Transactions D:
Computer Science & Engineering and
Electrical Engineering
Vol. 17, No. 2, pp. 99–104
© Sharif University of Technology, December 2010

Minimum Height Path Partitioning of Trees

A. Bagheri[1,*] and M. Razzazi[1]

Abstract. *Graph partitioning is a well-known problem in the literature. In this paper, path partitioning of trees in which the given tree is partitioned into edge-disjoint paths is considered. A linear time algorithm is given for computing a path partitioning of minimum height.*

Keywords: *Graph partitioning; Path partitioning; Minimum height path partitioning.*

INTRODUCTION

Graph partitioning is a well-known problem that has many applications specially in parallel computing, telecommunication networks, data storage, image processing and operation research [1,2]. In general, in this problem either the edge set of a graph, *edge partitioning*, or the vertex set of a graph, *vertex partitioning*, is partitioned into disjoint subsets such that some constraints are satisfied and some criteria are optimized. Examples of these constraints, criteria, and required notations are given in the following.

Let $T = (V, E)$ denote a free tree with vertex set V and edge set E. We define $P = \{p_1, p_2, \cdots, p_r\}$ to be a *path partitioning* of T if, and only if, for $1 \leq i \leq r$ the conditions hold as follows:

1. Each p_i is a path of T.
2. The paths p_i are edge-disjoint.
3. There is exactly one path, called *the root path*, whose two end-vertices are leaves of T.
4. Any other path p has exactly one end-vertex that is a leaf of T and the other end-vertex is a non end-vertex of another path q; q is called the parent path of p.
5. Set P covers all the edges of T.

By end-vertices of a path we mean the two vertices of the path that have only one adjacent vertex in the path. The cardinality of set P is called the cardinality of partitioning. The height of a vertex v of a rooted tree T, $h(v, T)$, is the length of the longest path to a leaf of T from vertex v. The height of a rooted tree T, $h(T)$ is the height of the root. For any path partitioning P of a free tree T, let T_p denote the rooted tree obtained from P by contracting each path of P into a single vertex. Edge (u, v) belongs to T_p if, and only if, the corresponding path of u in P is the parent of the corresponding path of v. The root of T_p is the vertex corresponding to the root path of P. The height of a path partitioning P, $h(P)$, is defined as the height of T_p. Figure 1a shows a path partitioning P of a free tree T. In this figure, each path is included in a dashed box. The path $1-2-3-4-5$ is the root path of P whose two end-vertices 1 and 5 are leaves of T. The corresponding contracted tree T_p of P is shown in Figure 1b. The root path of P has been contracted to a single vertex a of T_p.

In this paper, we present a linear time algorithm for constructing a path partitioning for a given free tree, such that the height of the partitioning is minimum among all possible path partitionings of the tree. In brief, our algorithm finds a root path for the given

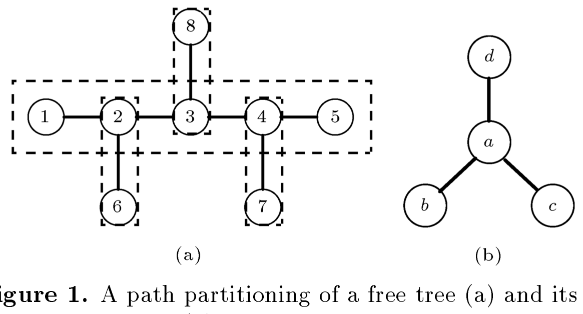

Figure 1. A path partitioning of a free tree (a) and its contracted tree T_p (b).

1. *Department of Computer Engineering and Information Technology, Amirkabir University of Technology, Tehran, P.O. Box 15875-4413, Iran.*
*. *Corresponding author. E-mail: ar_bagheri@aut.ac.ir*

Received 4 July 2009; received in revised form 17 July 2010; accepted 23 August 2010

tree and remove the path from the tree, then for each remained sub-tree recursively finds a root path. The root paths of the tree and the sub-trees are selected such that the height of partitioning is minimized. The detail of the algorithm is given in the next section. One of the applications of this path partitioning can be in (layered) drawing of trees [3-5]. An h-layer drawing of a graph G is a planar drawing of G in which each vertex is placed on one of h parallel lines and each edge is drawn as a straight line between its end-vertices.

Our motivation to research on the path partitioning problem is its application in a point-set embedding problem. In this problem, the vertices of a given n-node free tree should be mapped one-to-one onto n given points in the plane, such that the total edge length of the tree is minimized. This is a generalization of the well-known Travelling Salesman Problem (TSP). Our main idea to solve this problem is to convert the given tree into a simple path by deleting and adding some edges. This step is done using a path partitioning of the tree. Then we embed the simple path onto the point-set using one of the algorithms of TSP. At last we add the previously removed edges and remove the previously added edges. This yields an approximation algorithm for the problem whose approximation factor is a function of the height of the path partitioning.

Some results have been presented in the literature which are close to our result, but differ from ours in some aspects. In [6] a linear time algorithm was given for vertex partitioning of a rooted tree into subtrees (connected sub-graphs). The partitioning had to satisfy the knapsack constraint, i.e. the sum of the weights of the vertices of each sub-tree had not to exceed a given value. The partitioning that satisfied this constraint was called *admissible*. The objective was to minimize *the height of the partitioning*. The height of the partitioning considered the height of the rooted tree which was obtained by contracting each sub-tree into a single vertex. The difference between our result and the result of [6] is that we do not consider the knapsack constraint and our partitions are paths instead of sub-trees.

The problem of finding an admissible partitioning of minimum cardinality was studied in [7]. A generalization in which there were multiple weight functions on the vertices was investigated in [8]. Problems involving dissimilarities between sub-trees of a partitioning was considered in [9]. The problem of minimizing (maximizing) the maximum (minimum) weight of a sub-tree with respect to some weight functions are considered in [10-13]. An open problem in [11] is the most uniform vertex partitioning problem for trees in which the objective is to minimize the difference between the maximum and the minimum weights of the vertex set in the partitioning. For a special case that the tree is a path, a solution was given in [14]. In [2] the problem of splitting a tree into k connected components with roughly equal size was studied. For general k, a simple algorithm that finds a k-split with ratio at most three in $O(n \log k)$ time was proposed.

The path partitioning that we defined is a set of edge-disjoint paths, and we would like to find a path partitioning of minimum height. Partitioning of graphs into minimum number of vertex-disjoint paths have also been investigated in the literature. Since the Hamiltonian path problem is NP-complete so is the minimum cardinality path partitioning problem. Many linear-time algorithms were given for this problem for special classes of graphs including trees [15,16], cographs [17], interval graphs [18], circular arc graphs [19], bipartite permutation graphs [20] and block graphs [21,22]. A generalization of the problem, in which a constraint on the size of paths is also given, was considered in [23-25]. A linear-time algorithm was given in [26] for finding a minimum-cardinality set of simple-paths that cover all the edges of a given tree and, secondarily, have smallest total path lengths. In covering problem, partitions do not need to be disjoint.

The reminder of the paper is as follows. In the next section, a linear-time algorithm is given for computing a minimum height path partitioning of trees. Then the correctness proof and the time complexity of the proposed algorithm is presented. At last the conclusion is given.

MINIMUM HEIGHT PATH PARTITIONING OF TREES

In this section, we describe how a minimum height path partitioning of trees can be computed in linear time. For each edge (u, v) of a given free tree $T(V, E)$ we define two labels $l_v(u, v)$ assigned to end-vertex v of edge (u, v), and $l_u(u, v)$ assigned to end-vertex u of edge (u, v). Let $l_v(u, v) = \max\{l^1_w(v, w), l^2_w(v, w) + 1\}$, where $l^1_w(v, w)$ and $l^2_w(v, w)$ are respectively the largest and the second largest elements of $\{l_w(v, w) | (v, w) \in E, w \neq u\}$. Initially, let $l_v(u, v) = 1$ for each leaf v of T.

All the labels of the edges of a free tree T can be computed in linear time, by applying the Depth First Search (DFS) algorithm twice in two different phases. The starting vertex is arbitrary but should be the same in both phases. This is essential for correct computation of the labels. The details and the correctness proof are given in Theorem 1 in the next section. In the first phase, the label of one end-vertex of each edge is computed. In the second phase, the label of the other end-vertex of each edge is computed. Figure 2a shows the labels of the edges of a free tree T after the first phase, and Figure 2b shows the labels of

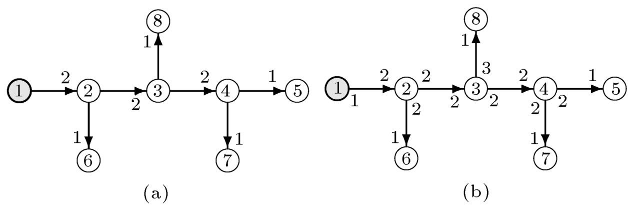

Figure 2. The labels of the edges of a free tree T.

the edges of T after the second phase. In this example, the DFS algorithm starts from vertex 1.

In the following, we describe our algorithm for computing a minimum height path partitioning of a given free tree $T = (V, E)$. If we remove an edge (u, v) from a free tree T, two sub-trees (connected sub-graphs) are left, say T_v and T_u which contain vertices v and u, respectively. Let $T_v(u, v)$ be the union of edge (u, v) and the sub-tree T_v. The pseudo-code of the partitioning algorithm is given by Algorithm 1. First, the labels of the edges of T are computed by calling procedure *ComputeLabels*. This procedure computes the labels by applying DFS twice as described before. Then, a leaf u of T is selected such that label $l_v(u, v)$, where v is the adjacent vertex of u, is minimum among all leaves of T. Next, procedure *ConstructPaths* is called with T and edge (u, v) as inputs to find a minimum height path partitioning of T.

In procedure *ConstructPaths*, first the root path of the path partitioning of T is computed by calling procedure *ConstructMainPath* with T and (u, v) as inputs. Edge (u, v) is the first edge of the root path of the path partitioning. The next edge of the root path is edge (v, w^*), where $w^* \neq u$ is an adjacent vertex of v, with the maximum label $l_{w^*}(v, w^*)$. The other edges of the root path are selected in the same way. Next, the other paths of the partitioning are constructed as follows. By removing the root path from the tree, some sub-trees are left. Each sub-tree was connected by an edge, *the connecting edge*, to the root path. For each sub-tree and its connecting edge, procedure *ConstructPaths* is called recursively.

Algorithm 1

```
PathPartitioning (T)

beginAlg
1. ComputeLabels (T)
2. Select the leaf u of the minimum label $l_v(u, v)$
3. ConstructPaths (T, (u, v))
endAlg

ConstructPaths (T,(u, v))
beginProc
1. MP=ConstructMainPath (T, (u, v))
2. For each edge (r, w), where $w \notin MP$ and $r \in MP$ do
3.     ConstructPaths ($T_w(r, w)$, (r, w))
endProc
```

THE CORRECTNESS PROOF AND THE TIME COMPLEXITY

In this section, we prove that the algorithm is correct and runs in linear time. The following theorem shows that the labels of the edges are computed correctly.

Theorem 1

Applying the DFS algorithm twice in two different phases on a free tree $T = (V, E)$, starting from a fixed arbitrary vertex s in both phases, all the labels of the edges of T are computed correctly.

Proof

We should prove that all the edges of T are assigned two labels, one for each end-vertex. In addition, we should prove that on computing a label, all necessary information are already available.

Let (u, v) be an arbitrary edge of tree T. Without loss of generality, assume that the DFS algorithm which starts from vertex s visits vertex u before visiting vertex v, see Figure 3. The label $l_v(u, v)$ of edge (u, v) is computed when DFS returns from vertex v to vertex u, in the first phase. If v is a leaf then simply $l_v(u, v) = 1$. Otherwise, to compute $l_v(u, v)$, we need all labels $l_w(v, w)$ where $w \neq u$ and $(v, w) \in E$. The labels $l_w(v, w)$ have been already computed, because DFS returns back from v to u after returning back from all adjacent vertices w of v $(w \neq v)$. So we already have all information we need for computing $l_v(u, v)$.

The label $l_u(u, v)$ of edge (u, v) is computed when DFS goes from vertex u to vertex v, in the second phase. Note that because the starting vertex s is fixed in both runs of DFS, the direction of traversing the edges is the same in both phases. This is essential

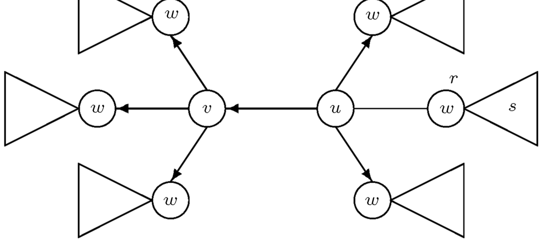

Figure 3. DFS on a free tree, Theorem 1.

for correct computation of the labels. Considering Figure 3, to compute $l_u(u,v)$, we need all the labels $l_w(u,w)$ where $w \neq v$ and $(u,w) \in E$. If $u = s$, then all the labels $l_w(u,w)$, $w \neq v$, have been already computed in the first phase. Otherwise, let r be the adjacent vertex of u that the DFS algorithm visits before visiting u. The label $l_r(r,u)$ has been already computed, because DFS goes from r to u before going from u to v. All the other labels $l_w(u,w)$, where $w \neq v$ and $w \neq r$, have been already computed in the first phase. So we already have all information we need for computing $l_u(u,v)$. Every edge is traversed twice by the DFS algorithm in each run, hence all the edges are assigned two labels, one for each end-vertex. □

In the following, we prove that our path partitioning is of minimum height. The following fact is concluded directly from the definitions.

Fact 1

For any edge (u,v) of a free tree T, label $l_v(u,v)$ depends only on sub-tree $T_v(u,v)$.

Lemma 1

For any edge (u,v) of a free tree T, the height of any minimum height path partitioning of sub-tree $T_v(u,v)$, that includes edge (u,v) in its root path, is $l_v(u,v)-1$.

Proof

The proof is, by induction on n, the number of vertices of the given free tree T. For trees of size 2, both $l_v(u,v)$ and $l_u(u,v)$ for the only edge (u,v) are one and the height of the only path partition is zero. Assuming that, the induction hypothesis is true for all trees of size less than n, we prove that it is true for any tree T of size n. For any edge (u,v) of T, if sub-tree $T_v(u,v)$ has less than n vertices, then we are done. Otherwise, suppose that $T_v(u,v)$ has n vertices, which is possible only when u is a leaf, see Figure 4.

Considering Figure 4, the root path of any minimum height path partitioning of $T_v(u,v)$ that starts

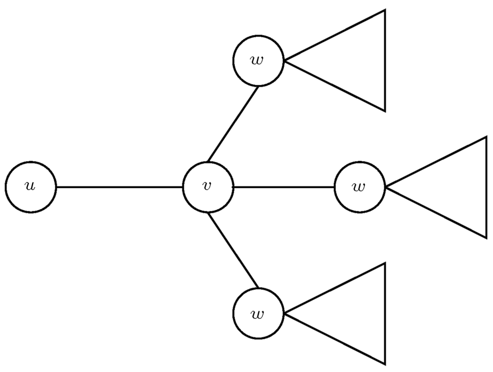

Figure 4. The sub-tree $T_v(u,v)$, Lemma 1.

from edge (u,v) continues by an edge (v,w^*) for an adjacent vertex $w^* \neq u$ of v. Now we should specify which adjacent vertex $w \neq u$ of v can be w^*. Let $P_u^{\min}(T)$ denote a path partitioning of tree T whose root path includes vertex u and has minimum height. Let $MP(P)$ denote the root path of a path partitioning P. We have $MP(P_u^{\min}(T_v(u,v))) = (u,v) \cup MP(P_v^{\min}(T_{w^*}(v,w^*)))$. One can see the height of $P_u^{\min}(T_v(u,v))$ will be minimum only when w^* is an adjacent vertex $w \neq u$ of v with $P_v^{\min}(T_w(v,w))$ of maximum height among other adjacent vertices $w \neq u$ of v. This is the optimal choice of our greedy algorithm. In this manner:

$$h(P_u^{\min}(T_v(u,v))) = \max\{h_w^1(v,w), h_w^2(v,w)+1\},$$

where $h_w^1(v,w)$ and $h_w^2(v,w)$ are, respectively, the largest and the second largest elements of $\{h_w(v,w) | (v,w) \in E, w \neq u\}$. From the definition we have:

$$l_v(u,v) = \max\{l_w^1(v,w), l_w^2(v,w)+1\}.$$

Any sub-tree $T_w(v,w)$ has less than n vertices, so:

$$h_w^1(v,w) = l_w^1(v,w) - 1,$$

and:

$$h_w^2(v,w) = l_w^2(v,w) - 1,$$

and we have

$$h_v(u,v) = l_v(u,v) - 1. \square$$

Theorem 2

The path partitioning P of a given free tree T, obtained by the proposed algorithm, is of minimum height.

Proof

By Lemma 1 for any leaf u of free tree T, label $l_v(u,v) - 1$ is the height of a minimum height path partitioning of sub-tree $T_v(u,v)$ that includes edge (u,v). Vertex u is a leaf of T, so sub-tree $T_v(u,v)$ is T itself. Because we select leaf u such that label $l_v(u,v)$ is minimum, so the computed path partitioning of T has minimum height among all possible path partitioning of T. □

Theorem 3

The time complexity of the proposed algorithm is linear in the number of vertices of the given free tree $T = (V,E)$.

Proof

Consider the pseudo-code of the partitioning algorithm which is given by Algorithm 1. In the first step, the labels of the edges of the tree are computed by applying the linear time DFS algorithm on the tree. In the second step, the leaf of minimum label is selected, which is done in linear time.

Then the recursive procedure *ConstructPaths* is called to construct the paths of the partitioning. The root path is constructed by the greedy method in time $O(\sum_{v\in mp} d(v))$, where mp is the root path and $d(v)$ is the degree of vertex v. Then the procedure is called recursively to construct the other paths of the partitioning. The time that is needed to construct each path p is bounded by $O(\sum_{v\in p} d(v))$. So the total time that is needed to construct all the paths, using the already computed labels, is $O(\sum_{v\in V} d(v)) = O(|E|) = O(|V|)$.

CONCLUSION

In this paper, we considered path partitioning of trees. In this problem, the given free tree was partitioned into edge-disjoint paths. A linear time algorithm was developed to compute a path partitioning for free trees such that the height of the partitioning was minimum.

REFERENCES

1. Johnson, D.S. and Niemi, K.A. "On knapsacks, partitions, and a new dynamic programming technique for trees", *Math. Oper. Res.*, **8**(1), pp. 1-14 (1983).
2. Wu, B.Y. et al. "On the uniform edge-partition of a tree", *Discrete Appl. Math.*, **155**, pp. 1213-1223 (2007).
3. Alam, M.J. et al. "Upward drawing of trees on the minimum number of layers", in *Proc. of the 2nd International Conference on Algorithms and Computation, Lect. Notes in Comput. Sci.*, **4921**, pp. 88-99 (2008).
4. Bagheri, A. and Razzazi, M. "Drawing free trees on 2D grids which are bounded by simple polygons", *Scientia Iranica*, **13**(4), pp. 387-394 (2006).
5. Suderman, M. "Pathwidth and layered drawing of trees", *Int. J. Comput. Geom. and App.*, **14**(3), pp. 203-225 (2004).
6. Kovacs, A. and Kis, T. "Partitioning of tress for minimizing height and cardinality", *Inform. Process. Lett.*, **89**, pp. 182-185 (2004).
7. Kundu, S. and Misra, J. "A linear tree partitioning algorithm", *SIAM J. Comput.*, **6**, pp. 151-154 (1977).
8. Hamacher, A. et al. "Tree partitioning under constraints- clustering for vehicle routing problems", *Discrete Appl. Math.*, **99**, pp. 55-59 (2000).
9. Maravalle, M. et al. "Clustering on trees", *Comput. Statist. Data Anal.*, **24**, pp. 217-234 (1997).
10. Becker, R.I. and Perl, Y. "Shifting algorithms for tree partitioning with general weighting functions", *J. Algorithms*, **4**, pp. 101-120 (1983).
11. Becker, R.I. and Perl, Y. "The shifting algorithm technique for the partitioning of trees", *Discrete Appl. Math.*, **62**, pp. 15-34 (1995).
12. Becker, R.I. et al. "A shifting algorithm for min-max tree partitioning", *J. ACM*, **29**, pp. 58-67 (1982).
13. Perl, Y. and Schach, S.R. "Max-min tree partitioning", *J. ACM*, **28**, pp. 5-15 (1981).
14. Lucertini, M. et al. "Most uniform path partitioning and its use in image processing", *Discrete Appl. Math.*, **42**, pp. 227-256 (1993).
15. Goodman, S.E. and Hedetniemi, S.T. "On the Hamiltonian completion problem", in *Proc. of the Capital Conference on Graph Theory and Combinatorics, Lecture Notes in Mathematics*, **406**, pp. 262-272 (1973).
16. Misra, J. and Tarjan, R.E. "Optimal chain partitions of trees", *Inform. Process. Lett.*, **4**, pp. 24-26 (1975).
17. Chang, G.J. and Kuo, D. "The $L(2,l)$-labeling problem on graphs", *SIAM J. Discrete Math.*, **9**(2), pp. 309-316 (1996).
18. Arikati, S.R. and Pandu Rangan, C. "Linear algorithm for optimal path cover problem on interval graphs", *Inform. Process. Lett.*, **35**(3), pp. 149-153 (1990).
19. Bonuccelli, M.A. and Bovet, D.P. "Minimum node disjoint path covering for circular arc graphs", *Inform. Process. Lett.*, **8**, pp. 159-161 (1979).
20. Srikant, R. et al. "Optimal path cover problem onblock graphs and bipartite permutation graphs", *Theoret. Comput. Sci.*, **115**, pp. 351-357 (1993).
21. Yan, J.H. "The path partition and related problems", PhD Thesis, Dept. Applied Math., National Chiao Tung Univ., Hsinchu, Taiwan (1994).
22. Yan, J.H. and Chang, G.J. "The path partition problem in block graphs", *Inform. Process. Lett.*, **52**(6), pp. 317-322 (1994).
23. Jin, Z. and Li, X. "On the k-path cover problem for cacti", *Theoret. Comput. Sci.*, **355**(3), pp. 354-363 (2006).
24. Steiner, G. "On the k-path partition of graphs", *Theoret. Comput. Sci.*, **290**(3), pp. 2147-2155 (2003).
25. Yan, J.H. et al. "k-path partitions in trees", *Discrete Appl. Math.*, **78**(1-3), pp. 227-233 (1997).
26. Atallah, M.J. et al. "A tree-covering problem arising in integrity of tree-structured data", *Inform. Process. Lett.*, **109**, pp. 79-82 (2008).

BIOGRAPHIES

Alireza Bagheri received the M.S. degree in computer engineering from Sharif University of Technology

(SUT) at Tehran and the Ph.D. degree in computer science from Amirkabir University of Technology (AUT) at Tehran. Currently he is an assistant professor in the computer engineering & IT department at Amirkabir University of Technology (AUT) at Tehran. His research interests include computational geometry, graph drawing and graph algorithms.

Mohammadreza Razzazi received the M.S. degree in computer science from Stanford University and the Ph.D. degree in computer science from the University of California at Santa Barbara. Currently he is an associate professor in the computer engineering & IT department at Amirkabir University of Technology (AUT) at Tehran. His primary areas of research are computational geometry, robotics, and computer graphics.