arXiv:1710.04333v1 [cs.DM] 12 Oct 2017

Modular decomposition of transitive graphs and transitively orienting their complements

Henning Koehler

Massey University, New Zealand

The modular decomposition of a graph is a canonical representation of its modules. Algorithms for computing the modular decomposition of directed and undirected graphs differ significantly, with the undirected case being simpler, and algorithms for directed graphs often work by reducing the problem to decomposing undirected graphs. In this paper we show that transitive acyclic digraphs have the same strong modules as their undirected versions. This simplifies reduction for transitive digraphs, requiring only the computation of strongly connected components.

Furthermore, we are interested in permutation graphs, where both the graph and its complement are transitively orientable. Such graphs may be represented indirectly, as the transitive closure of a given graph. For non-transitive graphs we present a linear-time algorithm which allows us to identify prime-free modules w.r.t their transitive closure, which speeds up both modular decomposition and transitive orientation for sparse graphs.

Finally, we show that any transitive orientation of a digraph's complement also transitively orients the complement of the digraph's transitive closure, allowing us to find such orientations in (near-)linear time.

Keywords: modular decomposition, transitive orientation, permutation graph

1 Introduction

A *module* M of a graph $G = (V, E)$, which may be directed or undirected, is a non-empty subset of V such that all vertices in M have the same predecessors and successors in $V \setminus M$. Decomposing a graph into modules can help with various graph problems, as it provides a way to reduce them to the same or similar (sometimes easier) problems on smaller graphs [8]. While graphs can possess exponentially many modules, all modules can be represented efficiently in linear space, and this representation, known as *modular decomposition*, can be found in linear time [8, 7]. As the algorithms needed to achieve linear running time are highly complex, later work has focused on achieving near-linear running times with simpler algorithms, for both undirected [10] and directed graphs [5, 12].

As pointed out in [10], the issue of *transitive orientation* is closely related. Here we seek to orient the edges of an undirected graph in such a way that the resulting digraph is transitive. Not every graph can be oriented this way, and transitively orientable graphs are known as comparability graphs. Comparability graphs whose complement is also transitively orientable are called permutation graphs.

We are particularly interested in digraphs whose complement can be transitively oriented. It turns out that the transitive closure of such a graph is a permutation graph, and that a pair of total orders characterizing the transitive closure can be computed in near-linear time.

To begin, we introduce some helpful tools and terminology. Throughout the paper we consider all graphs to be simple and directed, and represent undirected edges as pairs of directed edges.

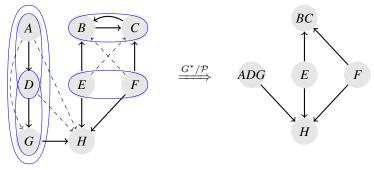**Definition 1** (inverse, undirected, complement, oriented, transitive)**.**

(i) *The* inverse *of an edge (a, b) is the edge $(a, b)^{-1} = (b, a)$. The inverse of a graph $G = (V, E)$ is the graph $G^{-1} = (V, E^{-1})$, where $E^{-1} = \{(a, b)^{-1} \mid (a, b) \in E\}$.*

(ii) *We say that a graph $G = (V, E)$ is* undirected *iff $E = E^{-1}$. The* undirected closure *of a graph $G = (V, E)$ is the graph $\mathcal{U}(G) = (V, E \cup E^{-1})$.*

(iii) *The* complement *of $G = (V, E)$ is the graph $\overline{G} = (V, V \rtimes V \setminus E)$, where $V \rtimes V = \{(a, b) \mid a, b \in V, a \neq b\}$. The* undirected complement *$\overline{G}^u$ of G is the complement of $\mathcal{U}(G)$.*

(iv) *A graph $G = (V, E)$ is* oriented *iff $E \cap E^{-1} = \emptyset$. A graph $G' = (V, E')$ is an* orientation *of $G = (V, E)$ iff E' is a maximal oriented subset of E.*

(v) *A graph $G = (V, E)$ is* transitive *iff for all edges $(a, b), (b, c) \in E$ with $a \neq c$ we also have $(a, c) \in E$. The* transitive closure *of a graph $G = (V, E)$ is $G^* = (V, E^*)$, where E^* is the minimal transitive superset of E.*

We shall describe the representation of modules next.

1.1   Modular Decomposition

The *trivial modules* of G are V and all singleton sets. A graph is called *prime* if it does not possess non-trivial modules. Two modules *overlap* if they intersect, but neither one contains the other. A module is a *strong module* if it does not overlap with any module, otherwise it is a *weak module*. Given a partition $\mathcal{P}$ of V into modules, also referred to as a *congruence partition*, the *quotient $G/\mathcal{P}$* is the graph obtained from G by merging the nodes of each module in $\mathcal{P}$.

Example 1. *Consider the transitive closure G^* of the graph G shown on the left below.*

The non-trivial modules of G^* are AD, DG, ADG, BC and EF. Its non-trivial strong modules are ADG, BC and EF. The set $\mathcal{P} = \{ADG, BC, E, F, H\}$ is one of many congruence partitions. The quotient graph $G^*/\mathcal{P}$ is shown on the right. Unlike its transitive closure, G is prime.   □
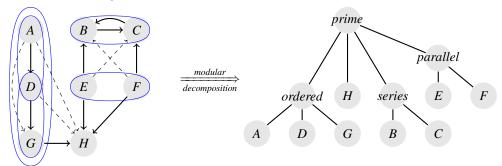
The *modular decomposition* of G is an $O(n)$-space representation of all modules of G. It was first described in [4], for a recent survey see [6]. The decomposition forms a tree, whose nodes are the strong modules of G, ordered by the subset relationship. In particular, the root of the modular decomposition tree is V, and its leaves are the singleton sets.

Furthermore, we can associate the *child quotient* graph $(G|N)/\text{children}(N)$ with each non-leaf node N, where $G|N$ denotes the subgraph of G on $N \subseteq V$, and children(N) the maximal strong modules properly contained in N. Each non-leaf node is classified based on the type of its child quotient graph:

- prime: the child quotient graph is a prime graph
- series: the child quotient graph is a clique
- parallel: the child quotient graph is independent (contains no edges)
- ordered: the child quotient graphs describes a total order (directed graph only)

No other cases exist, and the types, together with an ordering of the vertices in the case of an ordered node, fully describe all modules of the child quotient graph, which in turn are modules of G.

Example 2. *Consider again the graph from Example 1. The modular decomposition tree of its transitive closure is shown on the right.*

Together with a total ordering $A < D < G$ of the children of each ordered node, this describes all modules of G^*. For parallel and series nodes all subsets of children are modules, and for ordered nodes all sub-intervals of children are, in our case AD and DG (but not AG). □
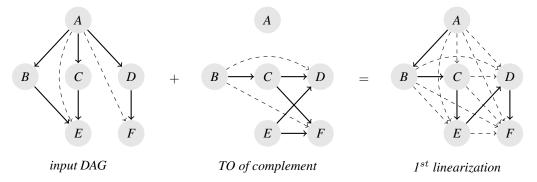
The modular decomposition of a graph is a useful tool for many graph problems. In particular, one may observe that the child quotient graphs of ordered, series and parallel nodes are permutation graphs, and that transitive orientations for them and their complements are trivial to find.
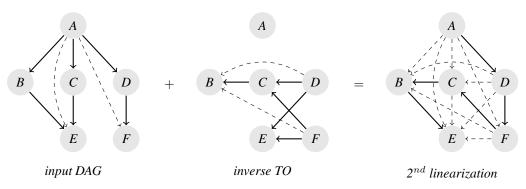
1.2   Transitive Orientation

We call a graph *transitively orientable* iff it possesses a transitive orientation. Transitive orientations of graphs are typically represented by providing a total ordering of vertices, which applies only to edges in the graph to be oriented. As this representation only requires space (near-)linear in the number of vertices, it can be computed efficiently even if the graph is large - e.g. the complement of a given sparse graph.

Existence of a transitive orientation is particularly interesting for the complement of a transitive DAG G, as it characterizes G as a permutation graph. In this case G has order dimension two (or less), i.e., can be characterized as the intersection of two linear orders [2]. The linear orders describing G can then be found by merging G with the transitive orientation $\mathcal{O}$ of its complement, and with its inverse $\mathcal{O}^{-1}$.

Example 3. *Consider the graph shown on the left. A transitive orientation of the undirected complement of its transitive closure is given in the center. Merging the two graphs results in two linear orderings of vertices, the intersection of which characterizes the original graph.*

input DAG   *TO of complement*   *1st linearization*

Merging G with the inverse orientation of its complement provides the second linearization.

input DAG   *inverse TO*   *2nd linearization*

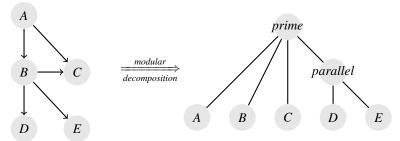*The partial ordering induced by G can now be represented as $\leq_{ABCEDF} \cap \leq_{ADFCBE}$.*   □

An efficient algorithm for combining the partial order described by G^* with a transitive orientation of its complement, given as a linear ordering, can e.g. be found in [3, Proof of Lemma 2].

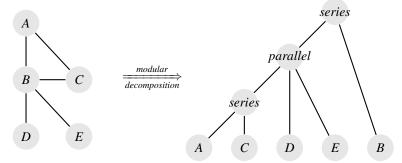## 2   Modular Decomposition of Transitive Graphs

Most algorithms for decomposing directed graphs do so by reducing the problem to finding the modular decomposition of an undirected graph [5, 12], or by adapting specific algorithms for undirected graphs [1]. This suggests that although modular decomposition of directed and undirected graphs can both be performed in (near-) linear time, decomposing undirected graphs is technically simpler.

An obvious candidate for an undirected graph to which we might reduce the problem of modular decomposition of a digraph is its undirected closure. Clearly, modules of the former must also be modules of the latter. However, as the following example shows, their modular decompositions may differ greatly if the graph in question is not transitive.

Example 4. *Consider the directed graph G shown below. Its only non-trivial module is DE, leading to the decomposition tree on the right.*

The undirected closure $\mathcal{U}(G)$ is shown below. In addition to DE, the sets AC, ACD, ACE and ACDE are non-trivial modules of $\mathcal{U}(G)$. Among these, only AC and ACDE are strong modules.

This shows that although modules of G must also be modules of $\mathcal{U}(G)$, the two graphs may not share any non-trivial strong modules, leading to very different decomposition trees.   □

However, we seek to decompose *transitive* graphs. Here the relationship between the modular decompositions of G and $\mathcal{U}(G)$ is much closer – as we will see later (proof of Theorem 1, *if* direction), strong modules of $\mathcal{U}(G)$ are strong modules of G. However, G may have strong modules which are only weak modules of $\mathcal{U}(G)$, and thus missing in the modular decomposition of $\mathcal{U}(G)$.

Example 5. *Consider the transitive graph G and its undirected closure below:*

The set $\{B, C\}$ is a strong module of G, but only a weak module of $\mathcal{U}(G)$.   □

The problem in Example 5 hails from the fact that ordered and series nodes become indistinguishable in the undirected closure. We can avoid this by identifying and merging strongly connected components. This leaves us with a transitive DAG, whose modular decomposition is free of series nodes. For such graphs, the modular decompositions of G and its undirected closure $\mathcal{U}(G)$ become isomorphic. Before we show this, we briefly establish some terminology and properties.

Definition 2 (child quotient graph)**.**

(i) *We denote the minimal strong module containing a module M by M^+.*

(ii) *The* child quotient graph *of a weak module M is $(G|M)/\mathcal{P}_M$, where $\mathcal{P}_M$ is the partition of M into the maximal strong modules contained in M.*

(iii) *We call a weak module M* series*,* parallel *or* ordered *if its child quotient graph is a clique, independent or totally ordered, respectively.*

One may easily confirm the following:

Proposition 1. *A weak module M is series/parallel/ordered iff M^+ is series/parallel/ordered.*
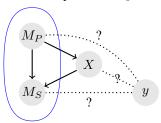
Proposition 2. *If a module M of $G = (V, E)$ can be partitioned into two non-empty subsets M_1, M_2 such that $M_1 \times M_2 \subseteq E$ and $M_2 \times M_1 \subseteq E$, then M is series.*

We are now ready to show our result.

Theorem 1. *Let $G = (V, E)$ be a transitive acyclic digraph. Then $M \subseteq V$ is a strong module of G if and only if M is a strong module of $\mathcal{U}(G)$.*

Proof: (if) Let M be a strong module of $\mathcal{U}(G)$, and assume to the contrary that M is not even a module of G. Then there must exist a *splitter* vertex $x \in V \setminus M$ for which the predecessor/successor relationships to vertices in M differ. As M is a module in $\mathcal{U}(G)$, x must be adjacent to all vertices in M. We can thus partition M into non-empty sets $M_P = M \cap P(x)$ and $M_S = M \cap S(x)$, where $P(x)$ and $S(x)$ denote the predecessors and successors of x, respectively. Since G is transitive, there must exist an edge from every vertex in M_P to every vertex in M_S. By Proposition 2, M must be a series module in $\mathcal{U}(G)$.

Denote by X the set of all vertices $x' \in V \setminus M$ with $M \cap P(x') = M_P$ and $M \cap S(x') = M_S$. In particular we have $x \in X$, so X is not empty. We claim that $M \cup X$ is a series module in $\mathcal{U}(G)$. To show this claim, let $y \in V \setminus (M \cup X)$. The relationships between y, X, M_P and M_S are visualized below.

If y is not adjacent to any vertex in $M \cup X$, then y does not split $M \cup X$. Otherwise, if y is adjacent to $x' \in X$, then either $(y, x') \in E$ or $(x', y) \in E$. As G is transitive, it follows that either $(y, s) \in E$ for all $s \in M_s$, or $(p, y) \in E$ for all $p \in M_p$. In both cases y is adjacent to a vertex in M.

Since M is a module in $\mathcal{U}(G)$, y must be adjacent to all vertices in M whenever it is adjacent to one. Since $y \notin X$, we either have $(y, p) \in E$ for some $p \in M_p$, or $(s, y) \in E$ for some $s \in M_S$. As G is transitive, it follows that either $(y, x') \in E$ for all $x' \in X$, or $(x', y) \in E$ for all $x' \in X$. In both cases, y is adjacent to all vertices in $M \cup X$, and hence does not split $M \cup X$ in $\mathcal{U}(G)$.

This shows the claim that $M \cup X$ is a module in $\mathcal{U}(G)$. Since every vertex in M is adjacent to every vertex in X, $M \cup X$ must be of the series type by Proposition 2. However, that would mean that both M and its parent node in the modular decomposition of $\mathcal{U}(G)$ are of the series type, which contradicts M being a strong module in $\mathcal{U}(G)$. This disproved our assumption that M is not a module in G.

Finally, since every module in G is a module in $\mathcal{U}(G)$, every weak (i.e., overlapping) module of G is a weak module of $\mathcal{U}(G)$. Thus M must be a strong module of G.

(only if) If M is a strong module in G, then M is clearly a module in $\mathcal{U}(G)$, and we only need to show that it is strong. Denote by M^+ the minimal strong module in $\mathcal{U}(G)$ containing M. If M^+ is prime, then $M = M^+$ follows immediately. If M^+ is parallel in $\mathcal{U}(G)$ then it is parallel in G, and so is M by Proposition 1, and $M = M^+$ follows again. Finally, if M^+ is series in $\mathcal{U}(G)$, then the child quotient graphs of M^+ and M in G are oriented cliques. Since G is acyclic, this orientation must induce a total order, so M^+ and M are both ordered modules in G, and $M = M^+$ follows once more. □

This immediately allows us to derive the modular decomposition of G from that of $\mathcal{U}(G)$.

Corollary 1. *Let $G = (V, E)$ be a transitive DAG. Then the modular decomposition tree of G can be obtained from the modular decomposition tree of $\mathcal{U}(G)$ by relabelling series nodes as ordered nodes.*

Proof: By Theorem 1 the modular decomposition trees of G and $\mathcal{U}(G)$ can only differ in their labelling. Prime and parallel nodes in $\mathcal{U}(G)$ are clearly prime and parallel nodes in G as well. As G is acyclic, series nodes in $\mathcal{U}(G)$ must be ordered nodes in G. □

3   Modular Decomposition w.r.t. Transitive Closure

Often the transitive graph we seek to decompose will not be given directly. Instead, the input is an arbitrary directed graph G, and we wish to find a modular decomposition of its transitive closure G^*.

This could be approached by first computing G^*, and then applying some modular decomposition algorithm. However, G^* can be much larger that G, especially if G is sparse. We therefore seek to pre-process G by identifying certain modules of G^*, which can then be used to reduce G before computing its transitive closure. While this does not improve the worst-case complexity for arbitrary graphs, it can greatly improve performance for sparse graphs, and is therefore highly relevant in practice.

We first identify the strongly connected components of G and contract them, in linear time. These are precisely the series nodes of G^*, which we insert into the modular decomposition tree afterwards.

Our main preprocessing algorithm then consists of two reduction rules, each of which merges two vertices into one. They are applied in any order until no more rule applications are possible – as we will show later, the resulting reduced graph is unique.

Rule 1 (sequential flow)**.**
If b is the only successor of a, and a the only predecessor of b, merge a and b.

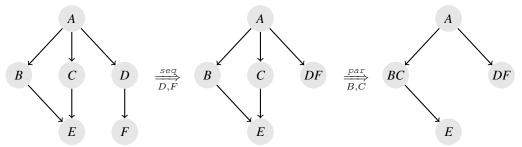**Rule 2** (parallel flow)**.**
If a, b with $a \neq b$ have the same sets of predecessors and successors, merge a and b.

One may easily observe that nodes merged using sequential flow form ordered modules w.r.t. G^*, while nodes merged using parallel flow are parallel modules w.r.t. G^*. As part of the reduction process, we may therefore construct an *unreduced* modular decomposition tree for each merged node-set. Unreduced modular decompositions may contain superfluous nodes for weak modules, but these can easily be eliminated by deleting ordered and parallel nodes with parents of the same type [10].

Applying these reduction rules as far as possible can be done in linear time, by keeping track of all possible rule application, with updates after each merge. Parallel nodes can be identified efficiently by maintaining a hash table, with the pairs of sets of predecessor and successor vertices as keys. To ensure

linear time complexity regardless of node degree, we identify predecessor / successor sets by a fixed-size hash value that can be updated in constant time when vertices are merged by either rule. One possible implementation is a simple XOR of cryptographic hash-values of vertices.

Example 6. *Consider once more the graph from Example 3. Initially, we can apply two reduction rules,* $seq(D, F)$ *and* $par(B, C)$, *in either order.*

After a reduction step, new rule applications involving the merged vertex may become available:

In this particular instance the graph reduces to a single vertex, making it trivial to decompose. □

We observe that no vertex can meet the conditions of both a sequential and parallel flow rule application at the same time. This simplifies processing as it restricts interaction between rule applications. We write $seq(a, b)$ ($par(a, b)$) to indicate that the conditions of the sequential (parallel) flow rule are met for a, b.

Lemma 1. *Let* a, b *be two vertices in* $G = (V, E)$ *with* $seq(a, b)$. *Then there does not exist any vertex* $x \in V$ *with* $par(a, x)$ *or* $par(b, x)$.

Proof: As a is the only predecessor of b, no other vertex x has the same successors as a. As b is the only successor of a, no other vertex x has the same predecessors as b. □

Next, we show that the order of rule application does not affect the final result. This property of rule systems is typically referred to as *confluence*.

Theorem 2. *The sequential/parallel flow rules are confluent.*

Proof: By Newman's diamond lemma [11] it suffices to show that the rules are locally confluent, i.e., that any two graphs obtained from a graph G via a single rule application can be reduced to the same graph by further rule applications. This is trivial if the two rule applications do not interact, and by Lemma 1 interaction can only occur for pairs of parallel or pairs of sequential rules that share a vertex. In both cases all three vertices involved in the pair of rule applications can be merged into a single vertex with another application of the same rule. □

Intuitively, our reduction steps identify non-prime modules in a bottom-up fashion, until they encounter a prime module or fail to identify an ordered or parallel node. While there is little to be done about prime modules, we find that failure to identify ordered or parallel nodes is always due to transitive edges.

Definition 3 (prime-free)**.**
We call a graph prime-free *iff it does not contain any prime modules. We call a module M of graph G* prime-free *iff $G|M$ is prime-free.*

Theorem 3. *Let G be acyclic and transitively reduced. Then sequential flow and parallel flow rules eliminate all prime-free modules of G^*.*

Proof: Since non-prime modules are made up of overlapping 2-sets of their children, all of which are modules, it suffices to show that all all modules of size 2 are merged. Denote by $\mathrm{anc}(x)$, $\mathrm{desc}(x)$ the ancestors and descendants of vertex x w.r.t. G, respectively. Now let $\{a, b\} \subseteq V$ be a module w.r.t. G^*. Then a, b have the same ancestors and descendants outside of $\{a, b\}$, that is

$$\mathrm{anc}(a) \setminus \{a, b\} = \mathrm{anc}(b) \setminus \{a, b\} \qquad \text{and} \qquad \mathrm{desc}(a) \setminus \{a, b\} = \mathrm{desc}(b) \setminus \{a, b\}$$

As G is acyclic, it cannot contain both arcs $a \to b$ and $b \to a$. This leaves two possibilities.

If G contains an arc between a and b, say $a \to b$, then $\mathrm{anc}(b) = \{a\} \cup \mathrm{anc}(a)$ and $\mathrm{desc}(a) = \{b\} \cup \mathrm{desc}(b)$. Since G is transitively reduced, it follows that a is the only parent of b, and b the only child of a. Thus a, b will be merged by the sequential flow rule.

If G contains no arc between a and b, then $\mathrm{anc}(a) = \mathrm{anc}(b)$ and $\mathrm{desc}(a) = \mathrm{desc}(b)$. Since G is transitively reduced, it follows that a and b share the same parents and children in G, and thus will be merged using the parallel flow rule. □

In particular we find that transitively reduced prime-free DAGs (e.g. all trees) are reduced to a single vertex. If G is transitively reduced but its modular decomposition contains a single prime-node as the root, then G reduces to the child-quotient graph of this prime node. In either case, no further decomposition is needed, provided it is known a priori that the reduced graph is prime.

4 Transitive Orientation of Complement Graphs

We now seek to find a transitive orientation of the complement of a transitive graph G^*, provided one exists. The challenge here is to do so in time (near-) linear in the size of G. We won't be able to achieve this for arbitrary digraphs, but we can for graphs G with transitively orientable complement.

The *Ordered Vertex Partitioning* (OVP) algorithm in [10] provides both a modular decomposition, and an orientation of a given undirected graph, with the provision that the orientation is transitive if the graph is transitively orientable. However, we do not wish to orient $\mathcal{U}(G^*)$, but its complement $\overline{G^*}^u$, which may be significantly bigger when G is sparse.

Luckily, it turns out that the OVP algorithm can easily be modified to operate on the complement of the given graph, by switching the role of neighbours and non-neighbours in the central *split* operation. As already observed in [1, Section 11.2], this does not affect the complexity of the algorithm, allowing us to compute a transitive orientation of $\overline{G^*}^u$ in time near-linear in the size of G^*.

The remaining problem is that G^* can be much larger than G, and while the reduction steps of Section 3 can be applied here as well (transitively orienting the complements of ordered or parallel quotient graphs is trivial), any approach involving transitive closure computation won't be able to achieve near-linear

complexity. To maintain the complexity bound one could simply apply the modified OVP algorithm to find a transitive orientation of $\mathcal{U}(G)$ rather than $\overline{G^*}^u$. As we will show in the following, this approach delivers a transitive orientation for $\overline{G^*}^u$ whenever $\mathcal{U}(G)$ is transitively orientable.
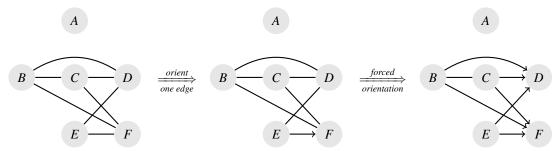
The basis for computing transitive orientations is the forcing relationship between edges, where orientation of one edge forces the orientation of another to maintain transitivity.

Definition 4 (forced orientation).
Let G be an undirected graph. We say that two edges $(a, b), (c, d)$ in G force each other, denoted as $(a, b) \ \Gamma \ (c, d)$, iff either $a = c$ and b, d are not adjacent in G_c, or $b = d$ and a, c are not adjacent in G. We denote the transitive closure of relation Γ by Γ^*.

It has been shown in [9, Proof of Lemma 2.1] that if G is prime, then orienting one edge forces the orientation of all other edges. Orienting the quotient graphs of series and parallel nodes is trivial.

Example 7. *Consider the graph G below, which is the undirected complement of the transitive closure of the graph from Example 3. If we pick an orientation of one edge, e.g. $E \to F$, then the orientation of most other edges is forced by it. E.g. $E \to F$ forces $E \to D$, which in turn forces $B \to D$.*

The only edge not forced this way is the one between B and C. This happens because G is not a prime graph, and BCE is a module of G. Orienting the missing edge as $B \to C$ yields the transitive orientation from Example 3. Orienting it as $C \to B$ yields another transitive orientation. □

A critical observation in relating transitive orientations of $\overline{G^*}^u$ and $\overline{G}^u$ is that the transitive closures of their forcing relationships are closely related.

Lemma 2. *Let $(a, b), (c, d)$ be two edges in $\overline{G^*}^u$ that indirectly force each other in $\overline{G^*}^u$, i.e., we have $(a, b) \ \Gamma^* \ (c, d)$. Then $(a, b), (c, d)$ indirectly force each other in $\overline{G}^u$.*

Proof: Let $(a, x), (b, x)$ be two edges in $\overline{G^*}^u$ that force each other directly. Then either (a, b) or (b, a) lies in G^*, say (a, b). This means G contains a path $a \to v_1 \to \ldots \to v_n \to b$, for some n.

Assume (x, v_i) does not lie in $\overline{G}^u$, for $1 \leq i \leq n$. Then either (x, v_i) or (v_i, x) must lie in G. If (x, v_i) lies in G, then (x, b) lies in G^*, contradicting $(b, x) \in \overline{G^*}^u$. If (v_i, x) lies in G, then (a, x) lies in G^*, contradicting $(a, x) \in \overline{G^*}^u$.

Hence (x, v_i) must lie in $\overline{G}^u$ for all $i = 1 \ldots n$, so the following forcing relationships hold in $\overline{G}^u$:

$$(a, x) \quad \Gamma \quad (v_1, x) \quad \Gamma \quad (v_2, x) \quad \Gamma \quad \ldots \quad \Gamma \quad (v_n, x) \quad \Gamma \quad (b, x)$$

This shows the claim for edges that directly force each other in $\overline{G^*}^u$. For edges that force each other indirectly, the claim then follows by transitivity of Γ^*. □

As a consequence of Lemma 2, any transitive orientation of $\overline{G}^u$ is a transitive orientation of $\overline{G^*}^u$ as well, so for graphs G where $\overline{G}^u$ is transitively orientable, we can find a transitive orientation of $\overline{G^*}^u$ in time near-linear in the size of G.

Theorem 4. *Any transitive orientation of $\overline{G}^u$ is a transitive orientation of $\overline{G^*}^u$.*

Proof: An orientation of an undirected graph is a transitive orientation iff all pairs of edges forcing each other are suitably oriented (i.e., according to the forcing relationship). By Lemma 2 any two edges in $\overline{G^*}^u$ forcing each other also force each other in $\overline{G}^u$, and thus are suitably oriented. □
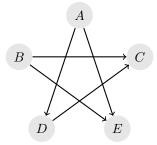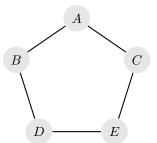
In particular this gives us:

Corollary 2. *Let G be a digraph such that $\overline{G}^u$ is transitively orientable. Then we can find a transitive orientation of $\overline{G^*}^u$ in near-linear time.*

Proof: As $\overline{G}^u$ is transitively orientable, we can find a transitive orientation of $\overline{G}^u$ in near-linear time using a modified OVP Algorithm. By Theorem 4 this gives us a transitive orientation of $\overline{G^*}^u$. □
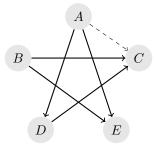
We must point out however that $\overline{G}^u$ may not be transitively orientable even though $\overline{G^*}^u$ is.
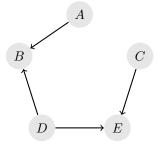
Example 8. *Consider the graph G shown below. Its complement $\overline{G}^u$ is shown on the right.*

Observe that $\overline{G}^u$ contains an odd-length cycle of forced edges, and thus cannot be transitively oriented. However, the transitive closure G^* of G does have a complement that is transitively orientable:

□

5   Conclusion

We have presented several results related to finding the modular decomposition of a transitive graph G, and a transitive orientation of its complement, in particular when G is given indirectly as the transitive closure of a sparse graph. This is particularly interesting for digraphs with transitively orientable complement, for which we can identify, in near-linear time, a pair of total orders characterizing the transitive closure. Among other applications, this allows answering of reachability queries in constant time.

References

[1] Elias Dahlhaus, Jens Gustedt, and Ross M. McConnell. Partially complemented representations of digraphs. *Discrete Mathematics & Theoretical Computer Science*, 5(1):147–168, 2002.

[2] Ben Dushnik and E. W. Miller. Partially ordered sets. *American Journal of Mathematics*, 63(3):600–610, 1941.

[3] Shimon Even, Amir Pnueli, and Abraham Lempel. Permutation graphs and transitive graphs. *J. ACM*, 19(3):400–410, 1972.

[4] T. Gallai. Transitiv orientierbare graphen. *Acta Mathematica Academiae Scientiarum Hungarica*, 18(1-2):25–66, 1967.

[5] Michel Habib, Fabien de Montgolfier, and Christophe Paul. A simple linear-time modular decomposition algorithm for graphs, using order extension. In *SWAT*, pages 187–198, 2004.

[6] Michel Habib and Christophe Paul. A survey of the algorithmic aspects of modular decomposition. *Computer Science Review*, 4(1):41–59, 2010.

[7] Ross M. McConnell and Fabien de Montgolfier. Linear-time modular decomposition of directed graphs. *Discrete Applied Mathematics*, 145(2):198–209, 2005.

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

[9] Ross M. McConnell and Jeremy P. Spinrad. Linear-time modular decomposition and efficient transitive orientation of comparability graphs. In *SODA*, pages 536–545, 1994.

[10] Ross M. McConnell and Jeremy P. Spinrad. Ordered vertex partitioning. *Discrete Mathematics & Theoretical Computer Science*, 4(1):45–60, 2000.

[11] M. H. A. Newman. On theories with a combinatorial definition of "equivalence". *Annals of Mathematics*, 43:223–243, 1942.

[12] Marc Tedder, Derek G. Corneil, Michel Habib, and Christophe Paul. Simpler linear-time modular decomposition via recursive factorizing permutations. In *Automata, Languages and Programming (ICALP)*, pages 634–645, 2008.