E) < d(y, s) \text{. Similarly the edges } \{(E, t), (F, t), (F, s)\} \text{ are included in } BP. \] The matching \( M = \{(E, s), (F, t)\} \) and \( |M| = |Y| \). So \( y \) is included in \( DS_A(C) \) and \( P_A(C) \). \( DS_A(C) = \{s, t, z, y\} \) and \( P_A(C) = \{s, t, z, y\} \). Also \( TE(y) = C, TE(s) = E, \text{ and } TE(t) = F \). Next we check whether \( x \) is included in \( DS_A(C) \). Again, we construct \( BP \) where \( X = \{E, F\} \) and \( Y = \{y, z\} \). Edge \( (E, y) \) is not included in \( BP \text{ since } y \notin P_A(E) \). Edge \( (E, z) \) is not included in \( BP \text{ as } d(C, E) > d(x, z) \). Similarly edges \( (F, y) \) and \( (F, z) \) are not included in \( BP \). So matching \( M = \{\phi\} \) and \( |M| \neq |Y| \). As a result, \( x \) is not included in \( DS_A(C) \) and \( P_A(C) \). Next we compute \( DS_A(B) = \{s, t, z, y\} \) and \( P_A(B) \). Now we compute \( DS_A(A) \). Initially \( DS_A(A) = \{s, t, z, y\} \). We construct \( BP \) where \( X = \{B, C\} \) and \( Y = \{y, z\} \). We have \( y \notin P_A(B) \). So we check the path \( (A - B - C) \) where \( y \in P_A(C) \) and \( d(A - B - C) \leq d(x, y) \). So edge \( (C, y) \) is included in \( BP \). We can also show that edge \( (D, z) \) is included in \( BP \). Matching \( M = \{(C, y), (D, z)\} \) and \( |M| = |Y| \). So \( x \) is included in \( DS_A(A) \) and \( P_A(A) \). \( DS_A(A) = \{s, t, z, y, x\} \) and \( P_A(A) = \{s, t, z, x\} \). We update \( TE(x) = A, TE(y) = C, \text{ and } TE(z) = D \). Since \( x \) is included in \( DS_A(A) \), the guest graph \( G \) can be homeomorphically embedded in \( H \) and
the embedding satisfies the delay constraint on \( G \). Now we call procedure \( \text{Fix}_G \). The procedure will find the final embedding for all the nodes in the guest graph \( G \) as \( FE(x) = A, FE(y) = C, FE(z) = D, FE(s) = E, \) and \( FE(t) = F \).

C. Time Complexity

In the algorithms presented in this paper, we assume that the weights are computable and comparable in constant time. The algorithm FDCH determines whether \( H_r \) has a subtree homeomorphic to \( G_r \) or not. Let \( |V_H| = n \) and \( |V_G| = m \). Now we compute the complexity of \( DS_r(v) \) and \( P_r(v) \) computation of each \( v \in H_r \). Suppose \( t_i \) and \( s_i \) be the number of children of node \( u \) in \( G_r \) and node \( v \) in \( H_r \), respectively. We can write \( \sum_{i=1} s_i = n - 1 \) and \( \sum_{i=1} t_i = m - 1 \). From [21], we know that the matching problem on a bipartite graph with node partition of sizes \( t \) and \( s \) can be solved in time \( cts \) for some constant \( c \). Hence the time complexity of computing \( DS_r(v) \) and \( P_r(v) \) will be bounded by \( \sum_{j=1}^{m} cs_i^{1.5} \leq cs_i m^{1.5} \). We compute \( DS_r(v_i) \) and \( P_r(v_i) \) for each node \( v \in H \) and \( |V_H| = n \). Also, when we construct the edges in the bipartite graph, if \( y_j \notin P_r(x) \), we traverse down the tree \( H_r \) and find out the node for which \( y_j \) belongs to its \( P_r(\cdot) \) value. The complexity of this operation is \( O(n) \). Hence, the total complexity of FDCH is \( \sum_{j=1}^{m} cs_i^{1.5} n \leq cs_i m^{1.5} n \). In the worst case, \( m \) will be equal to \( n \). The total complexity of the algorithm is \( O(n^{3.5}) \).

V. EXTENSION TO PSEUDO-HOMEOMORPHISM

The algorithm FDCH returns an embedding when we find the homeomorphic subtree of the guest tree on the host tree and a ‘Yes’ otherwise. The ‘No’ answer could be due to the structural constraint of the guest tree, the delay constraints imposed by the guest tree, or both. In Figure 3 we have provided a sample guest tree and a host tree. The FDCH algorithm will return a ‘No’ for the host graph given in Figure 3 (b). Figure 3(b) also shows a possible assignment wherein the edge \((y, t)\) is mapped to the path \( P(B \sim F) \) and the edge \((y, s)\) is mapped to the edge \((B, C)\). Note that the delay constraints specified in Figure 3 (a) are satisfied in the embedding. Since the edge \((B, C)\) is shared, it is not a strict homeomorphism problem and hence the name pseudo-homeomorphism.

A. Towards a solution for Pseudo-Homeomorphism

Consider the tree in Figure 4 which is a result of adding a new node \( F_B \) as a neighbor of node \( B \) and a link weight equal to \( d(F_B \sim B) \). The node \( F_B \) can be thought of as an alias to node \( F \). The mapping shown in Figure 4(b) satisfies the requirement imposed by the graph in Figure 4(a). We can generalize this approach and construct an augmented tree \( A_T \) from the host tree \( T \) as follows. Copy the host tree \( T \) into the augmented tree \( A_T \). For each non-leaf node \( x \) in the tree \( T \) and for each node \( y \) in the subtree rooted at \( x \) that is not a child of \( x \), add a node \( y_x \) (the alias node) as a child of \( x \). Make \( d(x, y_x) \) equal to \( d(P(x \sim y)) \) in \( T \). If the node \( y \) is a non-leaf node, then the entire tree rooted at \( y \) is copied at the node \( y_x \) for homeomorphic embedding. Clearly, the size of the tree augmented can be exponential in the number of nodes of the tree \( T \). We soon see, that the computation of \( P_r(v) \) for each node need not be done for alias nodes since they would already been computed for their real equivalents. This will avoid the algorithm to have to deal with exponential number of nodes.

![Fig. 3. (a) A guest tree, (b) a host network, and a possible pseudo-homeomorphic embedding.](image)

![Fig. 4. (a) A guest tree, (b) the augmented host network, and a possible homeomorphic embedding.](image)

The augmented tree for the tree in Figure 5(a) is shown in Figure 5(b). As evident, the number of nodes in \( A_T \) can be exponential. If we only consider the alias nodes without the nodes in its subtree the number of nodes in \( A_T \) is \( O(n^2) \). Additionally, the maximum number of children for any node in the tree \( A_T \) is proportional to the height of the node in the tree and this value is \( O(n) \).

Lemma V.1. A tree \( G \) is pseudo-homeomorphic to a subtree of \( T \) if and only if there exists a subtree of \( A_T \) that is isomorphic to \( G \).
The delay-constrained subtree pseudo-homeomorphic and from Lemma V.1 we have the following corollary.

**Corollary V.2.** A tree $G$ is pseudo-homeomorphic to a subtree of $T$ if there exists a subtree of $A_T$ that is homeomorphic to $G$.

B. An Algorithm for solving Delay constrained subtree Pseudo-homeomorphism

Given the Corollary V.2, we can now use the FDCH algorithm to determine if the guest is a delay constrained subtree homeomorphic to the augmented tree of the host tree (without the subtrees of the alias nodes). When we determine the set $P_r(v)$ for each node $v$ in the host tree, the same set can be copied for its aliases and hence the computation can be avoided on the entire subtree rooted at the alias node. The formal description of the algorithm is presented in Algorithm A. In Section IV.C where the time-complexity of the algorithm is presented, it was shown that the overall time required for FDCH to complete is dependent multiplicatively on the number of children nodes in the host tree. In an earlier observation on the size of the tree $A_T$, we had noted that the number of nodes in the tree $A_T$ is $O(n^2)$. This bounds the number the number of children nodes to $O(n^2)$ and more precisely we have $\sum_{i=1}^{n} s_i = O(n^2)$. From this observation and the time complexity result of FDCH, we have the following lemma.

**Lemma V.3.** The delay-constrained subtree pseudo-homeomorphism testing can be done in $O(n^{4.5})$ time.

VI. HOMEOMORPHISM AND PSEUDO-HOMEOMORPHISM WITH LOAD FACTORS

In this section, we will consider the case in which the host tree can accommodate more than one guest node, that is, it has a load factor. We will begin with the assumption that the guest tree can be homeomorphically embedded onto the host tree. Our approach for this problem would be to partition the guest tree into a set of connected subtrees such that the size of each subtree does not exceed the load factor on each node of the host tree. A tree is constructed with each partition as a node and this partition tree is used as a guest tree for finding the homeomorphic subtree on the host tree. While there are many ways in which the tree can be partitioned into connected subtrees, we propose a simple technique that does not increase the degree of the original guest tree.

Our partition method proceeds by numbering all the nodes of the guest tree $G$ using breadth-first search (BFS numbering). In the BFS numbering scheme the root is given the number 1. Now starting from the second level each node is numbered sequentially from left to right starting from the highest number given to the previous level. The partitions are formed starting from the root as follows. Pick nodes numbered 1 through $K$ (the load factor on each node of the host tree) to form the first partition $P_1$. The nodes of the first partition are removed.

**Algorithm A:** Find Delay Constrained Pseudo-Homeomorphism

- **Input:** Rooted trees $H_r = (V_H, E_H, r)$ and $G_{r'} = (V_{r'}, E_{r'}, r')$ with delay function $d(.)$
- **Output:** $FE(u)\forall u \in V_{G}$ if $H_r$ has a subtree which is delay constrained pseudo-homeomorphic to $G_{r'}$.
- **1.** $A_T \leftarrow H_r$;
- **2.** Mark all the nodes in $A_T$ to indicate ‘original’;
- **3.** foreach non-leaf and marked node $v \in A_T$ do
  - **4.** foreach node $u$ that is marked in subtree rooted at $v$ do
    - **5.** if node $u$ is not a child of node $v$ then
      - **6.** Add $u_v$ as a child of node $v$ in $A_T$;
      - **7.** $d(v, u_v) = d(P(v \sim u))$ in $A_T$;
    - **8.** end
  - **9.** end
- **10.** end
- **11.** Initially unmark all the nodes in $A_T$;
- **12.** foreach leaf node $v$ of $A_T$ do
  - **13.** $DS_r(v) = \{x|x$ is a leaf node of $G_{r'}\}$;
  - **14.** $P_r(v) = \{x|x$ is a leaf node of $G_{r'}\}$;
  - **15.** end
- **16.** foreach non-leaf node $v$ of $A_T$ do
  - **17.** $P_r(v) = \{x|x$ is a leaf node of $G_{r'}\}$;
  - **18.** end
- **19.** Mark all leaf nodes in $A_T$;
- **20.** foreach non-leaf node $v$ of $A_T$ do
  - **21.** if $v$ is an alias of node $v_a$ then
    - **22.** $P_r(v) = \{x|x \in P_r(v_a)\}$;
    - **23.** end
  - **24.** if all the children of $v$ are marked then
    - **25.** $\text{Compute } DS_r(v)_{P_r(v)}$;
    - **26.** Mark $v$;
    - **27.** end
- **28.** end
- **29.** if $r' \in DS_r(r)$ then
  - **30.** $\text{Fix Embedding}_G(r')$;
- **31.** else return NO;
- **32.** end

Fig. 5. (a) A host network and (b) the corresponding augmented tree
from the tree $G$ which results in a forest of trees. Each tree in the forest is processed similar to the original tree $G$ forming partitions along the way. The process continues until all the nodes in $G$ belong to some partition. A partition $P_j$ is a child of $P_i$ in the partition tree $P_G$, if there is a node $a$ in $P_j$ whose parent is $b$ in $G$ and $b$ is in $P_i$. The edge weight $W_{P_G}(P_i, P_j) = W_G(a, b)$. We have the following lemma.

**Lemma VI.1.** Let $P_G$ be the partition tree of a tree $G$. If $G$ is subtree homeomorphic to $T$, then $P_G$ is subtree homeomorphic to $T$.

**Proof:** If $G$ is subtree homeomorphic to $T$, any subtree of $G$ is also subtree homeomorphic to $T$. Based on the construction of the partition tree, any edge in $P_G$ is also an edge in $G$. This implies that $P_G$ is subtree isomorphic to $G$ and hence $P_G$ is also subtree homeomorphic to $T$.

**Input:** Rooted trees $H_r=(V_H,E_H,r)$ and $G_{r'}=(V_G,E_G,r')$ with delay function $d(.)$ and load factor value of $K$.

**Output:** Yes if $H_r$ has a subtree which is delay constrained homeomorphic to $G_{r'}$ with load factors.

1. if $(FDCH(H_r, G_{r'}) = 'NO')$ then
2. return ‘NO’;
3. end
4. Construct the partition tree $P_G$ of $G$ with load factor $K$;
5. if $(FDCH(P_G, G_{r'}) = 'NO')$ then
6. Execute Algorithm A ($P_G, G_{r'}$);
7. end
8. return ‘YES’;

**Algorithm B:** Find Delay Constrained Pseudo-Homeomorphism with load factors.

The partition tree is now treated as the guest tree for the algorithm FDCH. If the algorithm FDCH fails to find the homeomorphic subtree we can use the Algorithm A that will allow for sharing links. The partition tree can be constructed in $O(n)$ time. After the partition tree has been constructed, the rest of the algorithm has the same complexity as that of FDCH which is $O(n^{3.5})$ for the homeomorphism version or $O(n^{4.5})$ for the pseudo-homeomorphism problem. A formal description of the process is given in Algorithm B.

**Lemma VI.2.** The delay-constrained subtree homeomorphism with load factors can be solved in $O(n^{3.5})$ for the homeomorphism problem and in $O(n^{4.5})$ time for the case of pseudo-homeomorphism problem with load factors.

**VII. Performance Evaluation**

For evaluation purposes, we implemented our algorithms ‘Find Delay Constrained Homeomorphism’ (FDCH) and ‘Find Delay Constrained Pseudo-Homeomorphism’ (FDCPH) and compared their execution times for different cases. We ran the algorithms on randomly generated graphs constructed with 20, 40, 60, 80, and 100 nodes. The average node degree for each graph was kept at 3. For each graph, shortest path trees were generated which were used as the host trees. Guest trees were generated from host trees by randomly selecting nodes. The ratio of number of nodes in the guest tree and the number of nodes in the host tree was varied from 1:2 to 1:4. For each guest tree, first we ran FDCH algorithm. If the algorithm returned NO, we ran FDCPH algorithm for the same guest tree. For each run with each algorithm, we computed the execution time. The algorithms were implemented using Library of Efficient Data type and Algorithms (LEDA) and VC++ 8.0. The experiments were conducted on PC (Intel Pentium Mobile 1.4 GHz and 1GB RAM) running Windows XP. The results are plotted in Figure 6 and Figure 7. Each point in the plots represents the average value taken over 30 runs. From Figure 6, we observe that as the number of nodes in the host tree increases, the execution time for FDCH Algorithm increases. We also observe that the execution time increases when the ratio of number of nodes in the guest tree and the number of nodes in the host tree decreases. Figure 7 shows the execution time for FDCPH algorithm for different cases. For this algorithm also, the execution time increases as the number of nodes in the host tree increases. If we compare the results of Figure 6 and Figure 7, we observe that the execution time of FDCPH algorithm is more than that of FDCH algorithm. This is quite expected as we create the augmented host tree from the original host tree at the beginning of FDCPH algorithm and the augmented host tree which contains more nodes than the original host tree is used for embedding in this algorithm.

![Graph showing execution time vs number of nodes in the host tree for different load factors](graph.png)

**Fig. 6.** Execution Time VS Number of Nodes in the Guest Tree: Number of Nodes in the Host Tree

**VIII. Conclusion and Future Research**

In this paper, we considered the problem of embedding an interaction tree from a virtual world application on an overlay network subject to latency constraints. We used the concept of homeomorphic embedding of trees and proposed an algorithm which determines whether a host tree is delay constrained homeomorphic to a guest tree. The complexity of our algorithm is $O(n^{3.5})$, where $n$ is the number of nodes in the overlay. We have also shown that if we relax...
As a future work, it would be interesting to consider the problem of pseudo-homeomorphism problem on a variety of different networks with special topologies.

**Fig. 7. Execution Time VS Number of Nodes in the Host tree for Find Delay Constrained Pseudo-Homeomorphism Algorithm**

the assumption of vertex disjoint paths, we can obtain a polynomial-time algorithm with a time-complexity of $O(n^{1.5})$. We also presented an algorithm to map more than one guest node onto the host node taking into account delay constraints. As a future work, it would be interesting to consider the problem of pseudo-homeomorphism problem on a variety of different networks with special topologies.

**REFERENCES**


