The complexity of finding the maximum spanning DAG and other restrictions for DAG parsing of natural language

Natalie Schluter
Center for Language Technology
University of Copenhagen
Copenhagen, Denmark
natalie.elaine.schluter@jur.ku.dk

Abstract

Recently, there has been renewed interest in semantic dependency parsing, among which one of the paradigms focuses on parsing directed acyclic graphs (DAGs). Consideration of the decoding problem in natural language semantic dependency parsing as finding a maximum spanning DAG of a weighted directed graph carries many complexities. In particular, the computational complexity (and approximability) of the problem has not been addressed in the literature to date. This paper helps to fill this gap, showing that this general problem is APX-hard, and is NP-hard even under the planar restriction, in the graph-theoretic sense. On the other hand, we show that under the restriction of projectivity, the problem has a straightforward $O(n^3)$ algorithm. We also give some empirical evidence of the algorithmic importance of these graph restrictions, on data from the SemEval 2014 task 8 on Broad Coverage Semantic Dependency Parsing.

1 Introduction

Consideration of the decoding problem in natural language semantic dependency parsing as finding a maximum spanning DAG of a weighted directed graph carries many complexities that have not been addressed in the literature to date. Amongst these are the problem's computational complexity (and its approximability). The decoding problem for semantic dependency parsing was first introduced as the maximum spanning directed acyclic graph problem (MSDAG) by McDonald and Pereira (2006), where it is stated to be NP-hard.[1] The MSDAG problem asks for the highest weighted spanning sub-DAG of an input weighted digraph.

In this paper, we explain the APX-hardness of MSDAG, by relating it to the almost identical minimum weighted feedback arc set and maximum weighted acyclic subgraph problems. The proof of MSDAG's APX-hardness seems to discourage its use for decoding in semantic dependency parsing. However, unlike in syntactic dependency (tree) parse decoding, where projective decoding given by Eisner (1996)'s algorithm has a slightly higher computational complexity ($O(n^3)$) than the non-projective (Tarjan) maximum spanning tree algorithm ($O(n^2)$) (Tarjan, 1977; Chu and Liu, 1965; Edmonds, 1967; McDonald et al., 2005), finding the maximum spanning projective dependency DAG is tractable and can be found in time $O(n^3)$, contrary to its APX-hard non-projective counterpart.

Projective MSDAG has been referred to in the semantic dependency parsing literature as "*planar* MSDAG", which is an unfortunate mismatch with long-established graph theoretical terminology, that we need in this paper. As we discuss below, the planar MSDAG problem is in fact NP-hard, where "planar" is used in the graph theoretical sense. We generalise the definition of projectivity from tree models of syntax theory, which forbids crossing edges, to digraphs.

The projectivity restriction itself has not been linguistically motivated to date. However, an efficient

[1] McDonald and Pereira (2006) provide a reference to (Heckerman et al., 1995) for this fact. This fact is actually indirectly shown much earlier, as we discuss in Section 4.

*Proceedings of the Fourth Joint Conference on Lexical and Computational Semantics (*SEM 2015)*, pages 259–268,
Denver, Colorado, June 4–5, 2015.

exact algorithm for this restriction would pose a starting point for various relaxations of the definition (also known as *mild non-projectivity*) that reflect linguistic description of the data, as has been done for the Eisner algorithm (in for example (Bodirsky et al., 2005; Gómez-Rodríguez et al., 2011; Pitler et al., 2012; Pitler et al., 2013; Satta and Kuhlmann, 2013)). This projectivity restriction has been inherent in transition-based approaches to semantic dependency parsing (for example, in (Sagae and Tsujii, 2008; Titov et al., 2009)), without any study of the complexity nor proof of the power of the automaton models in recognising all projective DAGs. So, in terms of computational efficiency, we provide theoretical justification for the already used restriction of projectivity in DAG parsing, exhibiting a dynamic programming algorithm for this task, which runs in polynomial time.

Previous automaton approaches to DAG parsing can roughly be separated into two camps: one similar to (Sagae and Tsujii, 2008), which assumes projectivity of graphs in the data and parses without carrying out any transformation to relax the constraint of projectivity, and another similar to (Titov et al., 2009), which attempts (online) to find a re-ordering of the words in the sentence such that the resulting graph is projective. The latter approach assumes, as we will explain, precisely *outerplanarity* of the graphs, which, it turns out, is also NP-hard (Cf. §2 and §7). With respect to the data that we consider here, it turns out that the assumption of graph outerplanarity is well represented (almost all graphs among three data sets being outerplanar), whereas the percentage of projective graphs differs greatly from one dataset to another (from 57% to 84%, Cf. Section 5).

The projective MSDAG algorithm presented here is a first-order decoding algorithm, and empirical research on semantic parsing seems to have already gone beyond this (for example, in (Martins and Almeida, 2014)); moreover, first-order decoding using MSDAG in general seems not to be appropriate (Schluter, 2014), though the empirical work presented by Martins and Almeida (2014) on digraph decoding using a second order model suggests the relevance of higher-order DAG decoding in semantic dependency parsing. Manufacturing higher-order parsing algorithms in the sense of (McDonald and Satta, 2007; Carreras, 2007) from the tree decoding literature, based on the algorithm presented here is straightforward. And we believe that it is these latter algorithms, rather, that would provide the basis for empirical studies based on the generally theoretical research presented here.

2 Preliminaries

A graph is called *planar* if it can be drawn in the plane with no crossing edges. Each maximal region of the plane surrounded by edges of the planar graph drawn in the plane is called a *face*. There is one *outer* or *unbounded* face and some number of *inner* or *bounded* faces. If a connected planar graph can be written in the plane so that all vertices are on the outer face, then we call the graph *outerplanar*. A *connected component* of a graph is a maximal subgraph in which any two vertices are connected to each other by a path. A *digraph* is a directed graph (where edges have an orientation). A *DAG* is a digraph without any directed cycles. Consider the underlying undirected graph H of a digraph G. A *weakly connected component* of G is a maximal sub-digraph whose underlying undirected graph is a connected component of H.

Notation. We put $[i, j] := \{i, i+1, \ldots, j-1, j\}$, for $i \leq j$, and $[i] := [1, i]$.

In this paper, edge weights can be positive or negative and not zero, unless otherwise stated.

Hardness of approximability. APX is the class of all NP optimisation problems that can be solved in polynomial time with approximation ratio bounded by some constant. A problem is APX-hard if there is a PTAS-reduction from every problem in APX to that problem.[2] In this paper, we use a simpler type of PTAS-reduction called an L-reduction (linear reduction), which intuitively is a mapping between problems so that (approximate) solutions differ only by some constant factor. Any L-reduction is also a PTAS-reduction (but not vice versa).

Definition 1. *An L-reduction from problem A to problem B with respective cost functions c_A and*

[2] A discussion of PTAS-reductions is out of the scope of this paper. The definition of a PTAS-reduction can be found, for example, in (Wegener, 2005).

c_B consists of a pair of polynomial-time computable functions f and g such that:

- *if $x \in A$ then $f(x) \in B$,*
- *if y is a solution to $f(x)$, then $g(y)$ is a solution to x,*
- *there exists a positive constant α such that the optimal solution for $f(x) \in B$ ($opt_B(f(x))$) is bounded by a factor α of the optimal solution for $x \in A$ ($opt_A(x)$), and*

$$opt_B(f(x)) \leq \alpha \cdot opt_A(x)$$

- *there exists a positive constant β bounding differences between solutions and optimal solutions*

$$|opt_A(x) - c_A(g(y))| \leq \beta \cdot |opt_B(f(x)) - c_B(y)|.$$

3 Generalising projectivity to DAGs

In statistical natural language syntactic or semantic dependency parsing decoding problems, the input is a sequence of n words, $W = \langle w_1, \ldots, w_n \rangle$, called the *sentence* and a further set of weighted asymmetric binary relations (directed edges) between these words (nodes). The task is to output a most likely connected and spanning digraph over those words, where the formal expressivity of the structure is defined with respect to the linguistic theory in question.

The order of words in the sentence is essential for the description of important restrictions of trees and DAGs for natural language. We therefore include the order in the sentence digraph structure, so $G_W = (V, E, \leq_W)$ is a *dependency digraph* for the sentence W, where V is the set of words tokens, E is the set of directed binary relations between words, and $\leq_W$ describes the order of the words in the sentence W ($\leq_W$ is the sentential order). For the remainder of this paper, when we talk about dependency digraphs (or *dependency DAGs* or *dependency trees*) the nodes of the underlying digraphs are associated with some fixed total order. Also we use the terms "word" and "node" synonymously in this context.

In linguistic terms, if (w_i, w_j) is an edge, then we say that w_i is a *head* of w_j and that w_j is a *dependent* of w_i. We can also write the edge (w_i, w_j) as $w_i \rightarrow w_j$. $w_i \xrightarrow{*} w_j$ is the *reflexive transitive closure* of the dependency relation.

A *dependency tree* then is just a connected dependency digraph in which every node has a unique head, except for a special node called the root, which has no head.

An interesting property yielding good coverage of some natural languages (for example, English) is that resulting dependency trees should be *projective*.

Definition 2. *A dependency tree $T = (V, E, \leq_W)$ is* projective *if for all edges $(w_i, w_j) \in E$, for all intervening words, w_k such that $k \in [\min\{i, j\}, \max\{i, j\}]$, we have $w_i \xrightarrow{*} w_k$.*

It turns out that the edges of a projective dependency tree can be written above the sentence (i.e., words written on a line segment in sentential order) without any crossing edges. This notion of avoiding crossing edges in "desirable" spanning DAGs has been considered in recent natural language parsing research, however the projectivity of edges as given in Definition 3 for dependency trees is no longer a sufficient condition to ensure this property in DAGs. As such, NLP researchers have adopted the unfortunate term "planar". Rather than assign a new meaning to the term planar, we generalise the definition of projectivity using the notion of crossing edges so that it applies to dependency digraphs, adopting this definition for the remainder of the paper. We then provide the correct restriction of planar digraphs that projective digraphs correspond to.

Definition 3. *For a dependency digraph $G_W = (V, E, \leq_W)$, an edge (w_i, w_j) or (w_j, w_i), with $i \leq j$ is* projective *if and only if for all words w_k such that $i < k < j$, there are no edges (w_l, w_k) or (w_k, w_l) such that $l < i$ or $l > j$.*
G_W is projective *if and only if all its edges are projective.*

Since the definition of projectivity excludes crossing edges when nodes are laid out on a line segment (sentential order), we see that the underlying digraph of a projective digraph is outerplanar. Moreover, it is easy to prove that for any planar digraph with all nodes on its outer face, we can choose a first node and define an ordering $\leq_W$ on the nodes by following the order of nodes (in a fixed direction) along an outer face, skipping repeats, until the original node

is met (recalling that one can find this outer face in linear time). So, there is a correspondence between the sets of projective digraphs and outerplanar digraphs.

Proposition 4. *For the projective digraph $G_W = (V, E, \leq_W)$, (V, E) is an outerplanar digraph. Also, every outerplanar digraph corresponds to some projective digraph with a sentential ordering defined by node traversal in a fixed direction along its outer face.*

On the other hand, given an outerplanar graph and some random sentential order, we of course do not necessarily have a projective digraph. In particular, an outerplanar drawing in the plane of a digraph does not necessarily have the specific desired order of vertices on its outer face. So, for example, finding the outerplanar MSDAG and the projective MSDAG are two *different* problems.

4 APX-hardness of MSDAG and its dual

In this section we give L-reductions from the APX-hard problems *maximum weighted directed acyclic subgraph* and its dual *minimum weighted feedback set* to MSDAG and its dual.

Minimum weighted feedback arc set. The dual problem of the MSDAG problem is almost identical with that of finding a minimum weighted feedback arc set. Given a directed graph, $G = (V, E)$, a *feedback arc set* (FAS) is a subset S of G's edges whose removal leaves a DAG (i.e., such that $(V, E(G) \backslash E(S))$ is a DAG). A *minimum feedback arc set* (MFAS) is the smallest among all possible feedback arc sets and a *minimum weighted feedback arc set* (MWFAS) is a feedback arc set of minimum weight. We call an MWFAS whose removal leaves a connected DAG a *nice MWFAS*. Finding a nice MWFAS is the dual of the MSDAG problem.

Already, the decision version of the FAS problem, which asks whether there is a feedback arc set of size k was listed as one of Karp's original 21 NP-complete problems (Karp, 1972), which shows the NP-hardness of the optimisation version. In fact, this optimisation problem is also shown to be APX-hard (Kann, 1992). The best approximation algorithm in the literature to date for this problem has an approximation guarantee of $O(\log n \log \log n)$ and the solution is NP-hard to approximate to within any factor smaller than $10\sqrt{5} - 21 \approx 1.36$ (Even et al., 1998).

An L-reduction from MWFAS to MSDAG for a weighted directed digraph G goes as follows. We first show that if the G has no edge cut consisting only of edges of negative weights, then the nice MWFAS is simply the MWFAS. Let D^* be an MSDAG for G. We show that the nice MWFAS $F := E(G) - E(D^*)$ must also form an MWFAS. Suppose otherwise, then there is some other F' of lower weight and such that $D' := (V(G), E(G) - F')$ is acyclic. So D' is disconnected and has at least two weakly connected components; we suppose without loss of generality that it has precisely two weakly connected components, C_1 and C_2. But then there must not be any positive weighted edge e in $E(G)$ between C_1 and C_2; otherwise we could remove that edge from F' to achieve an MWFAS of lower weight, since $C_1 \cup C_2 \cup \{e\}$ is still acyclic.

Now suppose that there is some edge cut consisting solely of edges of negative weight in G and therefore in D^*. Without loss of generality, we can suppose that there is only one such edge cut. Clearly, D^* contains at most one edge from this edge cut, the rest of them being in the nice MWFAS. An MWFAS would contain every negative weighted edge. So, the difference between a nice MWFAS and an MWFAS is just this one edge, for G. In particular, we have given an L-reduction from MWFAS to nice MWFAS, where the optimal solution for MWFAS is just the nice MWFAS with negative weighted edges removed (so both α and β from Definition 1 are equals to 1).

This shows the APX-hardness of the dual of the MSDAG problem, nice MWFAS.

Maximum weighted directed acyclic subgraph. An almost identical problem to MSDAG is the maximum weighted directed acyclic subgraph problem (MWDAS), which aims to find the (not necessarily spanning) DAG of highest weight. The decision version of the maximum directed acyclic subgraph problem (MDAS) problem which asks whether there is a directed acyclic subgraph on k edges can be solved by the decision version of the FAS problem, with $m - k$ as the parameter (where m is the number of edges in the input graph), and hence has

long been shown to be NP-complete (Karp, 1972). By the same token, the optimisation problem has been shown to be APX-hard (Kann, 1992). The best approximation algorithm in the literature to date for this problem has an approximation guarantee of $\frac{1}{2} + \Omega\left(\frac{1}{\sqrt{d_{\max}}}\right)$ where $d_{\max}$ is the maximum vertex degree for the graph (Berger and Shor, 1997). Moreover, the solution is Unique Games-hard to approximate to within any factor smaller than $1/2$, which is a tight bound (Guruswami et al., 2008).

Again, if there are no edge cuts in the graph consisting only of negative weighted edges, then the MSDAG and MWDAS solutions are identical. Otherwise, the MWDAS is just the MSDAG without its negative weighted edges, by the discussion above L-reducing MWFAS to nice MWFAS.

We have therefore shown the following fact.

Theorem 5. *MSDAG and its dual, nice MWFAS, are APX-hard.*

5 Planarity, outerplanarity and projectivity of DAGs in English data

APX-hardness of the MSDAG problem not only means that the problem is essentially infeasible, but also that it theoretically cannot be very well approximated. However, with some structural assumptions, such as the planarity, outerplanarity or projectivity of the objective DAGS, either approximation algorithms with a good approximation guarantee or even feasible algorithms might be achievable. The authors are not aware of any specific theoretical linguistic evidence for the planarity or outerplanarity of semantic dependency DAGs. However, we consider the three datasets from SemEval 2014 task 8 on Broad Coverage Semantic Dependency Parsing (Oepen et al., 2014), referred to as PAS, DM and PCEDT, following the packing conversion into DAGs described in (Schluter et al., 2014), without the actual label packing or edge removal heuristic, finding that almost all DAGs are outerplanar (and therefore also planar). Moreover, the datasets consist of a majority of projective DAGs, with those graphs that are not projective having proportionally on average a large projective subgraph (respectively 0.555, 0.580,and 0.605 for the PAS, DM, and PCEDT datasets). This large projective proportion suggests future avenues for algorithms for mildly projective DAGs, based on the projective algorithm presented in this paper.

	PAS	DM	PCEDT
% projective	58.461	56.840	84.192
% outerplanar	97.796	99.160	95.390
% planar	99.997	100	99.904

Table 1: Percentage of projective, planar, and outerplanar DAGs in the data.

6 The NP-hardness of finding a planar or outerplanar MSDAG

By a similar discussion to that in Section 4 on the relationship between MWDAS and MSDAG, finding the both the maximum weighted planar or outerplanar spanning DAG of a directed weighted graph can simply be shown to be NP-hard, where "planar" (and "outerplanar") is used in the graph-theoretical sense (Garey and Johnson, 1979). To our knowledge, approximability of finding a maximum weighted planar or outerplanar acyclic *directed* subgraph is still an open problem. The NP-hardness of finding a maximum weighted outerplanar spanning DAG is somewhat discouraging. But the next section provides a polynomial algorithm if there is a restriction on the the order of nodes on the outer face of the output (which is just projective MSDAG).

7 Finding a projective MSDAG or digraph in $O(n^3)$ time

We now turn our attention to the projective MSDAG problem, which can be solved efficiently. Our algorithm employs bottom-up dynamic programming across spans of words, where a span consists of a segment of words of the input sentence $W = \langle w_1, w_2, \ldots, w_n \rangle$ along with any attributed edges, similarly to the CKY-algorithm for context-free language parsing (Cocke, 1969; Kasami, 1965; Younger, 1967) and projective maximum spanning tree algorithms (Eisner, 1996), though the proof of correctness is slightly more complex. Following this, we explain how to simplify the algorithm to the task of finding the maximum weighted spanning digraph of digraph within the same time complexity.

Let $G_W = (V, E, \leq_W)$ be a weighted dependency digraph over the input sentence $W =$

$\langle w_1, w_2, \ldots, w_n \rangle$ and we suppose without loss of generality that $|E| = n(n-1)$. An (i,j)-span (with $i \leq j$) for S is the subsequence of consecutive words $w_i, w_{i+1}, \ldots, w_{j-1}, w_j$. We construct an algorithm **proj-MSDAG** (Algorithm 1) which takes G_W as input and outputs a highest weighted projective spanning dependency DAG for G_W: a projective MSDAG for G_W. For simplicity, instead of G_W, we just write G.

We call a directed path from node i to node j, an $i - j$ path. The algorithm constructs the upper triangular square matrix $A = \{a_{i,j}\}$ $(i, j \in [n])$, from left to right and from the diagonal upwards, where component $a_{i,j}$ contains at most three different restrictions of optimal projective spanning DAGs for the (i,j) span along with their associated weights:

1. $a_{i,j}.G_1$ is a projective MSDAG of the spanning subgraph $G[i,j]$, and $a_{i,j}.w_1$ is its weight
2. $a_{i,j}.G_2$ is a maximum projective spanning DAG for which there is no $i - j$ path, and $a_{i,j}.w_2$ is its weight, and
3. $a_{i,j}.G_3$ is a maximum projective spanning DAG for which there is no $j - i$ path, and $a_{i,j}.w_3$ is its weight.

The solution to the problem is then $a_{1,n}.G_1$. The motivation for distinguishing between these three types of graphs is to allow restricted combinations of them which ensure that no cycles are introduced.

We claim that these three restrictions on projective MSDAGs for the span (i,j) can be constructed from those of shorter spans within (i,j) using the three following operations, (A1), (A2), and (A3) (Lemma 6).

(A1) Concatenate sub-DAGs $H_{a_{i,k}}$ and $H_{a_{k,j}}$ from among the graphs in the cells $a_{i,k}$ and $a_{k,j}$ respectively $(i < k < j)$, creating the graph

$$(\{w_i, \ldots, w_j\}, E(H_{a_{i,k}}) \cup E(H_{a_{k,j}})),$$

(A2) Concatenate a single edge $e \in \{(i,j),(j,i)\}$ with the sub-DAGs $H_{a_{i,k}}$ and $H_{a_{k,j}}$ from among the graphs in the cells $a_{i,k}$ and $a_{k,j}$ respectively $(i < k < j)$, creating the graph

$$(\{w_i, \ldots, w_j\}, \{e\} \cup E(H_{a_{i,k}}) \cup E(H_{a_{k,j}})),$$

(A3) Connect two sub-DAGs $H_{a_{i,k}}$ and $H_{a_{k+1,j}}$ from among the graphs in the cells $a_{i,k}$ and $a_{k+1,j}$ respectively $(i \leq k < j)$ with a single edge $e \in \{(i,j),(j,i)\}$, creating the graph

$$(\{w_i, \ldots, w_j\}, \{e\} \cup E(H_{a_{i,k}}) \cup E(H_{a_{k+1,j}})).$$

Lemma 6. *The projective DAGs $a_{i,j}.G_1, a_{i,j}.G_2$, and $a_{i,j}.G_3$ can be de-constructed, by reversing a single operation (A1), (A2) or (A3), to obtain two sub-DAGs D_1 and D_2 which span their vertices and where either*

- *D_1 is on vertices $\{w_i, \ldots, w_k\}$ and D_2 is on vertices $\{w_k, \ldots, w_j\}$, for $i < k < j$, or*
- *D_1 is on vertices $\{w_i, \ldots, w_k\}$ and D_2 is on vertices $\{w_{k+1}, \ldots, w_j\}$, for $i \leq k < j$.*

Furthermore, for $p \in [2]$ and $1 \leq a < b \leq n$, let $G[a,b]$ be the spanning subgraph of G on the vertices $\{w_a, w_{a+1}, \ldots, w_{b-1}, w_b\}$, let D_p be a DAG on the vertices $w_a, w_{a+1}, \ldots, w_{b-1}, w_b$.

1. *D_p is a projective MSDAG for $G[a,b]$, or*
2. *D_p is the a highest projective spanning DAG with no $a - b$ path for $G[a,b]$, or*
3. *D_p is the a highest projective spanning DAG with no $b - a$ path for $G[a,b]$.*

Proof. We separate the proof into two parts: Part 1 for the graph $a_{i,j}.G1$ and Part 2 for the graphs $a_{i,j}.G2$, and $a_{i,j}.G3$.

Part 1. Let us denote $a_{i,j}.G1$ by D for ease in notation. We denote by $E_D(u)$ the set of edges in $E(D)$ having u for some endpoint. Consider the word w_i. By D's connectivity, $E_D(w_i)$ is non-empty. Let e be the edge in $E_D(w_i)$ of longest span, and suppose without loss of generality in edge direction that $e = (w_i, w_k)$ for some $k \in \{i+1, \ldots, j\}$.

There are two cases to consider. Either $k < j$ in which case we can reverse (A1) (Case 1), or $k = j$ in which case we can reverse (A2) or (A3) (Case 2).

Consider first Case 1, where $k < j$. By D's projectivity and the fact that e has the longest span in $E_D(w_1)$, there are no edges with one endpoint among $w_i, \ldots, w_{k-1}$ and the other endpoint among $w_{k+1}, \ldots, w_j$. So we can partition D into two

sub-DAGs $D[i,k]$, which is a spanning subgraph over the nodes $w_i, \ldots, w_k$ and $D[k,j]$, which is a spanning subgraph over the nodes $w_k, \ldots, w_j$. Both $D[i,k]$ and $D[k,j]$ are projective MSDAGs for $G[i,k]$ and $G[k,j]$ respectively, otherwise we can construct a projective MSDAG D' for G of higher weight than D, by taking the respective projective MSDAGs to form D'.

Otherwise, we have Case 2, with $k = j$. Note that there must not be any $w_j - w_i$ path in D (for acyclicity). We remove the edge (w_i, w_j) from D, the result of which is either connected or disconnected.

If $D - \{(w_i, w_j)\}$ is connected, it must be the maximum spanning DAG for G not containing any $w_j - w_i$ path. In the same manner as for the case where $k < j$, we can partition $D - \{(w_i, w_j)\}$ into two sub-DAGs $D[i,k]$, which is a spanning sub-DAG over the nodes $w_i, \ldots, w_k$ and $D[k,j]$, which is a spanning sub-DAG over the nodes $w_k, \ldots, w_j$, where either $D[i,k]$ does not have a $k - i$ path or $D[k,j]$ does not have a $j - k$ path. Clearly $D[i,k]$ and $D[k,j]$ are the maximum weighted projective spanning DAGs with this property for $G[i,k]$ and $G[k,j]$ respectively. This is the reversal of (A2).

Otherwise $D - \{(w_i, w_j)\}$ is disconnected into two weakly connected components $D[i,k]$ and $D[k+1,j]$, with $k \in [j-1]$, such that $D[i,k]$ is a projective MSDAG for $G[i,k]$ and $D[k+1,j]$ is a projective MSDAG for $G[k+1,j]$ (which is the reversal of (A3)).

Part 2. We prove, without loss of generality, the result for the graph $a_{i,j}.G_3$, the proof for the graph $a_{i,j}.G_2$ being symmetric. The proof follows almost exactly the one in Part 1, so we simply indicate the differences here. Again, for ease in notation, we denote $a_{i,j}.G_2$ by D, and carry out the same partition of D as in Part 1.

The difference for Case 1 is that either the resulting $D[i,k]$ must be a maximum weighted projective spanning DAG for $G[i,k]$ with no $k - i$ path, or $D[k,j]$ must be a maximum weighted projective spanning DAG for $G[k,j]$ with no $j - k$ path.

Case 2 is precisely the same if the result of removing the edge (w_i, w_j) is disconnected. If the result is connected, then the difference is that $D[i,k]$ must be a maximum projective weighted spanning DAG for $G[i,k]$ with no $k - i$ path, or $D[k,j]$ must be a maximum weighted projective spanning DAG for $G[k,j]$ with no $j - k$ path. □

Algorithm 1 uses the operations (A1), (A2) and (A3) to fill the matrix A. In Figure 1, we define three different subroutines corresponding to each of these operations, which take the matrix A, the span, and the forbidden direction for the edge of the span, if there is one (and the empty set otherwise). Clearly each of these runs in $O(n)$ time.

We observe that Lines 9 through 14 in Algorithm 1 dominate the time complexity, taking $O(n^3)$. Lemma 7 shows that Algorithm 1 fills the table cells with the correct projective MSDAG restrictions.

Lemma 7. *For $i \leq j$, Algorithm 1 fills matrix entry $a_{i,j}$ with:*

1. *a projective MSDAG for $G[i,j]$,*
2. *a projective MSDAG for $G[i,j]$ with no $i - j$ path, and*
3. *a projective MSDAG for $G[i,j]$ with no $j - i$ path.*

Proof. The proof is by induction on the span size. For the table cells $a_{i,i}$ for $i \in [n]$, we have $a_{i,i}.G_1 = a_{i,i}.G_2 = a_{i,i}.G_3 = (\{w_i\}, \emptyset)$; the graphs are all single nodes. For our base case, $a_{i,i+1}, i \in [n-1]$, we construct the cell contents as follows:

1. $a_{i,i+1}.G_1$ is the highest weighted edge among (w_i, w_{i+1}) and (w_{i+1}, w_i).
2. $a_{i,i+1}.G_2$ is (w_i, w_{i+1}) if this edge is in G.
3. $a_{i,i+1}.G_3$ is (w_{i+1}, w_i) if this edge is in G.

This is Lines 1 through 8 in Algorithm 1.

Suppose now that the entries $a_{i,k}$ and $a_{k,j}$ contain the appropriate graphs G_1, G_2 and G_3, for $i < k < j$. For each k ($i < k < j$), Lines 9 through 14 in Algorithm 1 construct the three graphs, $a_{i,j}.G_1(k), a_{i,j}.G_2(k)$, and $a_{i,j}.G_3(k)$, as follows, which is possible according to Lemma 6:

1. $a_{i,j}.G_1(k)$ is a highest weighted projective dependency DAG for $G[i,j]$ that can be constructed from the graphs stored in $a_{i,k}$ and $a_{k,j}$.
2. $a_{i,j}.G_2(k)$ is a highest weighted projective dependency DAG for $G[i,j]$ that can be constructed from the graphs stored in $a_{i,k}$, $a_{i,k+1}$ and $a_{k,j}$, which avoids an $i - j$ path.

$$\texttt{A1}(G,A,i,j,\emptyset) = \underset{\{H|H=(\{w_i,\ldots,w_j\},E(a_{i,k}.G_1)\cup E(a_{k,j}.G_1)),\ i<k<j\}}{\arg\max} w(H)$$

$$\texttt{A1}(G,A,i,j,\rightarrow) = \underset{\substack{\{H|H=(\{w_i,\ldots,w_j\},E(a_{i,k}.G_2)\cup E(a_{k,j}.G_1)),\ i<k<j\} \\ \cup\{H|H=(\{w_i,\ldots,w_j\},E(a_{i,k}.G_1)\cup E(a_{k,j}.G_2)),\ i<k<j\}}}{\arg\max} w(H)$$

$$\texttt{A1}(G,A,i,j,\leftarrow) = \underset{\substack{\{H|H=(\{w_i,\ldots,w_j\},E(a_{i,k}.G_3)\cup E(a_{k,j}.G_1)),\ i<k<j\} \\ \cup\{H|H=(\{w_i,\ldots,w_j\},E(a_{i,k}.G_1)\cup E(a_{k,j}.G_3)),\ i<k<j\}}}{\arg\max} w(H)$$

$$\texttt{A2}(G,A,i,j,\emptyset) = \underset{\{H|H=(\{w_i,\ldots,w_j\},\ \{(w_i,w_j)\}\cup E(a_{i,k}.G_1)\cup E(a_{k,j}.G_1)),\ i<k<j\}}{\arg\max} w(H)$$

$$\texttt{A2}(G,A,i,j,\rightarrow) = \underset{\substack{\{H|H=(\{w_i,\ldots,w_j\},\ \{(w_j,w_i)\}\cup E(a_{i,k}.G_2)\cup E(a_{k,j}.G_1)),\ i<k<j\} \\ \cup\{H|H=(\{w_i,\ldots,w_j\},\ \{(w_j,w_i)\}\cup E(a_{i,k}.G_1)\cup E(a_{k,j}.G_2)),\ i<k<j\}}}{\arg\max} w(H)$$

$$\texttt{A2}(G,A,i,j,\leftarrow) = \underset{\substack{\{H|H=(\{w_i,\ldots,w_j\},\ \{(w_i,w_j)\}\cup E(a_{i,k}.G_3)\cup E(a_{k,j}.G_1)),\ i<k<j\} \\ \cup\{H|H=(\{w_i,\ldots,w_j\},\ \{(w_i,w_j)\}\cup E(a_{i,k}.G_1)\cup E(a_{k,j}.G_3)),\ i<k<j\}}}{\arg\max} w(H)$$

$$\texttt{A3}(G,A,i,j,\emptyset) = \underset{\{H|H=(\{w_i,\ldots,w_j\},\ \{(w_i,w_j)\}\cup E(a_{i,k}.G_1)\cup E(a_{k+1,j}.G_1)),\ i\leq k<j\}}{\arg\max} w(H)$$

$$\texttt{A3}(G,A,i,j,\rightarrow) = \underset{\substack{\{H|H=(\{w_i,\ldots,w_j\},\ \{(w_j,w_i)\}\cup E(a_{i,k}.G_2)\cup E(a_{k+1,j}.G_1)),\ i\leq k<j\} \\ \cup\{H|H=(\{w_i,\ldots,w_j\},\ \{(w_j,w_i)\}\cup E(a_{i,k}.G_1)\cup E(a_{k+1,j}.G_2)),\ i\leq k<j\}}}{\arg\max} w(H)$$

$$\texttt{A3}(G,A,i,j,\leftarrow) = \underset{\substack{\{H|H=(\{w_i,\ldots,w_j\},\ \{(w_i,w_j)\}\cup E(a_{i,k}.G_3)\cup E(a_{k+1,j}.G_1)),\ i\leq k<j\} \\ \cup\{H|H=(\{w_i,\ldots,w_j\},\ \{(w_i,w_j)\}\cup E(a_{i,k}.G_1)\cup E(a_{k+1,j}.G_3)),\ i\leq k<j\}}}{\arg\max} w(H)$$

Figure 1: Subroutines corresponding to the operations (A1), (A2), and (A3) as used by Algorithm 1.

Algorithm 1 `proj-MSDAG`(G)

for $i \leftarrow 1,\ldots,n$ **do**
  $a_{i,i}.G_1 \leftarrow a_{i,i}.G_2 \leftarrow a_{i,i}.G_3 = (\{i\},\emptyset)$
  if $i < n$ **then**
    $a_{i,i+1}.G_1 \leftarrow (\{w_i,w_{i+1}\},\{\arg\max_{e\in\{(w_i,w_{i+1}),(w_{i+1},w_i)\}} w(e)\})$
    $a_{i,i+1}.G_2 \leftarrow (\{w_i,w_{i+1}\},\{(w_{i+1},w_i)\})$
    $a_{i,i+1}.G_3 \leftarrow (\{w_i,w_{i+1}\},\{(w_i,w_{i+1})\})$
  end if
end for
for $i \leftarrow 1,\ldots,n$ **do**
  for $j \leftarrow i-1,\ldots,1$ **do**
    $a_{i,j}.G_1 \leftarrow \arg\max_{H\in\{\texttt{A1}(G,A,i,j,\emptyset),\texttt{A2}(G,A,i,j,\emptyset),\texttt{A3}(G,A,i,j,\emptyset)\}} w(H)$
    $a_{i,j}.G_2 \leftarrow \arg\max_{H\in\{\texttt{A1}(G,A,i,j,\rightarrow),\texttt{A2}(G,A,i,j,\rightarrow),\texttt{A3}(G,A,i,j,\rightarrow)\}} w(H)$
    $a_{i,j}.G_3 \leftarrow \arg\max_{H\in\{\texttt{A1}(G,A,i,j,\leftarrow),\texttt{A2}(G,A,i,j,\leftarrow),\texttt{A3}(G,A,i,j,\leftarrow)\}} w(H)$
  end for
end for
return $a_{1,n}.G_1$

3. $a_{i,j}.G_3(k)$ is a highest weighted projective dependency DAG for $G[i,j]$ that can be constructed from the graphs stored in $a_{i,k}$, $a_{i,k+1}$ and $a_{k,j}$, which avoids an $j-i$ path.

□

With Lemmata 6 and 7, the proof of correctness of the following theorem is complete.

Theorem 8. *There is an algorithm for finding a projective MSDAG of a weighted digraph in $O(n^3)$ time, where n is the number of word tokens of the input sentence.*

The proof of Theorem 8 can be simplified to adapt it to that of projective digraphs. We simply substitute the word "DAG" by "digraph", and disregard all graphs in table entries avoiding certain paths, since cycles are permitted: in the entry $a_{i,j}$, we need only construct and record the graph $a_{i,j}.G_1$. We have thus also shown the following fact.

Theorem 9. *There is an algorithm for finding a projective maximum weighted spanning digraph of a weighted digraph in $O(n^3)$ time, where n is the number of word tokens of the input sentence.*

8 Concluding Remarks

Understanding the complexity of the problem of finding a maximum spanning DAG as well as important restrictions provides a basis for both theoretical and empirical studies using restrictions or relaxations of the DAG parsing paradigm. We have provided the first direct discussion of this problem's complexity, showing that the problem is APX-hard as well as the first algorithm for the projective MSDAG problem proven to be exact and polynomial time. Additionally, we briefly discussed the complexity of finding a planar and outerplanar MSDAG, the approximability of which remains open.

References

Bonnie Berger and Peter W. Shor. 1997. Tight bounds for the maximum acyclic subgraph problem. *Journal of Algorithms*, 25:1–18.

Manuel Bodirsky, Marco Kuhlmann, and Mathias Möhl. 2005. Well-nested drawings as models of syntactic structure. In *Proc. of the Tenth Conference on FOrmal Grammar and Ninth Meeting on Mathematics of Language*.

Xavier Carreras. 2007. Experiments with a higher-order projective dependency parser. In *Proc. of CoNLL Shared Task Session of EMNLP-CoNLL*, pages 957–961, Prague, Czech Republic.

Y.J. Chu and T. H. Liu. 1965. On the shortest arborescence of a directed graph. *Science Sinica*, 14:1396–1400.

John Cocke. 1969. Programming languages and their compilers: Preliminary notes. Technical Report B0007F4UOA, Courant Institute of Mathematical Sciences, New York University.

Jack Edmonds. 1967. Optimum branchings. *J. Res. Nat. Bur. Standards*, 71B:233–240.

Jason Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of COLING-96)*, pages 340–345, Copenhagen, Denmark.

Guy Even, Joseph Naor, Baruch Schieber, and Madhu Sudan. 1998. Approximating minimum feedback sets and multicuts in directed graphs. *Algorithmica*, 20:151–174.

Michael R. Garey and David S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-completeness*. Freeman & Co., San Francisco.

Carlos Gómez-Rodríguez, John Carroll, and David Weir. 2011. Dependency parsing schemata and mildly non-projective dependency parsing. *Computational Linguistics*, 37:541–586.

Venkatesan Guruswami, Rajsekar Manokaran, and Prasad Raghavendra. 2008. Beating the random ordering is hard: Inapproximability of maximum acyclic subgraph. In *Proc. of FOCS*.

D. Heckerman, D. Geiger, and D. M. Chickering. 1995. Learning bayesian networks: The combination of knowledge and statistical data. Technical report, Microsoft Research. MSR-TR-94-09.

Viggo Kann. 1992. *On the approximability on NP-complete Optimization Problems*. Ph.D. thesis, Department of Numerical Analysis and Computing Science, Royal Institute of Technology, Stockholm, Sweden.

Richard M. Karp. 1972. Reducibility among combinatorial problems. In *Complexity of Computer Computations, Proc. Sympos. IBM Thomas J. Watson Res. Center*, pages 85–103, New York, NY.

Tadao Kasami. 1965. An efficient recognition and syntax-analysis algorithm for context-free languages. Technical Report AFCRL-65-758, Air Force Cambridge Research Lab, Bedford, MA.

André F. T. Martins and Mariana S. C. Almeida. 2014. Priberam: A turbo semantic parser with second order features. In *Proc of SemEval*, Dublin, Ireland.

Ryan McDonald and Fernando Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *Proc. of EACL*, pages 81–88.

Ryan McDonald and Giorgio Satta. 2007. On the complexity of non-projective data-driven dependency parsing. In *Proceedings of the Tenth International Conference on Parsing Technologies*, pages 121–132, Prague, Czech Republic, June.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Haji. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proc. of HLT-EMNLP*, pages 523–530, Vancouver, BC, Canada.

Stephan Oepen, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Dan Flickinger, Jan Hajič, Angelina Ivanova, and Yi Zhang. 2014. Semeval 2014 task 8: Broad-coverage semantic dependency parsing. In *Proc. of SemEval*, pages 63–71, Dublin, Ireland.

Emily Pitler, Sampath Kannan, and Mitchell Marcus. 2012. Dynamic programming for higher order parsing of gap-minding trees. In *Proceedings of EMNLP/CoNLL*, pages 478–488, Jeju Island, Republic of Korea.

Emily Pitler, Sampath Kannan, and Mitchell Marcus. 2013. Finding optimal 1-endpoint-crossing trees. *Transactions of the Association for Computational Linguistics*.

Kenji Sagae and Jun'ichi Tsujii. 2008. Shift-reduce dependency dag parsing. In *22nd International Conference on Computational Linguistics (Coling 2008)*, Manchester, UK.

Giorgio Satta and Marco Kuhlmann. 2013. Efficient parsing for head-split dependency trees. *Trans. ACL*, pages 267–278.

Natalie Schluter, Anders Søgaard, Jakob Elming, Dirk Hovy, Barbara Plank, Hector Martinez Alonso, Anders Johanssen, and Sigrid Klerke. 2014. Copenhagen: Tree approximations of semantic parsing problems. In *Proceedings of SemEval*, Dublin, Ireland.

Natalie Schluter. 2014. On maximum spanning dag algorithms for semantic dag parsing. In *ACL 2014 Workshop on Semantic Parsing*.

Robert Tarjan. 1977. Finding optimum branchings. *Networks*, pages 25–35.

Ivan Titov, James Henderson, Paola Merlo, and Gabrielle Musillo. 2009. Online graph planarization for synchronous parsing of semantic and synactic dependencies. In *Proceedings of IJCAI 2009*, pages 1562–1567.

Ingo Wegener. 2005. *Complexity Theory*. Springer.

Daniel H. Younger. 1967. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10(2):189– 208.