RC23979 (W0606-080) June 15, 2006
Computer Science

IBM Research Report

An Algorithm for Partitioning Trees Augmented with Sibling Edges

Rajesh Bordawekar, Oded Shmueli
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

An Algorithm for Partitioning Trees Augmented with Sibling Edges

Rajesh Bordawekar * Oded Shmueli

IBM T.J. Watson Research Center, Hawthorne, NY 10532

Abstract

We investigate a special case of the graph partitioning problem: the partitioning of a sibling graph which is an ordered tree augmented with edges connecting consecutive nodes that share a common parent. We present XS, a dynamic programming algorithm for partitioning an n-node sibling graph in $O(nW^3)$ time and $O(nW + W^2)$ space. We describe the algorithm and present a proof of its correctness.

Key words:
Algorithms, Graph Algorithms

1 Introduction

Graph partitioning problems arise in many applications, e.g., logical circuit design, computational fluid dynamics, and physical data clustering in databases. The graph partitioning problem is defined as follows. Given a graph with non-negative node and edge integer weights, partition the set of graph nodes into disjoint sets, termed clusters, so that the sum of the node weights in each cluster does not exceed a given bound and the sum of the weights of the edges connecting nodes in different clusters, called the cost, is minimal, or, equivalently, the sum of the weights of the edges whose endpoints are in the same cluster, called the value, is maximum [1].

We investigate a special case of the graph partitioning problem: the partitioning of a *sibling graph* which is an ordered tree augmented with sibling

* Corresponding author. Address: IBM T.J. Watson Research Center, Hawthorne, NY 10532

Email addresses: bordaw@us.ibm.com (Rajesh Bordawekar), oshmueli@us.ibm.com (Oded Shmueli).

edges, i.e., edges connecting consecutive nodes that share a common parent. This problem has gained importance due to the increasing number of applications that process XML documents using navigational languages such as XPath, XQuery, or XSLT [2]. These languages induce interactions among sibling nodes of an abstract XML tree which may be modeled by weighted sibling edges. Although the graph partitioning problem and its restricted version, partitioning a tree, have been studied extensively, the problem of partitioning a tree augmented with sibling edges has not been explored.

We present a dynamic programming algorithm, XS, for this problem. XS is inspired by Lukes' dynamic programming algorithm for partitioning trees [3]. Lukes' algorithm partitions an n-node tree in $O(nW^2)$ time and $O(nW^2)$ space. XS partitions an n-node sibling graph, i.e., a tree augmented with sibling edges, in $O(nW^3)$ time and $O(nW+W^2)$ space. If all sibling edge weights have a weight of 0, XS computes the same result as Lukes' algorithm.

Many algorithms have been proposed for partitioning trees and graphs. Kernighan and Lin present a heuristic algorithm for partitioning general graphs [4]. Kernighan presents a linear time algorithm for optimally partitioning a graph into clusters consisting of consecutive nodes (e.g., nodes model instructions in a computer program) [5]. Lukes' treatment of partitioning general graphs also uses dynamic programming [1]. The general problem of determining if there exists a tree partition with a value at least v is NP-complete in the ordinary sense (Problem ND15, acyclic partitioning) [6]. Johnson and Niemi treat this problem and present two algorithms that use: (1) an $O(nP^{*2})$ bottom-up, (2) $O(n^2P^*)$ left-right order dynamic programming, where P^* is the minimum cost [7]. [8] presents linear-time algorithms for partitioning trees with edge weights so as to minimize (maximize) the weight of the largest (smallest) cluster by removing k edges. Kundu and Misra proposed an $O(n)$ algorithm that partitions a tree into a minimum number of components each of whose total weights is less than or equal to k [9]. Unfortunately, the $O(n)$ running time of their algorithm is possible only for unit edge weights.

2 The Tree Partitioning Problem

For graph terminology we follow Harary [10]. A graph is a pair (V, E) where V is a set of *nodes* and $E \subseteq V \times V$ is a set of *undirected edges*. We use standard graph theory concepts including path, connected, cycle, acyclic graph, and subgraph. A *tree* is a connected acyclic graph. An *undirected rooted tree* is a triple (r, V, E) where (V, E) is a tree and $r \in V$ is called the *root*. An undirected rooted tree T can be transformed into a *directed rooted tree* T' by transforming each edge $\{u, v\}$ of T into a directed edge (u, v) so that it points away from the root. The *height* of a directed rooted tree T is the length of

the longest root to leaf path in T. The *descendants* of a node v in a directed rooted tree T are all nodes reachable from v via a directed path. An *ordered tree* is a 4-tuple (r, V, E, CI) such that (r, V, E) is a directed rooted tree and $CI : V \rightarrow Natural$ (*Natural* is the set of natural numbers) is a function which assigns to a node its child sequence number among the children of its parent, $CI(r) = 0$. Let $c(u)$ denote the number of children of a node u. If a node u has $k = c(u)$ children $v_1, \cdots, v_k$, then CI maps each child $v_i, 1 \leq i \leq k$, to a unique number in $\{1, \cdots, k\}$. A *subtree T' rooted at r_1* of an ordered tree $T = (r, V, E, CI)$ is the induced ordered subtree rooted at r_1; formally, it is an ordered tree (r_1, V_1, E_1, CI) such that $V_1 \subseteq V$, where V_1 consistes of r_1 and its descendants in T, $E_1 = E \bigcap (V_1 \times V_1)$ and CI is the restriction of CI to V_1. An *i-complete subtree*, $T'(r_1, i)$, of a subtree $T' = (r_1, V_1, E_1, CI)$ of an ordered tree T is obtained from T' by deleting from V_1 all nodes other than r_1, the first i children of r_1 (according to CI) and their descendants, and deleting from E_1 all edges in which at least one node was deleted. [1]

Let $T = (r, V, E, CI)$ be an ordered tree. A tuple (i, j) is a *sibling edge* of T if $i, j \in V$, there is a node $v \in V$ such that (v, i) and $(v, j) \in E$ and $CI(i) + 1 = CI(j)$. A *sibling graph* of T is a 5-tuple $S = (r, V, E \bigcup SE, CI)$ where SE is the set of all possible sibling edges of T. Clearly, a sibling graph is a directed acyclic graph that is formed by taking an ordered tree and augmenting it with all the possible sibling edges. When no confusion arises, a sibling graph $S = (r, V, E \bigcup SE, CI)$ is denoted by $S(r)$. The abovementioned concepts are illustrated in Fig. 1 (a-d).

A *sibling subgraph* of $S = (r, V, E \bigcup SE, CI)$ is a sibling graph $S' = (r_1, V_1, E_1, CI_1)$ such that $r_1 \in V_1$, V_1 is the set of nodes that includes node r_1, and its descendants, $E_1 = (E \bigcup SE) \bigcap (V_1 \times V_1)$, and CI_1 is the restriction of CI to V_1. When no confusion arises, a sibling subgraph $S' = (r_1, V_1, E_1, CI_1)$ is denoted by $S'(r_1)$; the subgraph root, r_1, is called the subgraph *pivot node*. The height h of a sibling graph (respectively, sibling subgraph) is the height of the tree (respectively, subtree) that induces it. A sibling subgraph is illustrated in Fig. 1 (e).

An *i-complete sibling subgraph*, $S'(r, i)$, of a subgraph $S' = (r_1, V_1, E_1, CI_1)$ of the sibling graph S is obtained from S' by deleting from V_1 all nodes other than r_1, the first i children of r_1 and their descendants, and deleting from E_1 all edges in which at least one node was deleted. The i-complete sibling subgraph, $S'(r_1, i)$, is said to be induced by the root r_1 and its first i children. The i-complete sibling subgraph, $S'(r_1, 0)$, is the sibling subgraph consisting of only node r_1 and no edges. An i-complete sibling subgraph, $S'(r_1, c(r_1))$, is the sibling subgraph, $S'(r_1)$. When no confusion arises, we refer to a sibling

[1] Note that including r_1 in the notation $T'(r_1, i)$ is redundant, as T' defines r_1; it is done for clarity of presentation.

graph(or subgraph) as simply a *graph* (or *subgraph*). An i-complete sibling subgraph is illustrated in Fig. 1 (f).

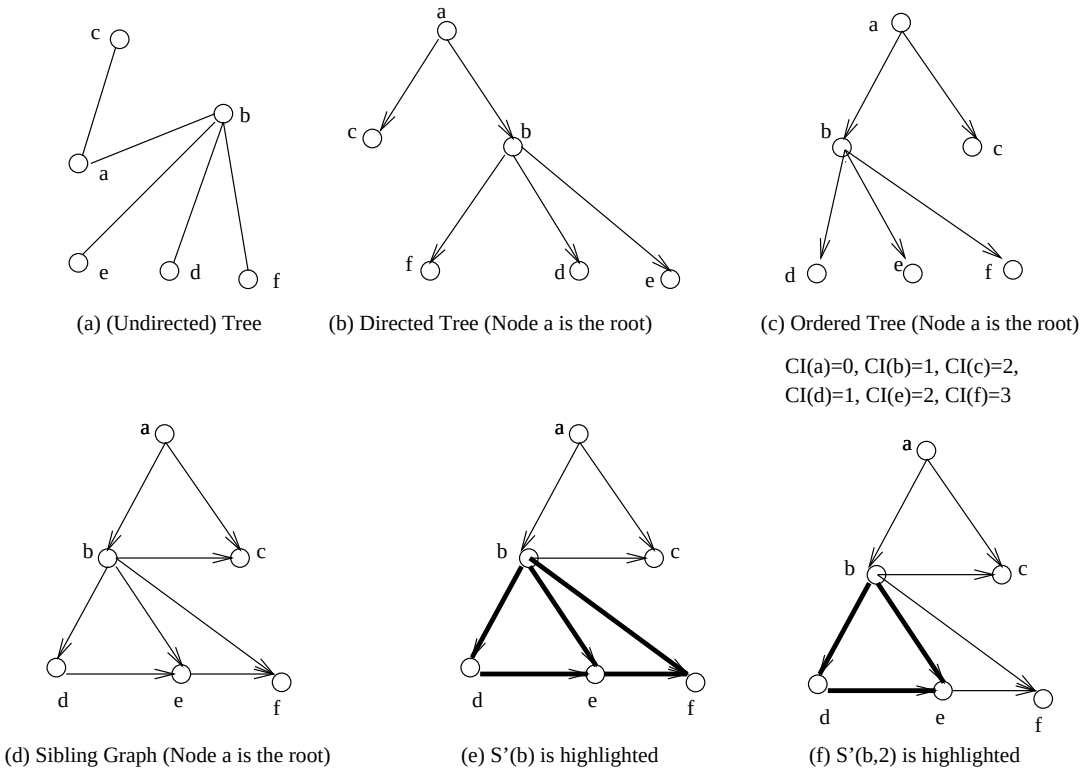

Fig. 1. Trees, directed trees, ordered trees and sibling graphs.

A *node cluster* of a sibling graph $S = (r, V, E \bigcup SE, CI)$ is a non-empty subset of V. When no confusion arises, we simply use the term *cluster*. A *partition of S*, P, is a set of pair-wise disjoint clusters of S whose union equals V, that is $P = \{c_1, \ldots, c_k\}$, $k \geq 1$, such that $\bigcup_{i=1}^{k} c_i = V$, and $c_i \bigcap c_j = \emptyset$, for $1 \leq i < j \leq k$. Each node i of S is associated with a non-negative integer *weight*, w_i. Each (parent-child or sibling) edge (i, j) of S is associated with a non-negative integer *value*, v_{ij}. The *size* of a cluster c, $size(c)$, is the sum of the weights of its nodes; formally, $size(c) = \Sigma_{m \in c}\ w_m$. The *value* of a cluster c, $value(c)$, is the sum of the values of its edges; formally, $value(c) = \Sigma_{(m,n) \in E\ \wedge m \in c\ \wedge n \in c}\ v_{mn}$. The *value* of a partition P, $value(P)$, is the sum of the values of its clusters; formally, $value(P) = \Sigma_{c \in P}\ value(c)$. The *cost* of a partition P, $cost(P)$, is the sum of the values of edges of S that connect nodes in two different clusters; formally, $cost(P) = \Sigma_{(m,n) \in E\ \wedge m \in c_i\ \wedge n \in c_j\ \wedge c_i \neq c_j, c_i \in P, c_j \in P}\ v_{mn}$.

Let W, the *cluster weight bound*, be a positive integer. The *sibling graph partitioning problem* is formulated thus. Find a maximam value partition, P_{opt}, of S such that the size of every cluster in P_{opt} does not exceed W. P_{opt} is said to be an *optimal partition* of S.[2] So, P_{opt}=$\{c_1, \ldots, c_n\}$ such that $size(c_i) \leq W$,

[2] In general, there may be more than one optimal partition.

$1 \leq i \leq n$, and $value(P_{opt}) = Max\{value(P) | P \; is \; a \; partition \; of \; S \; and \; \forall c \in P, \; size(c) \leq W\}$.

Figure 2 illustrates some of the concepts discussed in this section.

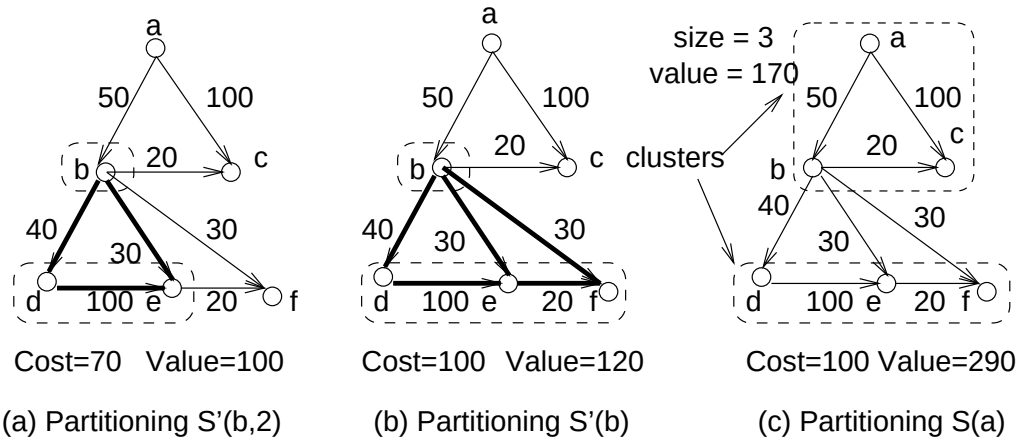

Fig. 2. Partitioning a sibling graph $S(a)$. Each graph node has unit weight and the cluster weight bound is 3. Edges weights are shown next to the edges. Clusters are illustrated via dashed ovals.

3 XS

XS is an iterative algorithm that partitions increasingly larger subgraphs until the subgraph being partitioned is the entire sibling graph. Each partitioning step applies to a sibling subgraph, say $S'(x)$. In each partitioning step, XS constructs a collection of partitions. Each such partition partitions the nodes of $S'(x)$ into a set of clusters. Each such cluster contains a sibling subgraph G of $S'(x)$; if edge directions are ignored, thereby making the underlying graph of G into an undirected graph G', then G' is connected.

A partition is usually denoted by P, possibly subscripted and superscripted. A superscript, as in P^x, identifies x, the root of a sibling subgraph whose nodes are partitioned in P^x. A partition may be classified by its *type*, which is a sequence enclosed in square brackets. The type is a predicate that is satisfied by the partition. One such predicate takes the form $[w, v]$. $P^x[w, v]$ is true if P^x has *weight* w and value v, where the *weight* of a partition P^x is the size of the cluster in P^x containing node x; furthermore, this cluster is called the *pivot cluster*. Similarly, we have a predicate of the form $[w, v, y, w_1]$. $P^x[w, v, y, w_1]$ is true if $P^x[w, v]$ is true and the cluster containing node y in P^x is of size w_1. For every partitioned subgraph $S'(x)$, XS computes a set of partitions, $\{P_1^x[w_x, v_0], \cdots, P_k^x[W, v_k]\}$, where $P_i^x[w_x + i, v_i]$, $0 \leq i \leq k$, is an *optimal* partition of $S'(x)$ of weight $w_x + i$ whose value v_i is the largest among all partitions of $S'(x)$ with weight $w_x + i$.

Consider a sibling graph rooted at a node r, $S(r)$, being partitioned by XS. First, for every node of the graph, x, XS creates an initial partition of a sibling subgraph that consists of the single node x and no edges; this partition consists of a single cluster containing node x with weight w_x and value 0. After initialization, XS chooses some node x of $S(r)$, all of whose children are roots of already partitioned sibling subgraphs. For the chosen node x, XS partitions increasingly larger i-complete sibling subgraphs, $S'(x,i)$, until the i-complete subgraph being partitioned is $S'(x)$. For partitioning $S'(x,(i+1))$, XS uses partitions computed for $S'(x,i)$ and the partitions computed for $S'(z)$, where node z is the $(i+1)^{th}$ child of node x. Once $S'(x)$ is partitioned, all children of node x are considered as deleted. XS then proceeds to partition another sibing subgraph of $S(r)$ rooted at a node all of whose children are roots of already partitioned sibling subgraphs. This process continues until the sibling subgraph to be partitioned is rooted at node r. After this subgraph is partitioned, XS completes the partitioning process. Intuitively, while partitioning a sibling graph, XS proceeds in a bottom-up manner in completing the processing of increasingly larger sibling subgraphs, and within a subgraph XS proceeds in a left-right order.

During the sibling graph partitioning process, XS repeatedly creates new partitions by using previously created partitions. A new partition can be created either by *concatenating* or by *merging* two existing partitions. Let node y be the $(i-1)^{th}$ child of node x and node z be the i^{th} child of node x. Therefore, nodes y and z are related by a sibling edge (y,z). Let $P^x[w_1,v_1]$ be a partition of $S'(x,i)$ and $P^z[w_2,v_2]$ be a partition of $S'(z)$. Note that $P^x[w_1,v_1]$ and $P^z[w_2,v_2]$ are partitions over disjoint node sets.

A partition $P^x_{concat}[w,v]$, of $S'(x)$, can be generated by a *concatenation* operation (henceforth denoted by $\oplus$) on the two partitions $P^x[w_1,v_1]$ and $P^z[w_2,v_2]$ as follows:

$P^x_{concat}[w,v] = P^x[w_1,v_1] \oplus P^z[w_2,v_2] = P^x[w_1,v_1] \bigcup P^z[w_2,v_2]$, where $v = v_1 + v_2$ and $w = w_1$.

Alternatively, two partitions $P^x[w_1,v_1]$ and $P^z[w_2,v_2]$ can be *merged* to create a new partition, $P^x_{merge}[w,v]$. The merging operation (henceforth denoted by $\otimes$ and $\odot$, as appropriate) involves creating a new cluster by merging two clusters from $P^x[w,v_1]$ and $P^z[w_2,v_2]$ containing nodes related by a parent-child ($\otimes$) or by a sibling edge ($\odot$). Let c_x denote the cluster of $P^x[w_1,v_1]$ containing node x, let c_z denote the cluster of $P^z[w_2,v_2]$ containing node z, and let c_y denote the cluster of $P^x[w_1,v_1]$ containing node y.

Two partitions, $P^x[w_1,v_1]$ and $P^z[w_2,v_2]$, can be merged along a parent-child edge (x,z) if $size(c_{xz}) = size(c_x) + size(c_z) \leq W$ as follows:

$P^x_{merge}[w,v] = P^x[w_1,v_1] \otimes P^z[w_2,v_2] = ((P^x[w_1,v_1] - \{c_x\}) \bigcup (P^z[w_2,v_2] - \{c_z\}) \bigcup \{c_{xz}\})$, where $c_{xz} = c_x \bigcup c_z$, $w = size(c_{xz})$. Note that $w_1 \leq w \leq W$, and $v \geq v_1 + v_2$.

Two partitions, $P^x[w_1,v_1]$ and $P^z[w_2,v_2]$, can be merged along a sibling edge (y,z) if $size(c_{yz}) = size(c_y) + size(c_z) \leq W$ as follows:

$P^x_{merge}[w,v] = P^x[w_1,v_1] \odot P^z[w_2,v_2] = ((P^x[w_1,v_1] - \{c_y\}) \bigcup (P^z[w_2,v_2] - \{c_z\}) \bigcup \{c_{yz}\})$, where $c_{yz} = c_y \bigcup c_z$. Note that $size(c_{yz}) \leq W$, $w_1 \leq w \leq W$, and $v \geq v_1 + v_2$.

The graphs of operand partitions for $\oplus$, $\otimes$ and $\odot$ are related via edges; therefore, if operand partitions are such that each partition is over a set of nodes that induces a connected subgraph, the resulting partition is such that it also induces a connected subgraph.

Partitions computed during the execution of XS are stored in *partition vectors* and *partition arrays*. Specifically, each node x of a sibling graph $S(r)$ maintains the following partition vector of size W:

- $\overrightarrow{P}[1,\cdots,W]$: This partition vector records the set of partitions once a sibling subgraph $S'(x)$ has been completely partitioned. An element $\overrightarrow{P}[w]$, $1 \leq w \leq W$, stores $P^x[w,v_{max}]$, a partition of $S'(x)$ that has the highest value among all the partitions of $S'(x)$ with weight w.

Let α be a special symbol which means "uninitialized". Let $S'(x,i)$ be the largest i-complete subgraph of $S'(x)$ partitioned thus far. Let y be the i^{th} child of node x. The following *temporary* array and vector are used while partitioning the subgraph rooted at x; note that vector entries may be uninitialized and the description is that of non-α entries.

- $\overrightarrow{L}[1\cdots W, 1\cdots W]$: $\overrightarrow{L}[w_1,w_2]$, $1 \leq w_1, w_2 \leq W$, stores a partition of $S'(x,i)$, $P^x[w_1,v,y,w_2]$, whose value v is maximal among all the partitions of $S'(x,i)$ with pivot cluster (containing x) weight w_1, the cluster containing node y having weight w_2, and node x and y in different clusters.
- $\overrightarrow{Q}[1\cdots W]$: $\overrightarrow{Q}[w]$, $1 \leq w \leq W$, stores a partition of $S'(x,i)$, $P^x[w,v]$, which has the highest value among the partitions of $S'(x,i)$ whose pivot cluster is of size w and it contains nodes x *and* y (and possibly other nodes).

Figure 3 details XS. XS uses a W-element vector, $\overrightarrow{INIT}$ and a $W \times W$-element array $\overrightarrow{INITA}$ all of whose entries are α. XS uses the merge ($\otimes$ and $\odot$) and concatenate ($\oplus$) operators on partitions that are stored in arrays and vectors. While partitioning a sibling graph by an n-node tree, step (b) in Figure 3 is invoked for every internal node, x, of the tree. Step (b) invokes two W^3 loops (lines 3.1–3.6 and 3.8–3.15, Figure 3). Temporary partitions created during these steps are stored in the $\overrightarrow{L}$ and $\overrightarrow{M}$ arrays and in the $\overrightarrow{Q}$ and $\overrightarrow{R}$ vectors,

by using the conditional assignment operator, $=_>$, which is defined as follows. Consider an assignment $v =_> P$, where v denotes a location containing either α or a partition, say P', and P is a partition. If $P' = \alpha$ then P replaces α as the content of v. Otherwise, if $value(P) > value(P')$ then P replaces P' as the content of v, else no change takes place. If one of the operands in a $\oplus$, $\otimes$ or $\odot$ operation is α, the result is α.

Operators $\oplus$, $\otimes$, and $\odot$ require constant time as unioned sets are always disjoint and at each point in the processing, we need to keep track of three nodes, the pivot x, the previous child y and the current child z, in partitions. On an n-node graph, XS has time complexity $O(nW^3)$. Similarly, each node in an n-node tree maintains a W-element partition vector for storing partitions of the sibling subgraph rooted at that node. In addition, two temporary partition vectors and two arrays are used while partitioning a subgraph. This leads to XS having $O(nW + W^2)$ space complexity.

4 Proof of Correctness

We observe the following characteristics of partitions.

Observation 1: Given a partition $P^x[w, v]$ with value v of an i-complete sibling subgraph, $S'(x, i)$, there is a partition $P^x_{correct}[w', v]$ with the same value v, whose every cluster induces a connected subgraph of $S(r)$ and such that $w' \leq w$.

Proof of Observation 1: Suppose there exists a partition $P^x[w, v]$ with value v of $S'(x, i)$ that includes a cluster inducing a disconnected graph. By definition, the value of a partition is the sum of the values of its clusters. The value of a cluster is the sum of the weights of its intra-cluster edges. So, any cluster c inducing a disconnected subgraph can be split into new clusters, each inducing a connected subgraph, such that the total of the values of the new clusters is the same as that of the pre-split cluster c. Hence, one can generate a new partition $P^x_{correct}[w', v]$ with the same value v as partition $P^x[w, v]$ but with clusters that induce only connected subgraphs and such that $w' \leq w$. □

For reasoning in the proof, we refer to vectors and arrays using the subscript i, e.g., $\overrightarrow{P}_i[q]$. The intention is the state of these structures after the i^{th} iteration of step (b), just before statement 3.17 (Figure 3). We use $value(\overrightarrow{P}_i[w])$ to denote the value of the partition stored at $\overrightarrow{P}_i[w]$. We call a partition *connected* if each of its clusters, where all directed edges are converted into undirected edges, induces a connected subgraph.

For the proof of correctness, we define an operator, $\ominus_i$, operating on a par-

Input: A sibling graph rooted at node r, $S(r)$ and cluster weight bound, W.
Output: An optimal partition P^r_{opt} of $S(r)$.

(a): For each node x of $S(r)$ with weight $w = w_x$, form a partition with weight w and value 0, $P^x[w,0] = \{\{x\}\}$. $P^x[w,0]$ is stored in the w^{th} location, namely $\overrightarrow{P}[w]$, of $\overrightarrow{P}$, all other locations are set to α.
(b): Choose some node x, all of whose children are roots of already partitioned sibling subgraphs. Let $n = c(x)$ be the number of children of x. Let y be the $(i-1)^{th}$ child of x. Let z be the i^{th} child of x. A partition vector $\overrightarrow{P}$ stores the set of partitions for $S'(z)$. For $S(x)$, compute optimal partitions for all possible weights w', $w_x \leq w' \leq W$ by following these steps:

1. Initialize the $W \times W$ array $\overrightarrow{L}$ and the W-element vector $\overrightarrow{Q}$ as follows:
   1.0 $\overrightarrow{L} = \overrightarrow{INITA}$; $\overrightarrow{Q} = \overrightarrow{INIT}$
   1.1 Assign the partition $P^x[w,0]$ to all column positions of row w in $\overrightarrow{L}$.
2. Let i=1.
3. while $(i \leq n)$ do
   3.0 Initialize the $W \times W$ array $\overrightarrow{M}$ and the W-element vector $\overrightarrow{R}$ as follows: $\overrightarrow{M} = \overrightarrow{INITA}$; $\overrightarrow{R} = \overrightarrow{INIT}$
   3.1 for $(j = 1; j \leq W; j++)$
   3.2 for $(k = 1; k \leq W; k++)$
   3.3 for $(m = 1; m \leq W; m++)$
   3.4 $\overrightarrow{M}[j,k] =_> \overrightarrow{L}[j,m] \oplus \overrightarrow{P}[k]$
   3.5 if $((j+k) \leq W)$
   3.6 $\overrightarrow{R}[j+k] =_> \overrightarrow{L}[j,m] \otimes \overrightarrow{P}[k]$
   3.7 if $(i > 1)$ /* a left sibling exists */
   3.8 for $(j = 1; j \leq W; j++)$
   3.9 for $(k = 1; k \leq W; k++)$
   3.10 $\overrightarrow{M}[j,k] =_> \overrightarrow{Q}[j] \oplus \overrightarrow{P}[k]$
   3.11 if $((j+k) \leq W)$
   3.12 $\overrightarrow{R}[j+k] =_> \overrightarrow{Q}[j] \otimes \overrightarrow{P}[k]$
   3.13 for $(m = 1; m \leq W; m++)$
   3.14 if $((m+k) \leq W)$
   3.15 $\overrightarrow{M}[j,m+k] =_> \overrightarrow{L}[j,m] \odot \overrightarrow{P}[k]$
   3.16 Assign $\overrightarrow{L} = \overrightarrow{M}$, $\overrightarrow{Q} = \overrightarrow{R}$
   3.17 $i = i + 1$
   3.18 Update $\overrightarrow{P}[w]$, $1 \leq w \leq W$, with a partition of highest value, chosen from $\overrightarrow{L}$ row w, and vector $\overrightarrow{Q}$, position w.

(c): Delete all n children of node x.
If node x is not r, the root of S, goto Step **(b)**. Otherwise, return the optimal partition $P^r_{opt}[w', v_{max}]$, $1 \leq w' \leq W$, which is the partition with highest value, v_{max}, among the partitions stored in $\overrightarrow{P}$, and terminate.

Fig. 3. The XS Partitioning Algorithm

tition $P^x[w,v]$ of an i-complete sibling subgraph $S'(x,i)$, $i > 0$. $\ominus_i$ splits the partition $P^x[w,v]$ into two partitions, $P_1^x[w_1,v_1]$ and $P_2^y[w_2,v_2]$, where $P_1^x[w_1,v_1]$ is a partition of the i-complete sibling subgraph $S'(x,i-1)$ (say of height h), and $P_2^y[w_2,v_2]$ is a partition of the sibling subgraph (of height $h-1$), $S'(y)$, where node y is the i^{th} child of node x. Formally, $\ominus_i(P^x[w,v]) = \{P_1^x[w_1,v_1], P_2^y[w_2,v_2]\}$. $P_1^x[w_1,v_1]$ is formed by intersecting each cluster of $P^x[w,v]$ with the nodes of $S'(x,i-1)$ and eliminating empty clusters. $P_2^y[w_2,v_2]$ is formed by intersecting each cluster of $P^x[w,v]$ with the nodes of $S'(y)$ and eliminating empty clusters.

Observation 2: Let P be a connected partition over an i-complete subgraph, $S'(x,i)$. Let P_1 and P_2 be the result of applying $\ominus_i$ to P. Then, P_1 and P_2 are connected partitions.

Proof of Observation 2: Let y be the $(i-1)^{th}$ child of x and let z be the i^{th} child of x. Suppose two nodes, u and v, in a cluster, c, become disconnected after deleting the $S'(z)$ nodes from c. Consider any shortest path p between u and v that uses only nodes in c. Path p must have used node z, and exactly once. But then, p must have passed through x and y, which therefore belong to c. However, (x,y) is an edge in $S'(x,i-1)$, so u and v are connected following the deletion, a contradiction.

Now suppose two nodes, u and v, in a cluster, c, become disconnected after deleting the $S'(x,i-1)$ nodes from c. Consider any shortest path p between u and v that uses only nodes in c. Since following the deletion u and v are not connected, path p must have used nodes in $S'(x,i-1)$. But then, p must have used node z. For p to connect u and v and use nodes in $S'(x,i-1)$, p must have used z more than once. So, p can be short-cut, contradicting it being a shortest path between u and v using nodes in c. □

The correctness of XS follows from the following Theorem.

Theorem 1: For every node x of a sibling graph $S(r)$, when XS completes processing $S'(x)$, for all q, $0 < q \leq W$, the partition stored at $\overrightarrow{P}[q]$, is a highest value connected partition of $S'(x)$ in which the cluster containing node x has weight q, if such a partition exists, and α otherwise.

Corollary: XS returns a highest value partition of $S(r)$.

Proof of Corollary: Suppose XS does not return the highest value overall partition and there exists a higher value partition P having value v. By Observation 1, there exists a connected partition with value v of, say, weight w. By the Theorem, XS computes a highest value connected partition $P^r[w,v']$, $v' \geq v$, and stores it in $\overrightarrow{P}[w]$. In calculating the return partition, XS examines $P^r[w,v']$, contradicting returning a partition whose value is less than v. □

Proof of Theorem 1:

The proof is by induction on the height, h, of the sibling subgraph, $S'(x)$.

Induction Hypothesis (A): When XS completes processing $S'(x)$, of height h, the partition stored at $\overrightarrow{P}[q]$ is a highest value partition of $S'(x)$ in which the cluster containing node x has weight q, if such a partition exists, and α otherwise.

Basis: (h=0.) In this case, node x is a leaf. The initialization step of XS produces the optimal partition for leaf x of value 0 and weight $w = w_x$ and stores it in $\overrightarrow{P}[w]$.

Induction Step: Suppose the hypothesis, (A), holds for all sibling subgraphs of height $h-1$. We prove that it holds for sibling subgraphs of height h. Consider a node x of height $h > 0$ having $k = c(x)$ children, $y_1, \cdots, y_k$. We claim that:

Claim: Following iteration i in step (b):3 of XS (Figure 3), for $1 \leq q \leq W$, the following holds:

(1) $\overrightarrow{L}[q_1, q_2]$ stores a highest value partition for $S'(x, i)$ among all connected partitions of $S'(x, i)$ having weight q_1 and in which the pivot cluster does not contain y_i, the i^{th} child node of x, and the cluster containing y_i has weight q_2.
(2) $\overrightarrow{Q}[q]$ stores a highest value partition for $S'(x, i)$ among all connected partitions of $S'(x, i)$ having weight q and in which the pivot cluster contains y_i, the i^{th} child node of x.
(3) $\overrightarrow{P}_i[q]$ stores a highest value partition for $S'(x, i)$ among all connected partitions having weight q.

The Claim establishes the Theorem's induction step as $\overrightarrow{P}_k = \overrightarrow{P}$. We prove the claim by induction on i, the iteration number in step (b):3.

(Claim) Induction hypothesis (B): The Claim holds when XS completes the processing of an i-complete sibling subgraph $S'(x, i)$.

(Claim) Basis: ($i = 1$.) After the first iteration in step (b:)3, XS has partitioned the i-complete sibling subgraph, $S'(x, 1)$. Let y be the first child of x.

We need to show that (1)-(3) hold.

(1) The content of $\overrightarrow{L}$ is determined by assignments made to $\overrightarrow{M}$ in step (b): 3.4 of XS. Suppose for some q_1, q_2, $1 \leq q_1, q_2 \leq W$, the final result stored

in $\overrightarrow{M}[q_1, q_2]$ (and $\overrightarrow{L}[q_1, q_2]$), $P^x_{final}[q_1, v, y, q_2]$, is not with the highest possible value among the connected partitions in which the pivot cluster contains x but does not contain y, and there exists a connected partition $P^x_{better}[q_1, v', y, q_2]$ among these partitions such that $v' > v$. Note that $q_1 = w_x$ as x must be in a cluster by itself. As x is in a cluster, say c, by itself in $P^x_{better}[q_1, v', y, q_2]$, if we delete this cluster c we are left with a connected partition $P^y_{better}[q_2, v']$ of $S'(y)$ such that $v' > v$. This implies that the highest value partition for $S'(y)$ among those with weight q_2 has value greater than or equal to v'.

By induction hypothesis (A), the value of the partition stored in the vector $\overrightarrow{P}[q_2]$ that is maintained at node y, $P^y_{computed}[a', v'']$, and which is used in step (b): 3.4, is such that $v'' \geq v'$. But then, in step (b): 3.4, XS must have generated a partition with weight w_x and value $v'' \geq v' > v$ in which x and y are in different clusters. This contradicts the final stored partition at $\overrightarrow{M}[q_1, q_2]$ having value v.

(2) The content of $\overrightarrow{Q}$ is determined by assignments made to $\overrightarrow{R}$ in step (b): 3.6 of XS. Suppose for some w, $1 \leq w \leq W$, the final result stored in $\overrightarrow{R}[w]$, $P^x_{final}[w, v]$, is not with the highest possible value among the partitions in which the pivot cluster contains both x and y and the pivot cluster having size w, and there exists a partition $P^x_{better}[w, v']$, among these partitions, such that $v' > v$. Split $P^x_{better}[w, v']$ using the operator $\ominus_1$ into two partitions, $P^x_{1,better}[w'_1, v'_1]$ and $P^y_{2,better}[w'_2, v'_2]$, where $w = w'_1 + w'_2$, and $v' = v'_1 + v'_2 + v_{xy}$.

XS computes two highest value connected partitions, $P^x_{xs}[w'_1, v_1]$ (initialization for x, $v_1 = 0$, $w' = w_x$), and $P^y_{xs}[w'_2, v_2]$ (by induction hypothesis (A)), for the subgraphs $S'(x, 0)$ and $S'(y)$, respectively. XS merges partitions $P^x_{xs}[w'_1, v_1]$ (stored, for example, at $\overrightarrow{L}[w'_1, 1]$) and $P^y_{xs}[w'_2, v_2]$ (stored at $\overrightarrow{P}[w'_2]$) in step (b:) 3.6, and generates a partition, $P^x_{g,xs}[w, v_{opt}]$, for $S'(x, 1)$, where $w = w'_1 + w'_2$ and $v_{opt} = v_1 + v_2 + v_{xy}$. Note that the pivot cluster of partition $P^x_{g,xs}[w, v_{opt}]$ contains both nodes x and y. Since $v_1 \geq v'_1$, $v_2 \geq v'_2$ and $v' = v'_1 + v'_2 + v_{xy}$, we have $v_{opt} \geq v'$. XS would have stored $P^x_{g,xs}[w, v_{opt}]$ at $\overrightarrow{P}_1[w]$ if $v_{opt} > v$, where v is the value of $P^x_{computed}[w, v]$, the partition stored at $\overrightarrow{P}_1[w]$. Hence, $v \geq v_{opt}$. Since $v_{opt} \geq v'$, our initial assumption $v' > v$ is contradicted.

(3) In any connected partition of weight q for $S'(x, i)$ either x and y are in the same cluster or in two different clusters. In the first case, by (2), such a partition with the highest value is in $\overrightarrow{Q}[q]$. In the second case, by (1), it is in $\overrightarrow{L}[q, q_1]$, for some q, such that $1 \leq q \leq W$. By definition of $\overrightarrow{P}_i$, in both cases $\overrightarrow{P}_i[q]$ will contain a highest possible value partition for $S'(x, i)$.

(Claim) Induction Step: Suppose hypothesis (B) holds for iteration $i - 1$ in step (b):3. We prove that (1)-(3) hold for iteration i in step (b):3.

(1) Suppose that following iteration i, for some w, $1 \leq q_1, q_2 \leq W$, $\overrightarrow{L}[q_1, q_2]$

does not store the highest value connected partition for $S'(x,i)$ among all connected partitions of $S'(x,i)$ having weight q_1 and in which the pivot cluster does not contain $y_i = z$, the i^{th} child node of x, which is in a cluster of weight q_2. We will derive a contradiction. Let $P^x_{computed}[q_1, v, z, q_2]$ be the partition computed by XS and recorded at $\overrightarrow{L}[q_1, q_2]$ (and $\overrightarrow{M}[q_1, q_2]$) after step (b):3.16. Let $P^x_{better}[q_1, v', z, q_2]$ be a connected partition of $S'(x,i)$ with value $v' > v$. Partition $P^x_{better}[q_1, v', z, q_2]$ has one of the following configurations:

a. (x-y-z): Nodes x, y, and z are in three different clusters of $P^x_{better}[q_1, v', z, q_2]$.
b. (x-yz): The pivot cluster of $P^x_{better}[q_1, v', z, q_2]$ contains node x, nodes y and z are together in a different cluster.
c. (xy-z): The pivot cluster of $P^x_{better}[q_1, v', z, q_2]$ contains nodes x and y, node z is in a separate cluster.

Split partition $P^x_{better}[q_1, v', z, q_2]$ using the operator $\ominus_i$ into two partitions: $P^x_{1,better}[w_1, v'_1, y, w_3]$ of $S'(x, i-1)$, and $P^z_{2,better}[w_2, v'_2]$ of $S'(z)$ where $w_1 \leq q_1$ and $w_2 \leq w$.

We consider configurations (a)-(c).

a. Consider configuration (a) where nodes x, y, and z are in different clusters of $P^x_{better}[q_1, v', z, q_2]$. Since the node set in each cluster induces a connected subgraph, the partition of $S'(x, i-1)$ generated by $\ominus_i$, $P^x_{1,better}[w_1, v'_1, y, w_3]$, has weight $w_1 = q_1$ and contains nodes x and y in different clusters. The other partition, $P^z_{2,better}[w_2, v'_2]$, of $S'(z)$, has value v'_2, such that $v' = v'_1 + v'_2$. Since clusters induce connected subgraphs, $w_2 = q_2$.

   By induction hypothesis (B)(1), XS generates a highest value connected partition for $S'(x, i-1)$, $P^x_{xs}[q_1, v_1, y, w_3]$, where nodes x and y, the $(i-1)^{th}$ child, are in different clusters and stores it at $\overrightarrow{L}_{i-1}[q_1, w_3]$. By induction hypothesis (A), XS generates a highest value connected partition, $P^z_{xs}[q_2, v_2]$, for $S'(z)$ and stores it in $\overrightarrow{P}[q_2]$ at node z. By using the concatenate operation (step (b): 3.4) on the partition $P^x_{xs}[q_1, v_1, y, w_3]$ and the partition $P^z_{xs}[q_2, v_2]$, XS generates a partition, $P^x_{xs}[q_1, v_{opt}, z, q_2]$, of $S'(x,i)$, such that nodes x, y, and z are in different clusters. This partition would have been stored in $\overrightarrow{M}[q_1, q_2]$ if $v < v_{opt}$, where v is the value of $P^x_{computed}[q_1, v]$, the partition currently stored at $\overrightarrow{L}_i[q_1, q_2]$. Hence $v \geq v_{opt}$. By induction hypothesis (B)(1), $v_1 \geq v'_1$. By induction hypothesis (A), $v_2 \geq v'_2$. So, $v \geq v_{opt} = v_1 + v_2 \geq v'_1 + v'_2 = v' > v$ and we have a contradiction.
b. Consider configuration (b) in which $P^x_{better}[q_1, v', z, q_2]$ has nodes y and z in a common cluster and node x in a different cluster. $P^x_{better}[q_1, v', z, q_2]$ contains node x in the pivot cluster of weight q_1 and nodes y and z in a separate cluster of weight q_2. $P^x_{1,better}[w_1, v'_1, y, w_3]$ of $S'(x, i-1)$ is such that $w_1 = q_1$ and $q_2 = w_3 + w_2$ where w_2 is the weight of $P^z_{2,better}[w_2, v'_2]$. By Observation 2, the results of $\ominus_i$ are connected partitions.

By induction hypothesis (B)(1), XS generates an optimal partition, $P^x_{xs}[w_1, v_1, y, w_3]$, for $S'(x, i-1)$, with nodes x in the pivot cluster of weight w_1 and node y in a separate cluster of weight w_3. This partition is stored at $\overrightarrow{L}_{i-1}[w_1, w_3]$. Similarly, by induction hypothsis (A), XS generates an optimal partition, $P^z_{xs}[w_2, v_2]$, of $S'(z)$, containing node z in the pivot cluster of weight w_2 and stores it at $\overrightarrow{P}[w_2]$ at node z. By merging these partitions over the sibling edge (y, z) in step (b):3.15, XS generates a partition of $S'(x, i)$ of weight w_1, $P^x_{xs}[w_1, v_{opt}, z, w_2 + w_3]$, in which nodes x and z are in different clusters and nodes y and z are in the same cluster.

This partition would have been stored in $\overrightarrow{M}_i[w_1, q_2]$ if $v < v_{opt}$, where v is the value of $P^x_{computed}[w_1, v, z, q_2]$, the partition currently stored at $\overrightarrow{L}_i[w_1, q_2]$. Hence, $v \geq v_{opt}$. By induction hypothesis (B)(1), $v_1 \geq v'_1$. By induction hypothesis (A), $v_2 \geq v'_2$. So, $v \geq v_{opt} = v_1 + v_2 \geq v'_1 + v'_2 = v'$ and we have a contradiction.

c. Consider configuration (c) in which $P^x_{better}[q_1, v', z, q_2]$ contains nodes x and y in one cluster, c, and node z is in another cluster. $P^x_{1,better}[w_1, v'_1, y, w_3]$ is such that $w_1 = w_3$ as x and y are in the same cluster, say c_1. Furthermore, due to node connectivity in cluster c, the clusters c and c_1 are identical. $P^z_{2,better}[w_2, v'_2]$ contains node z in the pivot cluster of weight w_2 such that $v' = v'_1 + v'_2$. By Observation 2, the results of $\ominus_i$ are connected partitions.

By induction hypothesis (B)(2), XS generates an optimal partition, $P^x_{xs}[w_1, v_1, y, w_1]$, of $S'(x, i)$, with nodes x and y in the same cluster and stores it at $\overrightarrow{Q}_{i-1}[w_1]$. By induction hypothesis (A), XS generates an optimal partition, $P^z_{xs}[w_2, v_2]$, $v_2 \geq v'_2$ of $S'(z)$ and stores it in $\overrightarrow{P}[w_2]$. By using the concatenate operations (step (b):3.10) on the partition $P^x_{xs}[w_1, v_1, y, w_1]$ and the partition $P^z_{xs}[w_2, v_2]$, XS generates a partition whose value, $v_{opt} = v_1 + v_2 \geq v'_1 + v'_2 = v' > v$, is at least v' in which x and y are in a cluster of weight w_1 and z is in a cluster of weight w_2. This partition would have been stored in $\overrightarrow{M}_i[w_1, w_2]$ (and $\overrightarrow{L}_i[w_1, w_2]$) since $v' > v$. But this contradicts a partition of weight v being stored at $\overrightarrow{M}_i[w_1, w_2]$.

(2) Suppose that following iteration i, for some w, $1 \leq w \leq W$, $\overrightarrow{Q}[w]$ does not store a highest value connected partition for $S'(x, i)$ among all connected partitions of $S'(x, i)$ having weight w and in which the pivot cluster contains $y_i = z$, the i^{th} child node of x. We will derive a contradiction. Let $P^x_{computed}[w, v, z, w]$ be the partition computed by XS and recorded at $\overrightarrow{Q}[w]$ (and $\overrightarrow{R}[w]$) after step (b):3.16. Let $P^x_{better}[w, v', z, w]$ be a connected partition of $S'(x, i)$ with weight w and value $v' > v$ in which the pivot cluster contains both x and z.

Partition $P^x_{better}[w, v', z, w]$ has one of the following configurations:

a. (xyz): Nodes x, y, and z are in one cluster of $P^x_{better}[w, v', z, w]$.

b. (xz-y): The pivot cluster of $P^x_{better}[w, v', z, w]$ contains nodes x and z, and node y is in a different cluster.

Split partition $P^x_{better}[w, v', z, w]$ using the operator $\ominus_i$ into two partitions: $P^x_{1,better}[w_1, v'_1, y, w_3]$ of $S'(x, i-1)$, and $P^z_{2,better}[w_2, v'_2]$ of $S'(z)$ where $w_1 \leq q_1$ and $w_2 \leq w$.

We consider configurations(a) and (b).

a. Consider configuration (a). Nodes x, y, and z are in one cluster of $P^x_{better}[w, v', z, w]$. $P^x_{1,better}[w_1, v'_1, y, w_3]$ is such that $w_1 = w_3$. $P^z_{2,better}[w_2, v'_2]$ is such that $w = w_1 + w_2$, $v' = v'_1 + v'_2 + v_{xz} + v_{yz}$. By Observation 2, the results of $\ominus_i$ are connected partitions.

   By induction hypothesis (B)(2), XS generates an optimal partition for $S'(x, i-1)$, $P^x_{xs}[w_1, v_1, y, w_1]$, where nodes x and y, the $(i-1)^{th}$ child, are in the same cluster and stores it at $\overrightarrow{Q}_{i-1}[w_1]$. By induction hypothesis (A), XS generates an optimal partition, $P^z_{xs}[w_2, v_2]$, for $S'(z)$ and stores it in $\overrightarrow{P}[w_2]$ at node z. By using the merge operation (step (b): 3.12) on the partition $P^x_{xs}[w_1, v_1, y, w_1]$ and the partition $P^z_{xs}[w_2, v_2]$, XS generates a partition, $P^x_{xs}[w, v_{opt}, z, w]$, of $S'(x, i)$, such that nodes x, y, and z are in the same cluster; $v_{opt} = v_1 + v_2 + v_{xz} + v_{yz}$.

   This partition would have been stored in $\overrightarrow{R}_i[w]$ if $v < v_{opt}$, where v is the value of $P^x_{computed}[w, v]$, the partition currently stored at $\overrightarrow{Q}_i[w]$. Hence $v \geq v_{opt}$. By induction hypothesis (B)(1), $v_1 \geq v'_1$. By induction hypothesis (A), $v_2 \geq v'_2$. So, $v \geq v_{opt} = v_1 + v_2 + v_{xz} + v_{yz} \geq v'_1 + v'_2 + v_{xz} + v_{yz} = v' > v$ and we have a contradiction.

b. Consider configuration (b). The pivot cluster, say c, of $P^x_{better}[w, v', z, w]$ contains nodes x and z, and node y is in a separate cluster, say c_1. $P^z_{2,better}[w_2, v'_2]$ contains node z in the pivot cluster of weight w_2 such that $v' = v'_1 + v'_2 + v_{xz}$. $P^x_{1,better}[w_1, v'_1, y, w_3]$ is such that $w = w_1 + w_2$. By Observation 2, the results of $\ominus_i$ are connected partitions.

   By induction hypothesis (B)(1), XS generates an optimal partition, $P^x_{xs}[w_1, v_1, y, w_3]$, of $S'(x, i-1)$, with nodes x and y in different clusters and stores it at $\overrightarrow{L}_{i-1}[w_1, w_3]$. By induction hypothesis (A), XS generates an optimal partition, $P^z_{xs}[w_2, v_2]$, $v_2 \geq v'_2$ of $S'(z)$ and stores it in $\overrightarrow{P}[w_2]$ at node z. By using the merge operations (step (b):3.6) on partition $P^x_{xs}[w_1, v_1, y, w_3]$ and partition $P^z_{xs}[w_2, v_2]$, XS generates a partition whose value, $v_{opt} = v_1 + v_2 + v_{xz} \geq v'_1 + v'_2 + v_{xz} = v' > v$, is at least v' in which x and z are in a cluster of weight w and y is in a cluster of weight w_3. This partition would have been stored in $\overrightarrow{R}_i[w]$ (and $\overrightarrow{Q}_i[w]$) since $v' > v$. But this contradicts a partition of weight v being stored at $\overrightarrow{Q}_i[w]$.

(3) The argument here is similar to that in the basis.

This concludes the proof of the Claim and the Theorem. □

5 Conclusions and Open Problems

We presented XS, a dynamic programming algorithm for partitioning an n-node sibling graph in $O(nW^3)$ time and $O(nW + W^2)$ space. The complexity of sibling graph partitioning is an open problem. In particular, the question of whether there exists an $O(nW^2)$ time algorithm for this problem remains open.

Acknowledgements

We thank Mukund Raghavachari for his helpful comments on earlier drafts of this paper.

References

[1] J. A. Lukes, Combinatiorial solution for the partitioning of general graphs, IBM Journal of Research and Development 19 (2) (1975) 170–180.

[2] W. W. W. Consortium, W3C architecture domain, www.w3c.org/xml, Online Documents (2003).

[3] J. A. Lukes, Efficient algorithm for the partitioning of trees, IBM Journal of Research and Development 13 (2) (1974) 163–178.

[4] B. Kernighan, S. Lin, An efficient heuristic procedure for partitioning graphs, Bell Systems Technical Journal 49 (2) (1970) 291–307.

[5] B. W. Kernighan, Optimal sequential partitions of graphs, J. Assoc. Comp. Mach. 18 (1) (1971) 34–40.

[6] M. S. Garey, D. S. Johnson, Computers and Intractability, W. H. Freeman and Co., 1979.

[7] D. S. Johnson, K. A. Niemi, On knapsacks, partitions, and a new dynamic programming technique for trees, Mathematics of Operations Research 8 (1).

[8] G. N. Frederickson, Optimal algorithms for tree partitioning, in: Proceedings of the Second Annual ACM-SIAM Symposium on Discrete Algorithms, 1991, pp. 168–177.

[9] S. Kundu, J. Misra, A linear tree partitioning algorithm, SIAM Journal of Computing 6 (1) (1977) 151–154.

[10] F. Harary, Graph Theory, Addison-Wesley, Reading, MA, 1969.