AKCE J. Graphs. Combin., 2, No. 1 (2005), pp. 45-56

On Simply Structured Bases of Tree Kernels

J. W. Sander and T. Sander
Institut für Mathematik
Technische Universität Clausthal
D-38678 Clausthal-Zellerfeld, Germany
E-mail: {juergen | torsten}.sander@math.tu-clausthal.de

Communicated by: Matthias Kriesell
Received 11 December 2004; accepted 1 June 2005

Abstract

It is shown that for every tree there exists a basis of the kernel that consists only of vectors with entries from $\{0, 1, -1\}$. An algorithm is presented that constructs such a basis.

Keywords: Tree, Kernel, Null space, Basis, Gaussian elimination.

2000 Mathematics Subject Classification: Primary 05C05, Secondary 15A03

1. Introduction

An interesting topic in algebraic graph theory is to find a simple basis for a considered graph eigenspace. For example, it can be shown [6] that both for the kernel and the eigenspace of eigenvalue -1 of a graph without induced path P_4 it is possible to construct a basis whose vectors contain only entries from the set $\{-1, 0, 1\}$. In [7] the existence of a similarly structured basis is proven for the kernel of a simply singular tree. However, it is noted that the proof technique used inevitably fails for multiply singular trees.

In the present work, we will use an alternate approach that works for arbitrary singular trees. Based on the observation that straight application of the Gauss algorithm to the adjacency matrix of a tree often yields a kernel basis of the intended structure, we seek to develop a procedure that guarantees the construction of such a basis. The key is a possibly incomplete Gaussian elimination scheme whose pivoting strategy only considers unit vector rows of the coefficient matrix. Note that for an adjacency matrix this only turns some one entries into zero entries.

For a formal analysis we want to avoid excessive use of matrix indices and therefore seek a presentation of the algorithm that carries out the elimination operations directly on a directed graph. If an undirected graph is given, then this approach uses the directed graph with the same adjacency matrix as the given undirected graph. Therefore it is clear that the described elimination procedure only removes edges from the digraph.

It is even shown that for a tree with n nodes a bit field of length n for each of the constructed kernel basis vectors is sufficient for storage in computer memory.

We would like to point out some loosely related work. Results on the kernel size of bipartite graphs in general can be found in [1]. In [5] an algorithm for the calculation of eigenspace bases of trees is presented. However, only part of this algorithm is carried out directly on the given tree. To obtain the final eigenspace basis one needs to calculate the kernel of an auxiliary matrix. Questions regarding basis structure are not covered. Making the transition from the given tree to its line graph we can find interesting results in [8] and [3]. For example, the order of eigenvalue zero is either 0 or 1 for the line graph of a tree. The singular line graphs of trees can even be partitioned in to two classes, depending on whether a node of the line graph can be deleted such that the remaining graph has either eigenvalue 0 or -1 multiple times. Further, results on the kernel of the incidence matrix – as opposed to the adjacency matrix – can be found in [4].

Throughout, we assume that graphs are finite, loopless and simple. Unless stated otherwise, all graphs are undirected.

2. The FOX Algorithm

Let $G = (V, E)$ be a connected graph. Construct the corresponding bidirectional orientation $\widehat{G}$, i.e. the unique digraph that has the same adjacency matrix as G.

Based on the intentions of the introduction chapter, we will introduce an algorithm that takes the graph $\widehat{G}$ as input and produces a labeled subgraph $\widetilde{D}$ of the input graph. Each node will be tagged with at least one of the three labels `F`, `O` and `X`.

Algorithm 1. (FOX Algorithm)

```
(1) Let all nodes of $\widehat{G}$ be untagged.
(2) Find a node $v$ without X tag that has exactly one outgoing edge $e$. If no such node exists,
    go to step (6).
(3) Tag node $v$ with X.
(4) Let $w$ be the unique node with $e = vw$. Except for $e$, remove all incoming edges of $w$.
(5) Tag node $w$ with O and go to step (2).
(6) Tag all untagged nodes with F.
```

Since for each run of the main loop from step (2) to (5) another node becomes `X` tagged, it is clear that the algorithm stops after at most $|V|$ iterations of the main loop.

Let $H_{\{\text{labels}\}}$ be the subgraph of $\widetilde{D}$ induced by all nodes that have each been tagged with all of the labels mentioned in the subscript.

We will say that v is an `O` tagged node if among its tags there is an `O` label. Conversely, we require an $\overline{\texttt{O}}$ tagged node to have no `O` labeled tag.

It is obvious that every node of $\widetilde{D}$ has been tagged.

An example is shown in figure 1, where the original graph G is a tree. The final graph $\widetilde{D}$ shown in the figure is obtained for the step (2) node sequence $0, 2, 4, 5, 13, 14, 17, 20, 22, 21$.

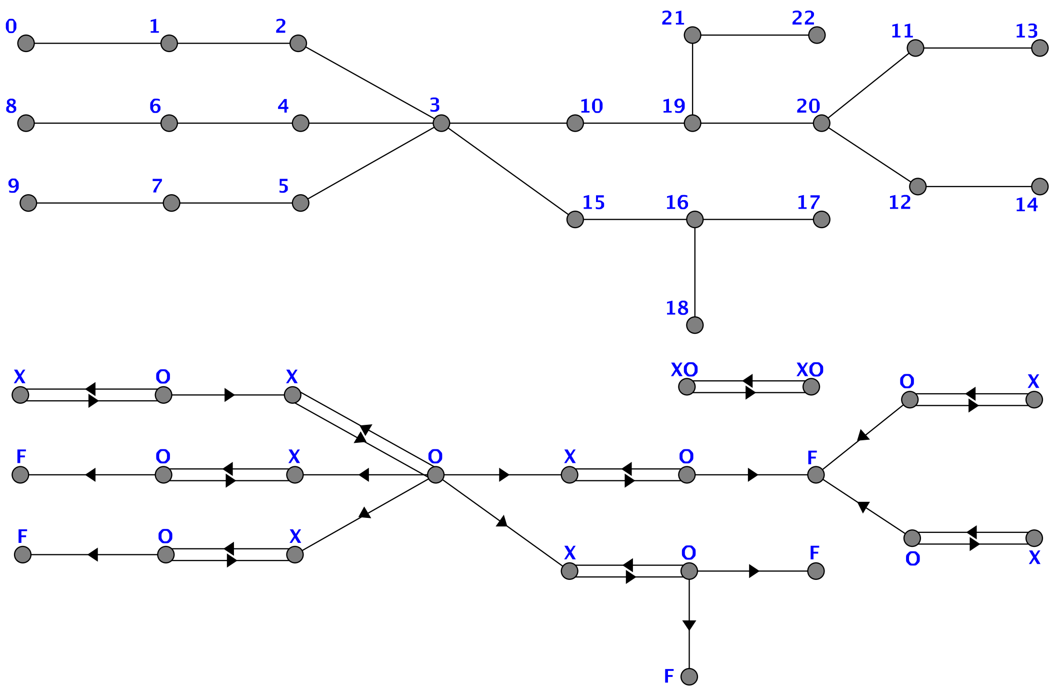

Figure 1: Example graphs G and $\widetilde{D}$

We will now look for further properties of $\widetilde{D}$. The construction of `X` and `O` tagged nodes immediately implies

Lemma 2. *Every* `X` *tagged node of* $\widetilde{D}$ *has exactly one outgoing neighbor, namely an* `O` *tagged node. Every* `O` *tagged node of* $\widetilde{D}$ *has exactly one incoming neighbor, namely an* `X` *tagged node. Every node has at least one incoming neighbor.*

Lemma 3. *Every weak component of* $H_{\mathtt{XO}}$ *is a 2-cycle and also a weak component of* $\widetilde{D}$.

Proof. Let v_1 be an `XO` tagged node. By Lemma 2 we have exactly one outgoing edge $e = v_1v_0$. The neighbor v_0 is `O` tagged. Now suppose that $\widetilde{D}$ does not contain the edge v_0v_1. Then, by Lemma 2, for v_1 there exists an incoming edge v_2v_1 from an `X` tagged node v_2. Since v_2v_1 is an edge of $\widetilde{D}$ but not v_0v_1 we see that $v_2 \neq v_0$. But v_2 misses v_1v_2 as incoming edge since v_1v_0 is the only outgoing edge of the `X` tagged node v_1. Consequently, v_2 must be an `O` tagged node.

Repeat this reasoning for v_2 instead of v_1 and so on. We obtain a sequence of distinct `XO` tagged nodes $v_1, v_2, \ldots$ with v_i adjacent to v_{i-1} for $i \geq 1$. Naturally, this sequence is finite. Note that the final node v_n may only be `O` tagged by v_0 since this is the only node left without an `X` tag towards the end of constructing the node sequence.

As a consequence, there exists a cycle of `XO` tagged nodes in $\widetilde{D}$. But in $\widehat{G}$ the nodes of this cycle originally each had at least two outgoing neighbors. Therefore, one of the `O` tagged nodes of the cycle must originate from an exterior `X` tagged node. But, by construction, each node of the cycle has been `O` tagged exactly by one of its neighbors within the cycle, a contradiction.

Therefore, there exists an edge v_0v_1 in $\widetilde{D}$. By Lemma 2 we see that v_0 and v_1 form a 2-cycle in $\widetilde{D}$. □

Lemma 4. *Let v_1 be an $\mathtt{F}$ tagged node of $\widetilde{D}$ and v_1v_2 be an outgoing edge of v_1. Then v_2 is an $\mathtt{F}$ tagged node as well and v_2v_1 is also an edge of $\widetilde{D}$.*

Proof. Let $e = v_1v_2$. Then v_2 cannot be an $\mathtt{O}$ tagged node since otherwise the edge e would have been removed in $\widetilde{D}$. Also, v_2v_1 is an edge in $\widetilde{D}$ because v_1 is not an $\mathtt{O}$ tagged node and therefore may not miss any of its original incoming edges. Consequently, v_1 and v_2 are mutually adjacent in $\widetilde{D}$.

Clearly, v_2 cannot be $\mathtt{X}$ tagged because its only outgoing neighbor v_1 would have to be an $\mathtt{O}$ tagged node. Therefore, v_2 must be $\mathtt{F}$ tagged. □

Lemma 5. *Let v be an $\mathtt{F}$ tagged node and let H be the strong component of $\widetilde{D}$ that contains v. Then all adjacencies between nodes of H are mutual. H contains only $\mathtt{F}$ tagged nodes. Further, $|H| = 1$ or $|H| \geq 3$. In the second case H contains a bidirectional cycle with at least 3 nodes.*

Incoming neighbors of H can only be $\overline{\mathtt{X}}\mathtt{O}$ tagged nodes. H has no outgoing neighbors.

Proof. Let w be a node of H, $w \neq v$. Then there exists a directed path from v to w. By Lemma 4 this path contains only $\mathtt{F}$ tagged nodes. Therefore, the entire strong component H contains only $\mathtt{F}$ tagged nodes, in particular w. By Lemma 4 it also follows that all adjacencies of nodes of H are mutual.

It is clear that H cannot have outgoing edges since by Lemma 4 these would only lead to $\mathtt{F}$ tagged nodes which cannot be weakly connected to H. Conversely, incoming edges cannot start from $\mathtt{F}$ tagged nodes but neither from $\mathtt{X}$ tagged nodes since that would require H to contain an $\mathtt{O}$ tagged node. The only remaining alternative is an $\overline{\mathtt{X}}\mathtt{O}$ tagged node.

Assume $|H| = 2$. Then each node would only have exactly one outgoing edge and should have been $\mathtt{X}$ tagged, a contradiction.

Now let $|H| \geq 3$. Then H contains two mutually adjacent nodes v_0, v_1. Since v_1 is not an $\mathtt{X}$ tagged node it must have another outgoing edge v_1v_2 besides v_1v_0. Note that $v_2 \neq v_1$ must be an $\mathtt{F}$ tagged node. Repeat this conclusion for v_2 and so on until a maximal sequence of distinct nodes $v_0, v_1, \ldots, v_n$ has been constructed. Since H is finite we see that v_n must be adjacent to one of the nodes $v_0, \ldots, v_{n-2}$. Thus, H contains a bidirectional cycle with at least 3 nodes. □

Corollary 6. *Let G be a tree. Then $H_{\mathtt{F}}$ contains only isolated points.*

Lemma 7. *Assume that $H_{\mathtt{F}}$ contains only isolated points.*

Then the $\mathtt{F}$ tagged nodes of $\widetilde{D}$ are exactly the $\overline{\mathtt{X}}$ tagged nodes that have no outgoing edges. Incoming edges of $\mathtt{F}$ tagged nodes always start from $\overline{\mathtt{X}}\mathtt{O}$ tagged nodes.

Proof. Let v be an $\overline{\mathtt{X}}$ tagged node of $\widetilde{D}$ that has no outgoing edges and assume that v is also $\mathtt{O}$ tagged. Then there exists an $\mathtt{X}$ tagged incoming neighbor w. Since by assumption there exists no edge vw in $\widetilde{D}$ we see that w is missing an incoming edge an therefore must be $\mathtt{O}$ tagged. But v lies in the same weak component of $\widetilde{D}$ as w so that by Lemma 3 there must be an $\mathtt{X}$ tag on v, a contradiction. Therefore v must be an $\mathtt{F}$ tagged node. The rest follows from Lemma 5. □

Lemma 8. *If the FOX algorithm is conducted in a way that avoids the assignment of multiple labels for as long as possible, then the set of unlabeled nodes at the time of the first assignment of a second tag to an already labeled node is identical to the set of nodes that become* `F` *tagged at the end of the FOX algorithm.*

Proof. We need to show that the assignment of a second tag to a node does not affect the set of unlabeled nodes.

The only situation where a second tag can be assigned is when there exists an `O` tagged node y with only one outgoing neighbor x. Note that x must be the `X` partner of y. By construction, all other neighbors of x must be incoming neighbors. These are necessarily `O` tagged since their incoming edge from x is missing.

After the FOX algorithm has assigned an `X` tag to y and an `O` tag to x we see that x and y form a separate 2-cycle. Therefore, if among the other formerly incoming neighbors there is one with only one outgoing neighbor, then the situation is the same as before and we may proceed in the same manner. Overall, only nodes of type `X` or `O` are affected. □

3. Graph Eigenvectors

Before we proceed to the application of the FOX algorithm to trees we need to introduce the notion of graph eigenvectors and eigenvalues (cf. [2]). The eigenvalues of a graph G are the eigenvalues of its adjacency matrix $A(G)$. Note that this definition is invariant under isomorphism so that the eigenvalues of a graph do not depend on the numbering of its nodes.

An eigenvector x of a graph G for eigenvalue λ is given by a nonzero solution of the equation $A(G)x = \lambda x$. This definition depends on the node numbering of the graph. Observe, however, that every vector $x = (x_i) \in \mathbb{R}^n$ can be represented by assigning weights to the nodes of G or $\widetilde{D}$ such that x_i is the weight of node v_i, given that $V(G) = \{v_1, \ldots, v_n\}$. This leads to a graph eigenvector definition that does not depend on the node order. Therefore, each time we determine an eigenvector of an adjacency matrix, we implicitly consider it as a function $V(G) \to \mathbb{R}$.

Reading the equation $A(G)x = \lambda x$ for each entry of x separately, we see that we may check if a given vector x is an eigenvector of G for eigenvalue λ as follows. Introduce node weights on G according to vector x and verify that for every node of G the sum of the weights of its neighbors equals λ times its own weight.

Since the FOX algorithm only performs row operations of the adjacency matrix of the given graph G, it is obvious that it has the same kernel as the adjacency matrix of the resulting graph $\widetilde{D}$.

Every `X` tagged node has exactly one outgoing neighbor, therefore its corresponding row of $A(\widetilde{D})$ is a unit vector. Consequently, the value of the `O` tagged partner of this `X` tagged node must be zero for every vector from the kernel of G:

Lemma 9. *Let G be a graph. Then the nodes of $H_{\mathtt{O}}$ obtained after a run of the FOX algorithm on G form a subset of the nodes whose value is zero for every vector from the kernel of G.*

In the next section we will see that this lemma can be strengthened considerably if the FOX algorithm is applied to a tree or forest.

4. FOX On Trees

In the following, we will assume that given a graph G we will automatically apply the FOX Algorithm 1 to its bidirectional orientation $\widehat{G}$ to get the final digraph $\widetilde{D}$. The number of nodes of G and the number of $\mathtt{F}$ tagged nodes of $\widetilde{D}$ will be denoted by $n = |G|$ and $k = |H_{\mathtt{F}}|$, respectively.

We may combine our findings up to this point in a straightforward manner to obtain the following lemma. Note that we call a directed path *maximal* if it is not contained in any other directed path.

Lemma 10. *Let G be a tree. For any $\mathtt{X}\overline{\mathtt{O}}$ tagged node v that has only one incoming edge let $\widetilde{R}_v$ be the subgraph of $\widetilde{D}$ spanned by the nodes of all directed paths in $\widetilde{D}$ starting from v. Likewise, for any $\mathtt{F}$ tagged node w let $\widetilde{S}_w$ be the subgraph of $\widetilde{D}$ spanned by the nodes of all directed paths in $\widetilde{D}$ ending at w. The undirected counterparts of $\widetilde{R}_v$ and $\widetilde{S}_w$ in G are called R_v and S_w, respectively.*

Then:

1. *The graphs R_v and S_w each induce a subtree of G.*
2. *Each maximal directed path in $\widetilde{R}_v$ that begins at v starts with an even number of consecutive nodes tagged $\mathtt{X}\overline{\mathtt{O}}$ and $\overline{\mathtt{X}}\mathtt{O}$ alternatively and terminates with an $\mathtt{F}$ tagged node.*
3. *Each maximal directed path in $\widetilde{S}_w$ that leads to w starts with an even number of consecutive nodes tagged $\mathtt{X}\overline{\mathtt{O}}$ and $\overline{\mathtt{X}}\mathtt{O}$ alternatively and terminates with an $\mathtt{F}$ tagged node.*

Before we advance to the construction of kernel eigenvectors, we will investigate some consequences of the previous lemma and definitions.

Lemma 11. *Let G be a tree. Then for every $\mathtt{X}\overline{\mathtt{O}}$ tagged node v of $\widetilde{D}$ there exists a node $w \in H_{\mathtt{F}}$ of $\widetilde{D}$ such that v lies in $\widetilde{S}_w$.*

Proof. Consider the last node w of a directed path P of maximum length within $\widetilde{D}$ that leads away from v. Let x be the immediate predecessor of w along P. We show that w can only be $\mathtt{F}$ tagged. Then by definition v would lie in $\widetilde{S}_w$.

Assume that w is $\mathtt{O}$ tagged. Then w cannot have any neighbors in $\widetilde{D}$ besides x. But then w and x must be of type $\mathtt{XO}$. Consequently, either $v = x$ so that it must also be of type $\mathtt{XO}$ or there exists no directed path from v to w because of Lemma 3.

Now assume that w is $\mathtt{X}$ tagged. Then due to the maximality of P the only outgoing neighbor of w must be x. Hence, x is the $\mathtt{O}$ partner of w. The case $v = x$ is impossible because then v would be $\mathtt{O}$ tagged. But since x is $\mathtt{O}$ tagged it cannot have an incoming edge from the third last node along P. □

Lemma 12. *Let G be a tree and x an $\mathtt{F}$ tagged node of G. Then, all $\overline{\mathtt{X}}\mathtt{O}$ tagged nodes of S_x have degree 2 within this subgraph of G. Consequently, all nodes of S_x whose degree is at least 3 are necessarily of type $\mathtt{X}\overline{\mathtt{O}}$ or $\mathtt{F}$.*

Proof. Clearly, S_x cannot contain any $\mathtt{XO}$ tagged nodes. Let u be a type $\overline{\mathtt{X}}\mathtt{O}$ node of S_x and let v be the $\mathtt{X}$ partner of u. Since v has no outgoing neighbors in $\widetilde{S}_x$ besides u it follows that a directed path from v to the $\mathtt{F}$ tagged node of $\widetilde{S}_x$ must lead via u. But then u must have another outgoing neighbor w besides v. It is impossible for u to have yet another outgoing neighbor y because then the `F` tagged node could not be reached in $\widetilde{S}_x$ without introducing a cycle in the tree G. Since u has no incoming neighbors besides v we see that it has degree two in S_x. □

We will now exhibit several properties of the adjacency matrix of the graph $\widetilde{D}$ obtained for a tree.

Lemma 13. *Let G be a tree and let A be the adjacency matrix of $\widetilde{D}$. Then the row vectors of A that correspond to the* $\overline{\mathtt{X}}\mathtt{O}$ *tagged of $\widetilde{D}$ are linearly independent among themselves and from the remaining row vectors of A.*

Proof. Let H be the subgraph of $\widetilde{D}$ that is spanned by all $\mathtt{X}\,\overline{\mathtt{O}}$ tagged and all $\overline{\mathtt{X}}\mathtt{O}$ tagged nodes. For each `X` tagged node of H remove the edge leading back from its unique outgoing neighbor. Then, as a consequence of Lemma 10 it is possible to assign integral height levels to the nodes of H such that the edges starting at each level may only lead to the nodes of the level directly below.

Construct a node sequence $v_1, \ldots, v_n$ as follows. First number the `X` tagged nodes of H level by level, starting from the top level. For the same order of the `X` tagged nodes, number the corresponding outgoing `O` tagged neighbors. Finally, number the `XO` tagged and then the `F` tagged nodes.

Assume that the adjacency matrix A reflects the above node numbering. Now form the submatrix $M = (m_{ij})$ of A that contains only the rows that correspond to the $\overline{\mathtt{X}}\mathtt{O}$ tagged nodes of $\widetilde{D}$ and the columns that correspond to the $\mathtt{X}\,\overline{\mathtt{O}}$ tagged nodes. Then by construction we have $m_{ij} = 0$ if $i > j$ and $m_{ii} = 1$. Thus, M takes upper triangular form with an all ones diagonal and therefore has maximum rank. Since $\mathtt{X}\,\overline{\mathtt{O}}$ tagged nodes can only have $\overline{\mathtt{X}}\mathtt{O}$ tagged nodes as incoming neighbors we see that within those columns of A that correspond to the $\mathtt{X}\,\overline{\mathtt{O}}$ tagged nodes all non-vanishing entries must necessarily belong to M as well. Consequently, the result follows. □

Construction 14. *Let G be a tree. Assign weight 0 to all* `O` *tagged nodes of $\widetilde{D}$ and also to all* `F` *tagged nodes except one node w. Assign weight 1 to w and construct $\widetilde{S}_w$ according to Lemma 10.*

All $\mathtt{X}\,\overline{\mathtt{O}}$ *tagged nodes that are not contained in $\widetilde{S}_w$ receive weights of 0. Conducting an incoming edge breadth first search on $\widetilde{S}_w$ with starting node w we can assign weights to the remaining* `X` *tagged nodes as follows. Let x be an unweighted* `X` *tagged node to be processed and let y be its outgoing* `O` *tagged neighbor. Then assign to x the negative sum of the weights of all non-incoming neighbors of y.*

Performing this procedure for every `F` *tagged node of $\widetilde{D}$ we obtain a set of k vectors $z_i \in \mathbb{R}^n$.*

Observation 15. *The following properties of Construction 14 are obvious:*

1. *At any time, all non-incoming neighbors of y have already been assigned weights.*
2. *The k vectors z_i are linearly independent.*

Lemma 16. *The vectors obtained by Construction 14 have only entries from the set $\{0, 1, -1\}$.*

Proof. Use the notation from Construction 14 and assume that we need to assign a weight to the $\mathtt{X}$ tagged node x. Consider the non-incoming neighbors of the associated node y. These can only be either $\mathtt{X}\overline{\mathtt{O}}$ tagged or $\mathtt{F}$ tagged. A non-zero weight can only occur at the neighbor that lies on the unique directed path from y to w. Using an inductive argument it is clear that only weights 1 or -1 can be assigned to $\mathtt{X}$ tagged nodes along this path and, consequently, to the node x. □

For a forest we will now strengthen Lemma 9 from the previous section.

Lemma 17. *Let G be a forest. Then the graph $H_{\mathtt{O}}$ contains exactly those nodes for which every vector from the graph kernel vanishes. In particular, this node set is invariant for all possible results of the FOX algorithm.*

Proof. Consider Construction 14 and note that according to Lemma 11 every $\mathtt{X}\overline{\mathtt{O}}$ or $\mathtt{F}$ tagged node v lies in a subgraph $\widetilde{S}_w$ for some node w. We see that for at least one basis vector the construction yields a nonzero weight on v. □

Example 18. *For graphs that contain cycles it is possible to find counterexamples regarding the above lemma. For instance, consider the graph that results by connecting a fifth node to one of the nodes of the cycle C_4. Its kernel has dimension one and the spanning vector is zero on an $\mathtt{X}$ tagged node, see figure 2.*

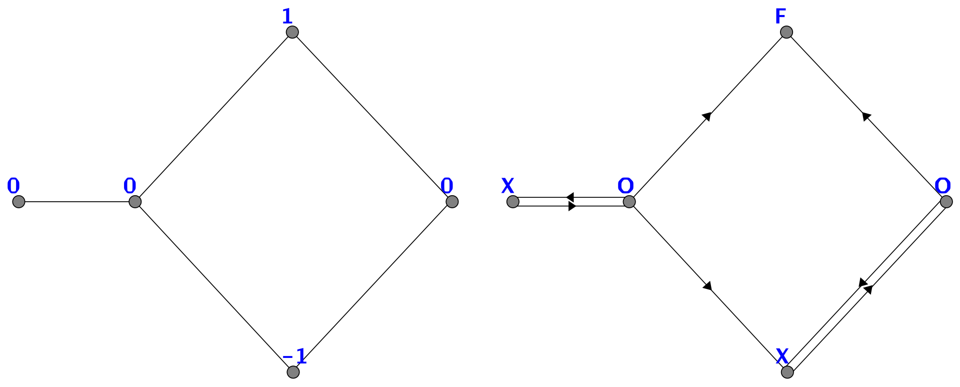

Figure 2: Counterexample for Lemma 17

Next, we present our main theorem. It is useful to note that a tree has a perfect matching if and only if it is nonsingular since the rank of the adjacency matrix of a tree is twice the size of a maximum matching [1], [5]. By iterative construction of the matching starting from the leaves and removing matched node pairs from the graph it becomes clear that a perfect matching in a tree must be unique.

Theorem 19. *Let G be a tree. Apply Algorithm 1 on its bidirectional orientation $\widehat{G}$ to get the final digraph $\widetilde{D}$.*

Then $k = |H_{\mathtt{F}}|$ equals the degree of singularity of G.

For $k \geq 1$ Construction 14 yields a basis of $\ker G$ that consists of vectors with entries from $\{0, 1, -1\}$ only.

For $k = 0$ the graph $\widetilde{D}$ consists only of XO *tagged nodes. The perfect matching of G can be constructed by pairing the nodes according to the 2-cycles in $H_{\mathtt{XO}}$.*

Proof. Construct row and column index permutations for the adjacency matrix as follows. First take all row indices that correspond to XO tagged nodes and arrange them as pairs according to the 2-cycles of $\widetilde{D}$. Use the same order of the node pairs for the column indices, but swap the node indices of each pair. Next, append the indices of the $\mathtt{X}\overline{\mathtt{O}}$ tagged nodes to the row indices and then in the same order append the indices of the corresponding unique outgoing ($\overline{\mathtt{X}}\mathtt{O}$ tagged) neighbors to the column indices. Now repeat this procedure vice versa (swap row and column indices) and additionally observe the node order described in the proof of Lemma 13. Finally append the indices of the F tagged nodes both to the row and column indices.

Using the above index permutation, the adjacency matrix of $\widetilde{D}$ transforms into the matrix

$$\begin{pmatrix} I & 0 & 0 & 0 \\ 0 & I & 0 & 0 \\ 0 & 0 & R & * \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

with an upper right triangular matrix R . This follows directly from Lemmas 3, 2, 5, 7, and 13.

Observing Lemma 13, we see that this matrix has kernel dimension k . Further, it has the same rank as the adjacency matrix of G since Algorithm 1, essentially, simply repeatedly looks for a unit vector row and uses it for pivoting.

Case $k \geq 1$ **.** Since the vectors constructed are linearly independent and Lemma 16 asserts that their entries are as required, it suffices to show that these vectors belong to the kernel of G . Using each vector as node weights on G , this is equivalent to showing that for each node the weight sum of its neighbors vanishes. Observe that two nodes that are adjacent in G are disconnected in $\widetilde{D}$ if and only if both have been O tagged. Therefore we may consider $\widetilde{D}$ instead (to be precise, its simple undirected representation). The neighbors of an F tagged or X tagged node are all O tagged nodes, therefore the weight sum vanishes. An $\overline{\mathtt{X}}\mathtt{O}$ tagged node may have both X tagged and F tagged neighbors. Among the latter there is at most one non-vanishing weight. Referring to Construction 14 we see that this weight has exactly been chosen so that the weight sum vanishes.

Case $k = 0$ **.** First observe that by pairing each X tagged node with its unique outgoing (O tagged) neighbor a perfect matching of G can be constructed since there are no F tagged nodes. As a consequence, every directed path in $\widetilde{D}$ that connects two leaves of G must contain an even number of nodes. Next, note that every O tagged leaf of $\widetilde{D}$ must also be X tagged since it has exactly one outgoing neighbor. Since the latter becomes O tagged this pair of nodes forms a 2-cycle. Repeat this argument for the remaining nodes. It is impossible to have only $\mathtt{X}\overline{\mathtt{O}}$ tagged leaves left since this would mean that we could find a directed leaf-to-leaf path with an odd number of nodes. Therefore, the whole graph $\widetilde{D}$ falls into XO tagged 2-cycles (cf. Lemma 3) so that indeed a perfect matching of G is obtained. □

Corollary 20. *The rank of the adjacency matrix of a tree is twice the size of a maximum matching.*

Proof. A matching that misses only k nodes can be constructed by pairing each X tagged node with its unique outgoing neighbor.

Conversely, consider a maximum matching of G. Without changing the size of the matching we may alter it such that every leaf neighbor is paired with one of its leaves. Apply Algorithm 1 and begin with the leaves that belong to the matching. These will become `X` tagged and their neighbors receive `O` tags. The subgraph of G that is spanned by the remaining untagged nodes is a forest F. Alter the matching of G such that every leaf neighbor of F is paired with one of its leaves. Proceed by alternatingly continuing Algorithm 1 with the leaves of F that belong to the matching and updating F. Finally, the forest F will only contain isolated nodes because otherwise the matching would not be maximal. Clearly, exactly the nodes of F become `F` tagged by Algorithm 1. □

We conclude this section with the following extension lemma:

Lemma 21. *Let G be a tree and v be an* `O` *tagged node of G. Construct a tree G' by connecting v to a new node w.*

Setting the weight of w to zero it is then possible to embed every vector from the kernel of G such that a vector from the kernel of G' is obtained.

Moreover,

$$\dim\ker G' = \dim\ker G + 1$$

and there exists a basis of $\ker G'$ such that all but one vector have zero weights on w.

Proof. Since according to Lemma 17 all vectors from the kernel of G have a zero entry on node v it is readily checked that assuming a zero weight on w will extend any eigenvector from $\ker G$ to one from $\ker G'$. Conversely, every vector from $\ker G'$ that has weight zero on w can be reduced to a vector from $\ker G$ by deleting the entry corresponding to w. Hence, $\dim\ker G \leq \dim\ker G'$. Moreover, we can find a basis of $\ker G'$ that does not contain more than one vector with a nonzero weight on w since a suitable linear combination of two such vectors would have a zero weight on w and therefore be a vector extended from $\ker G$. Consequently, $\dim\ker G' \leq \dim\ker G + 1$.

Now run the FOX Algorithm 1 on G as described in Lemma 8. The operations that occur until the first multiple tag is assigned are also valid for G'. But then v would be `O` labelled on G' so that w would become `F` tagged at the end of the FOX run. In view of Lemma 8 we see that we have gained exactly one degree of freedom and may construct a vector from $\ker G'$ with a nonzero weight on w. □

5. Storage Aspects

Given a basis whose vectors have length n and contain only entries from the set $\{-1, 0, 1\}$, it is clear that on a computer these vectors can be stored very efficiently. By separating the positive and negative components we get two binary vectors so that the overall information of a basis vector can be stored in a bit field of length $2n$.

Considering a tree kernel basis with entries only from the set $\{-1, 0, 1\}$, it is even sufficient to save only the zero/non-zero pattern of these basis vectors. Using a recursive technique it is possible to adjust the signs of the entries such that from every bit field a proper kernel vector can be reconstructed. However, the following example shows that the reconstructed vectors need not be linearly independent.

Example 22. *Consider the tree $K_{1,4}$. The leftmost three pictures in figure 3 represent a kernel basis of $K_{1,4}$. Given the zero/non-zero patterns of these basis vectors (e.g. read from suitable* *bit fields in memory), clearly, the first two basis vectors can be reconstructed up to a sign factor. Reconstruction of the third vector from its bit field may, however, also yield the vector shown in the rightmost picture. Since this vector can be obtained by subtracting the first basis vector from the second, we may fail to reconstruct a kernel basis from the bit fields of the original basis vectors.*

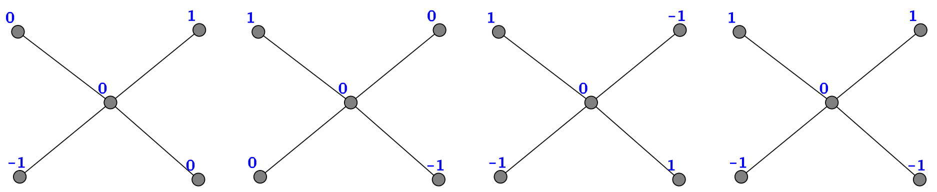

Figure 3: Kernel basis reconstruction for $K_{1,4}$

The next theorem asserts for a tree kernel basis obtained by the FOX algorithm and Construction 14 that up to sign factors the entire basis can be reconstructed from the stored zero/nonzero patterns.

Theorem 23. *Let G be a tree with n nodes. Having stored the zero-nonzero pattern of a vector from the kernel of G obtained by Construction 14 in a bit field of length n, the original eigenvector may be reconstructed up to a factor of ± 1 by the following procedure:*

1. *Interpreting the bit field as a function on the nodes of G, let G' be the tree that is obtained by removing all branches from G that have only zero weights.*
2. *Choose a node w of G' with weight 1 and negate the weights of all nodes whose distance to w is $2, 6, 10,$ and so on.*

Proof. First note that in view of Construction 14 the nonzero entries of the bit field represent $\overline{\texttt{0}}$ tagged nodes of G. Hence, node w must be either $\texttt{F}$ or $\texttt{X}\,\overline{\texttt{0}}$ tagged. According to Lemma 10 and Construction 14 the graph G' is one of the subtrees S_x of G in which w lies according to Lemma 11.

In view of this, we see that the neighbors (within G) of a nonzero node of G' must be $\texttt{0}$ tagged so that their weight sum vanishes. Now we need to check the neighbor weight sums for $\texttt{0}$ tagged nodes. Note that all node weights outside G' are zero. By definition of G' any node z outside G' can only have one neighbor in G'. This neighbor in turn must be $\texttt{0}$ tagged or otherwise the neighbor weight sum for z could not vanish. It remains to check the neighbor weight sums for the nodes of G'. But since all node weights outside G' are zero we may entirely restrict ourselves to the subgraph G'. According to Lemma 12 all $\texttt{0}$ tagged nodes of G' have degree 2 in G'. But by construction these neighbors have weights 1 and -1.

We see that we have constructed an eigenvector from $\ker G$ with the same zero/nonzero pattern as the original vector. Moreover, Lemma 12 ensures that after assigning only one nonzero weight to a node there is no choice during the reconstruction of the remaining entries. $\square$

Remark 24. *Note that although in the proof of the previous theorem we rely on the node tags created by the FOX algorithm, these tags are not needed for the eigenvector reconstruction itself.*

6. Conclusion

In the previous sections it has been shown that for the kernel of a given tree a basis can be constructed whose vectors contain only entries from the set $\{0, 1, -1\}$. The algorithm used is a variant of Gaussian elimination and is carried out directly on a digraph. It has been shown that if such a basis is to be stored along with the given tree, then only the zero/nonzero patterns of the vectors are needed in order to efficiently reconstruct the basis vectors up to sign factors.

An interesting open question is if the FOX algorithm can be modified such that the existence of simply structured kernel bases can be shown for a larger class of graphs. Overall, a characterization of all graphs that admit simply structured kernel (or other eigenspace) bases would be very desirable. We are currently investigating the situation for unicyclic graphs. Among these it is not difficult to find examples that do not admit kernel bases of the mentioned form.

References

[1] D. M. Cvetković and I. M. Gutman, The algebraic multiplicity of the number zero in the spectrum of a bipartite graph, *Mat. Vesnik,* **9**(1972), 141-150.

[2] C. Godsil and G. Royle, Algebraic graph theory, Vol.207 of Graduate Texts in Mathematics, Springer-Verlag, New York, 2001.

[3] I. Gutman and I. Sciriha, On the nullity of line graphs of trees, *Discrete Math.,* **232**(2001), 34-45.

[4] F. Hazama, On the kernels of the incidence matrices of graphs, *Discrete Math.,* **254**(2002), 165-174.

[5] P. E. John and G. Schild, Calculating the characteristic polynomial and the eigenvectors of a tree, *Match*, **34**(1996), 217-237.

[6] T. Sander, On certain eigenvalues of graphs without induced P_4. *(To appear).*

[7] T. Sander, Eigenspace Structure of Certain Graph Classes, Ph.D Thesis, TU Clausthal, 2004.

[8] I. Sciriha, The two classes of singular line graphs of trees, Vth Workshop on Combinatorics (Messina, 1999),*Rend. Sem. Mat. Messina Ser. II,* **5**(1999), 167-180.