Graph Algorithms and the Outward Equivalence Relation

Elias Dahlhaus* Jens Gustedt† Ross M. McConnell‡

Abstract

Let n denote the number of vertices and m denote the number of edges of a graph. It is not customary to claim that a $\Theta(n^2)$ graph algorithm runs in linear time, even though most graphs have $\Theta(n^2)$ size. The traditional measure against which linearity of a time bound is judged is $\Theta(n+m)$, since this measure is $O(n^2)$, but often much smaller. We give an even tighter measure of complexity of a graph that is $O(n+m)$, but often much smaller, and we show how many graph algorithms can be adapted to run in time that is linear in this complexity measure, provided that the graph is represented in an appropriate way.

1 Introduction

A directed graph with vertex set V can be viewed as a partition of all elements of $V \times V$ into two sets: "arcs" and "non-arcs". It also assigns a special status to the arcs; otherwise a graph would be indistinguishable from its complement. In modeling some types of problems where this distinction is artificial, it is sometimes convenient to ignore it. The partition of $V \times V$ into two arc classes is a special case of a "two-structure," which is the partition of $V \times V$ into an arbitrary number of classes, but we will deal only with two-structures that have two arc classes.

Let an *outward complementation operation* at a vertex v be a reassignment of each member of $\{(v,x) : x \in V - \{v\}\}$ to the opposite arc class from the one that contained it previously. This corresponds to changing every edge directed out of v into a non-edge, and every non-edge directed out of V to an edge. This differs from an *inward complementation operation*, which performs an analogous operation on $\{(x,v) : x \in V - \{v\}\}$, and which we will not discuss here. Both of these operations are special cases of *dynamic two-structure* operations (Ehrenfeucht and Rozenberg (1994)).

The relation of two two-structures or two digraphs being obtainable from each other by a sequence of these operations is an equivalence relation, and we may call the induced equivalence classes *outward equivalence classes*.

Any member G' of the outward equivalence class can be represented with an adjacency-list representation of a member G of the class that has the minimum

*Dept. of Computer Science and Dept. of Mathematics, University of Cologne, Cologne, Germany, dahlhaus@suenner.informatik.uni-koeln.de

†LORIA and INRIA Lorraine, campus scientifique, BP 239, 54506 Vandœuvre lès Nancy, France, Jens.Gustedt@loria.fr. Part of this work was done while Jens Gustedt was employed by the Technische Universität Berlin.

‡Department of Computer Science and Engineering, University of Colorado at Denver, Denver, CO 80217-3364 USA, rmcconne@carbon.cudenver.edu.

number of arcs, and a bit vector of length n that indicates which vertices of G must be outwardly complemented to obtain G'. The size of G is therefore a measure of the complexity of every member of the equivalence class. This measure is never larger than the traditional one, and is often much smaller. In this paper, we show that many graph algorithms can be adapted to run in time that is linear in the size of this new measure, given the representation using G. We have illustrated this bound for finding depth-first and breadth-first spanning forests, recognizing membership in several well-known graph classes, finding the chromatic number of a comparability graph, and computing modular decomposition (Dahlhaus et al. (1999b)). The present paper gives a survey of the techniques on a subset of these problems.

2 Modular Decomposition

Two sets X and Y *overlap* if $X \cap Y$, $X - Y$, and $Y - X$ are all nonempty. $X \Delta Y$ denotes $(X - Y) \cup (Y - X)$. An undirected graph is the special case of a symmetric directed graph where each edge corresponds to two symmetric arcs. If v is a vertex, $N(v)$ denotes $\{x : (v, x)\}$ is an edge. A two-structure is *symmetric* if for any distinct vertices x and y, (x, y) and (y, x) lie in the same arc class. If G is a two-structure or digraph and X is a subset of its vertices, then $G|X$ denotes the substructure or subgraph induced by X. A *component* of a two-structure is a proper subset of V that is a connected component of either of the two complementary graphs the two-structure represents. At most one of any two complementary graphs is disconnected, so this definition is unambiguous.

A *module* in a two-structure is a nonempty set X such that for any $y \in V - X$, all elements of $X \times \{y\}$ are in the same arc class, and all elements of $\{y\} \times X$ are in the same arc class. These are also considered modules in the two complementary graphs it represents. For simplicity, we restrict discussion of modules in this paper to symmetric two-structures and undirected graphs. A graph or two-structure is *prime* if it has only V and its one-element subsets as modules.

Proofs of the properties of modules in the remainder of this section can be found in Möhring (1985) or Ehrenfeucht and Rozenberg (1990).

Theorem 1 *If X is a module then $Y \subseteq X$ is a module of $G|X$ if and only if Y is a module of G.*

Theorem 2 *If X and Y are disjoint modules, then every member of the cartesian product $X \times Y$ is in the same arc class.*

If $\mathcal{P}$ is a partition of the vertices of a digraph G, then the *quotient* $G/\mathcal{P}$ is a graph whose vertices are members of $\mathcal{P}$, and where (X, Y) is an arc if there is an arc in G from X to Y. If $\mathcal{P}$ is the strongly-connected components of G, then $G/\mathcal{P}$ is a dag (Cormen et al. (1990)). If every member of $\mathcal{P}$ is a module, the quotient is a *modular quotient*. Because of Theorem 2, the definition of a modular quotient fails to give any special status to arcs over non-arcs, so the modular quotient of a two-structure is well-defined.

Theorem 3 *If $\mathcal{P}$ is a partition of the vertices of a two-structure where each part is a module, and $\mathcal{X} \subseteq \mathcal{P}$, then $\bigcup \mathcal{X}$ is a module of G if and only if $\mathcal{X}$ is a module of $G/\mathcal{P}$.*

Algorithm 2 defines the *modular decomposition* of a two-structure. The decomposition is a tree, where each leaf corresponds to a vertex of the two-structure, while each internal node represents the set of vertices given by its leaf descendants. It is convenient to view each internal node as a set of vertices, even if it is represented with an $O(1)$-sized node in the data structure.

Algorithm 1: MD(G)

---

if *G has only one vertex* **then return** v;
else
  create a tree node r;
  if *G has more than one component* **then**
    let $\mathcal{F}$ be the connected components;
    let r be labeled a parallel node
  else
    Let $\mathcal{F}$ be the maximal modules of G that are proper subsets of V;
    Let r be labeled a prime node
  for *each $X \in \mathcal{F}$* **do**
    Let $r_X = MD(G|X)$;
    add r_X as a child of r
  return r

A key property of the decomposition is that if G has only one component, the maximal modules that are proper subsets of V are a partition of V. Each internal node of the tree is a module, as is any union of children of a single degenerate node. There are no other modules of G other than those given in this way by the tree. The tree is therefore an implicit representation of all the modules of G. For each internal prime node X, let $\mathcal{C}$ be its children. The quotient $(G|X)/\mathcal{C}$ is a prime two-structure.

Below, we give an $O(n + m \log n)$ algorithm for computing the modular decomposition that requires that a solution to a subproblem be solved in time that is linear in the tighter sense we have proposed.

3 Vertex Partitioning

Let V be the vertices of an arbitrary graph G, and let $\mathcal{P} = (\{v\}, V - \{v\})$, where v is an arbitrary member of G. $\mathcal{P}$ is a partition of V where the partition classes are linearly ordered. We may refine $\mathcal{P}$ inductively to produce a finer and finer partition of V, as follows. Suppose that at some point $\mathcal{P} = (\{v\}, X_1, X_2, ..., X_k)$. Select $x \in X_i$ as a *pivot vertex.* For each partition class $X_j \neq X_i$, split X_j into $A_j = X_j \cap N(x)$ and $N_j = X_j - N(x)$. The refinement of $\mathcal{P}$ is obtained by removing any empty sets from the sequence ($\{v\}$, A_1, N_1, A_2, N_2, ..., A_{i-1}, N_{i-1}, X_i, N_{i+1}, A_{i+1}, ..., N_k, A_k). That is, those pairs of sets preceeding X_i are ordered so that the neighbors of x preceed non-neighbors, and those following X_i are ordered so that neighbors of x follow non-neighbors. X_i is not split.

Clearly, no proper refinement of $\mathcal{P}$ is possible if each partition class is a module. Conversely, if some partition class is not a module, there exists a choice of pivot x such that a proper refinement of $\mathcal{P}$ occurs. The procedure may be applied for different values of x until all partition classes are modules.

The final partition consists of $\{v\}$ and the maximal modules of G that do not contain v. The procedure is called *vertex partitioning.*

Spinrad (1985) suggests the strategy of selecting a vertex as a pivot only if the class X_i that currently contains it is at most half as large as the class that contained it the last time it was selected as a pivot. He shows that even with this restriction, all sets are modules when the procedure halts. This guarantees that any vertex is selected as a pivot $O(\log n)$ times, from which an $O(n + m \log n)$ bound for the procedure can be derived. An $O(n + m)$ bound that uses a different pivot selection strategy is given in McConnell and Spinrad (1999), but it requires precomputing the modular decomposition and is much more involved. A simple linear-time approach would be an interesting development.

Theorem 4 *A sequence of choices of pivots produces the same sequence of refinements to $\mathcal{P}$ in all digraphs of an outward equivalence class. Only the ordering of the partition classes depends on the particular graph.*

4 Transitive Orientation

A *source* in a digraph is a vertex with no incoming arcs, and a *sink* is one with no outgoing arcs. A digraph is transitive if whenever (u, v) and (v, w) are arcs, it follows that (u, w) is an arc. An undirected graph is a comparability graph if there is a way to assign orientations to its edges so that the resulting digraph is transitive. One example is the complement of an interval graph; a graph is an interval graph if and only if it is chordal and its complement is a comparability graph. Another example is a permutation graph; a graph is a permutation graph if and only if it and its complement are both comparability graphs.

Given a transitive orientation of G, many otherwise hard problems, such as finding a maximum clique can be solved in linear time. Finding a transitive orientation of G reduces to the problem of transitively orienting all quotients of the form $\{(G|X)/\mathcal{P} : X$ is an internal node of the modular decomposition of G and $\mathcal{C}$ is its children (see, for example, Möhring (1985)). When X is a degenerate node, the transitive orientation of the quotient is trivial to compute, since it is either a complete or an empty graph. The only difficult step is when X is a prime node; in this case the quotient is prime. Henceforth, we may assume that G is a prime comparability graph.

The transitive orientation procedure of McConnell and Spinrad (1999) uses the following observation. Since G is prime, the vertex partitioning procedure above yields a partition of V such that every partition class is a singleton set. The ordering of the partition classes gives a linear ordering of V. If v is arbitrary, the initial partition $(\{v\}, V - \{v\})$ gives rise to an ordering where the last vertex is a source in a transitive orientation. On the other hand, if v is a source, then the resulting linear order on V is a topological sort of a transitive orientation. The topological sort gives an implicit orientation to the edges of G that yields a transitive orientation.

Given modular decomposition, recognizition of permutation graphs reduces to the problem of recognizing prime permutation graphs. To recognize prime permutation graphs, we may use the property that a graph is a permutation graph if and only if G and its complement are both comparability graphs. Using Theorem 4 and the fact that G and its complement are in the same graph equivalence class, we may run the procedure twice, once for G and once for $\overline{G}$. The difference in the rules for ordering the partition classes the second time is trivial. If G is a permutation graph, this produces a type of certificate of this fact. Running an algorithm that verifies the certificate tells whether G is, in fact, a permutation graph.

A similar procedure gives an alternative to the algorithm of Booth and Lueker (1976) for recognizing interval graphs. The complement of an interval graph is a comparability graph, and a linear extension of a transitive orientation provides a type of certificate that the graph is an interval graph if it is one.

5 Depth-First Search and Strongly-Connected Components

In this section we describe how to compute a depth-first forest for a graph in time that is linear in the size of the tighter complexity measure.

One way to implement depth-first search is to simulate recursion by using a stack of vertices that are neighbors of vertices that have been visited. Vertices are popped from the top of the stack until an unvisited vertex v is encountered. The unvisited neighbors of v are then pushed to the stack. It is easy to see that if multiple copies of a vertex x appear on the stack, all but the top instance are irrelevant, since x will have been visited by the time a lower instance is popped. Thus, we may allow the stack to delete any lower instances of x when x is pushed.

The complement $\overline{G}$ is in the same equivalence class as G is, so let us begin by examining how to perform the search on $\overline{G}$ in time that is linear in the number of vertices and edges of G. The only change that must be made is that when we visit a vertex v, we must push its unvisited non-neighbors in G, rather than its unvisited neighbors.

The trick is to use a data structure for the stack that does not require physically touching some of the vertices that need to be pushed, and analyzing its behavior in an amortized sense (Tarjan (1983)). Let L be a doubly-linked list containing the unvisited vertices that have not yet been pushed, let T be the unvisited vertices that were pushed when the most recently visited vertex was visited, in any order, and let B be the remaining unvisited vertices on the stack, in stack order. $\{L, T, B\}$ is a partition of the unvisited vertices, and the sequence of vertices in the stack is given by the concatenation of T and B, which stand for "Top" and "Bottom". Each unvisited vertex is marked with the list that contains it. When v is visited, mark its neighbors in G, and, using the neighbor list of v, move any of them that are in T to the top of B. Then traverse L and B, moving any unmarked vertices to T.

The vertices that are implicitly pushed are those that are in T after the step, that is, the non-neighbors of v. This is what is desired, but no time was spent touching vertices that were already in T before the step. The time spent touching any marked vertex is charged to the arc to the vertex from v. Any unmarked vertex that is touched moves from L or B to T. This cost is charged either to the arc that subsequently causes it to be marked and removed from T, or to the pop operation that subsequently removes it from T. There are $O(n)$ pop operations and $O(m)$ arcs. Each arc and pop is charged $O(1)$ times. We conclude that the cost of the depth-first search on $\overline{G}$ takes time that is linear in the size of G.

Next, suppose that G' is an arbitrary member of the equivalence class of G. We simulate the stack S with two smaller stacks S' and S''. S' is implemented as an ordinary stack, while S'' is implemented with T and B as described above. When a vertex v is visited, its non-neighbors are pushed to S'' if v is outwardly complemented in getting G' from G, and its neighbors are pushed to S' otherwise. Each of these stacks is equipped with a means of returning a timestamp that tells when its top element was pushed. A pop operation pops from the stack with the most recent timestamp. The timestamp is also useful for establishing the parent of the popped vertex in the depth-first forest. The analysis of the time bound is similar.

In Dahlhaus et al. (1999b), we also describe a linear-time variant to find a depth-first spanning forest for a graph G' in the equivalence class of the transpose G^T of G, given an adjacency-list representation of G. The algorithm for strongly-connected components given in Cormen et al. (1990) reduces the problem to finding depth-first forests on G and G^T. Thus, we may compute the strongly-connected components of any member of G's outward equivalence class in time linear in the size of G. Although it is not stated explicitly in Cormen et al. (1990), that algorithm returns the strongly-connected components in an order that is a topological sort of $G/\mathcal{C}$, where $\mathcal{C}$ is the strongly-connected components. Thus, we may find a topological sort of this quotient within the same time bound.

6 Computing Modular Decomposition

At any degenerate internal node X, the children are the components of $G|X$. A *redundant node* could be obtained by removing a subset of the children of X, making them children of a new node, and making the new node a child of X. The new node then represents the union Y of the children that were removed from X, but the decomposition tree is no longer correct.

Algorithm 6 gives an algorithm for computing a modular decomposition tree that may contain redundant nodes, but these are easily identified and removed in a postprocessing step. The key to understanding why it works is the observation that the maximal modules of G that do not contain v are each a sibling of an ancestor of v whose parent is a prime node, or the union of the siblings of an ancestor of v whose parent is a degenerate node. Therefore $G/\mathcal{P}$ is a modular quotient. The correctness can be derived from Theorem 3, which says that the ancestors of v in G are given by the ancestors of $\{v\}$ in $G/\mathcal{P}$, and from Theorem 1, which says that the remainder of the tree is given by the recursive calls.

The algorithm first appeared in Ehrenfeucht et al. (1994), where an $O(n^2)$ implementation was given. We now show an $O(n + m \log n)$ bound, which is obtained by solving a subproblem in time that is linear under the tighter complexity measure proposed in the introduction.

For simplicitly, let us assume that the graph is connected. $\mathcal{P}$ is found by the vertex partitioning algorithm described above. A variant of Spinrad's vertex partitioning strategy that is given in McConnell and Spinrad (2000) makes sure that the cost of this step, summed over all recursive calls, is $O(m \log n)$.

The key obstacle to the time bound is finding the ancestors of $\{v\}$ in $H = G/\mathcal{P}$. Let us denote $\{v\}$ as v' when we refer to it as a vertex of H. The *forcing graph* $F(H, v')$ has the same vertex set as H, minus v'. For any pair $\{x, y\}$ of vertices, there is an arc from y to x in $F(H, v')$ if and only if y is adjacent to exactly one of v and x. Theorem 3 implies that all non-singleton modules of H are ancestors of v' in the modular decomposition of H. It is not hard to see that

Algorithm 2: *Decomp*(G), Compute the modular decomposition of G.

if *G has only one node* **then return** a one-node tree;
else
    Select a vertex v of G;
    Let $\mathcal{P}$ be $\{v\}$ and the maximal modules of G that don't contain v ;
    for *each ancestor $\mathcal{U}$ of $\{v\}$ in $G/\mathcal{P}$ in descending order* **do**
        Create a tree node t_u;
        Make t_u a child of the tree node corresponding to its parent in $G/\mathcal{P}$;
        for *each member X of $\mathcal{P}$ that is a child of U* **do**
            $t_x = Decomp(G|X)$;
            Let t_x be a child of t_u
    return the root of the resulting tree

X is a module of H that contains v' if and only if there is no arc directed from a non-member of X to a member of X in $F(H, v')$. These two facts imply that there is a unique topological sort of the quotient F' of $F(H, v')$ induced by its strongly-connected components, and that they give the siblings of the ancestors of v' in ascending order of depth in the tree.

All that is required for the step is computing the topological sort of F'. The problem is that $F(H, v')$, hence F', could have many more arcs than the number of edges in H, or even in G. Fortunately, $F(H, v')$ is in the same outward equivalence class as H, since the neighbors of a vertex y are its neighbors in H if y is a non-neighbor of v, and its non-neighbors in H if y is a neighbor of v. The cost of finding a topological sort of F' therefore takes time that is linear in the size of H. The size of H, summed over all recursive calls, is easily seen to be linear in the size of G. The time spent finding ancestors of $\{v\}$, summed over all recursive calls, is therefore $O(n + m)$. The only bottleneck is the $O(n + m \log n)$ bound for vertex partitioning.

7 Open Problems

Dahlhaus et al. (1999b) show that modular decomposition of G itself can be computed in time that is an $O(\log n)$ factor greater than the tighter complexity measure of G that we propose. The algorithm of Dahlhaus et al. (1999a) finds modular decomposition in linear time by solving a subproblem in time linear in our tighter complexity measure. Like Algorithm 6, this algorithm makes use of equivalence classes on graphs that are based on complementing edges incident to a vertex. It therefore seems likely that one can get a modular decomposition algorithm that is linear in the tighter complexity measure by modifying that algorithm, but this is still open.

Another open problem is to modify the linear-time transitive orientation algorithm of McConnell and Spinrad (1999). This is more ambitious because the algorithm is more difficult. However, it seems possible in principle, since the algorithm is based on vertex partitioning, modules, and induced P_4s, which all depend only on the properties of two-structures, and not the additional property of graphs.

Another area to investigate is what happens when one is allowed to complement either the outgoing arcs, or the incoming arcs, or both, at a vertex.

This also defines *two-way* equivalence classes on graphs that are each unions of many outward equivalence classes, and that have many interesting mathematical properties. We have noticed that some algorithms can be modified to run in time that is linear in the size of the minimum member of one of these larger classes. The tricks are similar, but the exercise is more challenging. Finding a minimum-sized member of an outward equivalence class is trivial, but this is not the case for the two-way equivalence classes.

References

[Booth and Lueker, 1976] Booth, S. and Lueker, S. (1976). Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *J. Comput. Syst. Sci.*, 13:335–379.

[Cormen et al., 1990] Cormen, T. H., Leiserson, C. E., and Rivest, R. L. (1990). *Algorithms*. MIT Press, Cambridge, Massachusetts.

[Dahlhaus et al., 1999a] Dahlhaus, E., Gustedt, J., and McConnell, R. M. (1999a). Efficient and practical algorithms for sequential modular decomposition. Technical report, INRIA. manuscript.

[Dahlhaus et al., 1999b] Dahlhaus, E., Gustedt, J., and McConnell, R. M. (1999b). Partially complemented representations graphs. Technical report, INRIA. submitted.

[Ehrenfeucht et al., 1994] Ehrenfeucht, A., Gabow, H. N., McConnell, R. M., and Sullivan, S. J. (1994). An $O(n^2)$ divide-and-conquer algorithm for the prime tree decomposition of two-structures and modular decomposition of graphs. *Journal of Algorithms*, 16:283–294.

[Ehrenfeucht and Rozenberg, 1990] Ehrenfeucht, A. and Rozenberg, G. (1990). Theory of 2-structures, part 1: Clans, basic subclasses, and morphisms. *Theoretical Computer Science*, 70:277–303.

[Ehrenfeucht and Rozenberg, 1994] Ehrenfeucht, A. and Rozenberg, G. (1994). Dynamic labeled 2-structures. *Math. Struct. in Comp. Science*, 4:433–455.

[McConnell and Spinrad, 1999] McConnell, R. M. and Spinrad, J. P. (1999). Modular decomposition and transitive orientation. *Discrete Mathematics*, 201(1-3):189–241.

[McConnell and Spinrad, 2000] McConnell, R. M. and Spinrad, J. P. (2000). Ordered vertex partitioning. Discrete Math and Theoretical Computer Science (to appear).

[Möhring, 1985] Möhring, R. H. (1985). Algorithmic aspects of the substitution decomposition in optimization over relations, set systems and boolean functions. *Annals of Operations Research*, 4:195–225.

[Spinrad, 1985] Spinrad, J. P. (1985). Graph partitioning. Unpublished manuscript.

[Tarjan, 1983] Tarjan, R. E. (1983). *Data structures and network algorithms*. Society for Industrial and Applied Math., Philadelphia.