An Efficient Algorithm for the Configuration Problem of Dominance Graphs[1]

Ernst Althaus[2] Denys Duchier[3] Alexander Koller[4] Kurt Mehlhorn[2] Joachim Niehren[3] Sven Thiel[2]

Abstract

Dominance constraints are logical tree descriptions originating from automata theory that have multiple applications in computational linguistics. The satisfiability problem of dominance constraints is NP-complete. In most applications, however, only *normal* dominance constraints are used. The satisfiability problem of normal dominance constraints can be reduced in linear time to the configuration problem of dominance graphs, as shown recently. In this paper, we give a polynomial time algorithm testing configurability of dominance graphs (and thus satisfiability of normal dominance constraints). Previous to our work no polynomial time algorithms were known.

1 Introduction

The dominance relation of a tree is the ancestor relation between its nodes. Dominance constraints are logical descriptions of trees talking about the dominance relation. Dominance based tree descriptions were first used in automata theory in the sixties [TW67] and rediscovered in computational linguistics in the early eighties [MHF83]. Since then, they have found numerous applications: they have been used for grammar formalisms [VS92, RVSW95], in semantics [Mus95, ENRX98], and for discourse analysis [GW98].

The satisfiability problem of dominance constraints is NP-complete [KNT98]. Earlier attempts at processing dominance constraints [Cor94, VSWR95, DN00] all suffer from this fact. But it turns out that *normal* dominance constraints, a restricted sublanguage, are sufficient for most applications. The starting point of the graph based approach of this paper is another recent result [KMN00] showing that the satisfiability problem of normal dominance constraints can be reduced in linear time to the configuration problem of dominance graphs.

Informally, a *dominance graph* is given by a collection of rooted trees and a set of dominance wishes. (A precise definition follows in Section 2.) A dominance wish is a directed edge from the leaf of some tree to the root of some other tree. A *configuration* of a dominance graph is obtained by assembling the trees of the graph into a forest, by hooking roots into leaves such that all dominance wishes are translated into ancestor-descendant relationships. The *configuration problem of dominance graphs* is the question whether there exists a configuration for a given dominance graph.

In this paper, we show that the configuration problem of dominance graphs is in polynomial time. This result immediately leads the way for a polynomial time and practically more efficient processing of normal dominance constraints in computational linguistics.

To get an idea of how configurations of dominance graphs arise in linguistics, consider the [ENRX98] analysis of the following English sentence:

(1.1) Every linguist speaks two languages.

a. . . . , namely English and German.

b. . . . , not necessarily the same ones.

Depending on the context (indicated by the continuations a. and b.), this sentence can be read in two different ways – it exhibits a *scope ambiguity.* It can mean either that there is a set of two languages spoken by every linguist, or it can mean that each linguist can pick his own pair of languages.

This ambiguity can be represented compactly by the graph in Figure 1, which we can read as a dominance graph by removing the node labels. Intuitively, the contributions of "every linguist" and "two languages" to the meaning of the sentence are represented as the two upper trees; the contribution of "speaks" is represented as the lower one. The tree configurations of this dominance graph are obtained by plugging the two upper trees in some order on top on the lower tree. The two ways to arrange them correspond to the two different readings of (1.1).

Our paper is organized as follows. In Section 2 we define the terms *dominance graph*, *solved form*, and *configuration* formally and give a preview of the results in

[1]Partially supported by the IST Programme of the EU under contract number IST-1999-14186 (ALCOM-FT)

[2]Max-Planck-Institute for Computer Science, Saarbrücken, Germany

[3]Programming Systems Lab, Fachbereich Informatik, Universität des Saarlandes, Saarbrücken, Germany

[4]Department of Computational Linguistics, Universität des Saarlandes, Saarbrücken, Germany

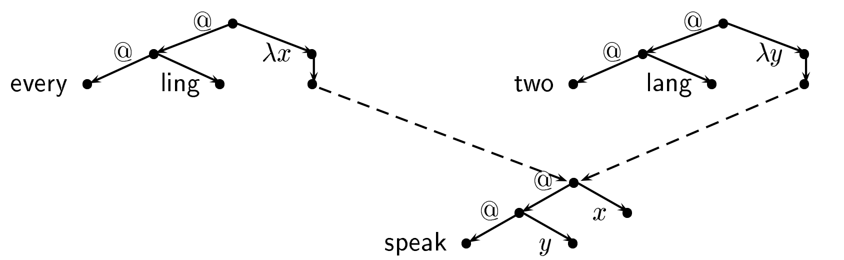

Figure 1: Dominance graph for a scope ambiguity.

the succeeding sections. In Section 3, we show how to enumerate all configurations of a dominance graph in exponential time; this provides a framework for the application of the later results. In Section 4, we characterize configurable dominance graphs (a dominance graph has a configuration iff it contains no hypernormal cycle), and then we show in Section 5 how the existence of a hypernormal cycle can be decided by solving a weighted matching problem in an auxiliary graph. This gives us a polynomial-time configurability test which we use in Section 6 to make the enumeration algorithm from Section 3 efficient. Section 7 shows that a slight extension of the configuration problem by closed leaves is again NP-complete. Finally, we offer a short conclusion.

2 Definitions

A *dominance graph* is defined by a directed graph $G = (V, E \dot\cup D)$ satisfying the following two conditions: (1) the graph $G = (V, E)$ defines[5] a collection T of node disjoint trees of height at least 1 and (2) each edge in D goes from a leaf of some tree in the collection to the root of some tree in the collection. In our figures, we draw the edges in E solid and the edges in D dashed. We call the edges in E *solid edges* or *tree edges* and we call the edges in D *dashed edges* or *dominance edges* or *dominance wishes*. A leaf is a node with no outgoing tree edge and a root is a node with no incoming tree edge.

Now the idea is that we want to assemble the trees in T by plugging roots into leaves. We say that a dominance graph G is *in solved form* iff it is a forest. If $G = (V, E \dot\cup D)$ is a dominance graph, we call a dominance graph $G' = (V', E' \dot\cup D')$ a *solved form of* G iff $V = V'$, $E = E'$, G' is in solved form, and G' realizes all dominance wishes in G – that is, for every dominance wish $(v, w) \in D$ there is a path from v to w in G'.

In particular, we call a solved form of G where the dominance edges in D' are a matching a *configuration* of G. Roots have at most one incoming dominance edge in configurations; the intuition is that the roots have been "plugged" into the leaves, and the remaining dominance edges indicate which root is plugged into which leaves.

A dominance graph is *configurable* if it has a configuration and *solvable* if it has a solved form. Figure 2 shows a configurable graph and one configuration. Figure 3 displays an unconfigurable graph, the heavy edges indicate an "unconfigurable cycle", as we shall see later.

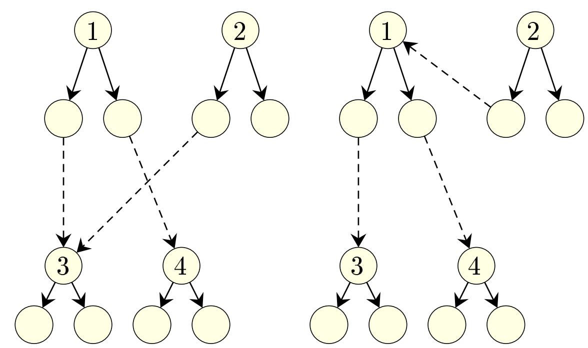

Figure 2: A configurable dominance graph and a configuration of it.

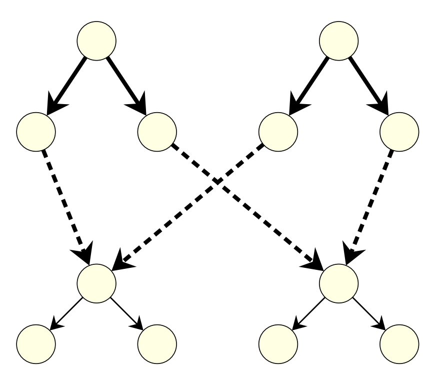

Figure 3: An unconfigurable dominance graph; the heavy edges form an "unconfigurable cycle".

[5]Edges are assumed to go from parents to children.

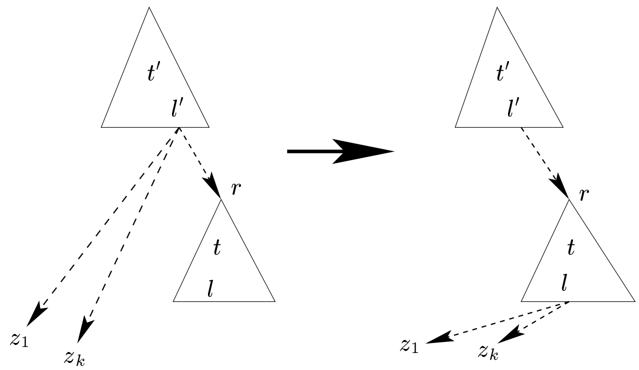

Figure 4: Application of Rule 1: All dominance wishes of l' except for (l', r) are shifted down to the leaf l.

The problem we investigate in this paper is to decide whether a given dominance graph has a configuration. More precisely, we are going to consider the problem of whether it has a solved form; but the following lemma expresses that this is the same problem.

LEMMA 2.1. *Every dominance graph in solved form is configurable.*

Conversely, every configurable graph is trivially solvable.

Proof. For the proof, we define a *problem leaf* to be a leaf with more than one outgoing dominance edge; our aim will be to eliminate problem leaves from solved forms.

The proof is by induction on weights (d, a) of graphs G, where d is the negative minimum depth of a problem leaf of G (or $-\infty$ if there aren't any), and a is the total number of dominance edges emanating from problem leaves of minimum depth (potentially 0). We consider the lexicographic order on these weights.

Solved forms without problem leaves (i.e. with weight $(-\infty, 0)$) are configurations, so the lemma is trivially true in this case. So let G be a solved form that does have problem leaves. Let G have weight (d, a), and assume that we know that all solved forms of lower weight do have configurations. Then we can apply the following rule to a problem leaf l' of minimum depth:

SIMPLIFICATION RULE 1. *Let $e = (l', r)$ be a dominance edge from the leaf l' of a tree t' to the root r of a tree t. Let l be an arbitrary leaf of t. Change any dominance edge (l', z) with $z \neq r$ into (l, z), see Figure 4.*

The result G' is still in solved form, and its weight is strictly lower than that of G; so by the induction hypothesis, G' has a configuration G_c. But G_c also realizes all dominance wishes of G. This is obvious for (l', r) and for all wishes which do not start in l'. For a wish (l', z) with $z \neq r$ we note that this wish is realized because there is a path from l' to l in G' and G_c realizes the wish (l, z). So G has a configuration as well. □

Finally, we call a dominance wish $d = (v, w)$ *redundant* if there is a path from v to w in $G \setminus d$. A dominance graph is called *reduced* if it contains no redundant dominance wish. As usual, we use n and m, respectively, to denote the number of nodes and edges of G.

In the following sections we show:

- Configurability of a dominance graph has a simple characterization.
- Configurability of a dominance graph can be decided in polynomial time. More precisely, it can be decided by solving a weighted matching problem in an auxiliary graph with $n' = O(m)$ nodes and $m' = \sum_{v \in V} \text{indeg}_v^2$ edges; here indeg_v is the degree of v in G. The matching problem can be solved in time $O(n'm' \log n')$, see [GMG86].
- A solved form of a (configurable) dominance graph can be constructed in polynomial time. More precisely, it can be found in time $O(n^2 n'm' \log n')$.
- All solved forms of a dominance graph can be enumerated in polynomial time per configuration. More precisely, if N denotes the number of solved forms then all configurations can be enumerated in time $O((N + 1)T)$, where T is the time to find a single one.
- Our theoretical results lead to a practically efficient algorithm for handling dominance graphs. The algorithm has been implemented. In our application, we have $m = O(n)$, and $\sum_{v \in V} \text{indeg}_v^2 = O(n)$. The existance of a configuration can therefore be tested in time $T = O(n^2 \log n)$, and a solved form can be constructed in time $O(n^2 T)$. The actual running times are smaller since the arising weighted matching problems seem to be fairly simple and the number of matching problems to be solved seems to be much less than n^2. Our implementation uses LEDA [MNSU, MN99] and the matching codes of T. Ziegler and G. Schäfer [Zie95, Sch00].

3 Enumeration of Solved Forms

In this section, we show how to enumerate the solved forms of a dominance graph G. The algorithm we present may take exponential time to produce even a single configuration because it blindly enumerates all cases. In Section 5, we will present a polynomial algorithm for determining configurability. By plugging this algorithm into the enumeration algorithm, we can enumerate configurations in polynomial time per configuration (Section 6).

The enumeration algorithm applies the following simplification rules:

SIMPLIFICATION RULE 2. (REDUNDANCY ELIM.) *All redundant dominance edges, i.e. edges that are implied by transitivity, can be removed. In particular, parallel edges can be combined into one.*

SIMPLIFICATION RULE 3. (CHOICE) *Let v be a root with at least two incoming dominance wishes (l,v) and (l',v) and let r and r' be the roots of the trees containing leaves l and l', respectively. Generate two new graphs H and H' by adding either (l',r) or (l,r') to D, see Figure 5.*

The enumeration of the solved forms can be carried out by a recursive algorithm:

1. Make the graph reduced, i.e. apply Rule 2.
2. If the graph contains a (directed) cycle, terminate this recursion since the graph has no configuration.
3. If the graph is in solved form, report it and terminate this recursion.
4. Otherwise, apply the choice rule and apply the algorithm to the two newly generated graphs.

Every solved form derived by the algorithm is clearly a solved form of the original graph. On the other hand, the algorithm enumerates *all* of its solved forms. This is because application of Rule 2 to a dominance graph does not change the set of solved forms. Application of Rule 3 partitions the set: The two new graphs have disjoint sets of solved forms, but the union of these sets is the same as the old set of solved forms. This can be seen as follows. In a solved form G_s of G the nodes l and l' are both ancestors of v and therefore either l' is ancestor of l and hence of r or vice versa. This implies that G_s is either a solved form of H or of H'.

To prove termination, we derive an upper bound for the maximum recursion depth. Consider for any dominance graph G its *reachability relation* R_G – the set of all pairs (u,v) of nodes such that there is a (directed) path from u to v in G. If G is acyclic, the cardinality of R_G is at most $\binom{n}{2} \le n^2$. Thus, whenever the size of the relation becomes greater than $\binom{n}{2}$, the recursion terminates immediately. But if we apply the choice rule to a reduced, acyclic dominance graph, the size of the relation increases strictly, i.e. $|R_G| < \min(|R_H|, |R_{H'}|)$. This is because $R_H \supseteq R_G$, and $(l',r) \in R_H$ but (l',r) cannot be in R_G, otherwise (l',v) would have been redundant. A similar argument holds for H'.

4 A Graph-Theoretic Characterization of Solvability

We give a graph theoretic characterization of solvability; as this is equivalent to configurability by Lemma 2.1, the result carries over to configurability. The characterization implies that the solvability problem for dominance graphs is in *NP* $\cap$ *co-NP*.

The *undirected dominance graph* G_u corresponding to the dominance graph $G = (V, E \dot\cup D)$ is the undirected graph obtained by making all edges of G undirected. Now, we want to define the notion of a *cycle* in an undirected graph, which may differ from the reader's usual notion. A cycle C in an undirected graph is a sequence of edges $e_0 \circ e_1 \circ \ldots \circ e_{n-1}$ with $n > 1$ such that for $i = 0, \ldots, n-1$ the following holds:

- there is a node v_i incident to both e_i and $e_{(i+1) \bmod n}$
- $e_i \neq e_{(i+1) \bmod n}$

We call C *edge-simple* if the edges in the sequence are pairwise different. C is said to be *simple* if all the visited nodes $v_0, \ldots, v_{n-1}$ are pairwise different.

4.1 Hypernormal Dominance Graphs

Let us first investigate a simpler subproblem of the solvability problem. A dominance graph $G = (V, E \dot\cup D)$ is *hypernormal* if for every leaf l in (V, E) there is at most one dominance wish $(l, .)$ in D. Hypernormal dominance graphs are reduced[6].

PROPOSITION 4.1. *Let $G = (V, E \dot\cup D)$ be a hypernormal dominance graph. If G_u contains a cycle then G is unsolvable.*

Proof. The proof is by induction on the minimal number k of dominance edges in a simple cycle C of G_u. Clearly, the case $k = 0$ cannot occur, and if $k = 1$ then G is not solvable. On the other hand, assume that we know the result to be true for $k - 1$. C either does not contain any nodes at which its edges change directions; then it is also a cycle in G and hence, G is clearly unsolvable. Or C does change directions, then it must contain two dominance edges (l,r) and (l',r) into the same root. Both results of applying the choice rule produce graphs

[6] An alternative definition of hypernormal graphs is as follows: Out of two dominance wishes (l,v) and (l,v') at least one is redundant.

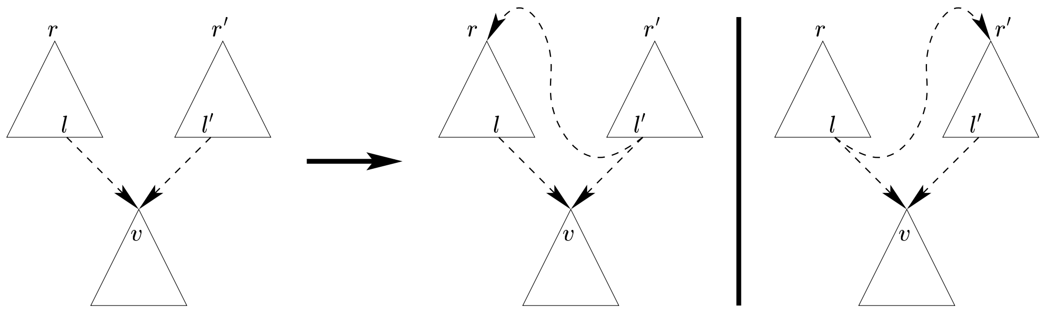

Figure 5: Two graphs are generated by applying the choice rule to the graph on the left hand side.

with a simple cycle containing $k-1$ dominance edges, so both are unsolvable. But then, G must be unsolvable as well. □

The converse of the above proposition is also true. If G is not solvable, then G_u contains a cycle. This statement will be a corollary of Theorem 4.1, which we will prove below.

4.2 Dominance Graphs

The Proposition 4.1 does not carry over literally to the general case: Figure 6 is a counterexample. In order to state our theorem for the general case, we call a subgraph H_u of G_u *hypernormal* if the corresponding directed subgraph H of G is hypernormal.

THEOREM 4.1. *Let $G = (V, E \dot\cup D)$ be a dominance graph.*
(a) G is solvable iff G_u does not contain a hypernormal cycle.
(b) G is solvable iff every hypernormal subgraph of G is.

Note that this implies that a graph G is configurable iff G_u has no hypernormal cycle, by Lemma 2.1.

Proof. Part (b) follows immediately from part (a). If some hypernormal subgraph of G is unsolvable, G is unsolvable. If every hypernormal subgraph of G is solvable, G_u contains no hypernormal cycle, and hence G is solvable by part (a). We turn to part (a).

Assume first that G_u contains a hypernormal cycle C. Let D' be the dominance edges of G corresponding to edges in C. Then $G' = (V, E \dot\cup D')$ is a hypernormal dominance graph such that G'_u contains C. By Proposition 4.1, G' is unsolvable and hence G is unsolvable.

It remains to prove the converse: If G is unsolvable, G_u contains a simple hypernormal cycle. Assume that the statement is false. W.l.o.g. we may restrict our attention to reduced dominance graphs. If the choice rule is not applicable to a graph, this graph is either in solved form or has a directed cycle, so the theorem is true for these cases. Hence, there must be a "minimal" counterexample G to the statement, in the following sense:

- G is reduced and unsolvable
- G_u does not contain a hypernormal cycle
- The choice rule can be applied to G. Both graphs H and H' which are generated by it are unsolvable, and both H_u and H'_u do contain hypernormal cycles.

We will derive a contradiction by showing that this implies that G_u contains a hypernormal cycle.

Suppose that v is the root and that (l, v) and (l', v) are the wishes considered in the above application of the choice rule. Let r be the root of the tree with the leaf l and r' be the root of the tree with the leaf l'.
We have a hypernormal cycle $C_1 = \{l, r'\} \circ P_1$ in H_u and a hypernormal cycle $C_2 = \{l', r\} \circ P_2$ in H'_u. Since C_1 is hypernormal, P_1 does not use any dominance edge incident to l. If P_1 does not use some dominance edge $\{l', w\}$ incident to l', we are done since G_u contains the hypernormal cycle $\{l, v\}\{v, l'\} \circ (l' \to r') \circ P_1$, where $l' \to r'$ is the tree-path from l' to r'. So assume we can decompose $P_1 = Q_1 \circ \{l', w\} \circ R_1$, then R_1 does not use any dominance edge incident to l or l'. A similar argument gives us a decomposition $P_2 = Q_2 \circ \{l, u\} \circ R_2$ such that R_2 avoids all dominance edges incident to l and l'.
Thus we have the cycle $C = \{l', w\} \circ R_1 \circ \{l, u\} \circ R_2$ in G_u. This cycle is not necessarily hypernormal but the paths $\{l', w\} \circ R_1 \circ \{l, u\}$ and $\{l, u\} \circ R_2 \circ \{l', w\}$ are. The following lemma shows that this suffices to prove that G_u contains a hypernormal cycle:

LEMMA 4.1. *If G_u contains a cycle $C = e_S \circ S \circ e_T \circ T$ where e_S, e_T are edges and S, T are paths such that*

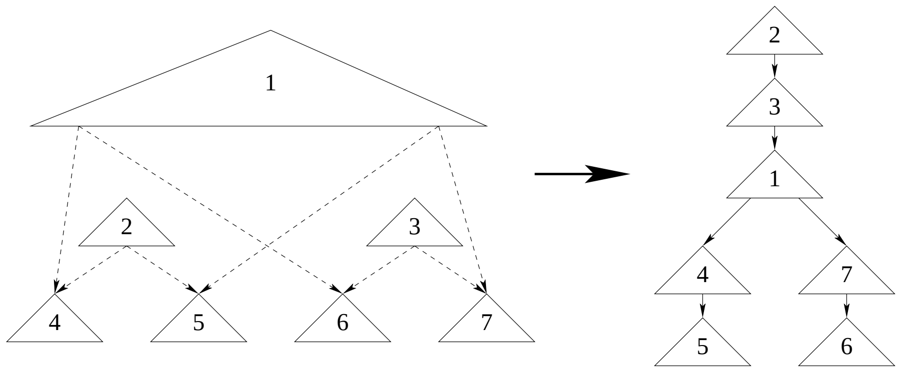

Figure 6: A solvable dominance graph and one of its solved forms. The graph contains an undirected cycle, but no hypernormal cycle.

$e_S \circ S \circ e_T$ and $e_T \circ T \circ e_S$ are hypernormal, then G_u contains a hypernormal cycle.

Before we prove the lemma, we want to give some intuition of its statement: If we can "glue" two hypernormal paths ($e_S \circ S$ and $e_T \circ T$) togehter such that they form a cycle which is hypernormal at the "glue" nodes, then G_u contains a hypernormal cycle.

Proof. We may assume that C is the smallest cycle (i.e. with the least number of edges) in G_u fulfilling the condition of the lemma. We consider two cases:

- C is simple:
  Since every node on C is an inner node of at least one of the two hypernormal paths, we have that C does not contain a consecutive pair of dominance edges incident to the same leaf. And hence, the simplicity of C implies that C is hypernormal.

- C is not simple:
  By the choice of C the paths $P_1 = e_S \circ S$ and $P_2 = e_T \circ T$ are simple. We may assume that none of the two is a cycle, otherwise we are done. Let v be a node which is visited twice by C. Then v must be an inner node of both P_1 and P_2, and we can decompose the paths at v: $P_1 = e_S \circ S' \circ f \circ S''$ and $P_2 = e_T \circ T' \circ g \circ T''$. Here f and g are edges and S', S'', T', T'' are (possibly empty) paths such that $e_S \circ S'$ and $e_T \circ T'$ end at v. Altogehter, we have the following decomposition:

  $$C = e_S \circ S' \circ f \circ S'' \circ e_T \circ T' \circ g \circ T''$$

  We have to distinguish two cases:

  1. f is not a dominance edge:
     Then $C' = e_T \circ T' \circ f \circ S''$ is a cycle and $e_T \circ T' \circ f$ and $f \circ S'' \circ e_T$ are hypernormal paths.
  2. f is a dominance edge:
     Then $e_S \circ S'$ does not end with a dominance edge. Hence $C' = e_S \circ S' \circ g \circ T''$ is a cycle and both $e_S \circ S' \circ g$ and $g \circ T'' \circ e_S$ are hypernomal.

  In both cases C' is smaller than C, and C' fulfills the condition of the lemma, a contradiction.

□

The characterization theorem has an interesting consequence. The solvability problem for dominance graphs is clearly in *NP*. Non-solvability is tantamount to the existance of a simple hypernormal cycle, and the existance of such a cycle is clearly in *NP*. Thus solvability is in *NP* $\cap$ *co-NP*.

5 Testing for Hypernormal Cycles

Now we show how to test for the presence of hypernormal cycles in a dominance graph in polynomial time. This immediately gives us a polynomial algorithm for testing solvability (and hence, configurability) of dominance graphs.

The test is by solving a weighted perfect matching problem on an auxiliary graph A which is constructed as follows. For every edge $e = (v, w) \in G$ we have two nodes $e_v = ((v, w), v)$ and $e_w = ((v, w), w)$ in A. We also have the following two kinds of edges.

(Type A) For every edge e we have the edge $\{e_v, e_w\}$.

(Type B) For every node v and distinct incident edges $e = (u, v)$ and $f = (v, w)$ we have the edge $\{e_v, f_v\}$ if either v is not a leaf or v is a leaf and either e or f is a tree edge.

The type A edges form a perfect matching in A. We give type A edges weight zero and type B edges weight one.

LEMMA 5.1. *The maximum weight of a perfect matching in A is positive iff G_u contains a simple hypernormal cycle.*

Proof. Suppose first that G_u contains a hypernormal cycle C. We may assume that C is simple. We construct a matching M of positive weight. For any pair $e = (u, v)$ and $f = (v, w)$ of consecutive edges in C, we put $\{e_v, f_v\}$ into M. Observe that $\{e_v, f_v\} \in A$ since C is hypernormal. For any edge $e = (v, w) \in (E \cup D) \setminus C$ we put the $\{e_v, e_w\}$ into M. M is clearly a perfect matching. It contains edges of type B and hence has positive weight.

Assume next that A has a perfect matching M of positive weight. We construct a simple hypernormal cycle in G_u. For any edge $\{e_v, f_v\} \in M$ we put the edges e and f into the set $\mathcal{C}$. Since for any edge $e = (v, w) \in G_u$ we have nodes e_v and e_w in A and since both nodes must be matched in a perfect matching, this rule will construct a collection of hypernormal cycles in G_u. □

We assume that all non-leaves in the dominance graph G have outdegree at most two[7]. Observe that we have one edge of type A for every edge of G, a complete graph on $2 + \mathrm{indeg}_v$ nodes for every root v, and a graph of size $1 + \mathrm{outdeg}_v$ for every leaf v. Thus the auxiliary graph A has $n' = m$ nodes and $m' = O(m) + \sum_{v \in V}(2 + \mathrm{indeg}_v)^2$ edges. The graph G can be reduced in time $O(nm)$, see [AGU72]. Then we have no parallel edges, and hence a root r with indegree greater than n must have two dominance edges from different leaves of the same tree to r, which is trivial to recognize in time $O(nm)$. So we can assume that the indegree of any root is at most n. Let us say that we have $r \leq n$ roots and let d_i be the indegree of the i-th root. We have $\sum_{i=1}^{r} d_i \leq m$ and $d_i \leq n$. What is the maximum value of $S = \sum_i (2 + d_i)^2$? We have $S = O(n + m) + \sum_i d_i^2$. The sum $\sum_i d_i^2$ is maximized if we make the d_is as unequal as possible. So we attain the maximum if we set m/n of the d_is equal to n and all others equal to zero. Thus $\sum_{v \in V}(2 + \mathrm{indeg}_v)^2 = O(n+m) + O(m/n \cdot n^2) = O(mn)$. A maximal weighted matching in a graph with n' nodes and m' edges can be found in time $O(n'm' \log n')$, see [GMG86].

THEOREM 5.1. *The existence of a hypernormal cycle can be decided in time $O(n'm' \log n')$, where $n' = m$ and $m' = O(m) + \sum_{v \in V}(2 + \mathrm{indeg}_v)^2$. A (simple) hypernormal cycle (if it exists) can be found in the same time bound.*

[7] We can replace each non-leaf with outdegree more than two and its children by a small binary tree. This construction increases the number of nodes and the number of edges only by a constant factor.

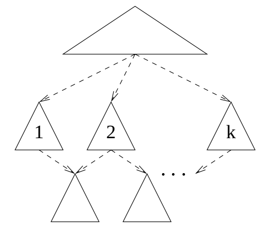

Figure 7: Embedded chain of length k.

In our application $m = O(n)$ and the indegrees are bounded (the outdegrees are not) and hence configurability can be decided in time $O(n^2 \log n)$. In the worst case, the running time is $O(nm^2 \log n)$.

6 Efficient Enumeration

A first application of the polynomial-time configurability test from the previous section is to make the enumeration of solved forms more efficient. We modify the enumeration algorithm from Section 3 by testing for (undirected) simple hypernormal cycles in step 2 instead of directed arbitrary cycles. The recursion will terminate immediately once the graph becomes unconfigurable, and we know that the recursion depth is bounded by n^2. Thus:

COROLLARY 6.1. *A solved form of a solvable dominance graph can be constructed in time $O(n^3 m^2 \log n)$. If a dominance graph has N solved forms, they can be enumerated in time $O(Nn^3 m^2 \log n)$.*

Note that N can still be exponential in n. Also note that we can get configurations instead of solved forms in the same asymptotic time, by applying Lemma 2.1: Simplification Rule 1 can only be applied at most n^2 times either, by a similar argument about the reachability relation.

Going back to the application in computational linguistics described in the introduction, the algorithm for enumerating configurations that we have just sketched gives us a straightforward algorithm for enumerating models of a normal dominance constraint. We have implemented this algorithm, and this gives us a significant improvement in runtimes over earlier solvers for dominance constraints. By way of example, consider the dominance graph in Figure 7. This graph is an *embedded chain* of length k. Such graphs appear in the application; for instance, the graph for "John says that every linguist speaks two languages" is an embedded chain of length 2. Runtimes for enumerating all configurations of

k	N	Time (new)	Time (old)
3	5	20	180
4	14	190	670
5	42	1210	5900
6	132	4130	12740
7	429	16630	46340
8	1430	255000	n/a

Figure 8: Runtimes on embedded chains of length k. N is the respective number of configurations. Times are in milliseconds CPU time.

embedded chains of various lengths (on a 550 MHz Pentium III) are displayed in Figure 8. In the table, "new" refers to the algorithm sketched above; "old" refers to the dominance constraint solver described in [DG99].

7 Dominance Graphs with Closed Leaves

A slight extension of the configuration problem by *closed leaves* becomes NP-complete again. A dominance graph with closed leaves is given by a dominance graph $G = (V, E \dot{\cup} D)$ and a set L of leaves. The members of L are called *closed*, all other leaves are called *open*. Closed leaves cannot be the source of dominance wishes. A solved form of (G, L) with closed leaves L is a solved form $G' = (V, E \cup D')$ of G which has the additional property that there is no edge $(l, v) \in E'$ with $l \in L$, but there is an edge $(l, v) \in D'$ for every $l \notin L$. In other words, it is not allowed to attach a tree to a closed leaf, and every open leaf must be "plugged" with some other tree. We show that the configuration problem of dominance graphs with closed leaves is NP-complete by reducing the 3-partition problem to it.[8]

FACT 7.1. (3-PARTITION) *Let A denote a multiset $\{a_1, \dots, a_{3m}\}$ of integers and $B \in \mathbb{N}$ such that $B/4 < a_i < B/2$ for all i; and $\sum_{i=1}^{3m} a_i = mB$. The question is whether there is a partition $A_1 \uplus \ldots \uplus A_m$ of A such that for all i, $\sum_{a \in A_i} a = B$. The problem is NP-complete in the strong sense [GJ79, problem SP15, page 224].*

We describe the reduction now, which is shown in Figure 9. The tree T has m leaves. Each leaf wants to dominate $B + 1$ closed subtrees (i.e., subtrees which have only closed leaves). T is required to be the child of some node l. This node l also wants to dominate the trees $t_1, \dots, t_{3m}$. For all i, the tree t_i has $a_i + 1$ open leaves.

[8]Exactly the same reduction works if we do not require open leaves to have outgoing dominance edges in solved forms; so this modified problem is NP-complete as well.

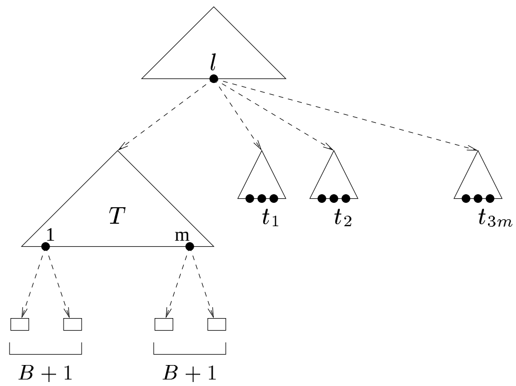

Figure 9: The dominance graph constructed in the reduction of 3-partition.

THEOREM 7.1. *The configurability problem for dominance graphs with closed leaves is NP-complete.*

Proof. Consider an instance (A, B) of the 3-partition problem and the dominance graph G constructed in the reduction. We show that the instance (A, B) has a solution iff G is configurable.

Assume first that the 3-partition problem has a solution. Observe that each of the sets A_i must have cardinality three. Let $A_i = \{a_{x_i}, a_{y_i}, a_{z_i}\}$ be one of the sets in the partition. Then $a_{x_i} + a_{y_i} + a_{z_i} = B$. We plug t_{x_i} as child into the i-th leaf of T, t_{y_i} into some leaf of t_{x_i} and t_{z_i} into some leaf of t_{y_i}. Then the tree T has $a_{x_i} + 1 + a_{y_i} + 1 + a_{z_i} + 1 - 2 = B + 1$ open leaves below its i-th leaf. These leaves are plugged with the $B+1$ closed subtrees which the i-th leaf of T wants to dominate. Finally, we plug l with T and obtain a configuration of G.

Assume next that the dominance graph G has a configuration. Consider the subtree plugged to the i-th leaf of T. It contains a subset A_i of the trees $\{t_1, \dots, t_{3m}\}$. We must have $\sum_{t_j \in A_i}(a_j + 1) \geq B + 1 + |A_i| - 1$, since $B+1$ closed subtrees must be plugged into some open leaf and since every subtree in A_i also requires an open leaf.

We next show that $|A_i| \geq 3$ for all i. It is clear that A_i cannot be empty (since $B > 0$). If A_i is a singleton, i.e. $A_i = \{t_x\}$, we have a contradiction since t_x has $a_x + 1 < B/2 + 1 \leq B + 1$ leaves. Now consider the case, where A_i consists of two elements t_x and t_y. By attaching t_x and t_y below the i-th leaf of T, we obtain $a_x + 1 + a_y + 1 - 1 < B/2 + 1 + B/2 = B + 1$ open leaves, which is also a contradiction.

Since each set A_i has cardinality at least three, since we have m sets, and since there are $3m$ elements to distribute, we conclude that $|A_i| = 3$ for all i. Thus

$\sum_{t_j \in A_i} a_j \geq B$ for every i. Finally, we observe that we have equality since $\sum_{a \in A} a = mB$. Thus we also have a solution for the 3-partition problem. $\square$

Note that for *solvability* of dominance graphs with closed leaves, Theorem 4.1 still holds. That is, solvability is still a polynomial problem. The difference to the unrestricted problem is that Lemma 2.1 breaks down: All the graphs we construct in the encoding of 3-partition are in solved form, but they may well be unconfigurable.

The relevance of this result is again in its relation to computational linguistics. There are alternative approaches to scope [Bos96] which require that the holes and roots of the trees must be paired uniquely: The roots must be "plugged" into the roots, and every hole must be plugged. This corresponds to making the holes open leaves, and all others closed leaves. Hence, we can show that the satisfiability problems of these alternative approaches must be NP-complete as well.

8 Conclusion

We have presented a polynomial time algorithm that solves the configuration problem of dominance graphs. This problem is of interest to applications: It can be used to encode satisfiability of normal dominance constraints, a formalism used in computational linguistics. Thus, our result establishes a difference in complexity between normal dominance constraints and unrestricted dominance constraints, whose satisfiability is NP-complete. Previously, no polynomial time algorithms for any interesting fragment of dominance constraints were known. Tests with a first implementation show that the presented graph algorithm also improves in runtime on a previous solver for (unrestricted) dominance constraints.

References

[AGU72] A. V. Aho, M. R. Garey, and J. D. Ullman. The transitive reduction of a directed graph. *SIAM Journal on Computing*, 1:131–137, 1972.

[Bos96] Johan Bos. Predicate logic unplugged. In *Proceedings of the 10th Amsterdam Colloquium*, pages 133–143, 1996.

[Cor94] Thomas Cornell. On determining the consistency of partial descriptions of trees. In *Proc. of the 32nd Annual Meeting of the Association for Computational Linguistics*, pages 163–170, 1994.

[DG99] Denys Duchier and Claire Gardent. A constraint-based treatment of descriptions. In *Proceedings of the 3rd Intern. Workshop on Comp. Semantics*, Tilburg, 1999.

[DN00] Denys Duchier and Joachim Niehren. Dominance constraints with set operators. In *Proceedings of the First International Conference on Computational Logic (CL2000)*, LNCS. Springer, July 2000.

[ENRX98] M. Egg, J. Niehren, P. Ruhrberg, and F. Xu. Constraints over Lambda-Structures in Semantic Underspecification. In *Proceedings COLING/ACL'98*, Montreal, 1998.

[GJ79] M.R. Garey and D.S Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness.* W.H. Freeman and Company, 1979.

[GMG86] Z. Galil, S. Micali, and H.N. Gabow. An $O(EV \log V)$ algorithm for finding a maximal weighted matching in general graphs. *SIAM Journal on Computing*, 15:120–130, 1986.

[GW98] Claire Gardent and Bonnie Webber. Describing discourse semantics. In *Proceedings of the 4th TAG+ Workshop*, Philadelphia, 1998. University of Pennsylvania.

[KMN00] Alexander Koller, Kurt Mehlhorn, and Joachim Niehren. A polynomial-time fragment of dominance constraints. In *Proc. of the 38th Ann. Meeting of the Ass. for Computational Linguistics*, 2000.

[KNT98] Alexander Koller, Joachim Niehren, and Ralf Treinen. Dominance constraints: Algorithms and complexity. In *Proc. of the 3th Conf. on Logical Aspects of Comp. Linguistics*, 1998. To appear as LNCS.

[MHF83] Mitchell P. Marcus, Donald Hindle, and Margaret M. Fleck. D-theory: Talking about talking about trees. In *Proc. of the 21st Ann. Meet. of the Ass. for Comp. Linguistics*, pages 129–136, 1983.

[MN99] K. Mehlhorn and S. Näher. *The LEDA Platform for Combinatorial and Geometric Computing.* Cambridge University Press, 1999. 1018 pages.

[MNSU] K. Mehlhorn, S. Näher, M. Seel, and C. Uhrig. The LEDA User Manual. Technical report, Max-Planck-Institut für Informatik. http://www.mpi-sb.mpg.de/LEDA/leda.html.

[Mus95] R.A. Muskens. Order-Independence and Underspecification. In J. Groenendijk, editor, *Ellipsis, Underspecification, Events and More in Dynamic Semantics.* DYANA Deliverable R.2.2.C, 1995.

[RVSW95] Owen Rambow, K. Vijay-Shanker, and David Weir. D-Tree Grammars. In *Proc. of the 33rd Annual Meeting of the Association for Computational Linguistics*, 1995.

[Sch00] G. Schäfer. Max-Weighted-Matching in general graphs. Master's thesis, Fachbereich Informatik, Universität des Saarlandes, Saarbrücken, 2000.

[TW67] J. W. Thatcher and J. B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2(1):57–81, August 1967.

[VS92] K. Vijay-Shanker. Using descriptions of trees in a tree adjoining grammar. *Computational Linguistics*, 18:481–518, 1992.

[VSWR95] K. Vijay-Shanker, David Weir, and Owen Rambow. Parsing D-Tree Grammars. In *Proceedings of the*

Intl. Workshop on Parsing Technologies, 1995.

[Zie95] T. Ziegler. Max-Weighted-Matching auf allgemeinen Graphen. Master's thesis, Fachbereich Informatik, Universität des Saarlandes, Saarbrücken, 1995.