Quotient Tree Partitioning of Undirected Graphs

Anders Edenbrandt[1]

TR 84-654
December 1984)

Department of Computer Science
Cornell University
Ithaca, New York 14853

[1] Work supported by an IBM graduate student fellowship and by NSF grant MCS-82-02948.

Quotient Tree Partitioning of Undirected Graphs

Anders Edenbrandt[1]

Computer Science Department
Cornell University
Ithaca, N.Y.

Abstract: The partitioning of the vertices of an undirected graph, in a way that makes its quotient graph a tree, mirrors a way of permuting a square symmetric matrix to allow its factoring with little fill-in. We analyze the complexity of finding the best partitioning and show that it is NP-complete. We also give a new and simpler implementation of an algorithm that finds a maximal quotient tree.

[1]Work supported by an IBM graduate student fellowship and by NSF grant MCS-82-02948.

1 Introduction

For certain classes of systems of linear equations $A\mathbf{x} = \mathbf{b}$, a solution can be computed efficiently by partitioning the matrix A into blocks, and solving the system by block factorization. The combinatorial problem we must solve to find a suitable partitioning is equivalent to quotient tree partitioning of undirected graphs. For a detailed description of the background of this problem, see [2].

A *partitioning* of the vertices of G is a division of the vertices into disjoint subsets: $\mathcal{F} = \{V_1, V_2, \ldots, V_p\}$. The *quotient graph* induced by the partitioning, written $G/\mathcal{F}$, is the graph $G' = (\mathcal{F}, \mathcal{E})$ of p vertices where $(V_i, V_j) \in \mathcal{E}$ if and only if there is some edge in E that connects a vertex in V_i with a vertex in V_j. If a quotient graph is a tree we call it a *quotient tree*. In what follows we will refer to vertices of the quotient graph as *nodes*, and reserve the term vertex for vertices of the original graph.

The problem we study is how to find tree partitionings that are "good" in some sense. Just exactly what determines a "good" partitioning depends of course upon our applications. We will suggest three different measures and show that for all three measures the problem of finding a best tree partitioning on an arbitrary undirected graph is NP-complete.

In the following section we will make use of reductions from the Maximum Independent Set problem. For our purposes this problem, which is known to be NP-complete [4], is defined as follows:

Definition 1.1: Maximum Independent Set (MIS).
INSTANCE: Undirected graph $G = (V, E)$ and an integer $K \leq |V|$.
QUESTION: Does there exist a subset $V' \subseteq V$, such that $|V'| \geq K$ and such that no two vertices in V' are connected by an edge in E?

2 Complexity of Quotient Tree Partitioning

In this section we investigate the computational complexity of the problem of finding a good quotient tree partitioning. We suggest three different measures and show that for each of them the problem of finding the best quotient tree partitioning is NP-complete.

The first measure is one where we want to maximize the number of nodes in the quotient tree. This is the measure used in an algorithm by George and Liu [1].

Definition 2.1: Maximum Quotient Tree (MQT).
INSTANCE: Undirected graph $G = (V, E)$ and an integer $K \leq |V|$.
QUESTION: Does there exist a partitioning $\mathcal{F} = \{V_1, V_2, \ldots, V_p\}$ of V into $p \geq K$ disjoint sets, such that $G/\mathcal{F}$ is a tree?

Theorem 2.1: Maximum Quotient Tree is NP-complete.

Proof: By a reduction from Maximum Independent Set.

Given a graph $G = (V, E)$ and an integer K as an instance of MIS, we form an instance of MQT by adding one new vertex r to G and making it adjacent to all the old vertices of G. We claim that this new graph G' has a quotient tree of size $K + 1$ if and only if G has an independent set of size K.

If the size of an independent set in G is K there exists a quotient tree of size $K + 1$, since we can take r plus the vertices not in the independent set to be in the root node of the quotient tree, and all the vertices of the independent set will be single-vertex leaves in the quotient tree.

Conversely, suppose there is a quotient tree of $K + 1$ nodes in G'. Consider the quotient tree to be rooted at the node that contains r. Since r is adjacent to all other vertices, all the other K nodes in the quotient tree are leaf nodes that are children of the root. Hence, no two vertices that reside in different leaf nodes are adjacent to each other. This means that we can form an independent set of size K by choosing one vertex from each of the K leaf nodes. □

With the maximum number of nodes as our measure of a good quotient tree we may of course find solutions in which a few nodes are very large. This consideration leads to our second measure of a good quotient tree, namely one where we try to minimize the size of each node.

Definition 2.2: Minimum Nodesize Quotient Tree (MNQT).
INSTANCE: Undirected graph $G = (V, E)$ and an integer K.
QUESTION: Does there exist a partitioning $\mathcal{F} = (V_1, V_2, \ldots, V_p)$ of V into p disjoint sets, for some $p \leq |V|$, such that $|V_i| \leq K$, and such that $G/\mathcal{F}$ is a tree?

Theorem 2.2: Minimum Nodesize Quotient Tree is NP-complete.

Proof: Again by a reduction from Maximum Independent Set.

Given a graph $G = (V, E)$ and an integer $K \leq |V|$ as an instance of MIS we construct an instance of MNQT as follows. First, add a copy of G to get a new graph $G_1 = (V_1, E_1)$ that consists of two disjoint copies of G. Clearly, G has an independent set of size K if and only if G_1 has an independent set of size $2K$.

Let $n = |V|$. Now, for each vertex v_i in V_1 construct a clique c_i of size $|c_i| = 4n - 1$, and make v_i adjacent to each vertex in its clique. Then, construct a new clique H of size $8n - 1$. Finally, add one more vertex, r, and make this new vertex adjacent to all the other vertices.

Call this new graph G'. It has $2n + 2n(4n - 1) + (8n - 1) + 1$ vertices, so its size is polynomial in the size of G. Setting $K' = 5n - K$ we now claim that G' has a quotient tree partitioning with its largest set of size $\leq K'$ if and only if G has an independent set of size $\geq K$.

Observation 1: Let $\mathcal{F}$ be any partitioning that makes $G'/\mathcal{F}$ a tree, T and let R be the set that contains r. Consider T to be rooted at R. Then, any other node in T is a child of R.

Observation 2: Let $\mathcal{F}$ be any partitioning and T and R as above. At least one set of $\mathcal{F}$ must have size $\geq 4n$. This is because the clique $H \cup \{r\}$ can be split over at most two subsets.

Observation 3: There exists a partitioning $\mathcal{F}$ of V' such that all sets in $\mathcal{F}$ have size $\leq 5n$. To see this, let each c_i-clique be in a set by itself, and let all vertices of V_1 plus r plus $3n-1$ of the vertices of H be in one set, and let the $5n$ remaining vertices of H be in the last set. It is easy to verify that this partitioning induces a tree and that no set is larger than $5n$.

Observation 4: If G has an independent set V_d of size d then there is a partitioning $\mathcal{F}$ which has all sets of size $\leq 5n - d$. A partitioning of this form is constructed as follows: For each vertex v_i of V_d take that vertex and its c_i-clique to form a separate set of size $4n$. Let each of the c_i-cliques of the remaining vertices of V_1 form a separate set of size $4n - 1$. Finally, form a root set R containing r, the $2n - 2d$ vertices of V_1 that are not in V_d, and $3n + d - 1$ of the vertices of H. The remaining $5n - d$ vertices of H will be in a separate set. The total size of R is then $2n - 2d + 3n + d - 1 + 1$ which is $5n - d$. Since $d \leq n$ each set of size $4n$ is also $\leq 5n - d$.

Observation 5: Let $\mathcal{F}$ be a Minimum Nodesize partitioning of G'. No set in the partitioning, except R, can contain two or more vertices from V_1. Assume that some set S other than R does contain two vertices from V_1. Then their c_i-cliques have to be split between being in S and being in R. Now, $|S| \leq 5n$ by observation 3 above and the fact that $\mathcal{F}$ is a Minimum Nodesize partitioning. Hence, at least $8n - 5n = 3n$ of the vertices of these cliques must fall in R. But then, since $|H| + 1 + 3n = 11n$ we must have that either R or the part of H outside of R is of size $\lceil 11n/2 \rceil$ which is greater than $5n$. This contradicts the assumption that $\mathcal{F}$ is a Minimum Nodesize partitioning.

Assume that there exists a partitioning of G' which is a MNQT partitioning and in which every set is of size $\leq p$. By observations 2 and 5 we can assume that every vertex from V_1 that is not in R is in a set containing only itself and vertices from its clique, forming a set of size $4n$. Furthermore, the c_i-cliques of all V_1-vertices must be outside R, forming sets of size $4n - 1$. Let S_H be the set formed by the vertices of H that are not in R. By assumption $|S_H| + |R| \leq 2p$ and among the vertices of $S_H \cup R$ are the $8n$ vertices of $H \cup \{r\}$. Thus, the number of vertices of V_1 in R is at most $2p - 8n$. Hence, at least one of the identical copies of the original graph has at most $p - 4n$ vertices in R. This means that for this graph we have an independent set of size at least $n - (p - 4n) = 5n - p$.

It follows from the observation above that G' has a Minimum Nodesize Quotient Tree partitioning with each set $\leq p$ if and only if G has an independent set of size $\geq 5n - p$. □

We come to our third, and last, measure of a good quotient tree. Here we want to minimize the sum of the squares of the set sizes in the partitioning. In the context of matrix block-factorization this is the area of the diagonal blocks; hence, an upper bound on the space required for these blocks. Not surprisingly, this measure does not make the problem any easier.

Definition 2.3: Minimum Sum Squares Quotient Tree (MSSQT).
INSTANCE: Undirected graph $G = (V, E)$, integer $K \leq |V|^2$.
QUESTION: Does there exist a partitioning $\mathcal{F} = (V_1, V_2, \ldots, V_p)$ of V into p disjoint sets, for some $p \leq |V|$, such that $G/\mathcal{F}$ is a tree and such that $|V_1|^2 + |V_2|^2 + \cdots + |V_p|^2 \leq K$?

Theorem 2.3: Minimum Sum Squares Quotient Tree is NP-complete.

Proof: By a reduction from Maximum Independent Set.

Given a graph $G = (V, E)$ and an integer $K \leq |V|$ as an instance of MIS we construct an instance of MSSQT as follows. Let $n = |V|$ and $s = \lceil \sqrt{n} \rceil$. For each vertex v_i in V make a clique c_i of size s, and make v_i adjacent to each vertex in its clique. Also, add an additional clique r of size s, and connect each vertex in r to all other vertices in the graph. The graph so constructed, $G' = (V', E')$, has $ns + n + s$ vertices—hence its size is polynomial in the size of G.

Setting $K' = K(s+1)^2 + (n-K)s^2 + (s+n-K)^2$, we claim that G' has a quotient tree partitioning with the sum of the squares of the set sizes at most K' if and only if G has an independent set of size K.

Let $\mathcal{F}_0$ be the partitioning $(r \cup V, c_1, \ldots, c_n)$. The cost of $\mathcal{F}_0$ is

$$C_0 = (s+n)^2 + ns^2 = (n+1)s^2 + 2ns + n^2.$$

Keeping this in mind, we can make the following claim:

Claim: In any optimal MSSQT partitioning, r lies completely within one set. Assume that we have a quotient tree partitioning of V' where two sets, S_1 and S_2, both contain vertices of r. Since each vertex of r is adjacent to every other vertex in V', there can be no other sets in the partitioning. Restricted to two sets, we get an optimal partitioning by making S_1 and S_2 equal in size. Hence the cost of this partitioning will be at least $2(\lfloor (ns+n+s)/2 \rfloor)^2$, which is greater than C_0 for $n \geq 2$.

Let $\mathcal{F}$ be any optimal MSSQT partitioning and let R be the set that contains r. If we consider the tree induced by $\mathcal{F}$ as rooted at R, then this tree has height at most 1.

Let S be a set in $\mathcal{F}$, $S \neq R$. If S contains part of a clique c_i, then v_i is either in S or in R. If v_i is in R we can construct a better partitioning by placing the part of c_i that is in S in a separate set. Hence, we can assume that each set S contains vertices v_i and parts of their cliques c_i. For each v_i its c_i-clique is either completely within R, completely within S, or partly within R and partly within S. We can assume without loss of generality that at most one vertex v in S has its clique partly in both R and S.

Assume that some vertex v in S has its clique completely within R. We construct a new partitioning $\mathcal{F}_1$, and claim that $\mathcal{F}_1$ is as least as good as $\mathcal{F}$. Remove the c-clique from R and let it form its own set, and move v from S to R. Clearly, this is still a quotient tree partitioning. We compare the costs by which these sets contribute to the total costs of the partitionings. Let $x = |R| - s$, $y = |S| - 1$, and let δ be the difference in cost between the old and the new partitioning. We get: $\delta = (x+s)^2 + (y+1)^2 - (x+1)^2 - s^2 - y^2 = 2sx + 2y - 2x \geq 0$.

Let v be a vertex in S whose c-clique straddles R and S. Let a be the number of vertices of the clique in R, and let b be the number of vertices of the clique in S. Also, let $x = |R| - a$, $y = |S| - b - 1$. We construct a new partitioning $\mathcal{F}_2$ by moving v from S to R and letting the c-clique be in a separate set. Then, $\delta = (x+a)^2 + (y+b+1)^2 - (x+1)^2 - (a+b)^2 - y^2 = 2x(a-1) + 2b - 2ab + 2y(b+1)$. Since R contains all of r, $x \geq s = a+b$ implies $\delta \geq 2(a+b)(a-1) + 2b - 2ab + 2y(b+1) = 2a(a-1) + 2y(b+1) \geq 0$.

This means that we now have a partitioning where R does not contain any part of any c-clique. Hence, S contains k vertices of V and all of their c-cliques. Assume $k \geq 2$ and let v be a vertex in S. Let $x = |R|$, $y = |S| - (s+1)$. Construct a new partitioning as follows: move v from S to R and let its c-clique be in a separate set. We have $\delta = x^2 + (y+s+1)^2 - (x+1)^2 - s^2 - y^2 = 2y(s+1) + 2s - 2x$. Since $k \geq 2$, $y \geq (s+1)$. This implies $\delta \geq 2(s+1)^2 + 2s - 2x = 2(s^2 + 3s + 1 - x)$. Furthermore, since R contains only vertices from r and V, $x \leq s + n$; and, by definition, $s^2 \geq n$. So, $\delta \geq 2(s^2 + 3s + 1 - s - n) \geq 2(2s+1) \geq 0$.

We can finally conclude that the optimal value of a MSSQT partitioning can be achieved through a partitioning of the following kind: we have a set R that contains r and some vertices of V; for each vertex v in R, its c-clique lies in a separate set; for each vertex v not in R, v is in a set that contains precisely v and its c-clique.

Assume that there are d vertices of V not in R. Clearly, from a partitioning of this kind we can find an independent set of size d in G. The cost of this partitioning is $C = (s+n-d)^2 + (n-d)s^2 + d(s+1)^2$.

Conversely, if G has an independent set of size d, we can easily construct a partitioning of the type and cost described above. □

3 Maximal Quotient Trees

The NP-completeness of the problems discussed above implies that we probably cannot find polynomial-time algorithms for MQT. Instead, we have to turn to approximation algorithms. One possibility then is to look for a maximal quotient tree rather than a maximum quotient tree.

A *maximal quotient tree* is a quotient tree partitioning in which we cannot partition any node further and still have a tree. A maximal quotient tree is not necessarily a maximum quotient tree, and can in fact be arbitrarily far from optimum. George and Liu [1] present an algorithm that will find a maximal quotient tree partitioning for an undirected graph. They offer no estimate of the running time of their algorithm, but an analysis of the implementation in [2] indicates that it has a worst-case running time of $\Omega(n^2)$, where n is the number of vertices in the original graph. We present in figure 1 an algorithm which is a different implementation of the same idea, and has a worst-case running-time of $O(m)$, where m is the number of edges in the graph.

The idea is as follows: We start with a level structure of G. Letting each level be a subset in the partitioning will certainly give us a quotient tree. However, to get a maximal quotient tree we refine the partitioning within each level, according to the following rule:

> Partitioning Rule: *Vertices v,w on level k are put in the same node if and only if there is a path from v to w going only through vertices on levels greater than or equal to k.*

George and Liu[1] prove that if we partition the vertices of G according to this rule the result will be a maximal quotient tree.

In the first part of the algorithm we use a breadth-first search to find a level structure for G. This part of the algorithm (not explicitly shown in the figure), returns an array of lists of vertices. Furthermore, for each vertex v in the graph we record in the field Level[v] the level of v.

In the second part of the algorithm we process the vertices looking at one level at a time. The first part of the body of the main loop builds an auxiliary graph H, whose vertices are the vertices on the present level plus the nodes of the next higher level.

In building the auxiliary graph we include all the edges that the vertices of H induce. To avoid adding an edge to H more than once, we take two precautions: First, if while processing vertex v on level k, we see an edge (v, w) within the level, we only add it if $w < v$. Second, assume that (v, w) is an interlevel edge, i.e., w is in a node u on level $k + 1$. We add the edge (v, u) to H and set the field Latest[u] to v. Now, if v should be adjacent to any other vertex in node u, we can detect that this edge has already been added simply by checking the value of Latest[u].

Having built H, we find its connected components using a depth-first search (not explicitly shown in the figure). Each new node in the quotient tree is made

MAXIMAL QUOTIENT TREE ALGORITHM:
Input: A connected graph $G = (V, E)$.
Output: A maximal quotient tree partitioning of V as given by the array Node.

```
1.  Use a breadth-first search to find a level structure for G.

2.  for i ← 1 to n do Latest[i] ← 0; od

3.  for k ← maxlevel to 1 do
4.      U ← {vertices on level k} ∪ {nodes on level k + 1};
5.      H ← (U, ∅);
6.      for each vertex v on level k do
7.          for each w ∈ Adj[v] do
8.              if Level[w] = Level[v] and w < v then
9.                  H ← H ∪ (w, v);
10.             else if Level[w] > Level[v] then
11.                 p ← Node[w];
12.                 if Latest[p] ≠ v then
13.                     H ← H ∪ (p, v)
14.                     Latest[p] ← v;
                    fi
                fi
            od
        od
15.     Determine the connected components of H,
        using a depth-first search.
        The level-k vertices of each connected component of H
        form a new node:
16.         for all vertices v in node p do Node[v] ← p; od
    od
```

Figure 1: Algorithm for finding a maximal quotient tree.

up of the level k vertices of some connected component of H.

Theorem 3.1: The algorithm in figure 1 computes a maximal quotient tree in time $O(m)$.

Proof: We first claim that vertices v,w on level k are in the same connected component of H if and only if there is a path from v to w going only through vertices of levels greater than or equal to k.

This is obvious for vertices on *maxlevel*. So, assume it is true for vertices on levels $k+1, ..., maxlevel$, and consider level k.

If v and w are in the same component, then obviously there is a path from v to w in H; and the path goes only through vertices of level k and nodes on level $k+1$. If v_i is a node on the path, then v_{i-1} and v_{i+1} must both be vertices on level k. Let (v_{i-1}, u) and (u', v_{i+1}) be the edges in G, where u,u' are in the same node p on level $k+1$. By induction hypothesis there is a path of the desired kind from u to u'. Hence there is a path from v to w going only through vertices on levels greater than or equal to k.

For the other direction, assume that a path $v = v_1, v_2, ..., v_l = w$ exists, where each v_i is on level k or higher. Let v_i be the first vertex on the path that is on level $k+1$. (If no such vertex exists, the path is completely within level k, and we are done.) Let v_{j+1} be the first vertex after v_i that is on level k. By inductive assumption, v_i and v_j are in the same node on level $k+1$, say node p. This means that (v_{i-1}, p) and (p, v_{j+1}) are both edges in H. By repeating this argument we can reduce the path in G from v to w to a path going only through vertices on level k and nodes on level $k+1$. It follows that v and w will be connected in H, and hence will be in the same node on level k.

We have proved that the algorithm makes a partitioning according to the Partitioning Rule above. As proved by George and Liu [1], this implies that the algorithm correctly computes a maximal quotient tree.

For the running time of the algorithm first note that determining the level structure of G can be done in time $O(m)$, since this is the running time of a breadth-first search.

In the part where we construct the auxiliary graphs each vertex of G is considered exactly once as v in the loop on line 6. For each adjacent vertex w of v we make a constant number of operations. Hence, the building of all the auxiliary graphs takes at most time $O(m)$.

Finally, it suffices to note that each edge that is added to an auxiliary graph corresponds uniquely to some edge in G. Hence, the total number of edges in all the auxiliary graphs is less than or equal to m. Since depth-first search takes time linear in the number of edges, the last part of the algorithm can also be done in time $O(m)$. It follows that the total worst-case running time of the algorithm is $O(m)$. □

Bibliography

[1] GEORGE, A., LIU, J.W. Algorithms for matrix partitioning and the numerical solution of finite element systems. *SIAM J. Numer. Analysis 15*, (1978), 297-327.

[2] GEORGE, A., LIU, J.W. *Computer Solution of Large Sparse Positive Definite Systems.* Prentice-Hall, Englewood Cliffs, N.J., 1981.

[3] GAREY, M.R., JOHNSON, D.S. *Computers and Intractability.* W.H. Freeman and Company, San Francisco, 1979.

[4] KARP, R.M. Reducibility among combinatorial problems. in R.E. MILLER AND J.W. THATCHER (eds.), *Complexity of Computer Computations*, Plenum Press, New York, 1972, 85-103.