Available online at www.sciencedirect.com

ScienceDirect

Artificial Intelligence 171 (2007) 185–196

Artificial Intelligence

www.elsevier.com/locate/artint

Determining the consistency of partial tree descriptions

Manuel Bodirsky [a,*], Martin Kutz [b]

[a] *Humboldt-Universität zu Berlin, Germany*
[b] *Max-Planck-Institut für Informatik, Saarbrücken, Germany*

Received 1 June 2006; received in revised form 27 November 2006; accepted 15 December 2006

Available online 22 December 2006

Abstract

We present an efficient algorithm that decides the consistency of partial descriptions of ordered trees. The constraint language of these descriptions was introduced by Cornell in computational linguistics; the constraints specify for pairs of nodes sets of admissible relative positions in an ordered tree. Cornell asked for an algorithm to find a tree structure satisfying these constraints. This computational problem generalizes the *common-supertree problem* studied in phylogenetic analysis, and also generalizes the network consistency problem of the so-called *left-linear point algebra*. We present the first polynomial time algorithm for Cornell's problem, which runs in time $O(mn)$, where m is the number of constraints and n the number of variables in the constraint.

Keywords: Tree descriptions; Constraint satisfaction problems; Graph algorithms; Left-linear point algebra

1. Introduction

Tree description languages became an important tool in computational linguistics over the last twenty years. Grammar formalisms have been proposed that derive logical descriptions of trees representing the syntax of a string [15,23,26]. Membership in a language is then equivalent to the satisfiability of the corresponding logical formula. In semantics, the paradigm of *underspecification* aims at manipulating the partial description of tree-structured semantic representations of a sentence rather than at manipulating the representations themselves [17,25]. One of the key issues in both constraint-based grammar and constraint-based semantic formalisms is to collect partial descriptions of trees and to *solve* them, i.e., to find a tree structure that satisfies all constraints.

Cornell [13] introduced a simple but powerful tree description language, which contains constraints for *dominance*, *precedence*, and *equality* between nodes, and disjunctive combinations of these (a formal definition is given is Section 2). Cornell also gave a saturation algorithm based on local propagations, which turned out to be incomplete. For an example of a tree description that shows this, see Section 3.4 in [3].

In this article we present the first polynomial-time algorithm that tests satisfiability of a tree description from Cornell's tree description language and directly constructs a solution to the problem instance, if one exists. A predecessor of this algorithm, which applies to a restricted language, was presented in [4]. The present algorithm, which solves the

[*] Corresponding author.
E-mail addresses: bodirsky@informatik.hu-berlin.de (M. Bodirsky), mkutz@mpi-inf.mpg.de (M. Kutz).

0004-3702/$ – see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.artint.2006.12.004

general problem of Cornell's full tree description language, runs in time $O(nm)$, where n is the number of variables and m the number of constraints in the input. The performance is achieved by a recursive strategy that works directly on the constraint graph, and avoids local consistency techniques a la [14,19] that are frequently used in constraint satisfaction.

Significance of the results. Computational linguistics is not the only area in computer science and artificial intelligence where partial tree descriptions become relevant. In fact, a fragment of Cornell's language has been introduced independently in [12] (also see [16,21]), and is known there as the *left-linear point algebra*. The best known algorithm for the so-called *network satisfaction problem* for the left-linear point algebra has a running time which is in $O(n^5)$, where n denotes the size of the input [21]. Since the left-linear point algebra is a fragment of Cornell's language, our quadratic time algorithm also yields a new and asymptotically faster algorithm for the left-linear point algebra. Details of this connection will be given in Section 3.1.

The consistency problem we study can also be posed as a constraint satisfaction problem (CSP), as formalized in e.g. [8]. The catch here is that the variables might take values from an infinite domain (it can be shown that the problem cannot be modeled as a CSP with a finite domain); see also Section 3. One important direction of research in this area is to systematically identify constraint languages that can be solved in polynomial time. In this context, our algorithmic result is interesting because it is neither based on group-theoretic techniques such as Gaussian elimination, nor on Datalog and local consistency techniques. This is in contrast to CSPs with finite domains, where all known algorithms for polynomial-time solvable algorithms involve at least one of the above two techniques [9,18].

In computational biology, *phylogenetic analysis* is a field where we have to deal with partial information about evolutionary trees. An *evolutionary tree* for a set of species is a rooted tree where the leaves are bijectively labeled with the species from the set. Constructing evolutionary trees from biological data is a difficult problem for a variety of reasons (see [20]). Many approaches assume that the evolutionary tree is built from a set of taxa based on the comparison of a single protein or a single position in aligned protein sequences, but very often the resulting tree will be different depending on which particular protein or position is used. Several trees, each from a different protein or position, must be built and be shown to be "generally consistent" before the implied evolutionary history is considered reliable. The question whether such consistency tests can be automated motivates the so-called *common-supertree problem* [20]. We will describe in Section 3.2 how the common-supertree problem can be modeled in (a fragment of) Cornell's tree description language. Therefore, the algorithm presented here also yields a new algorithm for (and a new perspective on) the common-supertree problem.

Outline. In Section 2, we introduce standard terminology for rooted trees and define Cornell's tree description language. This allows us to clearly describe in Section 3 the relationship between our results and results in qualitative temporal reasoning in artificial intelligence, phylogenetic analysis in computational biology, the left-linear point algebra in the theory of relation algebras, and the general framework of constraint satisfaction problems. In Section 5, we introduce an algorithm for a small fragment of Cornell's language. This fragment is already expressive enough to capture the common-supertree problem mentioned above. The corresponding consistency-problem is non-trivial in the sense that the algorithm proposed by Cornell is inconsistent already for this fragment (see [3]). However, the simplicity of the language allows for a smaller and simpler description of an algorithm that decides consistency. Discussing this language first will be instructive to deal with the consistency problem for the full language, which is far more involved. For the full language, we first reduce the problem to a simpler tree description language (Section 5), prove fundamental results for constraint-graphs that are associated to a partial tree description (Section 6), and finally use these results to present our algorithm and prove its correctness (Section 7). Section 8 summarizes and poses questions for future research.

2. Tree descriptions

The trees considered here are always rooted, and we consider the edges as directed, pointing away from the root. By an *ordered tree* we mean a rooted tree with a linear order on the children of each vertex and we use the terms *left* and *right* to compare them.

We follow the notation of [2]. The set of vertices of a tree T is denoted by V_T, and the vertices are usually called u, v, or w. The expression $u \lhd v$ denotes that u is the father of v and $u \lhd^* v$ (and $v \,{}^*\!\rhd\, u$) means that u *dominates* v, i.e., u is an ancestor of v in the tree (including $u = v$). We write $u \lhd^+ v$ (and $v \,{}^+\!\rhd\, u$) if $u \lhd^* v$ and $u \neq v$. If for two vertices u and v neither $u \lhd^* v$ nor $v \lhd^* u$, we say that u and v are *disjoint*, in symbols $u \perp v$. In this situation we distinguish

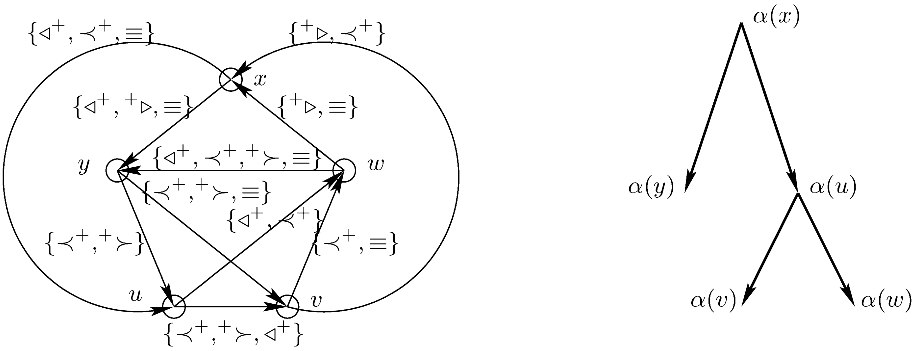
$\{\lhd^+, \prec^+, \equiv\}$
$\{{}^+\!\rhd, \prec^+\}$
x
$\{\lhd^+, {}^+\!\rhd, \equiv\}$
$\{{}^+\!\rhd, \equiv\}$
y
$\{\lhd^+, \prec^+, {}^+\!\succ, \equiv\}$
w
$\{\prec^+, {}^+\!\succ, \equiv\}$
$\{\lhd^+, \prec^+\}$
$\{\prec^+, {}^+\!\succ\}$
$\{\prec^+, \equiv\}$
u
v
$\{\prec^+, {}^+\!\succ, \lhd^+\}$
$\alpha(x)$
$\alpha(y)$
$\alpha(u)$
$\alpha(v)$
$\alpha(w)$

Fig. 1. The translation of the partial tree description of Fig. 1 into the restricted language.

two cases: either u *precedes* or *succeeds* v. A vertex u *precedes* a vertex v (and v *succeeds* u), in symbols $u \prec^+ v$ (and $v \,{}^+\!\!\succ u$), if there is a common ancestor of u and v in the tree that has two children w_1 and w_2 with $w_1 \lhd^* u$ and $w_2 \lhd^* v$ and such that u is to the left of v. We write $u \prec^* v$ if either $u \prec^+ v$ or $u = v$.

The right picture in Fig. 1 shows an example of a rooted tree with root $\alpha(x)$. In pictures we indicate the ordering of the children of a vertex by distinguishing between left and right. Here, $\alpha(v) \prec^+ \alpha(w)$ and $\alpha(y) \prec^+ \alpha(v)$, and $\alpha(x) \lhd^+ \alpha(w)$, for example.

In an ordered tree, for every pair u, v of vertices exactly one of the following relations holds:

$$u \lhd^+ v, \qquad u \,{}^+\!\!\rhd v, \qquad u \prec^+ v, \qquad u \,{}^+\!\!\succ v, \qquad u = v.$$

It is important to note that the union of the relations $\lhd^+$ and $\prec^+$ forms a strict linear order on the set of all vertices of an ordered tree, which is easily seen to be the pre-order that results from a (recursive) tree traversal that lists each node before its descendants.

We now define partial tree descriptions, due to Cornell [13], that allow to partially describe the structure of an ordered tree using arbitrary disjunctions of these five cases. To distinguish clearly between equality in this language and the common usage of the symbol '$=$', we denote the former relation by '$\equiv$'.

Definition 1. A *partial tree description* (V, C) consists of a set of variables V and a set C of binary constraints of the form $x\mathcal{R}y$, where $x, y \in V$ and $\mathcal{R} \subseteq \{\lhd^+, {}^+\!\rhd, \prec^+, {}^+\!\succ, \equiv\}$. A constraint $x\mathcal{R}y$ is *satisfied* by a pair (T, α), where T is an ordered tree and $\alpha : V \to V_T$ is a mapping from the variables to the vertices of the tree, if $\alpha(x)R\alpha(y)$ holds in the tree for *some* relation $R \in \mathcal{R}$. A pair (T, α) that satisfies all the constraints in C is called a *solution* for (V, C).

Note that the mapping α in the above definition is not required to be surjective. In particular, the induced graph on the image of α need not be connected but can be a forest. A partial tree description that has a solution is called *satisfiable* or *consistent*.

Fig. 1 shows a partial tree description and one of its solutions on the right. In this picture, a directed edge from x to x' that is labeled by $\mathcal{R}$ denotes a constraint $x\mathcal{R}x'$.

3. Related work

Cornell's tree description language is closely related to problems in various fields of computer science and artificial intelligence. We focus here on fields that are distinct from the origins of Cornell's language in computational linguistics.

3.1. The left-linear point algebra

The *left-linear point algebra* is studied in the area of binary relation algebras and algebraic logic, and can be used to model flows of time which branch into the future, but where the past is fixed [12,16,21]. Formally, a *semi-linear* (or *left-linear*) order $(O, <)$ is a partial order such that if $s, t, u \in O$, $s < u$, and $t < u$, then either $s < t$, $t < s$, or $s = t$.

In [21], the case that neither $u < v$ nor $v < u$ holds for two distinct elements u and v in a semilinear order is denoted by $u \# t$. Semi-linear orders give rise to an (abstract) relation algebra, called the *left-linear point algebra*, which has eight elements, and three atoms, represented by $<$, $>$, and #. A typical question in this context is the complexity of the so-called *network satisfaction problem*. In case of the left-linear point algebra, the network satisfaction problem can be solved in quintic time, i.e., in time $O(n^5)$, where n denotes the size of the network.

It is straightforward to see from the definitions given in [21] that the network satisfaction problem for the left-linear point algebra is just another formulation of the consistency problem for partial tree descriptions (V, C), where we restrict the constraint in C to be of the form $x\{\lhd^+\}y$ or $x\{\prec^+, {}^+\succ\}y$. In Section 4, we discuss an algorithm that solves the consistency problem for partial tree descriptions for this restricted language. The algorithm we present has a running time which is quadratic in the size of the partial tree description, and therefore outperforms the previously known algorithm for the network satisfaction problem for the left-linear point algebra.

3.2. The common-supertree problem

The *common-supertree problem* is a computational problem that was introduced and studied in computational biology [7,27], and is defined as follows. Let S be a set of trees with common leaf set L. The computational task is to decide whether there is a tree T on the leaf set L such that every tree in S is a *refinement* of T, i.e., can be obtained by a series of contractions of edges from T. If such a refinement exists, we say that the trees in S are *compatible*.

The common-supertree problem can be easily formulated with tree descriptions. In fact, we only have to use the constraint types $\{\lhd^+\}$ and $\{\prec^+, {}^+\succ\}$. Let S be a set of trees over a common leaf set L. The variables of the tree description are the vertices of the trees in S, where the inner nodes of all the trees become different variables, but the leaves are represented by the same set of variables for all trees. Tree-edges xy translate to dominance constraints $x\{\lhd^+\}y$. Siblings x, y in a tree from S will be related via the constraint $x\{\prec^+, {}^+\succ\}y$.

See Fig. 2 for an example of an incompatible set that contains two rooted trees, and its translation into an (unsatisfiable) partial tree description. In this picture, solid arcs stand for the constraint $\{\lhd^+\}$ and dashed arcs for the constraint $\{\prec^+, {}^+\succ\}$.

The size of the resulting tree description is clearly $O(\Delta^2 n)$, where n is the total number of nodes in S and Δ denotes the maximum degree in any tree of S. It is not hard to show that it has a solution if and only if there is a common supertree for the trees from S.

Note that the above definition of *compatible* should not be confused with the notion of compatible given in [24]. They say that a tree T is *compatible* with T' if there exists a subset A of the leaves of T such that the minimal subtree of T that connects all nodes in A and suppresses all nodes of degree two equals T'. In contrast, the notion of compatibility we used here is studied e.g. in [27], and was called *weak compatibility* in [24].

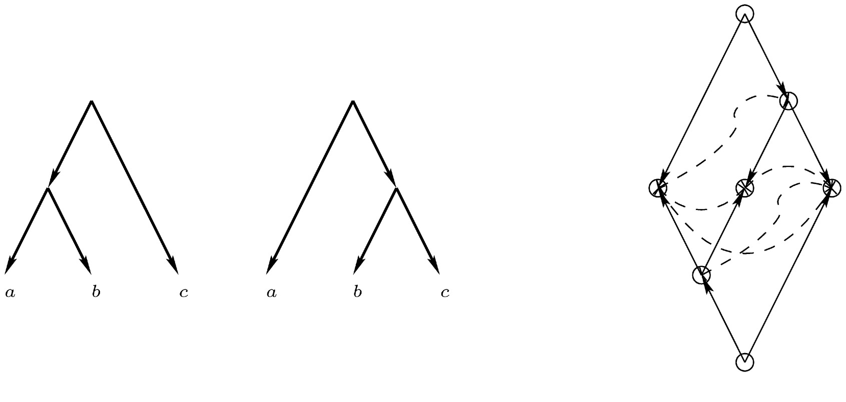

Fig. 2. Two incompatible trees and the translation into an (unsatisfiable) partial tree description.

3.3. Allen's Interval Algebra

As already observed by Cornell [13], it is possible to translate a partial tree description into a set of constraints from *Allen's Interval Algebra* [1]. In Allen's Interval Algebra, the variables denote intervals over the real line. The constraints from Allen's Interval Algebra describe possible relationships between intervals, e.g., one can impose the constraint that two intervals do not overlap. Allen's full interval constraint logic is NP-complete in its unrestricted form, but there are fragments of this logic that can be solved in polynomial time [10,22].

If the translation mentioned above only produces constraints that fall into one of the tractable fragments of Allen's Interval Algebra, we could use the translation to obtain a polynomial-time algorithm for partial tree descriptions. The idea for the translation of partial tree description (V, C) into Allen's Interval Algebra is as follows. We use V as the set of variables that denote intervals in the translation. If C contains $x\{\lhd^+\}y$, then the translation contains an interval constraint that requires that the interval for x contains the interval for y. If C contains $x\{\prec^+, {}^+\succ\}y$, then the translation contains an interval constraint that requires that the interval for x and the interval for y are disjoint. All the other partial tree descriptions can be translated analogously.

Solutions the translation only correspond to trees, if the set of intervals in the translation is *laminar*, i.e., for any pair of intervals that is not disjoint, one interval must completely contain the other. In other words, intervals must not overlap. To ensure this, we also have to add interval constraints that require that the intervals for x and y do not overlap, for all distinct variables x and y in V. It is then easy to show that the translation preserves satisfiability.

Consulting the classification of the tractable fragments of Allen's Interval Algebra given in [22], it turns out that the interval constraints obtained from translating constraints of the form $x\{\lhd^+\}y$ and $x\{\prec^+, {}^+\succ\}y$ in the above way, together with the necessary non-overlap interval constraints, do not fall into one of the tractable fragments of Allen's Interval Algebra. Hence, we cannot use this approach to solve partial tree descriptions in polynomial time.

3.4. General constraint satisfaction problems (CSPs)

The problems studied in this article (as well as the constraint satisfaction problem for Allen's Interval Algebra and its fragment) fall into a class of computational problems known as *constraint satisfaction problems* (*CSPs*), see e.g. [8]. Every problem in this class can be described as a homomorphism problem with respect to a fixed relational structure. To be precise, for a fixed structure Γ over a relational signature τ, also called the *template*, the constraint satisfaction problem of Γ is the following computational problem:

CSP(Γ)
INSTANCE: A finite τ-structure S.
QUESTION: Is there a homomorphism from S to Γ?

For relational structures Γ over a *finite* domain the computational complexity of the problems CSP(Γ) was intensively studied; see e.g. [9,18]. The most systematic approach here is based on a connection to universal algebra, which essentially says that the computational complexity of CSP(Γ) is described by a universal-algebraic object called the *clone of polymorphisms* of Γ [9].

However, the consistency problem for partial tree descriptions cannot be formulated with a finite template. But it can be formulated with a ω-categorical countable template described e.g. in [11]; see [3,6] for constraint satisfaction with infinite templates. The concept of ω-categoricity is fundamental in model-theory. It turns out that the universal-algebraic approach can also be applied to study the complexity of CSP(Γ) if Γ is ω-categorical [5].

It is interesting to observe that the algorithm presented here does not apply local consistency techniques or group-theoretic techniques, whereas all known tractable CSPs with finite templates having a finite domain make use of at least one of these two techniques [9,18].

4. An algorithm for a restricted signature

We first illustrate the idea of our algorithm for a small fragment of the tree description language, which is the fragment that corresponds to the left linear point algebra described in Section 3. We have also seen that this fragment

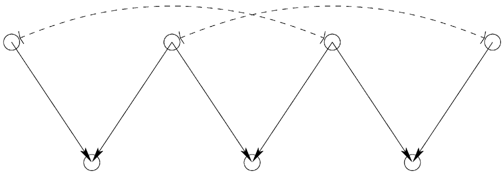

Fig. 3. An inconsistent partial tree description.

is already expressive enough to cover the common-supertree problem from phylogenetic analysis mentioned in the introduction. In this reduced constraint language we only allow constraints of the types

$$x\{\lhd^+\}y \quad \text{and} \quad x\{\prec^+, {}^+\succ\}y.$$

The first constraint is called (*strict*) *dominance* and the second *disjointness*, and we use the shorthands $x \lhd^+ y$ and $x \perp y$, respectively, for these two types of literals.

Such a set of constraints can be viewed as a graph $(V; \lhd^+, \perp)$ with two types of edges, namely directed edges $\lhd^+$ and undirected edges $\perp$. Observe that the binary relations $\lhd^+$ and $\perp$ are now defined both on the vertices of a tree and on the nodes of an instance. The reference, however, should always be clear. We would like to stress that the two constraints do not allow for equality, i.e., $x \lhd^+ y$ means that in any solution, x must lie strictly above y. We say that a constraint graph G has a solution (T, α) (and is satisfiable) if the corresponding partial tree description has this solution (and is satisfiable). We also use some of the standard graph theory notation, e.g., the subgraph of $G = (V; \lhd^+, \perp)$ induced by a subset S of V is denoted by $G[S]$, and $G - S$ denotes the constraint graph $G[V \setminus S]$.

In pictures we draw this *constraint graph* with two different types of arcs. For a dominance edge $x \lhd^+ y$ we draw a directed arc from x to y, for a disjointness edge $x \perp y$ we draw a dashed line without direction. Fig. 3, for instance, shows such a constraint graph.

The basic idea behind our algorithm to find a solution to a partial tree description (V, C) (if one exists) is to pick a node x of V as the root of our solution, then to decompose (V, C) into smaller parts, and to recursively determine solutions to these parts, which will become the subtrees below x. With this perspective, the following definition comes naturally.

Definition 2. Let (V, C) be a partial tree description. Then a variable $x \in V$ is *free* if (V, C) has a solution (T, α) in which $\alpha(x)$ is the root of T.

By definition, a partial tree description with a free node is satisfiable. We shall see soon that under a simple connectivity condition for the constraint graph, the converse is also true: a satisfiable instance with a "connected" constraint graph must have a free node, i.e., a node that dominates all others.

The crucial point here is, of course, in what sense the constraint graph, which contains directed and undirected edges, should be connected. Let us briefly recall some conventions. An *undirected* path in a directed graph (also called *digraph*) may use arcs in any direction, ignoring their orientation. A digraph $D = (V; E)$ is *strongly* (*weakly*) *connected* if there is a directed (an undirected) path from a to b for any two vertices $a, b \in V$. A *strongly* (*weakly*) *connected component of* D is a maximal strongly (weakly) connected induced subgraph U of D.

Now, we say that a constraint graph $(V; \lhd^+, \perp)$ is *dominance-connected* if the digraph $(V; \lhd^+)$ is weakly connected. This notion of connectivity is crucial for our desired characterization of free nodes. The following lemma marks the first step.

Lemma 1. *Let $G = (V; \lhd^+, \perp)$ be a dominance-connected constraint graph, and let y and y' be variables in V. Then for every solution (T, α) of G there exists a variable $x \in V$ such that $\alpha(x) \lhd^* \alpha(y)$ and $\alpha(x) \lhd^* \alpha(y')$ in T.*

Proof. Since the vertices y and y' are weakly connected in $(V; \lhd^+)$ there exists a chain of nodes $(y_1, \ldots, y_r)$ that starts at $y = y_1$, ends at $y' = y_r$, and where $(y_i, y_{i+1}) \in \lhd^+$ or $(y_{i+1}, y_i) \in {}^+\rhd$ for $1 \leqslant i \leqslant r - 1$. We prove by induction on r that for every solution (T, α) of (V, C) there exists an index $j \in \{1, \ldots, r\}$ with $\alpha(y_j) \lhd^+ \alpha(y_0)$ and $\alpha(y_j) \lhd^+ \alpha(y_r)$ in T. If $r = 0$ or $r = 1$ then we can choose x to be either $\alpha(y_0)$ or $\alpha(y_r)$. Otherwise, we can apply the induction hypothesis to the chain $(y_1, \ldots, y_{r-1})$. Thus, there exists a j, $0 \leqslant j \leqslant r - 1$, such that $\alpha(y_j)$ is a common ancestor of $\alpha(y_1)$ and $\alpha(y_{r-1})$ in T. If $\alpha(y_{r-1}) \lhd^+ \alpha(y_r)$ then $\alpha(y_j)$ is also a common ancestor of $\alpha(y_1)$ and $\alpha(y_r)$, so we can choose $x = y_j$. Otherwise, $\alpha(y_r) \lhd^+ \alpha(y_{r-1})$ in T. Hence, both $\alpha(y_1)$ and $\alpha(y_r)$ are ancestors of y_{r-1} in T. Since T is a tree, it follows that $\alpha(y_r) \lhd^* \alpha(y_j)$ or $\alpha(y_j) \lhd^* \alpha(y_r)$ in T. In the first case, we choose $x = y_r$, and in the second $x = y_j$. □

As promised, we now show that every satisfiable dominance-connected constraint graph has a free node.

Proposition 1. *A satisfiable partial tree description (V, C) with a dominance-connected constraint graph has a free node.*

Proof. Assume there is a solution (T, α) for (V, C) and consider the vertices v that are topmost in T with respect to $\lhd^+$ such that $v = \alpha(x)$ for some node $x \in V$. If there is only one such vertex v, then the subtree rooted at v together with α is also a solution of (V, C). But since there is a node $x \in V$ such that $v = \alpha(x)$, this contradicts the assumption that x is not free. If there are two distinct topmost nodes v, v', then v and v' are disjoint in T. Since the constraint graph is dominance-connected, we can apply Lemma 1, and obtain a contradiction to the assumption that v and v' lie topmost in T. □

For the algorithm we need a concise graph-theoretic characterization of free nodes that can be checked efficiently.

Proposition 2. *Let $G = (V; \lhd^+, \perp)$ be the constraint graph of a satisfiable partial tree description. Then a node $x \in V$ is free if and only if*

(C1) *there is no arc $y \lhd^+ x$ in G and*
(C2) *there is no edge $x \perp y$ in G.*

Proof. If there is another y such that $y \lhd^+ x$, then the vertex x cannot be topmost in any solution of G, and thus can not be free. If the vertex x is involved in a disjointness constraint it can also not be free, since the root of a tree is not disjoint to the other nodes in the tree.

Conversely, assume that the node x of a satisfiable set of constraints satisfies (C1) and (C2). The dominance-connected components of $G[V - \{x\}]$ are also satisfiable and must thus by Proposition 2 have free nodes. Then we have the following solution for G: introduce a tree node x and let the solutions of the dominance-connected components become the subtrees of this node. The disjointness constraints between the different dominance-connected components are thus satisfied by construction and it is clear from (C1) and (C2) that all the constraints on x are also satisfied. □

Example. Let us revisit the example shown in Fig. 3. Note that instance does not contain a node x that satisfies (C1) and (C2). By Proposition 2, there is no free node, and hence the instance has no solution. But if any constraint in the instance is removed, we find a node satisfying both conditions, and hence any such instance has a solution.

Fig. 4 shows the algorithm for the restricted constraint language with the two relations $\lhd^+, \perp$. For a dominance-connected constraint graph, the function Solve first selects a free node x as root and then links the recursively computed

```
Solve($G$)
// input: a dominance-connected constraint graph $G = (V; \lhd^+, \perp)$
// constructs a tree $T$ and a mapping $\alpha\colon V \to V_T$

  pick a node $x$ of $G$ satisfying (C1) and (C2);
  if no such free node exists then return "problem has no solution";
  create a new tree node $r$ and let $\alpha(x) := r$;
  compute the dominance-connected components $C_1, \dots, C_k$ of $G - \{x\}$;
  for $i = 1$ to $k$ do
     call Solve($G[C_i]$) and make the returned root a new child of $r$;
  od
  return root $r$;
```

Fig. 4. The function Solve for constraints of the type $\lhd^+$ and $\perp$.

solutions to the dominance-connected components C_i of the remainder $G - \{x\}$ below the α-image r of x in the correct order.

Theorem 1. *There is an algorithm that decides satisfiability of a given partial tree description* (V, C) *in the restricted constraint language in* $\mathrm{O}(|V||C|)$ *time.*

Proof. It is clear that we can construct the constraint graph $(V; \lhd^+, \perp)$ of the given partial tree description in linear time. We then apply the algorithm shown in Fig. 4 on the constraint graph. For an instance that is not dominance-connected, we can first introduce a dummy node z together with a $\lhd^+$-arc from z to each other node to make the graph connected. The node z then becomes the root of the output, and since α is not required to be onto the solution will be valid also for the original graph without z.

If the algorithm detects a weakly connected component without a free node, we know by Proposition 1 that the constraints do not have a solution. Otherwise Proposition 2 guarantees that we can proceed and make a node x satisfying (C1) and (C2) the root of our solution. The running time is dominated by the repeated computations of the connected components of constraint graphs. We can use depth first search to compute the connected components in time $\mathrm{O}(|V| + |C|)$, and this will be done at most $|V|$ times. We can assume that $|V|$ is smaller than $|C| - 1$, otherwise the first call of the algorithm divides the problem into instances where the assumption holds. Hence the algorithm runs in time $\mathrm{O}(|V||C|)$. □

5. Reduction to four basic constraints

We now turn to the general problem of deciding the consistency of a partial tree description for the full language introduced by Cornell. In order to get control over the 2^5 subsets of the relation set $\mathcal{R} \subseteq \{\lhd^+, {}^+\!\rhd, \prec^+, {}^+\!\succ, \equiv\}$, we first show how to express each of them with a smaller language that contains the following four different constraint types only:

$$x\{\lhd^+, \equiv\}y, \qquad x\{\prec^+, \equiv\}y,$$
$$x\{\lhd^+, {}^+\!\rhd, \prec^+, {}^+\!\succ\}y, \qquad x\{\prec^+, {}^+\!\succ, \equiv\}y. \tag{1}$$

We reduce each of the 32 subsets either directly to an intersection of constraints from (1), or we build small gadgets with new dummy variables that can simulate the original constraint.

The constraints $\{{}^+\!\rhd, \equiv\}$ and $\{{}^+\!\succ, \equiv\}$ are simply the first and second constraint of (1) flipped, and the singletons $\{\lhd^+\}$, $\{\prec^+\}$, and $\{\equiv\}$ can be written as intersections of these. The two extremal sets $\{{}^+\!\rhd, \lhd^+, \prec^+, {}^+\!\succ, \equiv\}$ and $\emptyset$ are not needed since the former imposes no restrictions on the tree and the latter is, by definition, unsatisfiable. If we show how to express the constraints

$$x\{\lhd^+, {}^+\!\rhd, \equiv\}y, \qquad x\{\lhd^+, \prec^+, {}^+\!\succ, \equiv\}y,$$
$$x\{\lhd^+, \prec^+, \equiv\}y, \qquad x\{\lhd^+, {}^+\!\succ, \equiv\}y \tag{2}$$

with those in (1) we are done, since the remaining constraints are easily representable as flippings and intersections of constraints from (1) and (2).

For each constraint $x\{\lhd^+, {}^+\!\rhd, \equiv\}y$, we introduce a new variable z and replace it by the two constraints $x\{\lhd^+, \equiv\}z$ and $y\{\lhd^+, \equiv\}z$. By the properties of a tree, these two constraints imply $x\{\lhd^+, {}^+\!\rhd, \equiv\}y$. Conversely, every solution for the original constraints can be modified to satisfy also the new constraints.

Constraints of the form $x\{\lhd^+, \prec^+, \equiv\}y$ are replaced by the two constraints $x\{\lhd^+, \equiv\}z$ and $z\{\prec^+, \equiv\}y$, where z is a new variable. Similarly, we can replace $x\{\lhd^+, {}^+\!\succ, \equiv\}y$ by the two constraints $x\{\lhd^+, \equiv\}z$ and $z\{{}^+\!\succ, \equiv\}y$. Finally, we replace $x\{\lhd^+, \prec^+, {}^+\!\succ, \equiv\}y$ by $x\{\lhd^+, \equiv\}y$ and $z\{\prec^+, {}^+\!\succ, \equiv\}y$, where z is again a new variable.

Thus we can express all constraints in Cornell's language with the four basic binary relations from (1). Our algorithm works on tree descriptions consisting of such basic constraints only. We therefore introduce special names and notation for these four types:

- $x\{\lhd^+, \equiv\}y$: the *dominance* constraint $x \lhd^* y$,
- $x\{\prec^+, \equiv\}y$: the *precedence* constraint $x \prec^* y$,

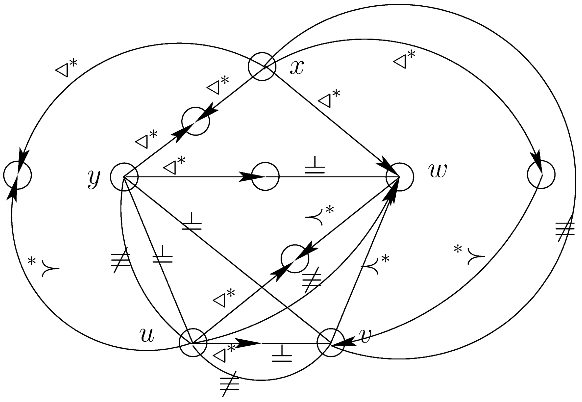

Fig. 5. The translation of the example from Fig. 1 into the basic constraints.

- $x\{\prec^+, {}^+\!\succ, \equiv\}y$: the *disjoint-or-equal* constraint $x \dot{\perp} y$,
- $x\{\lhd^+, {}^+\!\rhd, \prec^+, {}^+\!\succ\}y$: the *inequality* constraint $x \not\equiv y$.

Note that from now on, the notions of *dominance* and *precedence* relation are to be considered non-strict, i.e., they stand for reflexive relations.

In the following we apply this reduction to the example presented in Fig. 1. The result is shown in Fig. 5. Since the disjoint-or-equal and the inequality constraint are symmetric, we indicate them as undirected edges and not as arcs.

6. Constraint graphs and freeness

Our algorithm for the full tree description language uses the ideas that we discussed for the restricted setting. The first important difference to the restricted setting is that now several variables may map to the same tree node. Hence, we shall need a generalized notion of freeness.

Definition 3. Let (V, C) be a partial tree description. Then $S \subseteq V$ is called *free* if there is a solution (T, α) for (V, C) such that $S = \alpha^{-1}(r)$, where r is the root of T.

Similarly to the restricted setting we want to show that under a certain connectedness condition free sets must exist, and we also want to find practical characteristics of free sets to identify them algorithmically. To this end, we adapt the concept of the constraint graph from Section 4. The concept of weak connectedness with respect to dominance edges must be replaced by a more complex definition that is based on an auxiliary graph containing dominance and precedence arcs. We define the directed *P-graph* (V, P) on V with arc set

$$P := \{xy \mid C \text{ contains } x \prec^* y \text{ or } x \lhd^* y \text{ or } y \lhd^* x\}$$

and call a partial tree description (V, C) *dominance connected* if its P-graph is strongly connected.

Example. Consider the instance drawn in Fig. 5. The P-graph of this example is strongly connected. However, if we consider the graph where the vertex x and all constraints imposed on x are removed, then the P-graph has four connected components.

The following proposition is the analog of Proposition 1 for the restricted case.

Proposition 3. *A satisfiable partial tree description with a strongly connected P-graph has a free set.*

Proof. Suppose (T, α) is a solution of C. We consider the set S of nodes in C that map to the minimum of the linear order $\lhd^+ \cup \prec^+$ in T, i.e., the nodes that map to the leftmost and topmost vertex u in T. If the vertex u dominates all vertices in $\alpha(V)$ then S is actually a free set. We claim that this must always be the case. So assume for contradiction that there is a variable $y \in V$ such that $\alpha(y)$ is not dominated by u. Since the P-graph of C is strongly connected it contains a path $y, x_1, x_2, \dots, x_k$ from $y \notin S$ to some $x_k \in S$. The path $\alpha(y), \alpha(x_1), \alpha(x_2), \dots, \alpha(x_k)$ must eventually enter the subtree rooted at u, with $\alpha(x_j)$, say. Then $\alpha(x_{j-1})$ comes before u in $\lhd^+ \cup \prec^+$, contradicting the minimality of u with respect to the linear order $\lhd^+ \cup \prec^+$. □

For the analogy to Proposition 2, the graph-theoretic characterization of free sets, we define a directed graph (V, D) (called the *D-graph*) that is based on the $\lhd^*$ relation but in addition also takes precedence constraints into account. Precisely, we let

$$D := \{xy \mid C \text{ contains } x \lhd^* y \text{ or } x \perp y \text{ or } x \prec^* y \text{ or } y \prec^* x\}.$$

Example. Again, consider the instance drawn in Fig. 5. The strongly connected components of the P-graph has five strongly connected components.

We would like to remark that the digraphs (V, D) and (P, D) have an interesting symmetry: D is based on dominance constraints and contains bidirected precedence constraints, while P is the union of the precedence constraints and all bidirected dominance constraints. However, a deeper reason for this similarity is elusive. The purposes of the two digraphs are very different in nature.

The characterization of Proposition 2 now becomes a similar statement about the D-graph, with the old conditions (C1) and (C2) merged into one, (C). On the other hand, inequality now has to be handled explicitly, which again gives a second condition, (I).

Proposition 4. *Let (V, C) be a satisfiable partial tree description. Then $S \subseteq V$ is a free set of nodes if and only if the following two conditions hold*:

(C) *there is no edge $xy \in D$ such that $x \notin S$ and $y \in S$ and*
(I) *there is no pair $x, y \in S$ such that $x \not\equiv y$.*

Proof. The two conditions are clearly necessary, because a free set denotes the root of a tree, which dominates all other vertices, and because a constraint $x \not\equiv y$ explicitly forbids to map x and y to the same tree node.

For the other implication, let S be a set of nodes that satisfies conditions (C) and (I). Let $C_1, \dots, C_k$ be the strongly connected components of the P-graph without S. Fix an arbitrary linear extension of the acyclic structure that is defined by the P-graph on $C_1, \dots, C_k$. Since C is satisfiable, the subgraph G_i of C induced by the component C_i has a solution (T_i, α_i), for all $1 \leqslant i \leqslant k$. Introduce a new vertex v, and add $T_1, \dots, T_k$ as subtrees according to this linear order of the components. The resulting tree is denoted by T. Now consider the mapping α that is the extension of all the α_i and maps the vertices in S to v.

We claim that (T, α) is a solution for (V, C), and therefore S is a free set of nodes. It is clear that (T, α) satisfies all inequality constraints, because by (I) all inequality constraints are either within one of the components, or between different components, or between S and a component, and in all these cases the inequality is satisfied in (T, α). If C contains a disjointness constraint between x and y, then condition (C) implies that one out of the following cases applies. Either x and y are both in S, in which case the disjointness is satisfied by (T, α). Or, x and y are in the same component C_i, in which case the constraint is satisfied, because it is satisfied in (T_i, α_i) as well. By the definition of the P-graph it cannot be that x and y are in different components.

If C contains a dominance constraint between x and y, then condition (C) implies that y cannot be in S. Moreover, x and y cannot be in different components $C_1, \dots, C_k$, by the definition of the P-graph. Therefore, either x and y are both in the same component or both in S, or x is in S and y is in S; in all cases the dominance constraint is satisfied by (T, α).

Finally, suppose C contains a precedence constraint between x and y. Again, we see that the constraint is satisfied if both x and y lie in S or both lie in the same component $C_1, \dots, C_k$. If they lie in different components, then the precedence constraint is satisfied because the subtrees below v in T were arranged according to a linear extension of the acyclic structure defined by the P-graph on the components $C_1, \dots, C_k$. □

```
Solve($C$):
    Compute the scc's of the $P$-graph of $(C)$,
    and let $C_1, \ldots, C_k$ be a linear extension of their acyclic structure
    create a new vertex $r$ with children
    Solve_con($C[C_i]$), for $1 \leqslant i \leqslant k$, in this order
    return $r$

Solve_con($C$):
    precondition: The $P$-graph of $C$ is strongly connected
    if no set of nodes satisfying (C) and (I) exists
    then return "problem has no solution"
    else choose a set of nodes $S$ satisfying $(C)$ and $(I)$
    let $r$ be the root of the tree Solve($C[V - S]$) and set $\alpha(S)$ to $r$
    return $r$
```

Fig. 6. The function Solve for the full tree description language.

7. The algorithm

We now turn the Propositions 4 and 3 on free sets of nodes into an algorithm for Cornell's full language. From a given input with constraints $\lhd^*$, $\prec^*$, $\perp$, and $\not\equiv$, we first create the D-graph and P-graph and then recursively create a tree by decomposing the vertex set into dominance-connected components and extracting free sets as roots.

Fig. 6 shows the algorithm, which consists of two procedures, Solve and Solve_con. We abuse notation and denote a partial tree description (V, C) simply by C. Initially, for a given instance C, we call Solve(C), which partitions the variable set into the strongly connected components of P-graph. If the P-graph of C is strongly connected, the procedure Solve_con can be applied to C. The algorithm contains a statement *choose*, that influences *which* free set of nodes is selected for the solution. We discuss the issue of how to find such a free set in the running-time analysis below.

Proposition 5. *The algorithm* Solve *of Fig.* 6 *decides satisfiability of a given partial tree description* (V, C) *in time* $\mathrm{O}(|V||C|)$.

Proof. In Solve, the strongly connected components of the precedence graph of C can be computed in linear time in the input size. We then call the procedure Solve_con on the strongly connected components. If it returns "problem has no solution", the constraints were unsatisfiable by Proposition 3. Otherwise, Proposition 4 guarantees that we can construct a solution by selecting a free set as root. Because of (C1), a free set must be the union of strongly connected components of the D-graph into which no D-arcs enter. Any strongly connected component contained in a free set is a free set as well. These strongly connected components can easily be found by depth-first search in time $\mathrm{O}(n + m)$.

The strongly connected components are computed at each level of the recursion. Since we take out at least one vertex in each call of Solve_con, and since in all calls of Solve_con (except possibly the first call) n is in $\mathrm{O}(m)$, the overall running time is in $\mathrm{O}(nm)$. □

8. Conclusion and outlook

We presented an efficient algorithm that decides whether a partial tree description is consistent; if this is the case, the algorithm constructs a solution. This solves an open problem by Cornell, who introduced these tree descriptions in computational linguistics [13]. The consistency test and construction of one solution can be done in quadratic time in the input size.

We would like to conclude with a series of open problems for future research that are motivated in the various related areas mentioned in Section 3.

- As mentioned in Section 3, the consistency problem for partial tree descriptions can be formulated as a constraint satisfaction problem for an ω-categorical template Λ. For ω-categorical templates, the universal-algebraic approach applies, and in particular it can be shown that the complexity of CSP(Λ) is captured by the polymorphism clone of Λ. The open problems in this context are: 1) Describe the polymorphism clone of Λ. 2) What are the polymorphism responsible for the tractability of CSP(Λ)? An answer to these questions would give us descriptions of constraint languages with higher-ary constraints (all constraints considered in this article were binary) that can still be solved in polynomial time.

- In Section 3, we pointed to two different notions of compatibility of phylogenetic trees that were studied in the literature. Only for one of them we described how it can be modeled with partial tree descriptions. Is it possible to come up with a tree-description language that allows to model the common-supertree problem for the other notion of compatibility that was used by [24]?
- The known algorithms for the tractable fragments of Allen's Interval Algebra are all based on local-consistency techniques. Is it possible to find algorithms with better running times for these fragments, using the algorithmic techniques developed in this article? In particular, can we avoid local consistency techniques, similarly to the algorithm presented here?

Acknowledgements

We would like to thank Pierre Flener for drawing our attention to the notion of compatibility of trees used in [24].

References

[1] J.F. Allen, Maintaining knowledge about temporal intervals, Communications of the ACM 26 (11) (1983) 832–843.
[2] R. Backofen, J. Rogers, K. Vijay-Shanker, A first-order axiomatization of the theory of finite trees, Journal of Logic, Language, and Information 4 (1995) 5–39.
[3] M. Bodirsky, Constraint satisfaction with infinite domains, Dissertation, Humboldt-Universität zu Berlin, 2004.
[4] M. Bodirsky, M. Kutz, Pure dominance constraints, in: Proceedings of the 19th Annual Symposium on Theoretical Aspects of Computer Science (STACS'02), in: Lecture Notes in Computer Science, vol. 2285, Antibes, Juan le Pins, Springer, Berlin, 2002, pp. 287–298.
[5] M. Bodirsky, J. Nešetřil, Constraint satisfaction with countable homogeneous templates, in: Proceedings of Computer Science Logic (CSL'03), Vienna, 2003, pp. 44–57.
[6] M. Bodirsky, J. Nešetřil, Constraint satisfaction with countable homogeneous templates, Journal of Logic and Computation 16 (3) (2006) 359–373.
[7] D. Bryant, Building trees, hunting for trees, and comparing trees, PhD thesis at the University of Canterbury, 1997.
[8] A. Bulatov, P. Jeavons, A. Krokhin, The complexity of constraint satisfaction: An algebraic approach, in: Structural Theory of Automata, Semigroups and Universal Algebra, Montreal, 2003, NATO Science Series II: Mathematics, Physics, Chemistry 207 (2005) 181–213.
[9] A. Bulatov, A. Krokhin, P.G. Jeavons, Classifying the complexity of constraints using finite algebras, SIAM Journal on Computing 34 (2005) 720–742.
[10] H.-J. Bürckert, B. Nebel, Reasoning about temporal relations: A maximal tractable subclass of Allen's interval algebra, Journal of the ACM 42 (1) (1995) 43–66.
[11] P.J. Cameron, Oligomorphic Permutation Groups, Cambridge University Press, Cambridge, 1990.
[12] S. Comer, A remark on chromatic polygroups, Congressus Numerantium (1983) 85.
[13] T. Cornell, On determining the consistency of partial descriptions of trees, in: 32nd Annual Meeting of the ACL (ACL'94), 1994, pp. 163–170.
[14] R. Dechter, P. van Beek, Local and global relational consistency, Journal of Theoretical Computer Science 173 (1) (1997) 283–308.
[15] D. Duchier, S. Thater, Parsing with tree descriptions: A constraint-based approach, in: Sixth International Workshop on Natural Language Understanding and Logic Programming (NLULP'99), 1999, pp. 17–32.
[16] I. Düntsch, Relation algebras and their application in temporal and spatial reasoning, Artificial Intelligence Review 23 (2005) 315–357.
[17] M. Egg, A. Koller, J. Niehren, The constraint language for lambda structures, Journal of Logic, Language, and Information 10 (2001) 457–485.
[18] T. Feder, M. Vardi, The computational structure of monotone monadic SNP and constraint satisfaction: A study through Datalog and group theory, SIAM Journal on Computing 28 (1999) 57–104.
[19] E.C. Freuder, A sufficient condition for backtrack-free search, Journal of the ACM 29 (1) (1982) 24–32.
[20] D. Gusfield, Algorithms on Strings, Trees, and Sequences, Computer Science and Computational Biology, Cambridge University Press, New York, 1997.
[21] R. Hirsch, Expressive power and complexity in algebraic logic, Journal of Logic and Computation 7 (3) (1997) 309–351.
[22] P. Jeavons, P. Jonsson, A.A. Krokhin, Reasoning about temporal relations: The tractable subalgebras of Allen's interval algebra, Journal of the ACM 50 (5) (2003) 591–640.
[23] M.P. Marcus, D. Hindle, M.M. Fleck, D-theory: Talking about talking about trees, in: Proceedings of the 21st Annual Meeting of the ACL (ACL'83), 1983, pp. 129–136.
[24] M.P. Ng, M. Steel, N.C. Wormald, The difficulty of constructing a leaf-labelled tree including or avoiding given subtrees, Discrete Applied Mathematics 98 (2000) 227–235.
[25] M. Pinkal, Radical underspecification, in: Proceedings of the 10th Amsterdam Colloquium, 1996, pp. 587–606.
[26] J. Rogers, V. Shanker, Reasoning with descriptions of trees, in: Proceedings of the 30th Meeting of the ACL (ACL'92), 1992, pp. 72–80.
[27] M. Steel, The complexity of reconstructing trees from qualitative characters and subtrees, Journal of Classification 9 (1992) 91–116.