Statistical A* Dependency Parsing

Péter Dienes
Computational Linguistics
Saarland University

Alexander Koller
Computational Linguistics
Saarland University

Marco Kuhlmann
Programming Systems Lab
Saarland University

1 Introduction

Extensible Dependency Grammar (XDG, Duchier and Debusmann (2001)) is a recently developed dependency grammar formalism that allows the characterization of linguistic structures along multiple dimensions of description. It can be implemented efficiently using constraint programming (CP, Koller and Niehren (2002)). In the CP context, parsing is cast as a search problem: The states of the search are partial parse trees, successful end states are complete and valid parses.

In this paper, we propose a probability model for XDG dependency trees and an A* search control regime for the XDG parsing algorithm that guarantees the best parse to be found first. Extending XDG with a statistical component has the benefit of bringing the formalism further into the grammatical mainstream; it also enables XDG to efficiently deal with large, corpus-induced grammars that come with a high degree of ambiguity.

On the processing side, to the best of our knowledge, the use of an A* heuristic in the context of a CP search is novel. In particular, the combination of CP with statistical guidance in the application area of computational linguistics is novel, except for an unpublished experiment by Brants and Duchier (p.c.). Other applications in the area of the syntax-semantics interface that could be solved with the same methods are the resolution of scope ambiguities with preferences, and statistical generation (Koller and Striegnitz, 2002).

Our probability model is a lexicalized one, building on bilexical probabilities of head–dependent pairs (for a general overview of such models, cf. Collins (1999)). The most closely related parametrization is Resnik's model for stochastic tree-adjoining grammars (Resnik, 1992). On the processing side, A* search has first been proposed for parsing by Klein and Manning (2003), who use it in conjunction with a chart parsing algorithm for a PCFG. We claim that employing A* search fits into the CP framework even more naturally.

The paper reports work in progress. While it defines an initial probability model and a parsing algorithm, implementation and evaluation remain to be done, and we only present some first ideas. We hope to finish these parts in time for the workshop itself.

2 Extensible Dependency Grammar

XDG is a dependency grammar formalism relating sentences to syntactic structures that have one node per word in the sentence. Each node is labeled with a lexicon entry for this word. Nodes are connected with labeled edges. The lexicon entries specify what edges can emanate from a node with this entry (the lexical entry's *valency*), and what edges can go into it.

The general XDG formalism allows the grammar writer to distinguish any number of different dimensions of description, such as immediate dominance (ID) and linear precedence (LP) structures, which can be related by declarative constraints in the grammar. We will concentrate on ID structures in the paper, but all our results continue to work for grammars with additional dimensions.

At the ID level, we take a *lexical entry* l of XDG to consist of a word, a set $\mathsf{inval}(l)$ of labels that an edge coming into a node labeled by l can have, and the *valency list* $\mathsf{val}(l)$, which specifies what edges can go out of the node. $\mathsf{val}(l)$ is an ordered list with elements of the form a (require exactly one outgoing edge with label a), $a?$ (allow an optional

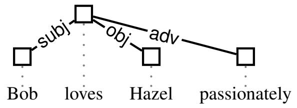

Figure 1: Sample XDG syntax tree.

edge with label a), or $a*$ (allow zero or arbitrarily many edges with label a).

A *dependency tree* τ of XDG is a tree, together with a *node labeling function* τ_n that assigns lexical entries to nodes and an *edge labeling function* τ_e that assigns edge labels to edges. A dependency tree is well-formed according to τ_n and τ_e iff (i) for each node v, the edge e that goes into v has a label acceptable by the lexical entry ($\tau_e(e) \in \mathsf{inval}(\tau_n(v))$); and (ii) for each node the set of outgoing edge labels is compatible with the valency list of the lexical entry associated with the node.

An XDG grammar G is a relation that connects words to lexical entries for these words. The set of lexical entries according to the grammar G is designated by L_G. The (ID) parsing problem for a grammar G can be stated as follows. Given a sentence $w_1 \cdots w_n$, find a tree and labeling functions τ_n, τ_e such that (i) the dependency tree thus specified is well-formed, and (ii) there is a bijection map from words to nodes in the tree such that $\tau_n(map(w_i))$ is a lexical entry for w_i, for all i.

As an example, consider the dependency tree in Fig. 1. It would be grammatical according to a grammar in which the only lexical entry for *love* allows no incoming edges (i.e., the finite verb must be the root of the tree), and has valency $[subj, obj, adv*]$ – i.e., it must take exactly one subject and exactly one object, and arbitrarily many adverbial modifiers. Lexical entries for *Bob* and *Hazel* could both accept incoming $subj$ and obj edges, and would at least not require any outgoing edges.

3 Probability Model

Now we add a generative probability model to the ID dimension of XDG. Given a well-formed (dependency) tree τ, the probability $P(\tau)$ of the tree is

$$P(\tau) = P_R(\tau_n(root)) \cdot \prod_{(v,v') \in \text{edges}(\tau)} P_L(l|\tau_n(v)) \cdot P_D(\tau_n(v')|\tau_n(v), l)$$

where $l = \tau_e(v, v')$ and

(a) the root probability $P_R(l)$ is the probability of a lexical entry being the root of the dependency tree, such that $\sum_{l \in L_G} P_R(l) = 1$;

(b) the labelling probability $P_L(lab|l)$ is the probability of lab being among the outgoing edge labels of the lexical entry l; and

(c) the bilexical dependency probabilities $P_D(l'|l, lab)$ designate the probability of the lexical entry l' being a child of the lexical entry l over an edge with label lab, such that for each l and for each label lab in the valency list of l, $\sum_{l' \in L_G} P_D(l'|l, lab) = 1$.

Intuitively, in the generative process, the first step is the generation of the root lexical entry l_{root} for a sentence with probability $P_R(l_{root})$. Then, for each node v in the tree, we go through each valency slot lab of its associated lexical entry $l = \tau_n(v)$ and generate the lexical entry l' filling this slot with probability $P_D(l'|l, lab)$. Note that we assume conditional independence between sisters, as well as between words not in immediate dependency relationship – as in the the standard head-lexicalized PCFG-model (e.g. (Magerman, 1995)). Our model resembles most closely Resnik's (1992) probability model for TAG, but we do not generate empty adjunction slots.

In order to ensure the consistency of the model (i.e., the probability of all trees for all grammatical sentences should sum to 1), reentrancies are not allowed. Note further that the probability model assigns probabilities only to well-formed structures according to the grammar. There is a strict division of labor between the grammar, which imposes hard constraints on possible dependency trees, and the probability model, which determines lexical preferences (cf. Eisner (1996)).

$$ecost(\sigma) = \sum_{v_2 \in \mathrm{dom}(\tau_n)} \min \quad \{-\log P_R(l) \mid l \in \tau^*_\sigma(v_2)\} \\ \cup \{C(l_1, lab, l_2) \mid l_2 \in \tau^*_\sigma(v_2), \exists\, v_1\colon (v_1, l_1, l_2, lab) \in \mathsf{in}_\sigma(v_2)\}$$

where the cost $C(l_1, lab, l_2) = -\log(P_D(l_2|l_1, lab) \cdot P_L(lab|l_1))$

Figure 2: Estimated cost

4 Parsing Algorithm

The next step is to efficiently compute the most probable dependency tree for a given sentence. Based on the implementation of XDG parsing as search for a solution of a constraint problem, we show how this can be done by imposing an A* control regime on this search.

Constraint-based dependency parsing may be visualized as exploring a search tree whose inner nodes are partially determined parse trees. In each such parse tree, some (initially all) of the nodes may not yet have a decided lexical entry, or may not yet be connected to a parent node (or both). In the course of the search, the set of possible lexical entries and parents for each node is eventually cut down by *constraint propagation* (inference-driven exclusion of choices according to the principles of the grammar) and *distribution* (non-deterministic choice). Leaves of the search tree are either failed (i.e., inconsistent with the grammar), or they are complete parses of the input, in which each node has exactly one lexical entry and each node except for the root has exactly one parent.

While the *shape* of the search tree is determined by propagation and distribution, the *order* in which the nodes of the search tree are expanded is an independent issue (Schulte, 1997). We choose to traverse the search tree under an A* regime, which guarantees that the first solution we find is optimal. We *evaluate* each partially determined parse tree by means of a cost function which estimates the negative logarithm of the most probable parse tree τ to which the partial tree can be extended. The search minimizes $-\log P(\tau)$, so the optimal solution has maximal probability.

We define $\tau^*_\sigma(v)$ to be the set of possible lexical entries for v that propagation and distribution haven't yet ruled out in the search state σ. $\mathsf{in}_\sigma(v_2)$ is the set of *potential incoming edges* of the node v_2. It contains all quadruples (v_1, l_1, l_2, lab) in which v_1 is a potential parent of v_2, l_1 and l_2 are potential lexical entries for v_1 and v_2, and lab is an edge label that can connect l_1 and l_2.

The estimated cost $ecost(\sigma)$ of a search state σ is defined in Fig. 2. The first line estimates the cost for the case when v_2 becomes the root of the dependency tree, and the second line estimates the cost of the best possible edge into v_2. $ecost(\sigma)$ underestimates $-\log(P(\tau))$, so it is an admissible A* heuristic.

Search regime Search begins with propagation in the initial search state and then proceeds in phases. Each phase begins with a distribution step that will create a set of new search states, called choices. After full propagation, *ecost* is computed for each of the choices, taking the remaining possible lexical entries and edges into account. Search then continues at that node in the search tree that carries the least costs.

Implementation In the rest of this section, we show that employing A* search does not increase asymptotic complexity (in the input size) of the parsing algorithm, although it requires some additional book-keeping which increases the complexity by a constant factor. Since in the worst case we still have to explore the whole search space (of exponential size), it is still an open question how much we actually benefit from using A* search, subject to empirical evaluation.

We annotate each node v in the search tree with a priority queue (PQ) that supports the efficient computation of $ecost(v)$. The elements of the

PQ for v in each search state σ are the entries of $\tau^*_\sigma(v)$ along with the negative logarithms of their P_R probabilities, and the entries of $\mathsf{in}_\sigma(v)$, along with the negative logarithms of their P_D probabilities. The queue is initialized before the search starts with all possible lexical entries and incoming edges. Then, during the search, every time a propagator removes a possible lexical entry or a possible edge, these changes are reflected in the queue by deleting the corresponding entries.

Because elements are only inserted into our queues in the initialization phase, we can implement a PQ as a doubly linked list that is sorted in increasing order of the negative logarithms of the probabilities. We can initialize all PQs for all nodes in time $O(n^2 \cdot k^2 \cdot (\log k + \log n))$, where n is the sentence length, and k is the maximal number of lexical entries per word in the grammar.

Computation of $ecost(\sigma)$ is $O(n)$, as the minimization in Fig. 2 only requires us to look at the head of the list for each node. This is dominated by the $\Omega(n^2)$ time that the standard XDG parser spends on propagation in each step, so it doesn't contribute additional asymptotic costs. Deletion of an item from a PQ is also $O(1)$. Because the queue entries must spell out possible combinations of lexical entries for nodes, a propagator deleting an edge between two nodes might have to delete multiple entries from a queue. This can increase the total runtime of propagation over the course of the whole search by a factor of $O(k^2)$.

5 Training and Evaluation

We plan to train and evaluate our system both on the Penn Treebank (using the dependency converter of (Buchholz, 2002)) and on the NEGRA Treebank. As it is very difficult to generalize lexical entries obtained from a treebank for optional complements and adjuncts, we expect that our first grammar will have no optional complements, and adjuncts will be subsumed under the general edge label *adjunct*. In addition, we will have to assume a simplified grammar for the LP dimension, e.g. with completely free or completely fixed word order.

Given the heavily lexicalized nature of the probability model, we are bound to run into a serious sparse data problem. We intend to tackle this problem by substituting infrequent words by their part-of-speech tags before training and parsing. Dependency probabilities are to be obtained by a maximal likelihood estimation using the modified training trees. Root probabilities could be obtained by considering head lexical entries for both matrix and embedded clauses.

We plan to evaluate the system according to its accuracy in determining labeled and unlabeled head-dependent relations (Carroll et al., 1998). Furthermore, in the case of the English experiment, we can convert the dependency trees to unlabeled phrase structure trees, which will enable us to use standard (unlabeled) PARSEVAL bracketing metrics to compare our results with the mainstream parsing approaches.

6 Conclusion

We have proposed a probability model for XDG and an extension of the CP-based XDG parsing algorithm by an A* search control regime. Our approach is guaranteed to find the optimal parse first, and potentially explores only a narrow part of the search space. It is interesting from the parsing point of view because it equips XDG with a statistical model and parser, and from a computational point of view because it explores the use of A* search in the context of constraint programming, an extension that is very natural, albeit previously unexplored.

There is a number of issues that we did not address in this paper, and a number of open questions. A particularly interesting one is that while our approach can deal with arbitrary multi-dimensional XDG grammars, the grammars we know how to derive from a corpus only have a meaningful ID dimension, and we can only assign probabilities to the ID dimension. If we assume that the probabilities of different dimensions are independent, we could simply derive a model for each dimension and then multiply the probabilities; the algorithm would generalize trivially. How close such an independence assumption is to the truth remains to be seen.

We see the work in the paper as a first step towards a unified framework for syntactic and semantic processing, which combines constraint propagation with statistical guidance. XDG is ideally suited to an integration with scope underspecification in the spirit of (Egg et al., 2001); we envision that the methods introduced here for statistical parsing could be used there to process scope preferences. On the other hand, Koller and Striegnitz (2002) have shown how to treat TAG generation from flat semantics to surface text as an XDG parsing problem. In this context, the present work can be applied to obtain a statistical generation system.

References

Sabine Buchholz. 2002. *Memory-Based Grammatical Relation Finding*. Ph.D. thesis, Tilburg University.

John Carroll, Ted Briscoe, and Antonio Sanfilippo. 1998. Parser evaluation: a survey and a new proposal. In *Proceedings of the 1st LREC*, pages 447–454, Granada, Spain.

Michael Collins. 1999. *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania.

Denys Duchier and Ralph Debusmann. 2001. Topological dependency trees: A constraint-based account of linear precedence. In *Proceedings of the 39th ACL*, Toulouse, France.

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

Jason Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th COLING*, pages 340–345, Copenhagen.

Dan Klein and Christopher D. Manning. 2003. A* parsing: Fast exact Viterbi parse selection. In *Proceedings of HLT-NAACL 03*.

Alexander Koller and Joachim Niehren. 2002. Constraint programming in computational linguistics. In D. Barker-Plummer, D. Beaver, J. van Benthem, and P. Scotto di Luzio, editors, *Words, Proofs, and Diagrams*, pages 95–122. CSLI Press.

Alexander Koller and Kristina Striegnitz. 2002. Generation as dependency parsing. In *Proceedings of the 40th ACL*, Philadelphia.

David Magerman. 1995. Statistical decision-tree models for parsing. In *Proceedings of the 33rd ACL*, pages 276–283, Cambridge, MA.

Philip Resnik. 1992. Probabilistic tree-adjoining grammars as a framework for statistical natural language processing. In *Proceedings of the 15th COLING*, pages 418–424, Nantes.

Christian Schulte. 1997. Programming constraint inference engines. In Gert Smolka, editor, *Proceedings of the Third CP Conference*, volume 1330 of *LNCS*, pages 519–533, Schloss Hagenberg, Austria.