Dominance Constraints and Dominance Graphs

David Steurer

Saarland University

Abstract. *Dominance constraints* logically describe trees in terms of their adjacency and dominance, i.e. reachability, relation. They have important applications in computational linguistics. Unfortunately, the satisfiability problem of dominance constraints is NP-complete.
In this extended abstract we describe efficient algorithms for natural subclasses of dominance constraints [BDNM03]. The instances of these subclasses are more appropriately formulated as configuration problems of so called *dominance graphs*. We show that for one particular subclass satisfiability can be characterized in terms of forbidden subgraphs in the corresponding dominance graph [ADKMNT]. These forbidden subgraphs can be found in linear time [Th04].

1 Introduction

Dominance constraints are conjunctions of literals of the form $X_i \triangleleft (X_{j_1}, \ldots, X_{j_k})$, $X_i \triangleleft^* X_j$ and $X_i \neq X_j$ where $X_1, X_2, \ldots$ are *variables*.

A dominance constraint ϕ is said to be *satisfiable* if there exists a mapping α from the variables to the vertices of a directed tree τ satisfing all literals of ϕ. A *labeling literal* $X_i \triangleleft (X_{j_1}, \ldots, X_{j_k})$ is satisfied if $\alpha(X_{j_1}), \ldots, \alpha(X_{j_k})$ are exactly the children of $\alpha(X_i)$ in τ. Similarly, *dominance literals* $X_i \triangleleft^* X_j$ are satisfied if $\alpha(X_i)$ *dominates* $\alpha(X_j)$, i.e., vertex $\alpha(X_i)$ can reach $\alpha(X_j)$ in τ. Of course, an *inequality literal* $X_i \neq X_j$ is satisfied whenever X_i and X_j are mapped to distinct vertices.

To test whether a given dominance constraint is satisfiable is provably hard. Indeed, the set of satisfiable dominance constraints was shown to be NP-complete in [Ko99].

Consider a dominance constraint ϕ that contains inequality literals for each pair of distinct variables. It is known that deciding satisfiability of ϕ is linear-time equivalent to deciding *solvability* of a *dominance graph* G_ϕ. Thus we turn our attention to graphs.

Dominance graphs and their configurations are defined as follows.

Definition 1. *A* dominance graph *is a directed graph G with*

- tree arcs $T(G)$ *and*
- dominance arcs $D(G)$

such that $A(G) = T(G) \mathbin{\dot\cup} D(G)$ are the arcs of G.

We denote the directed graph induced by the tree arcs of G by

$$G^T := (V(G), T(G)).$$

Definition 2. *A* configuration *of G is a directed graph C such that*

- *C is a rooted tree on the vertices of G,*
- *G^T is a subgraph of C and*
- *the reachability relation of G is contained in the reachability relation of C.*

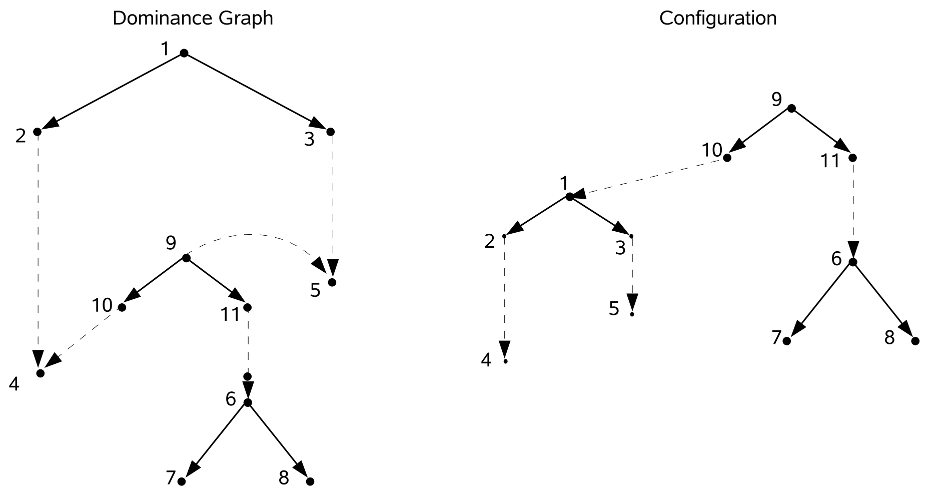
A dominance graph that has a configuration is called solvable.

Observe that the third condition in the definition of configuration can be replaced equivalently by the following.

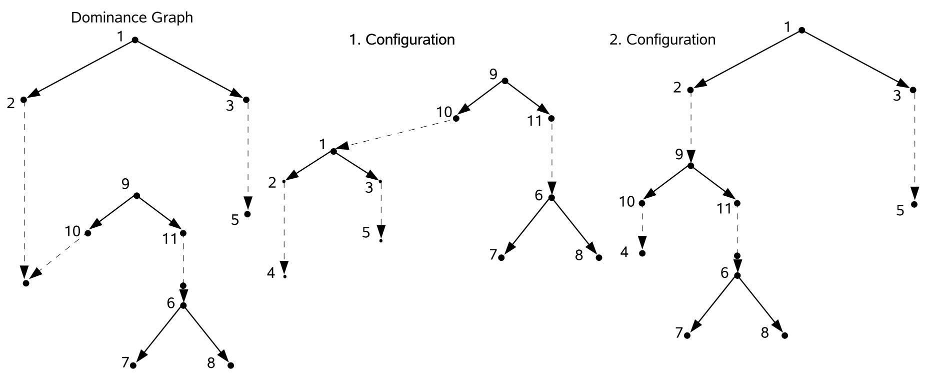
$$\forall (u, v) \in D(G).\ (u, v) \in \text{Reachability}_H.$$

For convenience we will treat all arcs of a configuration of G that are contained in G^T as tree arcs. All other arcs of the configuration are considered to be dominance arcs.

Example 1. Consider the following dominance graph G on the left and its unique configuration C on the right.



Clearly, C is a tree and contains G^T. The reachability condition can be verified easily. We want to point out that by deleting arc $(9, 5)$ we obtain a dominance graph with exactly two configurations. One is identical to the configuration above, the other has vertex 9 as its root. See figure below.



Problems From the definition of dominance graphs and configurations the following two problems arise naturally.

Problem 1 (Dominance Graph Configuration Decision Problem, DGCDP).
Given a dominance graph G, decide if there exists a configuration of G.

The corresponding enumeration problem is even more important for the applications in computational linguistics. It is defined as follows.

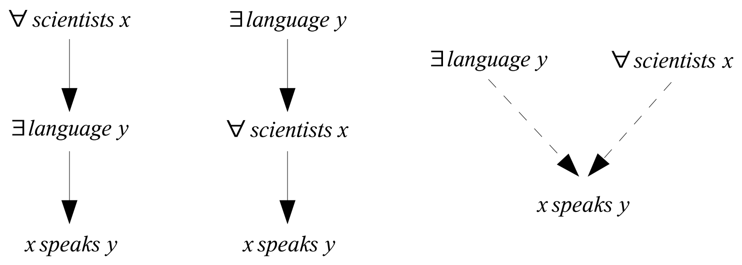
Problem 2 (Dominance Graph Configuration Enumeration Problem, DGCEP).
Given a dominance graph G, enumerate all configurations of G.

Applications. Dominance constraints have important applications in computational linguistics, especially in the context of semantic underspecification. It is often possible to consider dominance graphs instead of dominance constraints. The following very simple example tries to illustrate the connection between semantic underspecification, here underspecified quantifier scopes, and configuration problems on dominance graphs.

Example 2. Consider the sentence "all scientists speak a language". It has two possibile meanings.

In the first case the sentence means quite tautologically that all scientists speak a language but not necessarily the same. In the second case it means that there exists an universal language, for example English, that all scientists can speak.

Corresponding to these two meanings we can construct two trees as seen on the left of the following figure.



It happens to be that these trees are exactly the *configurations* of the dominance graph on the right of the figure above.

2 Algorithms for Dominance Graphs

The idea of the presented algorithm is as follows. First, it determines a set of vertices, called *free* vertices, that are candidates for being a root in a configuration. Then one of the free vertices, say u, is choosen to become the root of the computed configuration. By this choice the "position" of all vertices reachable from u within G^T, denoted by $\mathrm{tf}(u)$, gets determined. Therefore we can focus on the rest of the vertices. In other words, the algorithm deletes all the vertices $\mathrm{tf}(u)$. It can happen that by deleting these vertices the dominance graph breaks into parts, that are not connected in any way. For these parts we recursively compute configurations. In the last step we have to assemble the configurations of the parts and the tree induced by $\mathrm{tf}(u)$. In this step we have to make sure that all the dominance arcs between the parts and $\mathrm{tf}(u)$ are satisfied. We show that we can always do so. In this way we have computed a configuration of the original dominance graph.

The algorithm will report the original dominance graph G unsolvable if it encounters in some recursive step a subgraph of G exhibiting free vertices. We show that this is correct.

We also show that for each minimal solved form S there exists a way to choose free vertices such that the algorithm will output S. This implies that we can solve the enumeration problem efficiently.

2.1 Solved Forms

Since enumeration of all configurations of G may be intractable, we first introduce the more general concept of *solved forms* and then determine the important solved forms, namely *minimal solved forms.*

We will use the following natural partial order on dominance graphs to define both solved forms and minimal solved forms.

Definition 3. *A dominance graph G is* less specific *than dominance graph G' if $G^T = G'^T$ and the reachability relation of G is properly contained in the reachability relation of G'.*

Example 3. If C is a configuration of G then C is more specific than G.

Example 4. If (u, w) is a dominance arc of G and v is a vertex that can be reached from u in G by a path not containing (u, w) then replacing (u, w) by dominance arc (v, w) yields a dominance graph that is more specific than G (cf. Lemma 1.2).

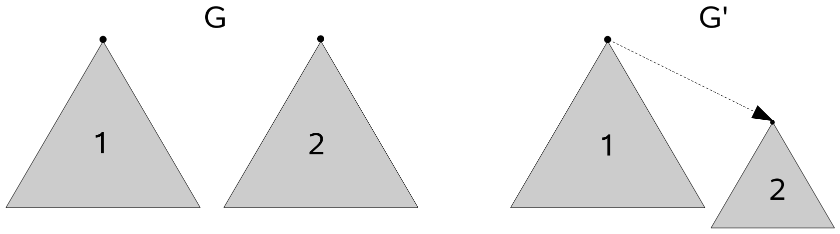
We can now formulate a slight generalization of configurations in terms of specificness.

Definition 4. *A* solved form *of a dominance graph G is a directed forest, that is more specific than G.*

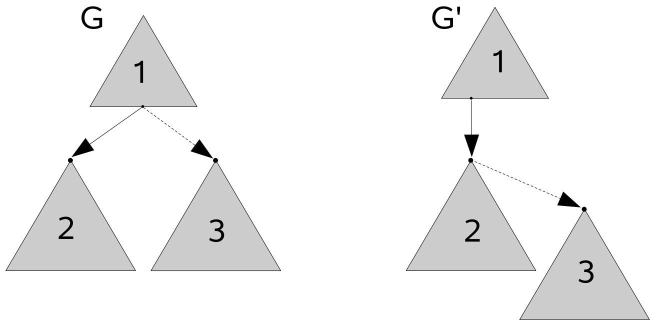
The next lemma shows that solved forms which are comparable by the partial order defined above do not differ much with respect to two *primitive transformations.*

Lemma 1. *From a solved form S any more specific solved forms S' can be generated by a sequence of $\mathcal{O}(n^2)$ transformations of the following kinds.*

1. *For two distinct roots r, r' in S we obtain a more specific solved form by adding dominance arc $e = (r, r')$.*



2. *For an arc $e = (u, v)$ and a dominance arc $f = (u, w)$ we obtain a more specific solved form when we replace f by $f' = (v, w)$.*



Proof. We do induction on the cardinality of $\text{Reachability}_{S'} \setminus \text{Reachability}_{S}$.

If the difference set is empty then, since $S \preceq S'$, both reachability relations are identical. Therefore the solved forms are identical.

Suppose the difference set is non-empty. Let u be a vertex of minimal depth in S, that can reach some vertex, v say, in S' but not in S. If u is a root we can simply connect u to the tree containing v by the first transformation.

Otherwise, if u is not a root then its parent w can reach v in S (since w can reach u and therefore v in S' and since w has smaller depth than u in S). Let $f = (w, w')$ be the first arc on the path connecting w to v in S. If f is a dominance arc we can apply the second transformation and replace f by $f' = (u, w')$ to obtain a solved form that is more specific than S and at most as specific as S'. Otherwise, if f is a tree arc then (w, u) must be a dominance arc (else S' would not be at least as specific as S). Again, we apply the second transformation but replace (w, u) by (w', u). This also yields a more specfic solved form that is at most as specific as S'.

In all the cases we obtained a solved form $\hat{S}$ with $S \prec \hat{S} \preceq S'$. Thus, we can apply induction hypothesis on $\hat{S}$. Since there are at most n^2 pairs in the reachability relation of S', we can increase the reachability relation at most n^2 times. □

Hence it suffices to look at solved forms that are minimal with respect to the partial order defined above.

Definition 5. *A* minimal solved form *of G is a least specific solved form of G.*

The minimal solved forms are natural representatives of the equivalence classes modulo the two primitive transformations of the lemma before.

We replace the enumeration problem DGCEP by the following more appropriate problem.

Problem 3 (Dominance Graphs Minimal Solved Forms (DGMSF)).

Given a dominance graph G, list all its minimal solved forms.

2.2 Basic Properties and Reductions

Within the algorithm we want to report the dominance graph G unsolvable as soon as we encounter a subgraph of G that is unsolvable. The next lemma proves this to be correct.

Lemma 2. *If a subgraph $H \subseteq G$ is unsolvable then so is G.*

Proof. If G has solved form S then $S[V(H)]$ is a forest that can be extended to a solved form of H by adding appropriate dominance arcs.

For convenience we exclude some degenerate cases from our considerations.

Definition 6. *A dominance graph G is* well-formed *if G^T is a forest and only roots of G^T have incoming dominance edges.*

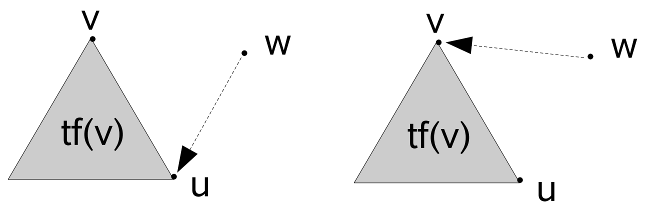
We observe that we can assume G to be well-formed without loss of generality.

Let $\mathrm{tf}(v)$ denote the *tree fragment* of v, i.e. the set of descendants of v in G^T.

Lemma 3. *The problems DGCDP and DGMSF for general dominance graphs and its restrictions to well-formed dominance graphs are linear time equivalent.*

Proof. If G^T is not a forest then G is clearly unsolvable.

Suppose v is a root in G^T and there exists a nonroot $u \in \mathrm{tf}(v)$ with incoming dominance arc (w, u). If w is contained in $\mathrm{tf}(v)$ then (w, u) is either implied by the tree arcs or it contradicts the tree arcs. Therefore suppose w is not contained in $\mathrm{tf}(v)$. Then, we can replace (w, u) by dominance arc (w, v) without changing solvability or solved forms, since in every solved form of G vertex w has to reach u through vertex v.



Thus, by scanning all the dominance arc we can get rid of all the dominance arcs incoming to nonroot vertices.

□

From now on we only consider well-formed dominance graphs.

2.3 Key Lemma

The following lemma is equally simple and powerful. The important concept of free vertices that is introduced in the next subsection heavily relies on it.

Lemma 4 (Key Lemma).
If G' is a solved form of G and $P = \langle v_1, \ldots, v_k \rangle$ is a undirected path in G then one of the v_i is in G' a (not necessarily proper) anchestor of both v_1 and v_k.

Proof. We do induction on the length of the path. Clearly, the statement holds true for $k = 1, 2$. For $k > 2$ apply induction hypothesis on $P_1 = \langle v_1, v_2 \rangle$ and $P_2 = \langle v_2, \ldots, v_k \rangle$. We obtain vertices v_{i_1} and v_{i_2}. Since v_{i_1} and v_{i_2} have common descendant v_2 and since G' is a forest one of v_{i_1} and v_{i_2} must be an anchestor of the other and this vertex is then a common anchestor of v_1 and v_k. □

When applying the Key Lemma the following terminology turns out to be very convenient.

Definition 7. *Vertices u and v are* weakly connected *in G if there is a undirected path in G connecting them. The* weakly connected components (wccs) *of G are the inclusionwise maximal weakly connected vertex sets.*

From the Key Lemma we get the following easy fact.

Corollary 1. *The weakly connected components of G and any minimal solved form of G are the same.*

2.4 Free Vertices

For computing solved forms of a dominance graphs it will be a crucial step to determine the possible roots in the solved forms. It will turn out that we can determine the set of possible roots efficiently, because there is a good characterization of possible roots captured in the definition of *free vertices*.

Definition 8. *A root v in G^T is* free *in dominance graph G, if*

- *v has no incoming dominance arc and*
- *for all distinct $d, d' \in \mathrm{tf}(v)$ each undirected path connecting d and d' in G contains at least one anchestor of d or d' in G^T.*

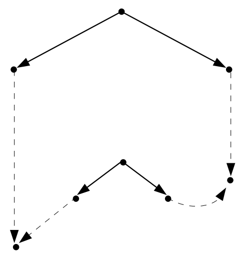
The following lemma enables us to exclude candidates for roots of minimal solved forms. To prove it we need to apply the Key Lemma.

Lemma 5. *Any root v in a solved form G' of G is free in G.*

Proof. Clearly, if v has any incoming arc then v cannot be a root in G'. Otherwise, if two distinct descendants d and d' of v are connected by a path P not containing any anchestor of d or d' in G^T then by the Key Lemma in all solved forms S of G a vertex $u_S \in P$ must be an anchestor of both d and d'.

Since $u_S \notin \mathrm{tf}(v)$ it must be a proper anchestor of v in S. And therefore v cannot be a root in S. □

Example 5. The following dominance graph has no free vertices. Thus it is unsolvable and so are all its supergraphs.



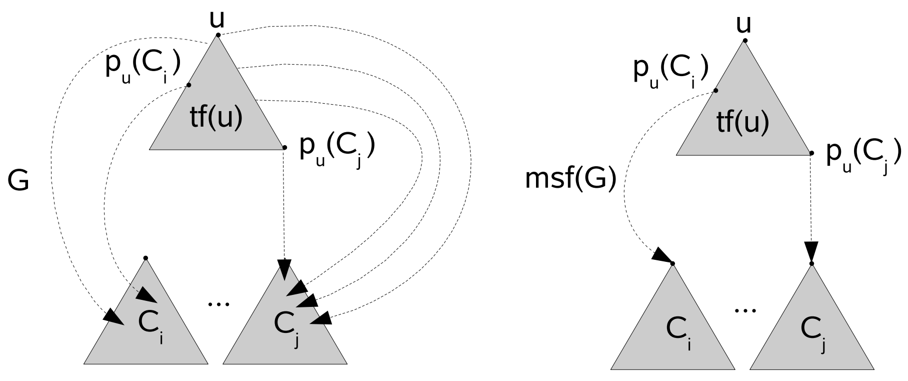
On the other hand, if a vertex is free then we can delete its tree fragment and recurse without getting troubles afterwards. Therefore it's always safe to choose a free vertex as root for a solved form. The following lemma shows that in the graph without the tree fragment for each weakly connected componented there is a unique way to connect the solved form of the component to the tree fragment such that all dominance arcs between the tree fragment and the component are satisfied.

We say a vertex dominates a vertex set C in G if it has a outgoing dominance arc in G directed to a vertex of C.

Lemma 6. *If u is free in G and C is a weakly connected component in $G \backslash \mathrm{tf}(u)$ then there is a unique vertex $p_u(C) := v \in \mathrm{tf}(u)$ dominating C such that any other vertex $w \in \mathrm{tf}(u)$, that dominates C, is a proper anchestor of v in G^T.*

Proof. Assume that there are distinct $v, w \in \mathrm{tf}(u)$ dominating C such that neither is an anchestor of the other in G^T. We have to prove that u cannot be free in G. This is clearly the case. Since C is weakly connected in $G \backslash \mathrm{tf}(u)$ and since v and w dominate C there is a undirected path from v to w containing only vertices of C besides v and w. This path obviously cannot contain an anchestor of v or w. Thus u is not free. □

For further illustration see also the figure below.



```
msf(G) ≡
    if $\mathcal{G}$ is not weakly connected then
        recurse on $G[C_i]$ for wccs $C_i$ of $G$
    else if there exists no free vertex in $G$ then
        print "$G$ and all its supergraphs are unsolvable"
    else
        // m.s.f. of G must be a tree
        choose some free vertex $u$
        let $C_1, \ldots, C_l$ be wccs of $G \backslash \mathrm{tf}(u)$
        recursively compute minimal solved forms $G'_i$ of $G[C_i]$ with roots $r_i$
        return $G'$ with $D(G') = \bigcup_i D(G'_i) \bigcup_i \{(p_u(C_i), r_i)\}$
```

Fig. 1. Generic algorithm for computing minimal solved forms of dominance graphs.

The following two lemmata establish the correctness of the algorithm.

Lemma 7. *For each minimal solved form G' of G there is a way to choose free vertices such that* `msf`*(G) returns G'.*

Proof. By Lemma 5 we can always choose the roots of G' as free vertices. Then we proceed recursively. In order to satisfy the dominance conditions between component any C_i and the tree fragment $\mathrm{tf}(u)$ we have to add a dominance arc from some descendant of $p_u(C_i)$ to the root r_i of C_i. The dominance arc $(p_u(C_i), r_i)$ yields minimum possible specificness. Therefore it must be present in the minimal solved form G'. □

Lemma 8. *For each way to choose free vertices the dominance graph returned by* `msf`*(G) is a minimal solved form of G. If there is no graph returned the input dominance graph is unsolvable.*

Proof. Clearly, unsolvability is reported correctly. Furthermore, by Lemma 6 all dominance conditions between $\mathrm{tf}(u)$ and component C_i are satisfied when the arc $(p_u(C_i), r_i)$ is added. As seen in the proof of Lemma 7, adding these arcs also yields the minimum possible specficness. Thus the output is a minimal solved form.

Clearly, algorithm `msf` runs in polynomial time.

Corollary 2. *The problems DGCDP and DGMSF can be solved in polynomial time.*

3 More Efficient Algorithms

In the section before we have seen that configuration problems on general dominance graph indeed have polynomial time complexity in contrast to problems on general dominance constraints.

Considering a important subclass of dominance graphs we can obtain algorithms with running time $\mathcal{O}(nm)$ per solved form. Further restrictions even enable us to test satisfiability in linear time.

Lemma 9. *If the trees in G^T have height at most 1 then a vertex $v \in V(G)$ is free in G if and only if*

- *v has no incoming arc in G*
- *all outgoing tree arcs of v are contained in pairwise distinct biconnected components of the undirected graph underlying G.*

Proof. We only have to consider the second condition.

If two tree arcs e, f of v are in the same biconnected component with respect to the undirected graph underlying G then there is a simple undirected cycle C in G containing both e and f. Of course, e and f are adjacent in C. Deleting e and f from C yields a undirected path P connecting two distinct children of v with $v \notin P$. Thus v is not free.

On the other hand, a simple undirected path connecting two distinct children of v not containing v can be extended to a simple cycle by adding two tree arcs of v. Now, these two tree arcs are contained in the same biconnected component.

Lemma 10. *If trees in G^T have height at most 1 then we can solve DGCDP in time $\mathcal{O}(n \cdot m)$ and DGMSF in time $\mathcal{O}(nm \cdot \#\text{minimal solved forms})$.*

Proof. In each recursive call of `msf` at least one vertex is deleted. Thus there are $\mathcal{O}(n)$ recursive calls.

In order to find the free vertices we just have to compute the biconnected components. This can be done in linear time. Also, all other steps in a recursive call of `msf` can be done in linear time.

Therefore we have a running time of $\mathcal{O}(nm)$ per solved form. □

Remark 1. We can obtain running times $\mathcal{O}(m + n \cdot \text{polylog}(n))$ respectively $\mathcal{O}(m + n \cdot \text{polylog}(n) \cdot \#\text{min. solved forms})$ if decremental biconnectivity algorithms are used.

3.1 Linear Time Decision Algorithm for Normal Dominance Graphs

Definition 9. *A well-formed dominance graph is normal if all trees in G^T have height 1 and the roots of G^T have no outgoing dominance arcs.*

For this class of dominance graph there exists a very nice charaterization for solvability based on the following kind of cycle.

Definition 10. *A* harmful cycle *in G is a undirected simple cycle not containing a vertex and two of its outgoing dominance arcs.*

One entailment is easy.

Lemma 11. *A normal dominance graph G containing a harmful cycle is unsolvable.*

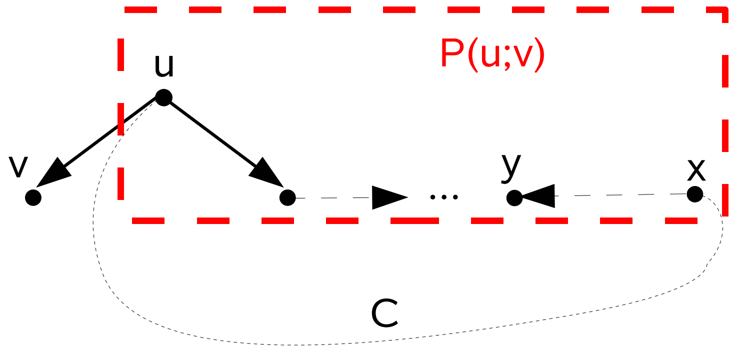
Proof. We show that a harmful cycle C has no free vertices. Let u be any vertex in C with no incoming arc. By definition of a harmful cycle it cannot have two outgoing dominance arcs. Since the host dominance graph G was normal it is not possible that u has an outgoing tree arc and an outgoing dominance arc. Thus u has two outgoing tree arcs directed to w_1, w_2. By deleting u from C we obtain an undirected path connecting w_1, w_2 not containing anchestor u of w_1, w_2. Therefore u is not free. Thus C is unsolvable and so is G.

The proof of the next lemma is more involved. Therefore we will omit it.

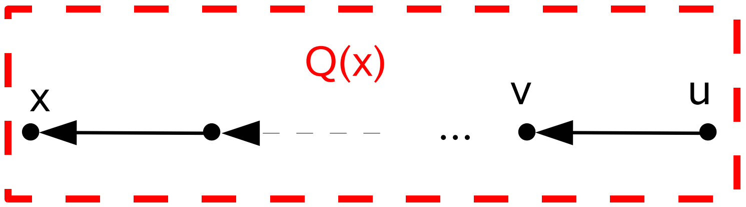
Lemma 12. *If a normal dominance graph G is not solvable then it contains a harmful cycle.*

Proof. We will prove that a dominance graph G not containing free vertices contains a harmful cycle.

The following three notions will be used. First, a undirected path is called *harmful* if it does not contain two outgoing dominance arcs of a vertex. Second, we define for all roots without incoming dominance arcs a maximal harmful walk. Let u be a root in G^T without incoming dominance arcs and (u, v) be a outgoing tree arc of v. Since u is not free there is a simple undirected cycle C containing two of its tree arcs. If C is harmful we are done. Suppose C is not harmful. Then we can extract from C a harmful undirected path $P(u;v)$ starting from u, not containing tree arc (u,v) and ending with a dominance arc (x,y) such that leaf x is the last vertex on $P(u;v)$ (cf. figure below).

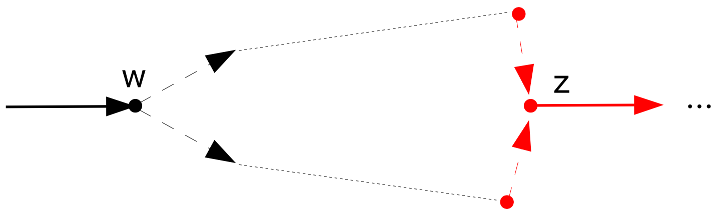


Third, we define maximal harmful paths for all leafs. Let x be a leaf in G^T. Then $Q(x)$ is a path following incoming arcs until a root without incoming dominance arc is reached. Clearly, $Q(x)$ is harmful path (cf. figure below).



Note the following two facts. If x is the last vertex on $P(u;v)$ then $P(u;v) \circ Q(x)$ is a harmful walk. Observe that adjacent arcs in $P(u;v) \circ Q(x)$ are distinct. Similarly, if (u,v) is the last tree arc in $Q(x)$ then $Q(x) \circ P(u;v)$ is a harmful walk where adjacent arcs are distinct.

Therefore starting at a root without incoming dominance arcs we can construct a harmful path until we reach a vertex, say w, which we already have seen before. This will close a harmful cycle unless the walk contains two outgoing dominance arcs of w (cf. figure below).



As shown in the picture, if two outgoing dominance arcs of w are contained in our walk then a vertex z and two of its incoming arcs are contained in the walk[1]. Thus we can continue constructing a harmful walk from z on. Note that the tree arc of z was not used before in the walk

This process eventually terminates because every time we "restart" our construction we see a new tree arc.

It is also easy to see that we found a harmful cycle when the process stops. This is because we maintain the invariant that we can reach the last vertex of the walk by a harmful path from all vertices seen so far in the process. □

Remark 2. Using a DFS-based cycle detection algorithm we can solve DGCDP for normal dominance graphs in linear time.

4 Conclusion

Using Dynamic Programming one can derive an algorithm that is more efficient in practice and solves the important problem of counting all minimal solved forms of a dominance graph.

We can enrich dominance graphs by disjointness edges. Then a disjointness edge (u, v) in G demands that in any configuration of G the subtrees rooted in u and v must be disjoint. When we refine the freeness conditions appropriately we can solve DGCDP and DGMSF for dominance graphs with disjointness edges using similar algorithmic ideas.

References

[ADKMNT] Ernst Althaus, Denys Duchier, Alexander Koller, Kurt Mehlhorn, Joachim Niehren, Sven Thiel: *An Efficient Graph Algorithm for Dominance Constraints.* Journal of Algorithms, to appear.

[BDNM03] M. Bodirsky, D. Duchier, J. Niehren, S. Miele. *A New Algorithm for Normal Dominance Constraints.* Symposium on Discrete Algorithms, 2003.

[Th04] S. Thiel. *Efficient Algorithms for Constraint Propagation and for Processing Tree Descriptions.* Doctoral Dissertation, Saarland University, 2004.

[Ko99] Alexander Koller. *Constraint Languages for Semantic Underspecification.* Diplomarbeit, Universität des Saarlandes, 1999.

[1] This is because we had to change the arc direction before we reached w the second time. In a normal dominance graph this implies the situation