ARTICLE IN PRESS

Computer Networks xxx (2008) xxx–xxx

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

A fast algorithm for computing minimum routing cost spanning trees

Rui Campos *, Manuel Ricardo

INESC Porto, Faculdade de Engenharia, Universidade do Porto, Rua Dr. Roberto Frias, 378, 4200-465 Porto, Portugal

ARTICLE INFO

Article history:
Received 18 November 2007
Received in revised form 28 May 2008
Accepted 17 August 2008
Available online xxxx

Responsible Editor: V.R. Syrotiuk

Keywords:
Minimum Routing Cost Tree
Shortest Path Tree
Minimum Spanning Tree
Bridging
Routing

ABSTRACT

Communication networks have been developed based on two networking approaches: bridging and routing. The convergence to an all-Ethernet paradigm in Personal and Local Area Networks and the increasing heterogeneity found in these networks emphasizes the current and future applicability of bridging. When bridging is used, a single active spanning tree needs to be defined. A Minimum Routing Cost Tree is known to be the optimal spanning tree if the probability of communication between any pair of network nodes is the same. Given that its computation is a NP-hard problem, approximation algorithms have been proposed.

We propose a new approximation Minimum Routing Cost Tree algorithm. Our algorithm has time complexity lower than the fastest known approximation algorithm and provides a spanning tree with the same routing cost in practice. In addition, it represents a better solution than the current spanning tree algorithm used in bridged networks.

1. Introduction

Along with the increasing convergence to an all-IP paradigm, in recent years, we have been witnessing a convergence to what can be called an all-Ethernet paradigm, especially in Personal Area Networks (PANs) and Local Area Networks (LANs). In fact, standards such as IEEE 802.11 [1] and Bluetooth with the specification of the PAN profile [2], and the recent draft specification of the WiMedia Network (WiNet), that builds on the WiMedia UWB (Ultra Wide Band) radio platform [3,4], have been defined to appear in the upper layers of the OSI (Open Systems Interconnection) model as legacy Ethernet links. This paradigm eases the use of bridging in scenarios beyond the traditional Ethernet LANs. For instance, IEEE 802.1D bridges [5] are being used to interconnect IEEE 802.11 networks to backhaul Ethernet infrastructures, to create Bluetooth PANs, and to interconnect Bluetooth PANs to IEEE 802 networks [2]. Additionally, they have been proposed to interconnect UWB WiNet networks to IEEE 802 networks [4] and pointed out by upcoming standards, such as IEEE 802.11s [6], as a solution to interconnect Layer 2 mesh networks to other IEEE 802 networks. The use of bridging enables the creation of a single Layer 2 network from the point of view of the upper layers and hides from them the multiplicity of underlying links that may form a PAN/LAN. New PAN/LAN wired or wireless technologies can be smoothly integrated with existing technologies, without modifications to the protocol stack above the data link layer of the OSI model. Also, IP and its companion protocols, such as Dynamic Host Configuration Protocol (DHCP) [7] and Address Resolution Protocol (ARP) [8], are enabled to run transparently over multiple underlying links.

When bridging is used a single spanning tree needs to be defined as the active network topology. Several spanning trees can be computed from the graph that models the network topology, also known as the topology graph. In the topology graph, the vertices model the network nodes, the edges model the network links connecting the network nodes, and the weights assigned to the edges represent costs computed based on metrics, such as bandwidth and delay. A Minimum Routing Cost Tree (MRCT) is, by definition, the optimal spanning tree from the

* Corresponding author. Tel.: +351222094258; fax: +351222094250.
E-mail addresses: rcampos@inescporto.pt (R. Campos), mricardo@inescporto.pt (M. Ricardo)

1389-1286/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2008.08.013

standpoint of the routing cost and always represents a spanning tree closer to the optimal solution defined by the union of the Shortest Path Trees.

The need for a single active spanning tree has been often presented as one of the major disadvantages of bridging. However, for heterogeneous PANs/LANs, i.e., networks whose corresponding topology graphs have heterogeneous edge weights, this may not represent a significant disadvantage. In Mieghem et al. [9,10] have demonstrated that, as the networks become increasingly heterogeneous, the union of the Shortest Path Trees tends to converge to a single spanning tree, a Minimum Spanning Tree (MST), which also defines an approximate or the exact MRCT in such cases. Therefore, with the increasing heterogeneity found in PANs/LANs – consider, for example, an Ethernet network where 100 Mbit/s and 1 Gbit/s links may coexist in the same LAN, or a future PAN where IEEE 802.11 (54 Mbit/s), Bluetooth (3 Mbit/s), and UWB WiNet (480 Mbit/s) links may coexist, the use of bridging gains a new interest.

Regarding the computation of a spanning tree from a mesh network topology, the IEEE has specified the Rapid Spanning Tree Protocol (RSTP) included in the IEEE 802.1D standard [5]. The devices participating in the protocol elect a root node (called the root bridge) and each device computes the shortest path towards the elected root. The union of these paths represents the final spanning tree, a Shortest Path Tree (SPT) from the root node's perspective. Nonetheless, an arbitrary SPT is selected to define the active network topology, as a consequence of the arbitrary selection of the root node. In practice, the arbitrary SPT does not define a good approximation to an MRCT, in particular for heterogeneous networks, precisely where the use of bridging would be of greater interest.

The computation of an MRCT is known to be a NP-hard problem [11]. That fact has led to the development of approximation algorithms. The fastest known approximation algorithm has been proposed by Wu et. al [12] but it only applies to metric graphs, i.e., graphs whose edges weights obey to the triangle inequality. In practice, edge weights may not obey to this condition since it may frequently happen that the direct path between two adjacent nodes is longer than some indirect path between them. As such, the algorithm is not generically applicable. The fastest known generic approximation algorithm has been proposed by Wong [13]. *Wong's* algorithm specifies the computation of n Shortest Path Trees (SPTs), where n is the number of nodes in the input graph, and selects the tree with the lowest routing cost. Still, the mandatory computation of n SPTs and the calculation of the routing cost for each of them represent its main disadvantages. More recently, Grout [14] has proposed a greedy approximation MRCT algorithm, called *Add* algorithm, that provides good results for homogeneous graphs (i.e., graphs whose edges weights are equal) and lower time complexity than *Wong's* algorithm. Nonetheless, it does not work for non-homogeneous graphs. We herein propose a new approximation MRCT algorithm, called *Campos's* algorithm, that has lower time complexity than *Wong's* algorithm while providing a spanning tree with similar routing cost for both homogeneous and heterogeneous graphs in practice. Furthermore, *Campos's* algorithm provides results better than the current spanning tree algorithm used in bridged networks, namely for heterogeneous networks.

The motivation for the present work is three-fold. Firstly, the relevance that bridging has and is envisioned to have in the future within the PAN/LAN scope. Secondly, the shortcomings found in the spanning tree algorithm currently used by IEEE 802.1D bridges; in general, it does not represent a good approximation MRCT algorithm. Thirdly, the lack of an approximation MRCT algorithm that provides similar results to *Wong's* algorithm (in practice) and has lower time complexity.

The contributions of our work are the following. Firstly, we propose *Campos's* algorithm as a new approximation MRCT algorithm. To the best of our knowledge, it is the first approximation MRCT algorithm with time complexity lower than *Wong's* algorithm while offering similar routing costs in practical cases. Furthermore, it represents a different approach concerning improvements to the current spanning tree algorithm used by IEEE 802.1D bridges; previous proposals have focused more on the provisioning of loop freedom or on the improvement of the performance of the spanning tree protocol itself, and less on the computation of the spanning tree used as active topology. Secondly, we make a comparative analysis of multiple spanning tree algorithms against the current spanning tree algorithm used in IEEE 802.1D bridges by using simulations. Previous studies have focused more on asymptotical analysis or have considered a limited scope; for instance, in [14] Grout has compared his algorithm with MST algorithms but did not compare it with IEEE algorithm and/or *Wong's* algorithm. Thirdly, we conclude about the realistic scenarios where *Add* and the classical MST algorithms can be used as faster alternatives to *Wong's* algorithm for computing approximate MRCTs.

Based on the simulation results we conclude that: (1) *Campos's* algorithm does represent the current fastest approximation MRCT algorithm and does provide results similar to those obtained using *Wong's* algorithm for practical cases; (2) as the number of vertices in the current graph increases, *Campos's* algorithm performs better than *Add* algorithm, when sparse homogeneous graphs are considered; (3) *Campos's* algorithm gives lower routing costs than the spanning tree algorithm currently used by IEEE 802.1D bridges and may be used in current and new bridging oriented solutions to define the active spanning tree; (4) *Add* algorithm is a good approximation MRCT algorithm as far as homogeneous graphs are concerned, but it performs poorly for heterogeneous graphs; (5) MST algorithms tend to approximate the result provided by *Wong's* algorithm as the weights in the input graph become increasingly heterogeneous.

Along the paper we take three major assumptions. Firstly, we limit our analysis to networks with up to 50 nodes. We focus on small scale networks, such as PANs and LANs, as this represents the typical applicability domain for bridging. Secondly, while computing an approximate MRCT our goal is to find the approximate unconstrained MRCT. That is, we do not assume any constraints, such as, minimum node degree in the final spanning tree. Thirdly, we assume a random communication

Please cite this article in press as: R. Campos, M. Ricardo, A fast algorithm for computing minimum routing cost spanning trees, Comput.Netw. (2008), doi:10.1016/j.comnet.2008.08.013

pattern within a bridged network: the probability of communication between each pair of network nodes in the active spanning tree is the same, as it has already been assumed by others [15,16], so that an MRCT can be considered the optimal active spanning tree.

The rest of the paper is organized as follows: In Section 2, we define some concepts and terms in the context of graph theory. In Section 3, we present the related work on approximation Minimum Routing Cost Tree algorithms and the major approaches proposed to improve the performance of bridged networks. In Section 4, we describe the spanning tree algorithms considered in our analysis. Section 5 describes our approximation MRCT algorithm. Section 6 provides the comparison methodology we used to evaluate it against the state of the art spanning tree algorithms. Section 7 describes STS, a new simulator developed to support our analysis. Section 8 presents the set of simulations we have performed. Section 9 shows and analyses the simulation results. Finally, Section 10 mentions the future work and Section 11 draws the conclusions.

2. Graph theoretic definitions

In this section, we present the concepts of graph, tree, and spanning tree, and define some terms used along the paper. In addition, we define three well known spanning trees: Shortest Path Tree (SPT), Minimum Spanning Tree (MST), and Minimum Routing Cost Tree (MRCT).

2.1. Graphs and spanning trees

Some types of graphs are defined in graph theory. Directed and undirected graphs, weighted and unweighted graphs as well as their combinations are defined. Here, we consider weighted, undirected graphs. Fig. 1 illustrates an example of a weighted, undirected graph.

Circles represent the vertices of the graph, lines represent the edges of the graph, and the numbers next to the lines specify the weights assigned to the corresponding edge. Formally, a weighted, undirected graph $G = (V, E, w)$ is defined as a set of vertices V connected by a set of edges $E \subseteq VxV$ (i.e., the elements of E are 2-element subsets of V) with weights w that represent costs assigned to the edges of G; $w(i,j)$ defines the weight assigned to the edge (i,j) and takes an integer value. The number of vertices in G is $n = |V|$ and the number of edges is $m = |E|$. The graph in Fig. 1 has $n = 6$ and $m = 7$. The set of vertices adjacent to a vertex v is denoted by A_v. The degree of a vertex v is defined as its number of adjacent edges, where an adjacent edge to v is an edge connecting it to a neighbor vertex. In Fig. 1 vertex 2, for example, has degree 3. A graph where all vertices have degree $n-1$ is called a complete graph. A path in G is defined as a sequence of connected adjacent vertices; the path starts at the first vertex of the sequence and ends at the last vertex of the sequence, vertices f and l, respectively. The cost of the path between f and l, $c_G(f,l)$, is equal to the sum of the weights of the edges belonging to the path. In Fig. 1, the sequence {1,2,5,3,6} represents a path between vertices 1 and 6 and $c_G(1,6) = 10 + 1 + 10 + 5 = 26$. A cycle in G is defined as a path where the first and the last vertex in the sequence are the same, i.e., $f = l$. In Fig. 1, the sequence {1,2,5,3,1} defines a cycle. A graph G in which edge weights satisfy the triangle inequality is called a metric graph. The triangle inequality is defined by the following expression:

$$c_G(i,j) \leqslant c_G(i,k) + c_G(k,j), \tag{1}$$

which states that the direct path between two vertices i and j in G has always lower or equal cost than any other path in G between the two vertices.

In graph theory, a tree T is defined as a connected graph without cycles. Any two vertices in T are connected by exactly one path and the number of edges in T is $n-1$. In Fig. 1, if the edges (1,2) and (2,4) are removed we get a tree. A spanning tree for a graph $G = (V, E, w)$ is defined as a subgraph of G that is a tree, contains all vertices of G, and has $|V| - 1$ edges. Some cost functions can be defined for a tree T. The cost between a source vertex s and a vertex v, with $s, v \in T$, is given by

$$c_T(s,v), \tag{2}$$

which expresses the cost of the path between s and v in T. The total cost for T is given by

$$C_{\text{total}}(T) = \sum_{e=1}^{n-1} w_T(e), \tag{3}$$

where $w_T(e)$ represents the weight assigned to edge $e \in E_T$ and E_T defines the set of edges of T. The routing cost for T is

$$C_r(T) = \sum_{i=1}^{n} \sum_{j=1}^{n} c_T(i,j), \quad i \neq j. \tag{4}$$

Based on the minimization of the cost functions defined in (2)–(4), three spanning trees have been defined for an input graph G.

A *Shortest Path Tree (SPT)* rooted at a vertex s, well known in the scope of network routing protocols, defines a tree composed by the union of the paths between s and each of the other vertices in G such that

$$c_{\text{SPT}}(s,v) = \min(c_G(s,v)). \tag{5}$$

A *Minimum Spanning Tree (MST)* represents a spanning tree T^* such that

$$C_{\text{total}}(T^*) = \min\left(\sum_{e=1}^{n-1} w_T(e)\right), \tag{6}$$

for all spanning trees T that can be computed from G.

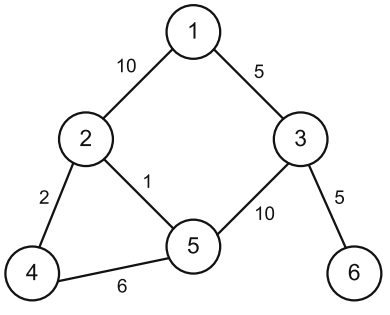

Fig. 1. Example weighted, undirected graph.

Please cite this article in press as: R. Campos, M. Ricardo, A fast algorithm for computing minimum routing cost spanning trees, Comput.Netw. (2008), doi:10.1016/j.comnet.2008.08.013

A *Minimum Routing Cost Tree (MRCT)* represents a spanning tree T^* such that

$$C_r(T^*) = \min\left(\sum_{i=1}^{n}\sum_{j=1}^{n} c_T(i,j), i \neq j\right), \quad (7)$$

for all spanning trees T that can be computed from G.

SPT, MST, and MRCT are not necessarily unique. For instance, a unique MST is only guaranteed when the weights of G are all different; otherwise, there can be multiple spanning trees that match the minimum total cost criterion. The algorithms used to compute these spanning trees are described in Section 4.

3. Related work

Below, we first present related work on approximation MRCT algorithms and then refer some approaches that have been proposed in order to improve the performance of bridged Ethernet networks.

In [13] Wong has proposed a 2-approximation algorithm, which means that the resulting spanning tree has at most 2 times the routing cost of the actual MRCT. *Wong's* algorithm defines the computation of n SPTs using a shortest path algorithm, such as *Dijkstra's* algorithm (see Section 4.4), and the calculation of the routing cost for each of them. The tree with lowest routing cost is selected as the 2-approximate MRCT. The time complexity $O(n^2 \log(n) + m \cdot n)$ is the major drawback of *Wong's* algorithm; n SPTs and the routing cost for each of them needs to be computed in order to find the best SPT. Better generic approximation MRCT algorithms, 15/8-approximation and 3/2-approximation running in time $O(n^3)$ and $O(n^4)$, respectively, are proposed in the literature [11]; yet, better approximations are achieved at a cost of increased time complexities.

Wu et al. [12] have demonstrated that to find an MRCT in a general weighted graph is equivalent to solving the same problem in a complete graph in which edge weights satisfy the triangle inequality, i.e., a complete metric graph. For metric graphs they have proposed the Polynomial Time Approximation Scheme (PTAS) for finding an $(1+\varepsilon)$-approximate MRCT in time $O(n^{2\lceil 2/\varepsilon \rceil - 2})$. Using this scheme a 2-approximate MRCT can be computed in time $O(n^2)$. However, concerning general graphs, firstly the input graph has to be converted into a metric graph, then the approximate MRCT is computed using the PTAS, and finally the approximate MRCT found for the metric graph can be transformed into a spanning tree of the original graph. This leads to an overall time complexity $O(n^3)$ [11].

In [14] Grout proposes a greedy algorithm – *Add* algorithm – that does not guarantee any asymptotical approximation but, in practice, provides good results for homogeneous graphs and has lower time complexity (see Section 4.5) than the previous algorithms. *Add* algorithm assumes that the way to approximate an MRCT is by minimizing the number of relay vertices – vertices with degree greater than one – in the final spanning tree. Thus, its objective is to compute a spanning tree with the minimal set of relay vertices. Despite performing well for homogeneous networks, *Add* algorithm does not work for non-homogeneous networks, as we will show later on. Consequently, it does represent a good approximation MRCT algorithm but only addresses homogeneous networks.

In [9,10] Mieghem et al. analyze the influence of the edge weights structure in the union of all SPTs (U_{SPTs}) used by routing approaches to define the active network topology. They show that U_{SPTs} exhibits different behaviors depending on the level of heterogeneity of the edge weights in the input graph. For increasingly heterogeneous input graphs they demonstrate that U_{SPTs} converges to or coincides with a single spanning tree, an MST; in other words, a MST approximates or coincides with U_{SPTs}. This means that under such conditions an MST algorithm can be used as an approximation MRCT algorithm or even as the MRCT algorithm.

The proposals for improving the performance of bridged Ethernet networks have mostly concentrated on overcoming the need to use a single spanning tree as the active network topology; less work has been performed on the improvement of the routing cost of the active spanning tree.

In [17] Lui et al. propose a new bridging device, called STAR bridge, that can coexist with current IEEE 802.1D bridges in the same network. STAR bridges take advantage of the possible existence of alternate paths shorter than the paths defined by the active spanning tree computed using IEEE algorithm. If such path exists, STAR bridges forward frames over it, instead of using the spanning tree path defined by IEEE spanning tree algorithm; through simulation the authors demonstrate the effectiveness of this approach. STAR bridges introduce a more complex operation model than IEEE 802.1D bridges, since they need to compute the possible alternate paths that may be used to forward frames, and need to explicitly exchange information about attached end-hosts, so that frames can be forwarded towards the right end-hosts over the alternate paths.

In [18] Rodeheffer et al. propose the so-called SmartBridge aiming at improving the scalability of Ethernet networks. SmartBridges overcome the single spanning tree limitation by always using a shortest path to forward frames, as it happens in routing. In addition, they introduce improvements in the convergence time when network topology changes occur; the authors claim that SmartBridges are three orders of magnitude faster than IEEE 802.1D bridges using the IEEE spanning tree protocol. Similarly to STAR bridges, SmartBridges introduce a more complex operation model and also need to exchange information about attached end-hosts. Moreover, they are not backwards compatible with IEEE 802.1D bridges; a similar backwards compatible solution has been proposed by Perlman [19].

In [20] Pellegrini et al. propose a new algorithm, called Tree-Based Turn-Prohibition (TBTP), targeting Gigabit Ethernet networks and aiming at tackling the non-efficient utilization of the underlying Gigabit links if using IEEE 802.1D bridges. The TBTP algorithm computes an active topology that besides the spanning tree links contains alternate links that enable shorter forwarding paths, as it happens in the STAR solution; for compatibility reasons the spanning tree is computed using IEEE spanning algorithm. As it happens for the previously described pro-

Please cite this article in press as: R. Campos, M. Ricardo, A fast algorithm for computing minimum routing cost spanning trees, Comput.Netw. (2008), doi:10.1016/j.comnet.2008.08.013

posals, bridges using the TBTP algorithm introduce an operation model more complex than the IEEE 802.1D operation model; for instance, the computation of the active topology has time complexity $O(m^2)$.

Huang and Cheng [21] proposed a new spanning tree algorithm that, for non-homogeneous networks, provides a spanning tree more balanced than the spanning tree achieved by using IEEE algorithm. Informally, a balanced spanning tree is defined as a tree where no leaf is much farther away from the root than any other leaf; an example of a balanced tree is the complete binary tree. By defining a more balanced spanning tree the authors claim that its corresponding routing cost will be lower. However, this conclusion does not always hold. Indeed, it is only true for homogeneous trees, where link weights are all equal and the topology of the tree is the only factor influencing its routing cost (see Section 5); in this case, a more balanced tree contributes to the reduction of the routing cost, since the average path between each pair of nodes tends to be shorter. Regarding heterogeneous networks the link weights also influence the routing cost of the tree and a more balanced tree does not necessarily have lower routing cost; it depends on its link weights. Thereby, in general, the authors' algorithm does not provide a spanning tree with lower routing cost than IEEE algorithm.

Recently, Elmeleegy et al. [22] proposed a new device, called EtherFuse, that can be inserted into existing bridged Ethernet networks to reduce the reconfiguration time and prevent congestion due to packet duplication during temporary formation of forwarding loops. EtherFuse is compatible with the current spanning tree protocol; it works as a watchdog by monitoring, for instance, the existence of temporary loops and the sudden leave of the root bridge. By means of an EtherFuse prototype the authors have shown the actual effectiveness of the device to tackle the current problems found in the RSTP protocol and the performance improvements achieved from common applications' perspective, such as web browsing and file transfer. However, EtherFuse only addresses the problems identified in the RSTP protocol; an Ethernet network including EtherFuse devices will still have an active spanning tree defined by current IEEE algorithm.

4. Spanning tree algorithms

This section describes the spanning tree algorithms considered in the comparative analysis we carried out to evaluate *Campos's* algorithm. We describe the current spanning tree algorithm used by IEEE 802.1D bridges, the two most relevant MST algorithms (*Kruskal's* and *Prim's* algorithms), *Dijkstra's* algorithm, on which *Wong's* algorithm is based, and *Add* algorithm.

4.1. IEEE 802.1D Spanning tree algorithm

The Rapid Spanning Tree Protocol (RSTP) protocol currently defined in IEEE 802.1D standard [5] defines an SPT as the active spanning tree within a network. The algorithm used to compute an SPT consists of two major steps: (1) the devices participating in the protocol elect a root node, named root bridge; (2) each device computes a shortest path towards the root node; the union of these paths represents the final spanning tree and defines a Shortest Path Tree from the root node's perspective. The root node is arbitrarily selected, what leads to an arbitrary SPT. The time complexity associated to the computation of a Shortest Path Tree is defined in Section 4.4.

4.2. Prim's algorithm

Prim's algorithm [23] by Prim is, in conjunction with *Kruskal's* algorithm, one of the classical greedy algorithms used to compute an MST for an input graph G. Initially *Prim's* algorithm selects an arbitrary vertex u (initial vertex) as the current candidate to make part of the spanning tree T. u is then added to the list of vertices L that includes the vertices candidates to make part of the spanning tree T at each step of the algorithm; at the beginning, u is the unique element in L. A generic vertex v in L is characterized by the weight w_v of the edge connecting v to its candidate parent vertex in T, p_v; the initial vertex has $w_u = 0$ and its candidate parent vertex p_u is undefined, since it is the root of the spanning tree T. At each step the algorithm works as follows. The vertex v with the lowest weight is removed from L and defined as the current spanned vertex. Each vertex a adjacent to v in G that does not belong to T yet is added to L; v represents the candidate parent vertex in T for each vertex a added to L. If a already belongs to L but the weight $w(v,a) < w_a$, w_a and the candidate parent vertex p_a are updated accordingly, i.e., $w_a = w(v,a)$ and $p_a = v$. This process is repeated until the n vertices have been added to T. At that stage T defines an MST, whose edges are defined by the union of the edges (v, p_v). *Prim's* algorithm runs in $O(n \cdot m)$ in its more naive implementation and in $O(n^2)$ time when ensuring that for each iteration at most n edges are searched. Yet, if binary heaps in conjunction with adjacency lists or Fibonacci heaps [24] in conjunction with adjacency lists are used, the time complexity can be improved to $O(m \log(n))$ and $O(m + n \cdot \log(n))$, respectively.

4.3. Kruskal's algorithm

Kruskal's algorithm [25] by Kruskal is another popular algorithm targeting the same problem. *Kruskal's* algorithm starts with a forest which consists of n trees, where each vertex is initially a separate tree. Afterwards, the edges in the graph are sorted by weight in non-decreasing order. At each step of the algorithm, the edge (u,v) with the lowest weight is selected. If vertices u and v belong to two different trees, the two trees are combined into a single tree. If the selected edge connects vertices which belong to the same tree the edge is rejected and not examined again, as it would produce a cycle. While running the algorithm, less and bigger trees are achieved until a single tree is found, an MST; the algorithm finishes when either all nodes have been spanned or all edges have been processed. Sorting the edges in non-decreasing order takes $O(m \cdot \log(m))$ time. This defines the asymptotic running time of the algorithm. In practice, *Kruskal's* algorithm has been implemented using priority queues, whose elements are the edges of the graph.

Please cite this article in press as: R. Campos, M. Ricardo, A fast algorithm for computing minimum routing cost spanning trees, Comput.Netw. (2008), doi:10.1016/j.comnet.2008.08.013

4.4. Dijkstra's algorithm

Dijkstra's algorithm [26] by Dijkstra is the classical shortest path algorithm used, for instance, in network routing protocols. *Dijkstra's* algorithm grows from a source vertex and extends outward within the graph, until all vertices have been spanned and an SPT has been found. For each vertex v the algorithm maintains two attributes: δ_v, the shortest path estimate from the source to v, and p_v, which stores the parent vertex of v in the SPT. At the beginning, the shortest path estimates for all vertices other than the source are set to ∞; the estimate for the source is set to 0. The algorithm also maintains a set S of vertices whose final shortest path costs from the source have not yet been determined. Initially, this set contains all vertices of G. Starting with the source vertex, the algorithm repeatedly selects the vertex $u \in S$ with the minimum shortest path estimate δ_u. Furthermore, it re-evaluates the shortest path estimates of the vertices adjacent to u, and updates them if $\delta_a > \delta_u + w(u,a)$, where $w(u,a)$ defines the weight of edge (u,a) and a is adjacent to u. Once a vertex is removed from S, its shortest path cost from the source has been found. When the algorithm finishes, δ_v will be the cost of the shortest path from the source to v. The union of the edges (v,p_v) defines the final SPT. A naive implementation that examines all nodes in S to find the minimum runs in $O(n^2)$. If, as in *Prim's* algorithm, binary heaps or Fibonacci heaps are used, $O(m \cdot \log(n))$ and $O(m + n \cdot \log(n))$ running times can be achieved, respectively.

4.5. Add algorithm

Add algorithm [14] by Grout represents a vertex oriented version of *Prim's* algorithm. It considers that the way to approximate an MRCT for G is by minimizing the number of relay nodes in the final spanning tree. *Add* algorithm ignores the edge weights and considers as the decision criterion the degree of the vertices in G. A *spanning relay* (a vertex with more than one adjacent edge) is chosen initially as the vertex of highest degree to start the construction of the spanning tree T. Next, all its adjacent edges as well as the adjacent vertices are selected to make part of T. From the vertices currently spanned, a new *spanning relay* is selected, adjacent to the maximum number of unspanned vertices. The process is repeated until all vertices belong to T. If an adjacent edge of the current *spanning relay* connects two vertices already in T, the edge is rejected. At the end, T represents the approximate MRCT. *Add* is not an exact greedy algorithm as it does not provide the optimal number of relays, in general [14]; nonetheless, it performs well in practice. It runs in $O(n \cdot \log(n))$ time in the worst case. Usually, it runs faster due to the tendency for many nodes to be added to T at each stage.

5. Campos's algorithm

Our proposal, *Campos's* algorithm, represents a new approximation MRCT algorithm that improves the performance of bridged Layer 2 networks, particularly when the edges of the graph modeling the network have heterogeneous weights. It aims at improving the routing cost of the active spanning tree assumed in these networks, taking into account that an MRCT is by definition the optimal spanning tree.

In [11] the routing cost of a spanning tree is shown to depend both on the edge weights and on the tree topology. These two factors have impact in the diameter of the tree – the cost of the longest path between any two vertices in the tree – which influences the routing cost of the tree; in general, a spanning tree with lower diameter has lower routing cost. Consider, for example, the trees shown in Fig. 2. They have the same set of edge weights but different topologies. The left-hand side tree has routing cost 104 and diameter 7, while the right-hand side tree has routing cost 89 and diameter 5. In this case, the right-hand side tree has lower routing cost due to its lower diameter. Thus, the computation of an MRCT for a graph G must consider both *edge weights* and *topology*. Nonetheless, when the graph G is homogeneous or extremely heterogeneous a single factor actually influences the selection of an MRCT, as demonstrated by Grout [14] and by the analysis and conclusions drawn by Mieghem et al. [9,10]. *Add* algorithm provides an approximate MRCT when G is homogeneous by considering the degree of the vertices as the selection criterion. On the other hand, an MST approximates or coincides with an MRCT when links are extremely heterogeneous and is selected based on the edge weights only. Between these two extreme cases both factors influence the selection of an MRCT. Therefore, a combination of the criterion used by MST algorithms (edge weights) and the criterion considered by *Add* algorithm (degree of the vertices) in a single algorithm appears as a promising approach to approximate

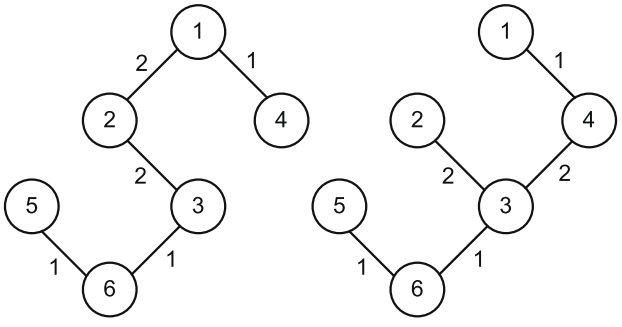

Fig. 2. Two example trees with the same set of edge weights but different topologies.

Please cite this article in press as: R. Campos, M. Ricardo, A fast algorithm for computing minimum routing cost spanning trees, Comput.Netw. (2008), doi:10.1016/j.comnet.2008.08.013

an MRCT. That is what *Campos's* algorithm does in order to consider both topology and edge weights in the general case; in fact, it takes into account an additional criterion in order to contribute to the reduction of the diameter of the final spanning tree (see Section 5.1).

5.1. Description of the algorithm

Campos's algorithm takes *Prim's* algorithm as basis but introduces two major modifications. Firstly, it defines a deterministic approach to find the initial vertex. Secondly, it considers parameters other than the edge weight to select the vertex to be extracted from the list of vertices L at each step of the algorithm. The new parameters are partially imported from other spanning tree algorithms, namely from *Add* and *Dijkstra's* algorithms. The algorithm defines basic parameters and composed parameters. The latter are obtained by combining the former.

As in *Prim's* algorithm, *Campos's* algorithm starts by selecting the initial vertex. Three basic parameters characterizing a vertex $v \in V$ are considered for that purpose:

- d_v – degree of the vertex
- s_v – sum of adjacent edge weights
- m_v – maximum adjacent edge weight

The combination of the basic parameters defines the composed parameter sp_v that characterizes the potential of v to span neighbor vertices; we call it the spanning potential. sp_v includes the three components shown in (8): the degree of the vertex, the inverse of the average adjacent edge weight, and the inverse of the maximum adjacent edge weight. These components are considered in inverse proportion since higher weight values imply lower spanning potential. C_1, C_2, and C_3 are coefficients whose values have been found by simulation: $C_1 = C_3 = 0.2$ and $C_2 = 0.6$. These were the values for which we could obtain the best routing cost results. Still, it should be emphasized that the latter are not strongly dependent on the actual values selected for C_1, C_2, and C_3; for instance, for $C_1 = C_3 = 0.25$ and $C_2 = 0.55$ the obtained results were not significantly different.

$$sp_v = C_1 \cdot d_v + C_2 \cdot \frac{d_v}{s_v} + C_3 \cdot \frac{1}{m_v}, \tag{8}$$

where

$$s_v = \sum_{j=1}^{n} w(v,j), \quad v \neq j,$$
$$m_v = w(v,l) : w(v,l) \geqslant w(v,k) \wedge k, \quad l, v \in V,$$

The vertex $f \in V : sp_f \geqslant sp_j, \forall j \in V$ is defined as the initial vertex. In other words, the vertex with the highest spanning potential is selected as the initial vertex, as in *Add* algorithm. This selection criterion represents a modified version of the criterion used by *Add* algorithm for selecting the initial spanning relay. Besides considering the degree of the vertices, our algorithm considers the average adjacent edge weight and the maximum adjacent edge weight. The two additional parameters are relevant for non-homogeneous graphs; for homogeneous graphs only the degree of the vertices influences the choice of the initial vertex since the other two parameters are constant for all vertices.

In order to select the vertex to be extracted from the list of vertices L at each step of the algorithm, *Campos's* algorithm defines the following set of basic parameters to characterize a vertex $v \in V$:

- w_v – weight (considered as in *Prim's* algorithm)
- d_v – degree of the vertex
- s_v – sum of the adjacent edge weights
- cf_v – estimate cost of the path between v and f in T
- pd_v – degree of the candidate parent vertex in T
- ps_v – sum of the adjacent edge weights of the candidate parent vertex in T

By combining them the algorithm defines the following composed parameters:

$$wd_v = C_4.w_v + C_5 \cdot cf_v, \tag{9}$$

$$jsp_v = sd_v + \frac{sd_v}{sw_v}, \tag{10}$$

where

$$sd_v = d_v + pd_v,$$
$$sw_v = s_v + ps_v.$$

The composed parameter wd_v (9) characterizes the weight of the edge connecting v and its candidate parent vertex p_v in T and the estimated cost of the path between v and f in T implicitly dictated by the candidate parent vertex p_v. The composed parameter jsp_v (10) characterizes the joint spanning potential of v and p_v.

In (9) the coefficients C_4 and C_5 take values which depend on the set of edge weights of the input graph G. After computing the mean (μ) and standard deviation (σ) for the set of edge weights of G, the algorithm evaluates the ratio σ/μ that is used to characterize the heterogeneity of the input graph. If it is below a threshold *Thr*, $C_4 = C_5 = 1$; otherwise, $C_4 = 0.9$ and $C_5 = 0.1$. These values have been obtained by simulation. Also based on simulations we found out that as the number of vertices in the input graph increases *Thr* should increase proportionally. Therefore, *Thr* has been defined as a function of $n = \mid V \mid$; the values 0.4 and 0.005 in (11) have been found by simulation. Again, it should be emphasized that the obtained results are not strongly dependent on the actual values selected for the various coefficients; for instance, if $C_4 = 0.85$ and $C_5 = 0.15$ the obtained results did not differ significantly from those presented herein. Nevertheless, the best results were obtained using the coefficients mentioned above.

$$Thr = 0.4 + 0.005 \times (n - 10) \tag{11}$$

The need to give more importance to w_v ($C_4 = 0.9$) when the input graph has highly heterogeneous edge weights comes from the fact that the major factor influencing the selection of an MRCT in those cases is the set of edge weights.

The algorithm proceeds as follows: Initially all vertices have w_v and cf_v set to ∞, except f that has w_f and cf_f set to 0; the other basic parameters listed above have undefined values at this stage. At each step of the algorithm, the vertex $u \in L : wd_u < wd_j, \forall j \in L$ is selected as the next

Please cite this article in press as: R. Campos, M. Ricardo, A fast algorithm for computing minimum routing cost spanning trees, Comput.Netw. (2008), doi:10.1016/j.comnet.2008.08.013

vertex to be extracted from L that initially only contains f. If there is more than one vertex in L with the same value of wd_v, jsp_v is considered to break the tie. Let $S \subseteq L$ define the set of vertices in L having the same lowest value of wd_v. The vertex $u \in S : jsp_u \geqslant jsp_j, \forall j \in S$ is then the vertex extracted from L in that case. Afterwards, for every vertex $a \in A_u \wedge a \notin T$ (A_u defines the set of vertices adjacent to u), if wd_a becomes lower while computed considering u as the candidate parent vertex, wd_a and jsp_a are updated according to the new values of the corresponding basic parameters, $w_a = w(u,a)$, $cf_a = cf_u + w(u,a)$, $pd_a = d_u$, and $ps_a = s_u$. If wd_a achieves the same value but jsp_a achieves a greater value if computed considering u as the candidate parent, only jsp_a is updated according to the new values of the pd_a and ps_a. In both cases u becomes the candidate parent vertex for a, i.e., $p_a = u$. a is then added to L if it does

Algorithm: *Campos's* algorithm
Input: A weighted, undirected graph G=(V,E,w).
Output: An approximate Minimum Routing Cost Tree.

```
// The following set of variables is assumed to be initialized to 0: d_v, s_v, m_v, and sp_max.
// The variables wd_t and jsp_t are used to store temporary values.
for each (i,j) ∈ E do
    d_i ← d_i+1
    s_i ← s_i+w(i,j)
    m_i ← max(m_i,w(i,j))
    d_j ← d_j+1
    s_j ← s_j+w(i,j)
    m_j ← max(m_j,w(i,j))
for each v ∈ V do
    w_v ← ∞
    cf_v ← ∞
    sp_v ← C1.d_v+C2.d_v/s_v+C3/m_v
    if sp_v > sp_max
        f ← v
        sp_max ← sp_v
w_f ← 0
cf_f ← 0
p_f ← NIL
L ← L ∪ {f}
while L ≠ ∅
    wd_min ← ∞
    jsp_max ← 0
    for each v ∈ L do
        if wd_v < wd_min
            S ← ∅ ∪ {v}
            wd_min ← wd_v
        else if wd_v = wd_min
            S ← S ∪ {v}
    for each v ∈ S
        if jsp_v ≥ jsp_max
            jsp_max ← jsp_v
            u ← v
    for each a ∈ A_u do
        if a ∉ T
            wd_t ← C4.w(u,a)+C5.(cf_u+w(u,a))
            jsp_t ← (d_a+d_u)+(d_a+d_u)/(s_a+s_u)
            if wd_t < wd_a
                wd_a ← wd_t
                jsp_a ← jsp_t
                p_a ← u
            else if (wd_t = wd_a ∧ jsp_t ≥ jsp_a)
                jsp_a ← jsp_t
                p_a ← u
            if a ∉ L
                L ← L ∪ {a}
    T ← T ∪ {(u,p_u)}
```

not yet belong to L and the edge (u, p_u) is added to T. This process is repeated until the n vertices have been added to T. At that stage T is the approximate MRCT.

The set of parameters listed above enables the algorithm to take both topology and edge weights into account during the computation of the approximate MRCT. The w_v parameter, already considered in *Prim's* algorithm, relates to the edge weights. By considering the $n-1$ lowest edge weights the algorithm tries to reduce the costs of the paths between the vertices in the final spanning tree and, consequently, reduce the routing cost of the final spanning tree. The cf_v parameter is imported from *Dijkstra's* algorithm and considers both topology and edge weights. By minimizing cf_v for each vertex the algorithm tries to reduce the diameter of the final spanning tree and, consequently, its routing cost; in fact, if we define the initial vertex as the root of the final spanning tree, a reduction in the cost of the path towards the root contributes to the reduction of the diameter of the tree. The parameters sd_v $(d_v + pd_v)$ and sw_v $(s_v + ps_v)$ are related to topology. They allow the algorithm to characterize the joint spanning potential of a vertex v and its candidate parent vertex pd_v. By selecting first

```
CAMPOS_ALGORITHM(in G, out T) // receives G as input and returns the spanning tree T as output

    // computes degree and sum of weights for each vertex -> O(m)
    for each (i,j) in E do {
         // the elements of d, s, and m are assumed to be initialized to 0
         d[i]=d[i]+1; s[i]=s[i]+w(i,j); m[i]=max(m_i,w(i,j));
         d[j]=d[j]+1; s[j]=s[j]+w(i,j); m[j]=max(m[j],w(i,j));
         sumWeights=sumWeights+w(i,j); // sumWeights is assumed to be initialized to 0
    }

    // compute mean and standard deviation for the set of edge weights of G
    mean=sumWeights/|E|; // divides overall sum of edge weights by number of edges
    for each w(i,j) do // -> O(m)
         sum=sum+(w(i,j) - mean)^2; // sum is assumed to be initialized to 0
    stdDev=SQRT(sum/(|E|-1)); // SQRT stands for square root
    ratio=stdDev/mean;

    // selects the vertex with higher spanning potential as the initial vertex -> O(n)
    for each v in V {
         w[v]=inf; // inf stands for a huge integer
         color[v]=WHITE;
         sp[v]=0.2*d[v]+0.6*(d[v]/s[v])+0.2*(1/m[v]);
         if(sp[v] > sp_max) // sp_max is assumed to be initialized to 0
                  sp_max=sp[v]; f=v;
    }

    w[f]=0; cf[f]=0; p[f]=f; pd[f]=0; ps[f]=1;
    color[f]=GRAY; // GRAY color means that f has been added to L
    L=INSERT(f); // insert vertex f into list L -> O(1) (if using Fibonacci Heaps [24])
    spanned_vertices = 0;

    // partial time complexity -> O(n.log(n)+m)
    while (spanned_vertices < |V|) {

         u=EXTRACT_MIN(L); // -> O(log(n)) amortized (if using Fibonacci Heaps [24])

         // Adj[u] represents an adjacency list containing the adjacent vertices to u
         for each v in Adj[u] do { // -> O(degree(u))
                  if(color[v]==BLACK) // v already belongs to T
                           continue;

                  // t has the same characteristics of v, but its candidate parent is u
                  wd[t]=C_4*w(u,v)+C_5*(cf[u]+w(u,v)); jsp[t]=(d[v]+d[u])+(d[v]+d[u])/(s[a]+s[u]);

                  // cmp() is the compare function used to sort elements in L; cmp() receives 2
                  // vertices as argument and compares them based on wd[] and jsp[] for each vertex
                  if(cmp(v,t) < 0) { // check whether u represents better candidate parent vertex for v
                           if(color[v] == WHITE) // v was not processed yet
                                    wd[v]=wd[t]; jsp[v]=jsp[t]; p[v]=u;
                                    L = INSERT(v); // -> O(1) amortized (if using Fibonacci Heaps [24])
                                    color[v] = GRAY;
                           else if(color[v] == GRAY)
                                    // update wd[v] and jsp[v] in L according to new candidate parent
                                    L = UPDATE(v); // -> O(1) amortized (if using Fibonacci Heaps [24])

                   }
         }

         color[u]=BLACK; // u has been added to T
         spanned_vertices=spanned_vertices+1;

    }

    T[spanned_vertices]=(u, p[u]); // add edge (u, p[u]) to T, which is defined by a set of edges
```

Please cite this article in press as: R. Campos, M. Ricardo, A fast algorithm for computing minimum routing cost spanning trees, Comput.Netw. (2008), doi:10.1016/j.comnet.2008.08.013

the vertices that together with their candidate parent vertices have the highest joint spanning potential, the algorithm reduces the number of relay vertices in the final spanning tree; this contributes to the reduction of its routing cost, as demonstrated by *Add* algorithm.

Campos's algorithm is formally defined above. Since the definition is independent of the actual implementation, we also define it using pseudo code in order to both show how we have implemented *Campos's* algorithm in the STS simulator (see Section 7) and analyze its time complexity.

Taking into account the partial time complexities provided in the pseudo code above, i.e., $O(m)$, $O(m)$, $O(n)$, and $O(n \cdot \log(n) + m)$, we conclude that *Campos's* algorithm has time complexity $O(m + n \cdot \log(n))$, when implemented using Fibonacci heaps in conjunction with adjacency lists. In spite of the computation of additional parameters characterizing the vertices ($O(m)$ time), the computation of the mean and standard deviation for the set of edge weights of G ($O(m)$ time), and the deterministic selection of the initial vertex ($O(n)$ time), the overall time complexity of the algorithm is of the same order of *Prim's* algorithm.

5.2. Illustrative example

In order to illustrate the use of *Campos's* algorithm in the computation of an approximate MRCT, let us consider the input graph shown in Fig. 3. We start by calculating the parameters d_v, s_v, and m_v, and the spanning potential for each vertex (sp_v). The values are presented in Table 1.

According to the algorithm, the vertex with the highest sp_v is selected as the initial vertex f; in this case vertex 1 is selected. Subsequently, f is added to the list of vertices L. Here, L is modeled by a table, where each column represents a vertex in the list, the shadowed column emphasizes the vertex selected at each step of the algorithm to make part of T, and the cells with bold borders highlight the composed parameter and the value that made it happen. The tables (a)–(h) below represent different instances of L. Initially, L only contains vertex 1. The values of the parameters characterizing vertex 1 are shown in table (a); some values are undefined since the initial vertex does not have parent vertex. In this example, the coefficients C_4 and C_5 used in the computation of wd_v are equal, as the mean for the set of edge weights is 2.00 and the standard deviation is 0.74 ($0.74/2.00 \approx 0.37 < Thr$). The algorithm proceeds by removing vertex 1 from L. Then, all adjacent vertices to 1 (vertices 2 and 4) are added to L; this is shown in table b). Vertex 4 is then selected to make part of T, since it is the vertex with the highest value of jsp_v and wd_v is equal for all vertices in L; the vertices adjacent to vertex 4 (vertex 3 and 5) are added to L. Vertex 2 is now selected to make part of T, as it is the vertex with the lowest value of wd_v. Vertex 6 is added to L; since the composed parameter wd_3 gets higher value if computed using vertex 2 as candidate parent vertex ($7 > 5$), the candidate parent for vertex 3 is unchanged. Vertex 3 is added to T, given that it is the vertex with the highest value of jsp_v and wd_v is equal for all vertices in L. Vertex 7 is added to L. Regarding vertex 5 nothing changes, as the composed parameter wd_5 does not get lower value if vertex 3 is considered as its candidate parent vertex; the other vertices adjacent to 3 already belong to T. Vertex 5 is selected next, given that it is the vertex with the highest value of jsp_v among the vertices in L with the lowest value of wd_v (vertices 5 and 6). Vertex 8 is added to L; the other adjacent vertices (3) and (4) already belong to T. Subsequently, as the vertex with the lowest value of wd_v, vertex 6 is added to T. Since wd_7 does not get lower value if vertex 6 is considered as its candidate parent vertex, its parameters remain unchanged. At this stage, vertex 7 is selected to make part of T. Since the composed parameter jsp_8 gets higher value if vertex 7 is considered as its candidate parent vertex (the value of wd_8 remains the same), the parameters w_8, p_8, pd_8, ps_8, and cf_8 are updated accordingly; this is illustrated in table h), where the values of the updated parameters are shown in bold. Vertex 8 is finally selected to make part of T. At this point, T defines the approximate MRCT shown in Fig. 3.

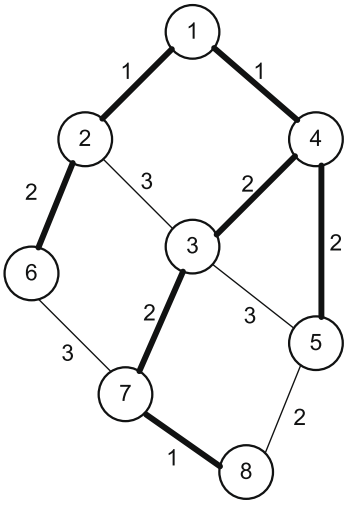

Fig. 3. Example graph and the corresponding approximate MRCT computed using *Campos's* algorithm (in bold).

6. Comparison methodology

Regarding the evaluation of *Campos's* algorithm against state of the art spanning tree algorithms targeting the

Table 1
Values of the parameters d_v, s_v, and m_v and the corresponding spanning potential (sp_v) for each vertex v

v	d_v	s_v	m_v	sp_v
1	2	2	1	1.20
2	3	6	3	0.97
3	4	10	3	1.11
4	3	5	2	1.06
5	3	7	3	0.92
6	2	5	3	0.71
7	3	6	3	0.97
8	2	3	2	0.90

Please cite this article in press as: R. Campos, M. Ricardo, A fast algorithm for computing minimum routing cost spanning trees, Comput.Netw. (2008), doi:10.1016/j.comnet.2008.08.013

a

v	1
w_v	0
d_v	2
s_v	2
p_v	-
pd_v	-
ps_v	-
cf_v	0
wd_v	0
jsp_v	-

b

v	2	4
w_v	1	1
d_v	3	2
s_v	6	5
p_v	1	1
pd_v	2	2
ps_v	2	2
cf_v	1	1
wd_v	2	2
jsp_v	5.6	5.7

c

v	2	3	5
w_v	1	2	2
d_v	3	4	3
s_v	6	10	7
p_v	1	4	4
pd_v	2	3	3
ps_v	2	5	5
cf_v	1	3	3
wd_v	2	5	5
jsp_v	5.6	7.5	6.5

d

v	3	5	6
w_v	2	2	2
d_v	4	3	2
s_v	10	7	5
p_v	4	4	2
pd_v	3	3	3
ps_v	5	5	6
cf_v	3	3	3
wd_v	5	5	5
jsp_v	7.5	6.5	5.5

e

v	5	6	7
w_v	2	2	2
d_v	3	2	3
s_v	7	5	6
p_v	4	2	3
pd_v	3	3	4
ps_v	5	6	10
cf_v	3	3	5
wd_v	5	5	7
jsp_v	6.5	5.5	7.4

f

v	6	7	8
w_v	2	2	2
d_v	2	3	2
s_v	5	6	3
p_v	2	3	5
pd_v	3	4	3
ps_v	6	10	7
cf_v	3	5	5
wd_v	5	7	7
jsp_v	5.5	7.4	5.5

g

v	7	8
w_v	2	2
d_v	3	2
s_v	6	3
p_v	3	5
pd_v	4	3
ps_v	10	7
cf_v	5	5
wd_v	7	7
jsp_v	7.4	5.5

h

v	8
w_v	**1**
d_v	2
s_v	3
p_v	**7**
pd_v	**3**
ps_v	**6**
cf_v	**6**
wd_v	7
jsp_v	5.6

same problem we used the following methodology. Firstly, we selected the spanning tree algorithms to be considered in our comparative analysis, besides current IEEE spanning tree algorithm. We considered the following algorithms: *Wong's* algorithm, as the current fastest approximation MRCT algorithm, and *Add* as the fastest approximation MRCT algorithm for the specific case of homogeneous input graphs, as well as the two most used MST algorithms, *Kruskal's* and *Prim's* algorithms, known to provide good approximation to an MRCT for highly heterogeneous graphs [9,10]. Secondly, we defined the parameters to be taken into account while comparing the algorithms. Two parameters were considered: (1) the routing cost of the spanning tree returned by each algorithm, given that we are comparing approximation MRCT algorithms; (2) the execution time for each algorithm, since our purpose is to demonstrate that *Campos's* algorithm requires lower execution time than the fastest known approximation MRCT algorithm and that it has time complexity of the same order of *Add*, *Kruskal's*, *Prim's*, and IEEE algorithms. Finally, we defined the type of analysis to be used in the evaluation of our algorithm. A possible approach could be to consider an asymptotical analysis from the perspective of the routing cost and a time complexity analysis regarding the execution time of the algorithms. However, such kind of analysis provides only information about the worst cases. Thereby, we considered a simulation oriented approach so that we could evaluate the performance of the algorithms in other cases, namely in practical cases; a wide range of randomly generated input graphs, from homogeneous to highly heterogeneous graphs, have been considered for that purpose. Since we did not find any existing and open simulator addressing our specific purpose we have developed a new simulator from the scratch.

7. Spanning trees simulator

The simulations have been performed using a new simulator, called Spanning Trees Simulator (STS). STS is written in C++ and uses the popular Boost Graph Library (BGL) [27] for the computation of MSTs and SPTs according to the classical spanning tree algorithms, i.e., *Dijkstra's*, *Prim's*, and *Kruskal's* algorithms; in addition, it includes *Add*, *Wong's*, and *Campos's* algorithms. *Campos's* algorithm is implemented using Fibonacci heaps in conjunction with adjacency lists so that a time complexity $O(m + n\log(n))$ can be achieved. STS accepts the following input parameters:

1. $s_{\max}$ – the maximum size for the input graphs expressed in number of vertices.
2. $s_{\min}$ – the minimum size for the input random graph. It also defines the granularity in the generation of the input graphs. For instance, if the maximum size is set to 30 and the minimum size is set to 10, the simulator will simulate input graphs with 10, 20, and 30 vertices.

Please cite this article in press as: R. Campos, M. Ricardo, A fast algorithm for computing minimum routing cost spanning trees, Comput.Netw. (2008), doi:10.1016/j.comnet.2008.08.013

3. e_{max} – the maximum number of edges to be simulated for each input graph with n vertices.
4. S – the set of edge weights considered while generating an input random graph.
5. *sta* – the spanning tree algorithm to be simulated, besides the two spanning tree algorithms considered as reference by the simulator, i.e., current IEEE spanning tree algorithm and *Wong's* algorithm.
6. *nr* – the number of simulation runs.

Taking into account these input parameters the simulator defines the following sequence of steps for each simulation run. For each $n \in \{s_{min}, 2s_{min}, \ldots, s_{max}\}$, STS generates $e_{max} - (n-1) + 1$ random graphs with increasing number of edges $m \in \{n-1, n, n+1, \ldots, e_{max}\}$. It employs one of the variants of the Erdös and Renyi [28] model for generating the random graphs. The first variant considers a random graph with n vertices and defines a probability p that an edge is present in the graph. The second sets an edge between each pair of vertices with equal probability, independently of the other edges. STS uses the latter variant and for each randomly selected edge assigns it a random weight from the set S with equal probability. This means that G is chosen uniformly at random from the collection of all graphs which have n vertices and m edges. For each G, the simulator computes n SPTs, the routing cost for each SPT, and selects the lowest routing cost as the routing cost for the approximate MRCT provided by *Wong's* algorithm. In addition, it calculates the expected value for the routing cost of an arbitrarily selected SPT, as considered by current IEEE spanning tree algorithm, and calculates the routing cost for a spanning tree computed using the *sta* algorithm; IEEE and *Wong's* algorithms are considered as a reference by STS, since they represent, respectively, the current used spanning tree algorithm in real networks and the fastest approximation MRCT algorithm. Besides the routing cost, the simulator calculates the execution time for each spanning tree algorithm. At each simulation run STS stores the values of the routing cost and execution time found for each (n, m). The whole process is repeated for *nr* simulations.

The STS simulator provides two output parameters: the average routing cost and the average execution time for each (n, m) and for each simulated spanning tree algorithm. These values are computed considering *nr* simulations; the margin of error for each value is also presented regarding a confidence interval of 95%. STS provides the routing costs and the execution times for *Wong's* and *sta* algorithms normalized to the values obtained for IEEE spanning tree algorithm, i.e., the time required to compute an SPT.

STS is open source and can be downloaded at [29], where further information about it is provided, namely regarding its utilization.

8. Simulation setup

We considered a wide set of simulations to evaluate *Campos's* algorithm against the spanning tree algorithms referred in Section 6. The STS simulator was used to carry out the simulations. The following input parameters were defined: (1) size of the input random graph; (2) set of edge weights used in the generation of the input random graph. Since our focus is on graphs modeling small scale networks (see Section 1), we considered input random graphs of the following sizes: 10, 30, and 50 vertices. For each n defining the size of the input graph, we performed simulations for each possible number of edges, i.e., from $n-1$ (tree) to $n \cdot (n-1)/2$ (complete graph) in order to cover the whole spectrum of topologies for a graph with n vertices. A wide range of input graphs, from homogeneous to highly heterogeneous graphs, have been covered by defining different input sets of edge weights generated using the following discrete functions:

$$u[n] = K^{n-1}, \quad 1 \leqslant n \leqslant 5 \wedge n \in N, \tag{12}$$

$$v[n] = n, \quad 1 \leqslant n \leqslant 5 \wedge n \in N, \tag{13}$$

$$w[n] = \begin{cases} 1, n = 0 \\ K_1.w[n-1], n \text{ odd}, 1 \leqslant n \leqslant 10 \wedge n \in N, \\ K_2.w[n-1], n \text{ even}, 2 \leqslant n \leqslant 10 \wedge n \in N, \end{cases} \tag{14}$$

where K, K_1, K_2 are constants and N represents the set of natural numbers. In our simulations, $K \in \{1, 10\}$, $K_1 = 5$, and $K_2 = 2$. For $K = 1$ the discrete function (12) was used to generate homogeneous input graphs; for $K = 10$ it was used to generate highly heterogeneous input graphs with edge weights selected from the sets $S1 = \{1, 10, 100\}$ and $S2 = \{1, 10, 100, 1000, 10000\}$. The functions 13 and 14 were used to generate slightly heterogeneous and highly heterogeneous input graphs, respectively. (13) was used to generate the sets $S3 = \{1, 2, 3\}$ and $S4 = \{1, 2, 3, 4, 5\}$ and (14) was employed to generate the set $S5 = \{1, 5, 10, 50, 100, 1000, 5000, 10000, 50000\}$. There is no particular reason to consider these specific discrete functions for generating the sets of edge weights. The only objective was to consider sets of edge weights with different levels of heterogeneity. Therefore, other sets could be considered, as long as they could be used to generate input random graphs with different levels of heterogeneity.

In the context of our simulations a set of edge weights represents the possible values for a given metric, e.g., bandwidth, delay, power consumption, typically used to characterize a link of a communication network. In practice, few possible values are usually defined for a given metric. Consider, for instance, the example of an Ethernet network. With the advent of 1Gbit/s and 10 Gbit/s links coexisting with legacy 100 Mbit/s links, we have a scenario where links of different capacities coexist. If we characterize each link according to its capacity we get the following possible values: 1, 10, and 100. Another example is a future Personal Area Network composed by different wired/wireless links. Technologies such as UWB, Ethernet, Bluetooth, and IEEE 802.11 may coexist in the same network. If we assign weights to each wired/wireless link according to some of the metrics above mentioned a few possible values will be defined as well. In that sense, the sets of edge weights considered in our simulations include a few values; however, in order to cover different cases, we took into account different sizes for the sets of edge weights by varying the number of values generated through the discrete functions (12)–(14).

Please cite this article in press as: R. Campos, M. Ricardo, A fast algorithm for computing minimum routing cost spanning trees, Comput.Netw. (2008), doi:10.1016/j.comnet.2008.08.013

The same set of simulations was performed for each spanning tree algorithm under analysis. One hundred simulations were performed for each set of input parameters (size of graph, set of edge weights).

9. Results and evaluation

This section presents the simulation results and evaluates the various spanning tree algorithms under analysis from the perspectives of the routing cost and execution time, with special focus on *Campos's* algorithm. The most relevant results are presented; further results can be found in [29]. Along the description and analysis of the simulation results we use the expressions "sparse graph" and "dense graph" to refer to the density of the input graphs; sparse graphs refer to graphs with the number of edges $m = \Theta(n)$, while dense graphs refer to graphs with $m = \Theta(n^2)$.

9.1. Routing cost

The routing cost of the spanning trees computed by each spanning tree algorithm was analyzed for different input graphs. Below, we present the simulation results for homogeneous, slightly heterogeneous, and highly heterogeneous input graphs. The curves for the different spanning tree algorithms under analysis, i.e., *Campos's*, *Prim's Kruskal's*, *Add*, and *Wong's* algorithms, are shown in Figs. 4–6 with the corresponding 95% confidence intervals. The results were obtained for input random graphs with 10, 30, and 50 vertices. They are normalized to the routing cost of the spanning tree obtained by using IEEE spanning tree algorithm and are presented as a function of the average degree per vertex (the maximum average degree per vertex is equal to $n - 1$ and corresponds to the complete input random graph). For the sake of clearness, the curves corresponding to spanning tree algorithms providing similar routing costs are presented in the same plot.

9.1.1. Homogeneous graphs

The plots shown in Fig. 4 were obtained considering the set of edge weights $S = \{1\}$ as input to the STS simulator; as mentioned in Section 6, the simulator uses the set of edge weights as input to generate the input random graphs. The set S was obtained by using the discrete function in (12) with $n = 1$.

From the plots in Fig. 4 we can firstly conclude that *Campos's* algorithm provides an approximate MRCT with the same routing cost as the spanning tree provided by *Wong's* algorithm regardless of the number of vertices of the input graph. Therefore, concerning homogeneous input graphs *Campos's* algorithm can in fact be used as an approximation MRCT algorithm providing the same routing cost as *Wong's* algorithm while running in $1/n$th of the time. In addition, we conclude that, in general, our algorithm provides a spanning tree with the same routing cost as *Add* algorithm. But, as the size of the input graph increases, it performs even better than *Add* algorithm for sparse input graphs. This can be seen in Fig. 4 for $n = 50$; the difference in the routing cost, on average, reaches

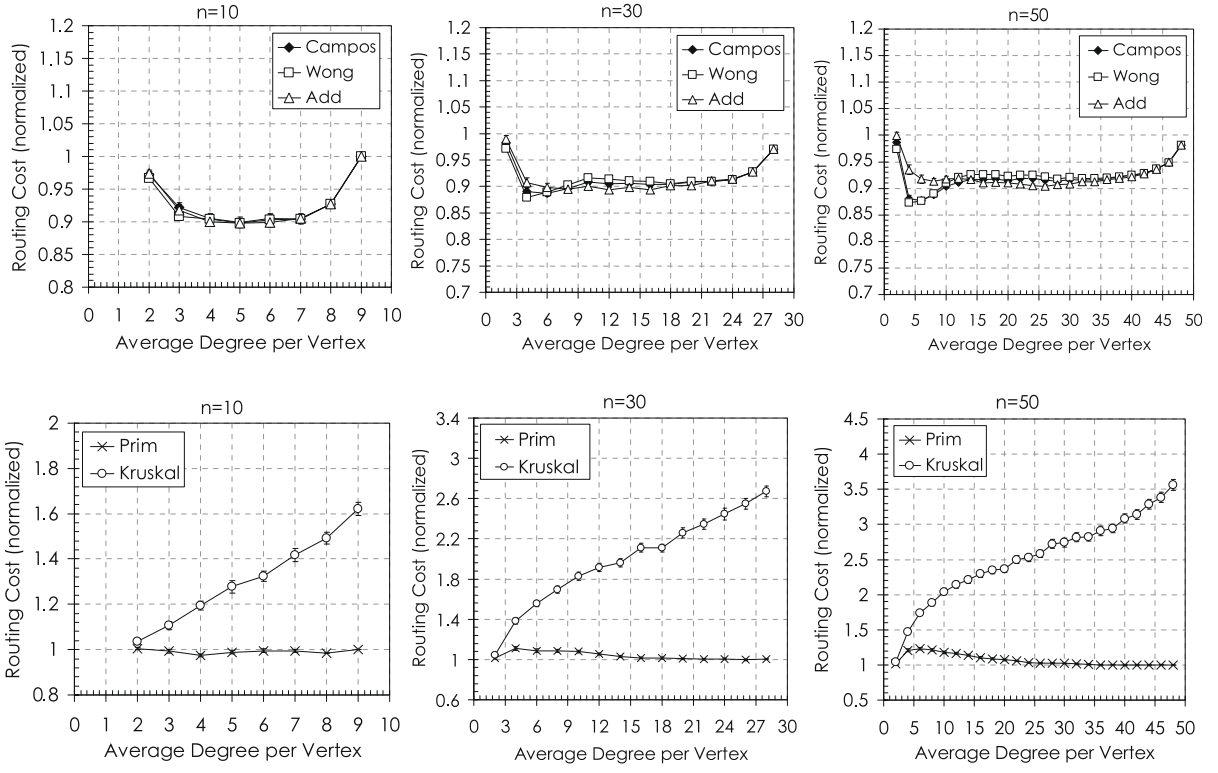

Fig. 4. Routing costs for each spanning tree algorithm under analysis when *homogeneous* input graphs of 10, 30, and 50 vertices are considered.

Please cite this article in press as: R. Campos, M. Ricardo, A fast algorithm for computing minimum routing cost spanning trees, Comput.Netw. (2008), doi:10.1016/j.comnet.2008.08.013

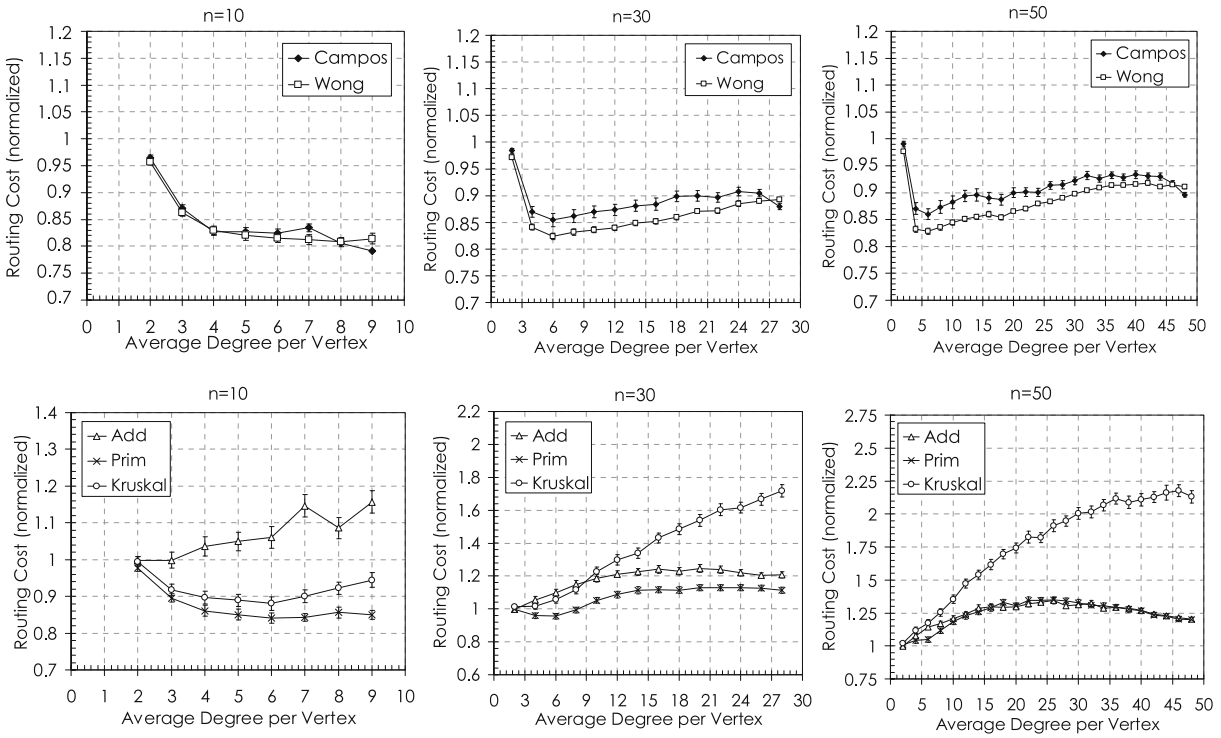

Fig. 5. Routing costs for each spanning tree algorithm under analysis when *slightly heterogeneous* input graphs of 10, 30, and 50 vertices are considered.

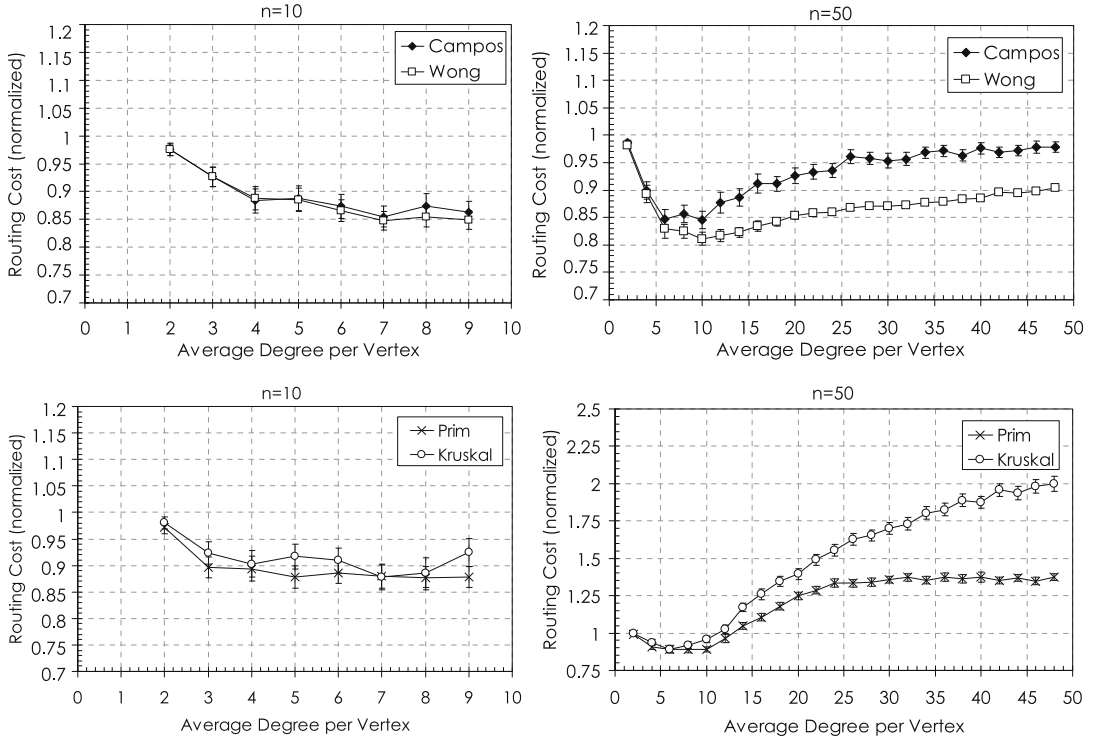

Fig. 6. Routing costs for each spanning tree algorithm under analysis when *highly heterogeneous* input graphs of 10 and 50 vertices are considered.

about 7% for an average degree per vertex equal to 4. The difference can be explained by the cf_v parameter considered in *Campos's* algorithm. Unlike *Add* algorithm, which only considers the degree of the vertices as the decision

Please cite this article in press as: R. Campos, M. Ricardo, A fast algorithm for computing minimum routing cost spanning trees, Comput.Netw. (2008), doi:10.1016/j.comnet.2008.08.013

criterion to select the next spanning relay, *Campos's* algorithm also considers the cost of the path to the initial vertex as criterion. This contributes to a reduction of the diameter of the final spanning tree and, consequently, of its routing cost, although it only takes relevant effect for sparse graphs; for dense graphs the cost of the path between each vertex and the initial vertex tend to be similar or equal, as it happens for the complete graph, and the degree of the vertices criterion is enough to find a good approximate MRCT. Finally, *Campos's* algorithm, as *Wong's* and *Add* algorithms, provides lower routing costs than current IEEE spanning tree algorithm, independently of the size of the input graph, even though the gain increases with the size of the input graph and is higher for sparse graphs; for $n = 10$ the gain may achieve 10%, while for $n = 50$ it may achieve about 13%.

As expected, MST algorithms represent a bad approximate MRCT when the input graph is homogeneous. Still, *Prim's* algorithm performs better than *Kruskal's* algorithm. The routing cost for *Kruskal's* algorithm diverges from the routing costs provided by any of the other spanning tree algorithms. The algorithm is only concerned with the selection of the $n-1$ edges with the lowest weight so that the total cost of the resulting spanning tree can be minimized. Since in this case the weights are all the same, the final spanning tree will result from the random selection of $n-1$ edges out of the input graph G that together form a spanning tree of G. Thus, with no selection criterion in place, the resulting spanning tree tends to have higher diameter and, consequently, higher routing cost. Regarding *Prim's* algorithm, for small graphs, it provides routing costs close or equal to IEEE algorithm. Nonetheless, concerning sparse graphs as the number of vertices in the input graph increases *Prim's* algorithm tends to provide higher routing costs and diverges from the routing costs provided by IEEE algorithm. For dense input graphs *Prim's* algorithm presents the same routing cost of IEEE algorithm and in the extreme case (complete graph), it provides the same routing cost of *Campos's*, *Wong's*, and *Add* algorithms, since it actually computes an SPT from the initial vertex's perspective that defines the best SPT (for the complete graph any SPT is the best SPT).

9.1.2. Slightly heterogeneous graphs

The plots shown in Fig. 5 were obtained by considering the set of edge weights $S = \{1, 2, 3\}$ as input to the STS simulator. The set S was obtained by using the discrete function in (13) with $1 \leqslant n \leqslant 3$.

Regarding *Campos's* algorithm we can conclude the following. When the number of vertices in the input graph is small ($n = 10$) *Campos's* algorithm provides the same results as *Wong's* algorithm, independently of the input graph density. As the size of the input graph increases *Campos's* algorithm tends to perform slightly worse than *Wong's* algorithm but provides similar or the same results when the input graph is dense. Then, our algorithm represents a suitable alternative to *Wong's* algorithm also when the input graph is slightly heterogeneous. On the other hand, *Campos's* algorithm provides lower routing costs than current IEEE spanning tree algorithm, independently of the size of the input graph. The gain may reach 21% but, in contrast to what happens for homogeneous input graphs, it decreases with the size of the input graph (this occurs for both *Campos's* and *Wong's* algorithms).

Concerning MST algorithms the following conclusions can be drawn. *Prim's* algorithm provides a good approximate MRCT when the size of the input graph is small. For $n = 10$ the routing cost is just about 5% greater than the routing cost provided by *Campos's* and *Wong's* algorithms; the results for *Kruskal's* algorithm are similar but diverge for dense input graphs. We conclude that the edge weights have the major impact in the selection of the approximate MRCT when the input graph has small size or, in other words, edge weights can almost be used as the only selection criterion regarding the construction of the approximate MRCT. However, as the size of the input graph increases MST algorithms provide worse results, even worse than current IEEE spanning tree algorithm. This can be explained by using the example given in Fig. 2. For a given input graph G with 6 vertices, if one of the spanning trees shown in Fig. 2 is an MST the other also defines an MST for G, since both have total cost equal to 7. An MST algorithm can select both with equal probability, although they have different topologies and, as such, have different routing costs; the spanning tree on the right-hand side has lower routing cost. This fact, on average, contributes to increase the routing cost of the final spanning tree. For small size input graphs the number of possible MSTs is small and the differences in topology are small too; thereby, the average routing cost does not greatly diverge from the routing cost of the best MST (from the perspective of the routing cost), which represents a good approximate MRCT, as implicitly shown by our simulation results. Yet, the number of possible MSTs increases for bigger input graphs, as well as the number of MSTs providing higher routing cost than the best MST. On average, the routing cost of the spanning tree computed by an MST algorithm increases accordingly. Nevertheless, for the input graphs considered in this case the differences between the routing cost for the MST algorithms and the other spanning tree algorithms are attenuated by the reduced heterogeneity of the edge weights; for highly heterogeneous the differences are greater when the input graphs are dense (see Section 9.1.3).

As theoretically expected, *Add* algorithm does not provide good results for heterogeneous graphs, even for slightly heterogeneous graphs. Regardless of the size of the input graph it provides routing costs that diverge from the results achieved by the other algorithms. This is because it does not consider the edge weights at all; only topology is considered which is in general insufficient for constructing a spanning tree with small routing cost as mentioned in Section 5. The conclusion is that, in fact, *Add* algorithm cannot be used as an approximate MRCT when the input graph is non-homogeneous.

9.1.3. Highly heterogeneous graphs

The plots shown in Fig. 6 were obtained by considering the set of edge weights $S = \{1, 10, 100, 1000, 10000\}$ as input to the STS simulator. S was obtained by using the discrete function in (12) with $K = 10$ and $1 \leqslant n \leqslant 5$. Only the

Please cite this article in press as: R. Campos, M. Ricardo, A fast algorithm for computing minimum routing cost spanning trees, Comput.Netw. (2008), doi:10.1016/j.comnet.2008.08.013

plots for $n = 10$ and $n = 50$ are presented, since for $n = 30$ the results are very similar to the results obtained for $n = 50$.

From the plots in Fig. 6 we conclude that, when the number of vertices in the input graph is small ($n = 10$), *Campos's* algorithm provides the same results as *Wong's* algorithm regardless of the density of the input graph. As the size of the input graph increases *Campos's* algorithm performs differently depending on the density of the input graph. For sparse graphs it provides the same routing costs as *Wong's* algorithm. This is illustrated by the plot of Fig. 6 for $n = 50$; for $n = 30$ the performance of the algorithm is very similar. As the average degree per vertex in the input graph increases *Campos's* algorithm slightly diverges from the routing costs provided by *Wong's* algorithm (differences of about 10% can be achieved). Then, for highly heterogeneous graphs our algorithm provides the same results as *Wong's* algorithm when the input graph is small or, for bigger input graphs, when it is sparse; for dense graphs it provides slightly worse results. When compared to IEEE spanning tree algorithm, it provides lower routing costs independently of the size of the input graph. The gain may reach about 17% for $n = 50$, although, for dense input graphs it decreases significantly. The worse performance for dense graphs can be explained as follows. For highly heterogeneous input graphs *Campos's* algorithm considers the edge weight as the major selection criterion (see Section 5). Thereby, it tends to follow the results provided by *Prim's* algorithm, which exhibits worse performance for dense input graphs; still, the increasing rate of the routing cost as the input graph becomes denser is attenuated by the other parameters considered in our algorithm.

Regarding MST algorithms the simulation results confirm that as the input graph becomes more heterogeneous they tend to approximate the routing costs provided by *Wong's* algorithm and can actually be used as an approximate MRCT. This is evident for $n = 10$, with *Prim's* algorithm providing routing costs that can be just 1% higher than the routing costs provided by *Wong's* algorithm; *Kruskal's* algorithm also shows a tendency to converge to the results given by *Wong's* algorithm. For bigger input graphs it can only be observed for sparse graphs in this case. This is due to the fact that for dense graphs the high heterogeneity inherent to the edge weights set S tends to be less reflected in the generated random input graphs as the number of edges in the graph increases. In other words, the probability of equal weights assigned to the edges of the graph increases. Thus, in order to reduce this probability and increase the level of heterogeneity in the input graphs we have performed further simulations using the input edge weight $S = \{1, 5, 10, 50, 100, 500, 1000, 5000, 10000, 50000\}$ generated using the discrete function in (14). The obtained results can be found in [29]. They confirm that both *Prim's* and *Kruskal's* algorithms tend to approximate *Wong's* algorithm results also for denser graphs when higher heterogeneity is in place; for instance, for $n = 30$ the routing costs provided by *Prim's* algorithm are below the routing costs for IEEE algorithm regardless of the density of the input graph and are only about 10% higher than the routing costs provided by *Wong's* algorithm; the exception are the input graphs with density close to the complete graph, where *Prim's* algorithm results diverge a bit more from *Wong's* algorithm results. Furthermore, the obtained results show that *Campos's* algorithm follows the tendency of *Prim's* algorithm (used as basis for the former) and indeed provides better results as the heterogeneity in the input graph increases. In fact, this was already expected since *Campos's* algorithm is basically an improved version of *Prim's* algorithm, namely for highly heterogeneous input graphs, where the edge weights represent the most relevant criterion in the algorithm. This let us conclude that, for even higher heterogeneous input graphs, *Campos's* algorithm will tend to approximate further *Prim's* algorithm results which was demonstrated to represent an approximation MRCT algorithm or even the exact MRCT algorithm in such cases (see Section 3).

The simulation results for *Add* algorithm are not presented, since it was already shown in the previous section (Fig. 5) that the algorithm does not work for heterogeneous input graphs. For highly heterogeneous input graph this is simply exacerbated; during our simulations we verified that, for instance, for $n = 30$ the routing costs provided by *Add* algorithm could reach two and even three orders of magnitude the routings costs of the other spanning tree algorithms under analysis.

9.2. Execution time

The results shown in Fig. 7 are independent of the set of edge weights used to generate the input random graphs. In fact, the execution time is influenced by the number of vertices and the number of edges of the input graph, as expressed in Section 4 for the different spanning tree algorithms. Therefore, the results were obtained by only varying these two parameters. The values for each algorithm are normalized to the time needed to compute an SPT using the *Dijkstra's* algorithm. The execution time for *Wong's* algorithm is not shown since it is simply n times the execution time needed to compute an SPT.

The obtained results are according to the time complexities of the algorithms provided in Section 4. *Campos's* algorithm essentially takes the same execution time as *Dijkstra's* algorithm used to compute an SPT and 2 times the execution time of *Prim's* algorithm; this is mostly due to the further calculations performed by *Campos's* algorithm regarding the selection of the initial vertex and the computation of the ratio σ/μ characterizing the heterogeneity of the input graph. As expected, the execution times of both *Campos's* and *Prim's* algorithms do not strongly depend on the number of edges in the input graph; the number of vertices has the major impact as demonstrated by the time complexities provided in Section 4. *Add* algorithm takes execution times similar to *Prim's* algorithm for sparse graphs, but runs faster as the density of the input graph increases. This is due to the tendency for many vertices to be added to the spanning tree at each step of the algorithm, as explained in [14], in contrast to what happens in *Prim's* algorithm where a single vertex is added to the spanning tree at each step. According to our simulation results for $n = 30$ and $n = 50$, *Add* algorithm can be about 5 and 10 times faster than *Prim's* and *Campos's* algorithms, respectively, concerning dense input graphs; the difference is even greater when the number of vertices in the input

Please cite this article in press as: R. Campos, M. Ricardo, A fast algorithm for computing minimum routing cost spanning trees, Comput.Netw. (2008), doi:10.1016/j.comnet.2008.08.013

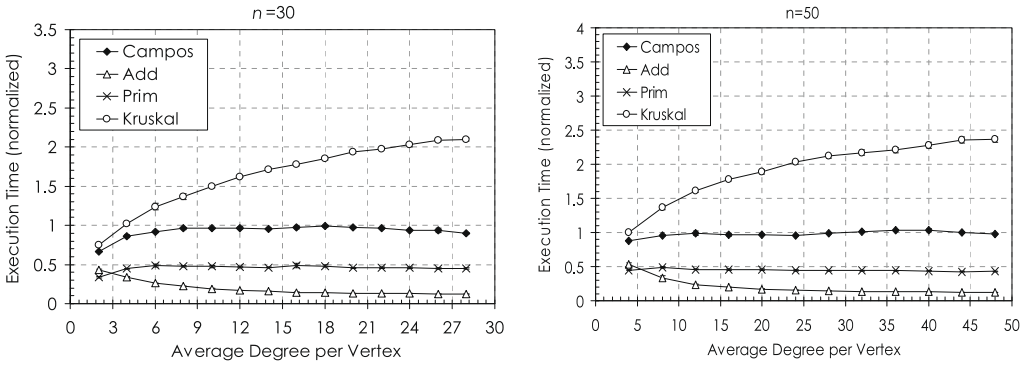

Fig. 7. Execution time for each spanning tree algorithm under analysis considering input graphs of 30 and 50 vertices.

graph increases. *Kruskal's* algorithm takes more time to compute a spanning tree. Also, its execution time increases with the number of edges in the input graph; this is according to the time complexity provided in Section 4. In spite of being simple to implement by using a priority queue, it is the least efficient spanning tree algorithm.

9.3. Discussion

Our early idea of combining the criteria used by the MST algorithms (edge weights) and *Add* algorithm (degree of the vertices) in a single algorithm revealed to be a good approach to design a new approximation MRCT algorithm. Besides performing well for the extreme cases already covered by *Add* and *Prim's* algorithms, *Campos's* algorithm provides good results for other cases. In fact, in general it provides results similar to those obtained using *Wong's* algorithm and runs much faster than the former. Nevertheless, regarding increasing heterogeneous input graphs it tends to perform better for sparse input graphs than for dense graphs, although this tendency disappears for extremely heterogeneous input graphs. In practice, the topology of a PAN or a LAN typically defines a sparse graph, since each network device is only directly linked to a fraction of the devices present in the network; this fraction even tends to decrease with the increasing number of nodes participating in the network. For instance, in current bridged Ethernet networks each bridge is only directly linked to a fraction of the bridges participating in the network. Another example is a future PAN envisioned to include multiple wired/wireless technologies [30]. The coexistence of different wired and wireless technologies and the limited number of network interfaces per network node precludes the establishment of direct links from each device to all the others. Thereby, *Campos's* algorithm represents a true faster alternative to *Wong's* algorithm in practical cases. On the other hand, in all cases we have analyzed, *Campos's* algorithm always provides a spanning tree with lower routing cost than the spanning tree computed using current IEEE algorithm, regardless of the number of vertices or the number of edges in the input graph; in particular, *Campos's* algorithm provides significantly lower routing costs for heterogeneous input graphs. As such, it may be used in bridging oriented solutions to define the active spanning tree with reductions in the routing cost that may reach 21%, on average, without introducing further time complexity from a pure algorithmic point of view. Additionally, for homogeneous input graphs it provides the same routing cost as the spanning tree constructed by using *Add* algorithm, in general, but performs better than the latter for sparse graphs as the number of vertices in the input graph increases. This means that, in practical cases, *Campos's* algorithm can provide lower routing costs than *Add* algorithm; the gain augments as the number of vertices in the input graph increases. On the other hand, concerning highly heterogeneous graphs it tends to approximate the results provided by the MST algorithms which in such conditions provide an approximate or the exact MRCT.

In practice, the reduction in execution time enabled by *Campos's* algorithm when compared to *Wong's* algorithm is important. Given that we may be talking about mobile and dynamic networks that may experience frequent topology changes, a new MRCT must be computed as fast as possible, so that the network reconfiguration time is minimal. In addition, while in highly capable devices the reduction in computation time may not make much difference, in PANs, for example, it may make difference indeed. The devices forming such networks may have limited computational capability, which may lead to a significant absolute reduction in computation time. On the other hand, a faster computation of an MRCT leads to lower energy consumption, which is an important point considering devices running on battery power, and releases the CPU earlier for other tasks.

In general, *Add* algorithm does define a good approximation MRCT algorithm for homogeneous graphs. When compared to *Wong's* algorithm it tends to provide worse results for sparse input graphs as the size of the input graph increases. Thereby, it may diverge from the results provided by *Wong's* algorithm for practical cases. On the other hand, it does not perform well for heterogeneous input graphs. Therefore, in spite of being faster than *Wong's* and *Campos's* algorithms, it has limited applicability in practice. It can only be applied to homogeneous networks that either contain small number of nodes or define a dense topology, where the major advantages of the algorithm come up, as far as execution time and routing cost are concerned.

The MST algorithms tend to approximate the result provided by *Wong's* algorithm as the weights in the graph

Please cite this article in press as: R. Campos, M. Ricardo, A fast algorithm for computing minimum routing cost spanning trees, Comput.Netw. (2008), doi:10.1016/j.comnet.2008.08.013

become increasingly heterogeneous; yet, for practical cases they can only be considered as approximation MRCT algorithms for small input graphs ($n = 10$). Conversely, they do not provide good results for homogeneous graphs. In spite of converging to the same results as the heterogeneity in the input graph increases, the two MST algorithms considered for analysis do not exhibit the same performance; *Kruskal's* algorithm generally provides worse results than *Prim's* algorithm and is less efficient than the latter. This is explained by the different procedures followed by each algorithm. *Kruskal's* algorithm does not take topology into account at all. It only cares about selecting the $n - 1$ edges with the lowest weight so that the total cost of the resulting spanning tree can be minimized. When there is more than one edge with equal weight the algorithm selects randomly one of them to be positioned first in the priority queue. On average, this leads to the construction of stretched spanning trees that have higher diameters and, consequently, higher routing costs. Rather, *Prim's* algorithm somehow considers topology as an implicit selection criterion. At each step, after selecting the next vertex v to be added to the partial spanning tree, the algorithm adds all the neighbor vertices of v to the list L. If a given neighbor vertex v_n is already in L and the edge connecting it to v has weight equal to the weight currently assigned to v_n the algorithm does not update the weight and the candidate parent vertex in L. This tends to reduce the number of parent vertices in the final spanning tree and, consequently, contributes to the construction of an MST with lower routing cost than the MST provided by *Kruskal's* algorithm.

In summary, *Campos's* algorithm actually represents a new faster approximation MRCT algorithm for practical cases. It provides the same results as the currently fastest known approximation MRCT algorithm, *Wong's* algorithm, and runs n times faster, where n defines the number of vertices in the input graph. On the other hand, *Campos's* algorithm has the same time complexity as *Add* and *Prim's* algorithms, but performs well also for other than the extreme cases covered by the formers. In practice, it can even provide better results than *Add* algorithm for homogeneous input graphs. In addition, it constructs a spanning tree with routing cost that may reach up to 79% the routing cost of the tree constructed by current IEEE spanning tree algorithm; the routing cost is particularly lower for heterogeneous networks, exactly where the use of bridging may be of greater interest. If, for instance, the edge weights represent the delay associated to each network link, such reduction in routing cost will mean a reduction on the average communication delay, and consequently an increase on the average throughput, between every pair of nodes.

10. Future work

The evaluation of *Campos's* algorithm and the various spanning tree algorithms considered herein, using real networks and real metrics, such as throughput or delay, may be a future work item. Furthermore, the comparative analysis presented may be extended by considering other approximation MRCT algorithms, namely the 15/8-approximation and 3/2-approximation algorithms running in time $O(n^3)$ and $O(n^4)$, respectively, proposed in the literature [11]. It would be interesting to evaluate whether the results provided by these algorithms are significantly better than the results provided by *Campos's* and *Wong's* algorithms, in practical cases. Finally, we may consider improvements to *Campos's* algorithm. Along the simulations we verified that the routing cost of the final spanning tree depends significantly on the selected initial vertex. Currently, for increasingly heterogeneous graphs, the algorithm cannot guarantee that the initial vertex is the right one, i.e., the vertex with the highest degree in the final spanning tree. Thus, improvements to the selection of the initial vertex may be considered in the future as a way to improve the results achieved by our algorithm. In addition, the definition of the composed parameters wd_v and jsp_v may be modified by considering other basic parameters either to be combined with current parameters or to replace them.

11. Conclusion

Communication networks have been developed based on two networking approaches: bridging and routing. The convergence to an all-Ethernet paradigm and the increasing heterogeneity found in Personal Area Networks (PANs) and Local Area Networks (LANs) emphasizes the current and future applicability of bridging oriented solutions based on IEEE 802.1D bridges. The use of IEEE 802.1D bridges requires that a single spanning tree is configured as the active network topology. A Minimum Routing Cost Tree is known to be the optimal spanning tree, though its computation is a NP-hard problem. We proposed a new approximation MRCT algorithm, named *Campos's* algorithm, that has time complexity lower than the fastest known approximation MRCT algorithm and provides a spanning tree with similar routing cost, in practice. Additionally, our algorithm provides a spanning tree with lower routing cost than the routing cost of the spanning tree computed using current IEEE algorithm; on average, the routing cost reductions may reach 21%. Thus, in practice, bridging oriented solutions have gains in using our algorithm instead of current IEEE algorithm, namely in current and future heterogeneous PANs/LANs. The new algorithm may be used together with a signaling protocol that enables it to operate in a distributed manner.

Acknowledgements

The authors would like to thank the support from the Portuguese Foundation for Science and Technology (FCT) under the fellowship SFRH/BD/19429/2004/. Also, they would like to thank to the anonymous reviewers for their valuable comments that did contribute to improve the quality of the paper.

References

[1] IEEE 802.11 Work Group, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, 1999.
[2] Bluetooth SIG, Personal Area Networking Profile, Version 1.0, 2003.
[3] Standard ECMA-368, High Rate Ultra Wideband PHY and MAC Standard, first ed., 2005.

[4] B. Brackenridge, UWB the WiMedia Way, 2007. <http://www.extremetech.com/article2/0,1697,2129903,00.asp>.
[5] IEEE 802.1D Work Group, IEEE Standard for Local and Metropolitan Area Networks: Media Access Control (MAC) Bridges, 2004.
[6] M. Bahr, Update on the hybrid wireless mesh protocol of IEEE 802.11s, in: Proceedings of the IEEE International Conference on Mobile Adhoc and Sensor Systems (MASS 2007), Pisa, 2007, pp. 1–6.
[7] R. Droms, Dynamic host configuration protocol, IETF RFC 2131 (1997).
[8] D.C. Plummer, An Ethernet address resolution protocol, IETF RFC 826 (1982).
[9] P. Mieghem, S. Langen, Influence of the link weight structure on the shortest path, Physical Review E 71 (056113) (2005) 1–13.
[10] P. Mieghem, S.M. Magdalena, Phase transition in the link weight structure of networks, Physical Review E 72 (056138) (2005) 2–7.
[11] B.Y. Wu, K. Chao, Spanning Trees and Optimization Problems, Chapman & Hall, 2004.
[12] B.Y. Wu et al., A polynomial time approximation scheme for minimum routing cost spanning trees, in: Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, 1998, pp. 21–32.
[13] R. Wong, Worst-case analysis of network design problem heuristics, SIAM Journal of Algebraic Discrete Mathematics 1 (1980) 51–63.
[14] V. Grout, Principles of cost minimization in wireless networks, Journal of Heuristics 11 (2005) 115–133.
[15] P. Gupta, P. Kumar, Capacity of wireless networks, IEEE Transactions on Information Theory 46 (2) (2000) 388–404.
[16] P. Stuedi, G. Alonso, Computing throughput capacity for realistic wireless multihop networks, in: Proceedings of the Ninth ACM International Symposium on Modeling Analysis and Simulation of Wireless and Mobile Systems, Terromolinos, Spain, 2006, pp. 191–198.
[17] K. Lui, W.C. Lee, K. Nahrstedt, STAR: a transparent spanning tree bridge protocol with alternate routing, ACM SIGCOMM Computer Communication Review 32 (3) (2002) 33–46.
[18] T.L. Rodeheffer, C.A. Thekkath, D.C. Anderson, SmartBridge: a scalable bridge architecture, in: Proceedings of ACM SIGCOMM, Stockholm, 2000, pp. 205–216.
[19] R. Perlman, Rbridges: transparent routing, in: Proceedings of INFOCOM, vol. 2, Hong Kong, 2004, pp. 1211–1218.
[20] F.D. Pellegrini, D. Starobinski, M.G. Karpovsky, L.B. Levitin, Scalable cycle-breaking algorithms for gigabit ethernet backbones, in: Proceedings of INFOCOM, vol. 4, Hong Kong, 2004, pp. 2175–2184.
[21] N. Huang, Y. Cheng, An effective spanning tree algorithm for a bridged LAN, in: Proceedings of the International Workshop on Advanced Communication and Applications for High Speed Networks, Munich, 1992, pp. 43–49.
[22] K. Elmeleegy, A.L. Cox, T.S. Eugene Ng, EtherFuse: an ethernet watchdog, in: Proceedings of ACM SIGCOMM, Kyoto, 2007, pp. 253–264.
[23] R. Prim, Shortest connection networks and some generalizations, Bell System Technical Journal 36 (1957) 1389–1401.
[24] M. Fredman, R. Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms, Journal of the ACM (JACM) 34 3 (1987) 596–615.
[25] J.B. Kruskal, On the shortest spanning subtree of a graph and the traveling salesman problem, in: Proceedings of the American Mathematical Society, vol. 7, 1956, pp. 48–50.
[26] E.W. Dijkstra, A note on two problems in connexion with graphs, in: Numerische Mathematik, vol. 1, 1959, pp. 269–271.
[27] Boost Graph Library (BGL). <http://www.boost.org/libs/graph/doc/index.html>.
[28] P. Erdos, A. Renyi, On random graphs I, Publicationes Mathematicae Debrecen 6 (1959) 290–297.
[29] Rui Campos. <http://telecom.inescporto.pt/~rcampos>.
[30] R. Campos, M. Ricardo, Dynamic and automatic connection of personal area networks to the global internet, in: Proceedings of the 2006 International Conference on Wireless communications and mobile computing, Vancouver, 2006, pp. 581–586.

Rui Campos received a Diploma degree in Electrical and Computers Engineering in 2003 from University of Porto, Portugal. He works as a researcher at INESC Porto and since 2004 he is pursuing his PhD in Electrical and Computers Engineering. His research interests include mobile communications, autoconfiguration and self-management of Personal Area Networks, and spanning tree algorithms.

Manuel Ricardo received a Diploma degree in 1988, an M.S. in 1992, and a Ph.D. in 2000, all in Electrical and Computers Engineering from University of Porto, Portugal. He is an associate professor at University of Porto, where he gives courses in mobile communications and computer networks. He also leads the Communication Networks and Services Area at INESC Porto.

Please cite this article in press as: R. Campos, M. Ricardo, A fast algorithm for computing minimum routing cost spanning trees, Comput.Netw. (2008), doi:10.1016/j.comnet.2008.08.013