A New Algorithm for Normal Dominance Constraints

Manuel Bodirsky, Denys Duchier, Joachim Niehren, Sebastian Miele

A New Algorithm for Normal Dominance Constraints

Manuel Bodirsky* Denys Duchier† Joachim Niehren‡ Sebastian Miele§

November 12, 2003

Dominance constraints are logical descriptions of trees. Efficient algorithms for the subclass of *normal dominance constraints* were recently proposed. We present a new and simpler graph algorithm solving these constraints more efficiently, in quadratic time per solved form. It also applies to weakly normal dominance constraints as needed for an application to computational linguistics. Subquadratic running time can be achieved employing decremental graph biconnectivity algorithms.

1 Introduction

Dominance constraints are logical descriptions of trees [2, 10] that can talk about the mother and ancestor relations between the nodes of a tree. They have numerous applications, most of which belong to the area of computational linguistics, e.g. in underspecified semantics [4–6], underspecified discourse [7], and parsing with tree adjoining grammar [12].

Satisfiability of dominance constraints was proved NP-complete in [9]. This shed doubts on the feasibility of dominance based applications. The doubts were removed by Mehlhorn et al [1], who distinguished the language of *normal dominance constraints* which suffices for many applications and has a polynomial time satisfiability problem.

The most relevant problem for normal dominance constraints is to enumerate solved forms, i.e., all trees satisfying a constraint. Mehlhorn et al [1] presented an enumeration algorithm whose running time is $O(n^4)$ per solved form. This algorithm relies on an efficient satisfiability test. Thiel [13] improved this result to $O(n^3)$ by faster satisfiability testing.

In this paper, we propose a novel graph algorithm relying on graph connectivity, and inspired by [3]. It can enumerate all solved forms of a normal dominance constraint in $O(n^2)$ per solved form, and thereby improves on the best previously known algorithm in efficiency. Subquadratic running time can be achieved employing decremental graph biconnectivity algorithms.

Our algorithm applies to the extended language of *weakly normal dominance constraints* that we introduce. This improves the applicability of dominance constraints in the area of natural language semantics. It underlies the first polynomial time algorithm for *minimal recursion semantics (MRS)* [5] presented in a follow up paper [11]. MRS is built into *head driven phrase structure grammar (HPSG)*, one of the mainstream grammar formalisms in computational linguistics.

*Humboldt Universität zu Berlin, Germany
†LORIA, Nancy, France
‡INRIA team Mostrare, Université de Lille, France
§Universität des Saarlandes, Germany

2 Weakly Normal Dominance Constraints

We start with a brief exposition of dominance constraints [2, 10], recall the notion of normality [1], and then introduce weak normality.

Let f, g range over the elements of some signature of function symbols with fixed arities and a, b over constants, i.e., function symbols of arity 0. A *constructor tree* over this signature is a ground term τ constructed from the function symbols, as for instance $f(g(a, a))$. We identify ground terms with trees that are rooted, ranked, edge-ordered, and node-labeled. See Fig. 1 for a graphical representation of the tree $f(g(a, a))$.

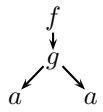


Figure 1: $f(g(a, a))$

Dominance constraints are logic formulas that describe constructor trees. They can talk about the mother-child relation $\triangleleft$ between the nodes of a tree, *dominance* $\triangleleft^*$ which is the reflexive ancestor relation, and inequality $\neq$. Let X, Y, Z range over an infinite set of *node variables*. A *dominance constraint* ϕ is then a conjunction of literals:

$$\phi \quad ::= \quad X{:}f(Y_1, \ldots, Y_n) \;\mid\; X \triangleleft^* Y \;\mid\; X \neq Y \;\mid\; \phi' \wedge \phi''$$

A solution of a dominance constraint consists of a tree τ and a variable assignment α to the nodes of τ. A

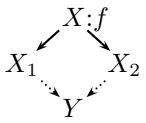


Figure 2: Unsatisfiable constraint: $X{:}f(X_1, X_2) \wedge X_1 \triangleleft^* Y \wedge X_2 \triangleleft^* Y$

labeling literal $X{:}f(Y_1, \ldots, Y_n)$ is satisfied if $\alpha(X)$ is labeled by f in τ and has the children $\alpha(Y_1), \ldots, \alpha(Y_n)$ in that order. A *dominance literal* $X \triangleleft^* Y$ requires that $\alpha(X)$ dominates $\alpha(Y)$ in τ. An *inequality literal* $X \neq Y$ requires $\alpha(X)$ and $\alpha(Y)$ to be distinct.

The constraint of Fig. 2 requires that the node values of X_1 and X_2 are sisters, and ancestors of the node value of Y. This is clearly impossible in a tree since the subtrees rooted in two sister nodes, the values of X_1 and X_2, are necessarily disjoint and therefore cannot share a common node Y.

Definition 2.1. A dominance constraint ϕ is *normal* if it satisfies:

1. (a) each variable of ϕ occurs at most once in the labeling literals of ϕ. A variable Y_i is a *hole* of ϕ whenever it occurs at the right of some labeling literal $X{:}f(Y_1, \ldots, Y_n)$ of ϕ; else it is a *root* of ϕ.
   (b) each variable of ϕ occurs at least once in the labeling literals of ϕ.
2. if X and Y are distinct roots in ϕ, $X \neq Y$ occurs in ϕ.
3. (a) if $X \triangleleft^* Y$ occurs in ϕ, Y is a root in ϕ.
   (b) if $X \triangleleft^* Y$ occurs in ϕ, X is a hole in ϕ.

A dominance constraint is *weakly normal* if it satisfies all above properties except for 1.(b) and 3.(b).

Dropping 3.(b) allows root-to-root dominances which enable the applications to MRS discussed in Section 5. Dropping condition 1.(b) adds convenience. The unsatisfiable constraint in Fig. 2, for instance, has holes X_1 and X_2 and roots X and Y. Extended by the entailed inequality $X \neq Y$, this constraint becomes weakly normal, but not normal as root Y violates condition 1.(b).

Compactification. Condition 1.(a) could be relaxed in order to support nested labelings, such as $X_1{:}f(Y_1, X_2) \wedge X_2{:}g(Y_2, Y_3)$ modeling the *nested* term $f(Y_1, g(Y_2, Y_3))$. We do not do so here, as nested labeling can always be eliminated by *compactification*: The idea is to replace $X_1{:}f(Y_1, X_2) \wedge X_2{:}g(Y_2, Y_3)$ by $X_1{:}h(Y_1, Y_2, Y_3)$ where h is a new function symbol. In general, compactification preserves (un)solvability.

3 Dominance Graphs

We now shift the view from weakly normal dominance constraints to dominance graphs. Solutions of dominance constraints will correspond to solved forms of dominance graphs.

Definition 3.1. *A* dominance graph $(V, \triangleleft^* \uplus \triangleleft)$ *is a directed graph with two kinds of directed edges,* dominance edges $(X, Y) \in \triangleleft^*$ *drawn as dotted arrows* $X \cdots\!\!\rightarrow Y$ *and* tree edges $(X, Y) \in \triangleleft$ *drawn as solid arrows* $X \longrightarrow Y$. *Subgraphs of the following forms are forbidden in dominance graphs:*

To every weakly normal dominance constraint we assign a unique dominance graph. The nodes of the graph of ϕ are the variables of the constraint ϕ. Labeling literals $X{:}f(X_1, \ldots, X_n)$ of ϕ contribute *tree edges* $X \longrightarrow X_i$ for $1 \leq i \leq n$, and dominance literals $X \triangleleft^* Y$ *dominance edges* $X \cdots\!\!\rightarrow Y$. Weak normality (Def. 2.1) ensures the absence of forbidden subgraphs (Def. 3.1).

We frequently draw dominance constraints as dominance graphs, as for instance in Fig. 2. In such drawings, we may freely include node labels f and the order of children in labeling literals even though these are ignored by the formal definition of dominance graphs. Note that inequality literals are also ignored.

The *reachability relation* R_G of a dominance graph $(V, \triangleleft^* \uplus \triangleleft)$ is the reflexive transitive closure of $\triangleleft^* \uplus \triangleleft$. A dominance graph G is *less specific* than G' if:

- G and G' differ only in their sets of dominance edges, and
- the reachability relation is extended: $R_G \subseteq R_{G'}$

Definition 3.2. *We call a dominance graph* $(V, \triangleleft^* \uplus \triangleleft)$ *a* solved form *if it is a forest, i.e., a collection of rooted trees. A* solved form of a dominance graph G *is a solved form that is more specific than* G. *We call a dominance graph* solvable *if it has a solved form.*

A *minimal solved form (of* G*)* is a solved form (of G) that is minimal with respect to specificity. Fig. 3 shows a minimal solved form and two others that are more specific. Fig. 4 presents a dominance graph and its two minimal solved forms.

4 Minimal Solved Forms versus Solutions.

In general, the solutions of a normal dominance constraint are partitioned by the minimal solved forms of its graph. This is important, as every dominance constraint has infinitely many solutions but only finitely many solved forms.

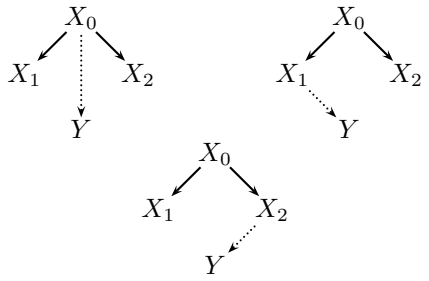


Figure 3: Three solved forms of which the left one is least specific.

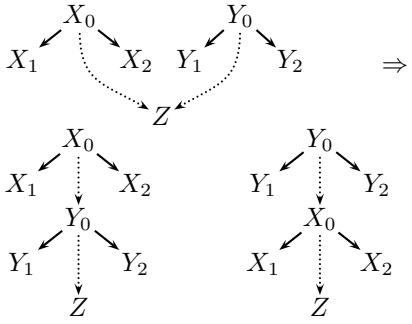


Figure 4: A dominance graph with two minimal solved forms.

For lack of space, we do not prove this claim here. Instead, we formalize the equivalence between satisfiability and solvability (Prop. 4.1 below). We call ϕ *well-formed* if it does not contain any subconstraint of the form $X{:}a \wedge X{\triangleleft^*}Y$. This does not restrict generality as every constraint can be made well-formed in a linear time: suppose that ϕ contains a subconstraint $X{:}a \wedge X{\triangleleft^*}Y$. If X is the same variable as Y then we can remove $X{\triangleleft^*}Y$. Otherwise, X and Y are distinct roots of ϕ so that $X{\neq}Y$ belongs to ϕ; this is unsatisfiable since X must denote a leaf (which does not properly dominate any other node).

Proposition 4.1. *A well-formed weakly normal dominance constraint is satisfiable (has a solution) if and only if its dominance graph is solvable (has a solved form).*

Proof. From a solution of a dominance constraint, one can easily read off a solved form of the corresponding dominance graph. It is sufficient to ignore all nodes of solutions that are not values of variables (together with all adjacent edges) and all redundant dominance edges induced by the solution. Vice versa, one can construct a solution for a solved form of a weakly normal dominance constraint inductively top down, as long as the constraint is well-formed. New function symbols are to be introduced for labeling leaf variables in order to satisfy multiple outgoing dominance wishes via branching. The complete proof for normal dominance constraints is elaborated in [1]. We omit the details for the weakly normal case. □

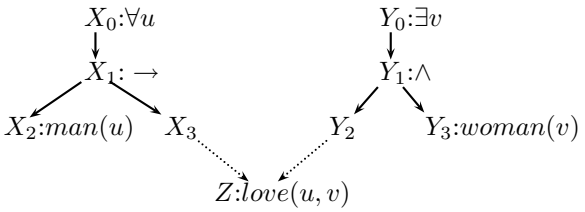


Figure 5: Dominance graph for *Every man loves a woman* (before compactification)

5 Applications

Dominance constraints have multiple applications in computational linguistics [10]. Here we focus on modeling semantic underspecification of scope in two independent approaches: the constraint language for lambda structures (CLLS) [6], a general framework for semantic underspecification, and MRS [5], a less general approach but more popular in practice, in particular in larger scale *head driven phrase structure grammars (HPSG)*.

The famous example *every man loves a woman* illustrates the notorious problem of scope ambiguity: either each man loves a possibly distinct woman $\forall u(man(u) \rightarrow \exists v(woman(v) \wedge love(u,v)))$ or there is a woman that is loved by every man $\exists v(woman(v) \wedge \forall u(man(u) \rightarrow love(u,v)))$.

The dominance graph in Fig. 5 is derived by a CLLS semantics construction [6]; it becomes normal by compactification. The two readings of the sentence correspond to the minimal solved forms of this dominance graph. The MRS approach [5] is similar to CLLS but differs in two respects. First, MRS semantics constructs weakly normal dominance graphs, and second, these graphs are understood to describe configurations rather than minimal solved forms.

Configurations are particular solved forms, where nodes cannot have more than one outgoing dominance edge. "Closed" leaves labeled by a constant have no outgoing dominance edges. The restriction to such configurations matters: it is NP-complete to decide whether a solved form permits a configuration (because of closed leaves in contrast to [13], see Section 10 of [1]). Nevertheless, Niehren and Thater [11] distinguish a large fragment of MRS graphs, where all minimal solved forms are configurations. This result relies on the algorithm presented here, and for the first time formally

treats the relationship between MRS and CLLS.

6 Weak Connectedness

We reduce solvability of dominance graphs to solvability of weakly connected dominance graphs. The constraint graph $G = (V, \lhd^* \uplus \lhd)$ is *weakly connected* if for any two nodes X and Y there is an undirected path from X to Y in $\lhd^* \uplus \lhd$. A weakly connected component (wcc) of G is a maximal weakly connected subgraph of G. Given a set of nodes $V' \subseteq V$, we write $G|_{V'}$ for the restriction of G to nodes in V' and edges in $V' \times V'$. The wccs of $G = (V, \lhd^* \uplus \lhd)$ form a proper partition of $V, \lhd^*$ and $\lhd$.

Proposition 6.1. *A dominance graph is solvable if and only if all its weakly connected components are solvable.*

Proof. Let G be a dominance graph. If all wccs are solvable then we can choose some solved form for each component. The union of these solved forms is a solved form of G. Conversely, if G' is a solved form of G, and W are the nodes of a wcc of G, then $G'|_W$ is a solved form of $G|_W$. □

We will now prove that a solved form of a weakly connected dominance graph is a rooted tree. This is equivalent to the following lemma, which states the key inductive property underlying the proofs of this paper.

Lemma 6.1. (Key) *Let $G = (V, E)$ be a dominance graph with solved form G'. If Y, Y' are weakly connected nodes in G then there exists a node Z that is a common ancestor of Y and Y' in G'.*

Proof. Since the vertices Y and Y' are weakly connected there exists a chain $(Y_0, Y_1, \ldots, Y_r)$ that starts at $Y = Y_0$, ends at $Y' = Y_r$, and is linked by edges $(Y_i, Y_{i+1}) \in E \cup E^{-1}$. We prove by induction on r that there exists an index $j \in \{0, \ldots, r\}$ with $(Y_j, Y_0) \in R_{G'}$ and $(Y_j, Y_r) \in R_{G'}$. If $r = 0$ or $r = 1$ then we can choose Z to be either Y_0 or Y_r. Otherwise, we can apply the induction hypothesis to the chain $(Y_1, \ldots, Y_r)$. Thus, there exists Z' that is a common ancestor of Y_1 and Y_r in G'. If (Y_0, Y_1) is an inverse edge then Z' is also a common ancestor of Y_0 and Y_r, so we can choose $Z = Z'$. Otherwise, $(Y_0, Y_1) \in R_G$. Thus, both Y_0 and Z' are ancestors of Y_1 in G'. Since this digraph is a forest, it follows that either $(Y_0, Z') \in R_{G'}$ or $(Z', Y_0) \in R_{G'}$. In the first case, we choose $Z = Y_0$, and in the second one $Z = Z'$. □

7 Freeness

This section prepares a solvability test for weakly connected dominance graphs. The idea of the algorithm is based on the notion of *freeness*.

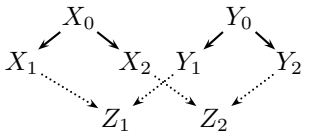


Figure 6: An unsatisfiable dominance graph.

Definition 7.1. *A node X of a dominance graph G is called* free *in G if there exists a solved form of G where X has in-degree zero.*

Proposition 7.1. *A weakly connected dominance graph without free nodes is unsolvable.*

Proof. If G is weakly connected and solvable then it has a solved form which is a tree (Lemma 6.1). The root of this tree is free for G. □

The absence of solved forms can thus be proved by showing the absence of free nodes.

Lemma 7.1. *A dominance graph $G = (V, \lhd^* \uplus \lhd)$ with free node X satisfies:*

- F1 *X has in-degree zero in G*
- F2 *no distinct nodes Y, Y' that are linked to X by tree edges in G are weakly connected in $G|_{V \setminus \{X\}}$*

Proof. We assume that X is a free node of G and show that both conditions hold. (F1) A free node cannot have any incoming edge in G since it would have one in all its solved forms. (F2) Let Y and Y' be distinct nodes that are linked to X in G via tree edges. If Y and Y' are weakly connected in $G|_{V \setminus \{X\}}$ then some node Z of $G|_{V \setminus \{X\}}$ must be a common ancestor of Y and Y' in all solved forms of G (Lemma 6.1), yet distinct from their mother X: therefore Z is an ancestor of X in all solved forms of G, in contradiction with the assumption that X is free. □

Prop 7.1 and Lemma 7.1 imply the unsolvability of the dominance graph in Fig. 6, where the roots X_0 and Y_0 violate (F2) while all others (Z_1 and Z_2) violate (F1).

8 Algorithm

The idea to construct a solved form of a given weakly connected dominance graph $G = (V, \lhd^* \uplus \lhd)$ is to remove a node satisfying (F1) and (F2) together with its neighborhood in $\lhd^*$, and to fail if there is no such node. We then decompose the remaining digraph of G into weakly connected components, and recursively solve these subgraphs. If successful it is easy to construct a solved form of the whole graph.

$\mathrm{sat}(G) \equiv$
forall weakly connected components $G' = (V', \lhd^* \uplus \lhd)$ of G:
choose some arbitrary node $X \in V'$ satisfying (F1) and (F2) in $G'|_{V'}$ **else fail**
Let $Y_1, \ldots, Y_n \in V'$ be all nodes s.t. $X \lhd Y_i$.
$\mathrm{sat}(G|_{V'-\{X,Y_1,\ldots,Y_n\}})$

Figure 7: Checking solvability of dominance graphs.

LEMMA 8.1. *Let G be a non-empty dominance graph. Then the following properties are equivalent:*

1. *The procedure* $\mathrm{sat}(G)$ *in Fig. 7 fails for some nondeterministic choice.*
2. *G is not solvable.*
3. *The procedure* $\mathrm{sat}(G)$ *fails for all nondeterministic choices.*

Proof. ($1 \Rightarrow 2$) If $\mathrm{sat}(G)$ fails for some nondeterministic choices, then G contains a weakly connected subgraph G' whose nodes all violate (F1) or (F2). By Lemma 7.1, G' has no free node and by Prop. 7.1 it is unsolvable. Since any graph which has an unsolvable subgraph is unsolvable, G must be unsolvable, too.

($2 \Rightarrow 3$) Suppose $\mathrm{sat}(G)$ is successful for some nondeterministic choices. We will prove by induction on the size of $G = (V, \lhd \uplus \lhd^*)$ that G has a solved form. If G is not weakly connected, then G' is solvable by induction hypothesis for all wcc's G' of G, and hence G is solvable by Prop. 6.1. Otherwise G is weakly connected. Since the algorithm did not fail, there exists some node X in G that satisfies (F1) and (F2). By induction hypothesis, the wcc's $G|_{V-\{X,Y_1,\ldots,Y_n\}}$ have solved forms $G'_1, \ldots, G'_k$. For each G'_i, if G has a dominance edge from Y_j to some node in G'_i, we attach G'_i with a dominance edge under Y_j; otherwise, we attach it with a dominance edge under X. Using (F1) and (F2) and the definition of dominance graphs we verify that (0) the resulting graph G' is a dominance graph, (1) G' is a tree that (2) contains all tree edges of G, and (3) refines dominance its wishes $\lhd^* \subseteq R_{G'}$. Thus G' is a solved form that is more specific than G.

($3 \Rightarrow 1$) Clearly, if $\mathrm{sat}(G)$ fails for all nondeterministic choices then it also fails for some nondeterministic choice. □

The algorithm contains a nondeterministic choice statement. We will now show that if we consider all different possibilities at this point, we can enumerate all minimal solved forms of a given dominance graph. The enumeration algorithm is given in Figure 9.

An example for a run of the algorithm is given in Figure 8. Procedure solved_form applied to the graph on the left first computes the weakly connected components - there is only one. It then applies solve to this component. The only node satisfying (F1) and (F2) in the graph on the left is X_3. Note in particular that X_0 violates (F2). The algorithm thus removes the tree fragment of X_3, i.e., nodes X_3, X_4, X_5. The resulting graph, drawn in the middle of Figure 8, has only one connected component, and we again apply solve. After several steps, a single solved form is returned (it equals the graph in the middle again). For the final result, the algorithm adds the previously removed tree fragment of X_0 on top of this graph. A dominance edge from X_4 to X_0 is inserted since X_4 dominates X_6 in G, and since X_6 belongs to the same component as X_0 in the middle.

THEOREM 8.1. *The algorithm solved_form of Figure 9 applied to a dominance graph G produces all and only the minimal solved-forms of G.*

Proof. The algorithm reflects the construction of a solved form in the proof of Lemma 8.1. By its recursive structure, the dominance graphs returned by solve(G) are rooted trees, and therefore solved_form(G) is a forest. It is also easy to see that it is more specific than G and thus the algorithm only produces solved forms of G. Different choices of free nodes lead to different solved forms, and thus each solved form is produced at most once. We only have to prove that the algorithm produces *all minimal solved forms of G*. Since at each recursive step in solve we process a weakly-connected graph, by Lemma 6.1 the algorithm must return a solved-form that is a tree; therefore some node X must be chosen to be top-most. If we want to make this node the root of our solved form, we *must* insert the dominance edges added in the last or next to last line of the algorithm. Therefore, we only enumerate *minimal* solved forms of G. The node X is by definition free, and thus the algorithm will eventually choose it. Thus we enumerate *all* minimal solved forms of G. □

COROLLARY 8.1. *For a node of a solvable dominance graph,* (F1) *and* (F2) *are necessary and sufficient conditions for freeness.*

Proof. Free nodes satisfy (F1) and (F2) by Lemma 7.1. Conversely, let X be a node of a dominance

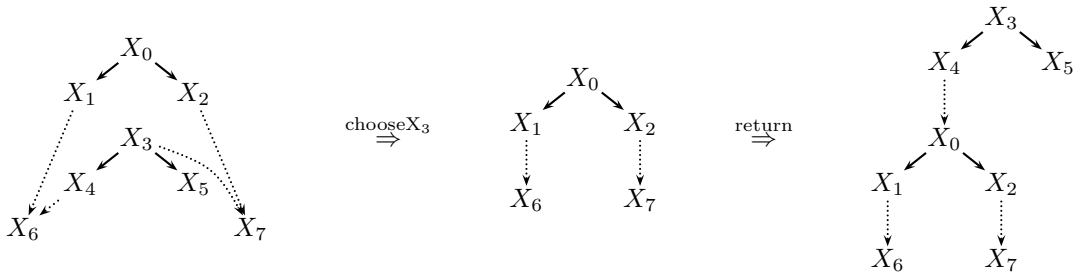


Figure 8: The algorithm in action.

graph G satisfying (F1) and (F2). Then algorithm solved_form(G) calls solve(G) for the weakly connected component G' of X, and in solve(G) the node X can be chosen to construct a solved form of G'. It will never fail since G is solvable (Lemma 8.1) and finally produces a solved form of G' (Theorem 8.1), which is a tree rooted at X. □

In Section 9 we will describe how to compute all nodes satisfying (F1) and (F2) in time $O(n+m)$ where n is the number of nodes and m the number of edges in a dominance graph G. This will prove:

THEOREM 8.2. *The overall running time of the enumeration algorithm for dominance graphs in Figure 9 is in $O(n \cdot (n+m))$ per solved-form.*

9 Testing Freeness Conditions

We now want to efficiently compute the set of nodes that satisfy the freeness conditions (F1) and (F2). We have to check for each node u with indegree zero whether it has a pair of children (linked to u by outgoing tree edges) that are weakly connected in the dominance graph of the constraint without u. Thus the naive way would be to compute for all these nodes u the weakly-connected components of the constraint graph without node u. This takes quadratic time per node, and thus cubic time to compute *all* free nodes.

We now show how to compute the set of free nodes in linear time. A *(vertex-)biconnected component* of a graph is a maximal subgraph that remains connected when one of its nodes is deleted. Biconnected components form a proper partition of the graph edges.

PROPOSITION 9.1. *A node X in a dominance graph G satisfies* (F2) *if and only if all different tree edges (X,Y) and (X,Y') of G lie in different biconnected components of G.*

Proof. If X does not have any children there is nothing to show. So first assume that Y and Y' are in the same weakly connected component of $G|_{V_G \setminus \{X\}}$. Then there is a path between Y and Y' not using X and thus the edges (X,Y) and (X,Y') must be in the same biconnected component. Conversely, assume that (X,Y) and (X,Y') are in the same biconnected component. Then there is a path from Y to Y' not using X and therefore there still must be a path from Y to Y' after removing X. □

A linear time algorithm that given a graph computes a mapping from each edge to a unique representant of its biconnected component can be found in any textbook that covers graph algorithms. To compute the set of free nodes, we use a control bit for each biconnected component. When processing a candidate X, we can use the bit to check for each edge $(X,Y) \in \lhd^*$ whether we already encountered another edge $(X,Y') \in \lhd^*$ in the same biconnected component. Since we spend only constant time on every tree edge, we have a linear time procedure that finds the free roots.

In each step, the algorithm computes the weakly connected components and biconnected components from scratch again. It is possible to further improve the running time and to avoid these redundant computations using dynamic graph connectivity algorithms, that allow to answer weak connectivity and biconnectivity queries in amortized sublinear time [8].

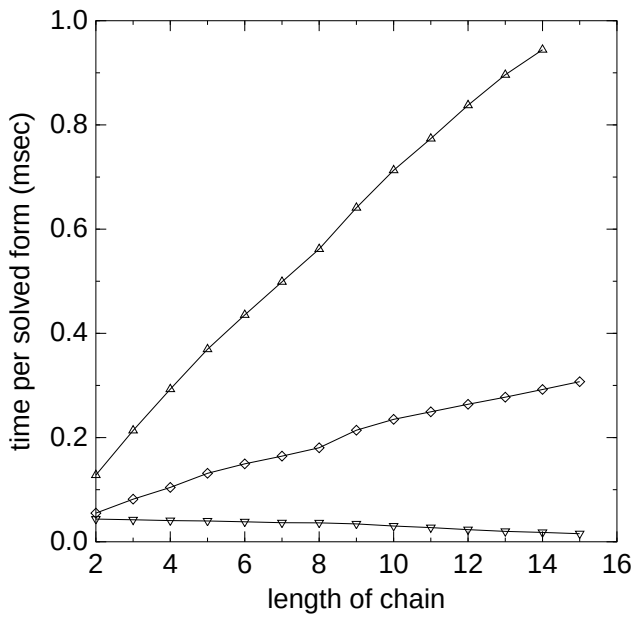
10 Implementation and Evaluation

We implemented the New algorithm (without using the mentioned decremental graph connectivity algorithms) and compared it to the best known previous implementations of Mehlhorn [1] and Thiel [13]. All implementations are done with C++/LEDA and were run on a Pentium-IV 2GHz. The software and the instances for experimentation are available at `http://ps.uni-sb.de/~smiele/dom-solving`.

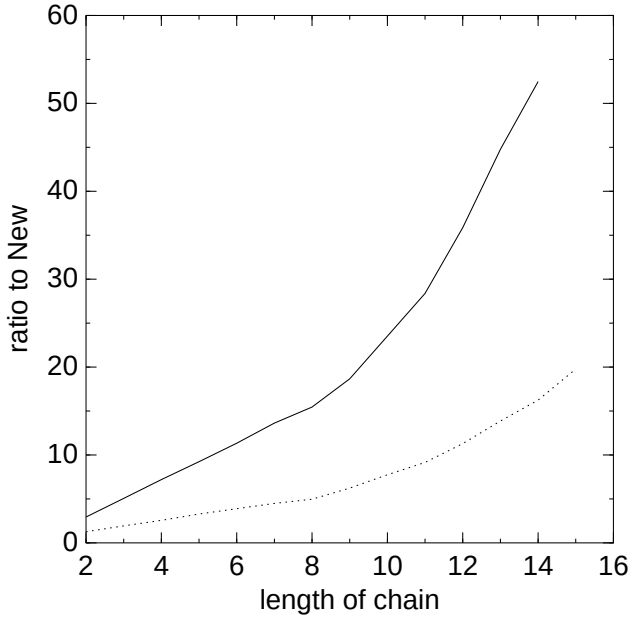
Table 1 and Figure 10 report on comparative benchmarks for a collection of prototypical examples in the application to scope underspecification. Chain k is a dominance constraint that models a natural language sentence with k quantifiers. Furthermore, we consider

solved_form(G) $\equiv$
Let $G_1, \ldots, G_k$ be the wcc's of $G = (V, \lhd^* \uplus \lhd)$
Let $(V_i, \lhd_i^* \uplus \lhd_i)$ be the result of solve(G_i)
return $(V, \cup_{i=1}^k \lhd_i^* \uplus \lhd)$
solve(G) $\equiv$
precond: $G = (V, \lhd^* \uplus \lhd)$ is weakly connected
choose a node X satisfying (F1) and (F2) in G **else fail**
Let $Y_1, \ldots, Y_n$ be all nodes s.t. $X \lhd Y_i$
Let $G_1, \ldots, G_k$ be the weakly connected components of $G|_{V-\{X,Y_1,\ldots,Y_n\}}$
Let $(W_j, \lhd_j^* \uplus \lhd_j)$ be the result of solve(G_j), and $X_j \in W_j$ its root
return $(V, \cup_{j=1}^k \lhd_j^* \cup \lhd_1^* \cup \lhd_2^* \uplus \lhd)$, where
$\lhd_1^* = \{(Y_i, X_j) \mid \exists X' : (Y_i, X') \in \lhd^* \wedge X' \in W_j\}$,
$\lhd_2^* = \{(X, X_j) \mid \neg\exists X' : (Y_i, X') \in \lhd^* \wedge X' \in W_j\}$

Figure 9: Algorithm enumerating the solved-forms of a dominance graph.



Average times in milliseconds per solved form, for chains of increasing lengths. Mehlhorn highest curve, Thiel middle, New lowest



Ratios of Mehlhorn and Thiel to our New algorithm. Mehlhorn/New higher curve, Thiel/New lower curve

Figure 10: Benchmarks

Problem	Mehlhorn	Thiel	New	n	m	N
Chain 2	0.128	0.055	0.0435	8	8	2
Chain 3	0.2136	0.0818	0.0422	12	13	5
Chain 4	0.292857	0.104286	0.040714	16	18	14
Chain 5	0.369524	0.131429	0.04	20	23	42
Chain 6	0.435379	0.149545	0.038409	24	28	132
Chain 7	0.498834	0.164336	0.036596	28	33	429
Chain 8	0.561818	0.18042	0.036363	32	38	1430
Chain 9	0.641094	0.214109	0.034348	36	43	4862
Chain 10	0.713027	0.234996	0.030304	40	48	16796
Chain 11	0.77379	0.249430	0.027268	44	53	58786
Chain 12	0.837788	0.263831	0.023364	48	58	208012
Chain 13	0.896218	0.277453	0.020016	52	63	742900
Chain 14	0.944325	0.292304	0.017992	56	68	2674440
Chain 15		0.307242	0.015540	60	73	9694845
H&S	0.342262	0.117262	0.03869	19	20	168
MRS	N/A	N/A	0.045	9	8	2

Table 1: Average time in milliseconds per solved form, for constraint graphs with n nodes, m edges, and N solved forms

a famous example of Hobbs and Shieber (H&S) and the MRS example of Fig. 5. The latter has a constraint graph that is not normal, but only weakly normal. Therefore it is beyond the reach of the Mehlhorn and Thiel algorithms, hence the N/A entries in Table 1.

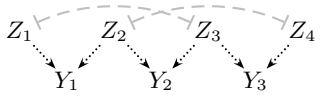


Figure 11: A dominance graph with *disjointness edges*.

11 Disjointness Constraints

The algorithm described here is fundamentally different from the one presented in [1] where satisfiability was characterized by the absence of so-called *hypernormal cycles* in the dominance graph. The difference can be further demonstrated with a natural extension of our constraint language by means of *disjointness literals*. These are part of various tree description languages [2, 3, 6].

A *disjointness literals* $X \perp Y$ is satisfied by a solution (τ, α) if neither $\alpha(X)$ dominates $\alpha(Y)$ nor $\alpha(Y)$ dominates $\alpha(X)$. Consider Figure 11. We draw disjointness constraints in the constraint graph with dashed lines. This graph is unsatisfiable, since it does not contain a free node. The approach of [1], which tests for the presence of certain kinds of 'bad cycles' in the constraint graph, does not naturally extend to graphs also containing disjointness edges. Observe that after deleting any edge in Figure 11 the constraint is satisfiable.

In contrast, our algorithm can easily be adapted to deal with this extended constraint language. We now have the following additional property of a free node X:

F3 X has no incident disjointness edge:
$\forall Y \in V : (X \perp Y) \notin \phi$

The proofs can readily be adapted to show that, for satisfiable constraints, (F1), (F2) and (F3) are again necessary and sufficient conditions for freeness.

12 Conclusion

We described an algorithm that enumerates the minimal solved forms of a normal dominance constraint in quadratic time per solved form. We thereby improved on the best previously known algorithm [13], which required cubic time per solved form. Subquadratic running time could be achieved by employing decremental graph biconnectivity algorithms.

Our approach is not only more efficient, but it is also more general than earlier approaches [1, 13]: firstly, it also applies to the language of *weakly* normal dominance constraints, which improves the applicability of dominance constraints in the area of natural language semantics. Secondly, it extends simply to a language admitting disjointness literals.

These results are especially significant for applications in computational linguistics. In particular, they underlie the first polynomial time algorithm for minimal recursion semantics (MRS) [5, 11].

References

[1] E. Althaus, D. Duchier, A. Koller, K. Mehlhorn, J. Niehren, and S. Thiel. An efficient graph algorithm for dominance constraints. *Journal of Algorithms*, 48(1):194–219, May 2003. Special Issue of SODA 2001.

[2] R. Backofen, J. Rogers, and K. Vijay-Shanker. A first-order axiomatization of the theory of finite trees. *Journal of Logic, Language, and Information*, 4:5–39, 1995.

[3] M. Bodirsky and M. Kutz. Pure dominance constraints. In *19th Annual Symposium on Theoretical Aspects of Computer Science (STACS'02), LNCS 2285*, pages 287–298, 2002.

[4] J. Bos. Predicate logic unplugged. In *Proceedings of the 10th Amsterdam Colloquium*, pages 133–143, 1996.

[5] A. Copestake, D. Flickinger, I. Sag, and C. Pollard. Minimal recursion semantics: An introduction. CSLI Draft, Sept. 1999.

[6] M. Egg, A. Koller, and J. Niehren. The constraint language for lambda structures. *Journal of Logic, Language, and Information*, 10:457–485, 2001.

[7] C. Gardent and B. Webber. Describing discourse semantics. In *Proceedings of the 4th TAG+ Workshop*, Philadelphia, 1998. University of Pennsylvania.

[8] J. Holm, K. de Lichtenberg, and M. Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. In *30th Annu. ACM Sympos. Theory Comput.*, pages 79–89, 1998.

[9] A. Koller, J. Niehren, and R. Treinen. Dominance constraints: Algorithms and complexity. In *Logical Aspects of Computational Linguistics*, volume 2014 of *LNAI*, pages 106–125, 2001.

[10] M. P. Marcus, D. Hindle, and M. M. Fleck. D-theory: Talking about talking about trees. In *21*st *Annu. Meeting of the Ass. of Comp. Ling.*, pages 129–136, 1983.

[11] J. Niehren and S. Thater. Bridging the gap between underspecification formalisms: Minimal recursion semantics as dominance constraints. In *41st Meeting of the Association of Computational Linguistics*, pages 367–374, July 2003.

[12] J. Rogers and K. Vijay-Shanker. Obtaining trees from their descriptions: An application to tree-adjoining grammars. *Computational Intelligence*, 10:401–421, 1994.

[13] S. Thiel. A linear time algorithm for the configuration problem of dominance graphs. Submitted, 2004. Max-Planck Institut, Saarbrücken.