Incremental Integer Linear Programming for Non-projective Dependency Parsing

Sebastian Riedel and **James Clarke**
School of Informatics, University of Edinburgh
2 Bucclecuch Place, Edinburgh EH8 9LW, UK
s.r.riedel@sms.ed.ac.uk, jclarke@ed.ac.uk

Abstract

Integer Linear Programming has recently been used for decoding in a number of probabilistic models in order to enforce global constraints. However, in certain applications, such as non-projective dependency parsing and machine translation, the complete formulation of the decoding problem as an integer linear program renders solving intractable. We present an approach which solves the problem incrementally, thus we avoid creating intractable integer linear programs. This approach is applied to Dutch dependency parsing and we show how the addition of linguistically motivated constraints can yield a significant improvement over state-of-the-art.

1 Introduction

Many inference algorithms require models to make strong assumptions of conditional independence between variables. For example, the Viterbi algorithm used for decoding in conditional random fields requires the model to be Markovian. Strong assumptions are also made in the case of McDonald et al.'s (2005b) non-projective dependency parsing model. Here attachment decisions are made independently of one another[1]. However, often such assumptions can not be justified. For example in dependency parsing, if a subject has already been identified for a given verb, then the probability of attaching a second subject to the verb is zero. Similarly, if we find that one coordination argument is a noun, then the other argument cannot be a verb. Thus decisions are often co-dependent.

Integer Linear Programming (ILP) has recently been applied to inference in sequential conditional random fields (Roth and Yih, 2004), this has allowed the use of truly global constraints during inference. However, it is not possible to use this approach directly for a complex task like non-projective dependency parsing due to the exponential number of constraints required to prevent cycles occurring in the dependency graph. To model all these constraints explicitly would result in an ILP formulation too large to solve efficiently (Williams, 2002). A similar problem also occurs in an ILP formulation for machine translation which treats decoding as the Travelling Salesman Problem (Germann et al., 2001).

In this paper we present a method which extends the applicability of ILP to a more complex set of problems. Instead of adding all the constraints we wish to capture to the formulation, we first solve the program with a fraction of the constraints. The solution is then examined and, if required, additional constraints are added. This procedure is repeated until all constraints are satisfied. We apply this dependency parsing approach to Dutch due to the language's non-projective nature, and take the parser of McDonald et al. (2005b) as a starting point for our model.

In the following section we introduce dependency parsing and review previous work. In Section 3 we present our model and formulate it as an ILP problem with a set of linguistically motivated constraints. We include details of an incremental algorithm used to solve this formulation. Our experimental set-up is provided in Section 4 and is followed by results in Section 5 along with runtime experiments. We finally discuss fu-

[1] If we ignore the constraint that dependency trees must be cycle-free (see sections 2 and 3 for details).

Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing (EMNLP 2006), pages 129–137,
Sydney, July 2006.

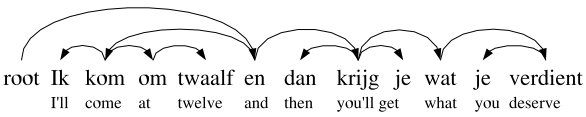

Figure 1: A Dutch dependency tree for "I'll come at twelve and then you'll get what you deserve"

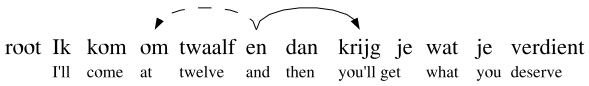

Figure 2: An incorrect partial dependency tree. The verb "krijg" is incorrectly coordinated with the preposition "om".

ture research and potential improvements to our approach.

2 Dependency Parsing

Dependency parsing is the task of attaching words to their arguments. Figure 1 shows a dependency graph for the Dutch sentence "I'll come at twelve and then you'll get what you deserve" (taken from the Alpino Corpus (van der Beek et al., 2002)). In this dependency graph the verb "kom" is attached to its subject "ik". "kom" is referred to as the head of the dependency and "ik" as its child. In labelled dependency parsing edges between words are labelled with the relation captured. In the case of the dependency between "kom" and "ik" the label would be "subject".

In a dependency tree every token must be the child of exactly one other node, either another token or the dummy root node. By definition, a dependency tree is free of cycles. For example, it must not contain dependency chains such as "en" → "kom" → "ik" → "en". For a more formal definition see previous work (Nivre et al., 2004).

An important distinction between dependency trees is whether they are projective or non-projective. Figure 1 is an example of a projective dependency tree, in such trees dependencies do not cross. In Dutch and other flexible word order languages such as German and Czech we also encounter non-projective trees, in these cases the trees contain crossing dependencies.

Dependency parsing is useful for applications such as relation extraction (Culotta and Sorensen, 2004) and machine translation (Ding and Palmer, 2005). Although less informative than lexicalised phrase structures, dependency structures still capture most of the predicate-argument information needed for applications. It has the advantage of being more efficient to learn and parse.

McDonald et al. (2005a) introduce a dependency parsing framework which treats the task as searching for the projective tree that maximises the sum of local dependency scores. This framework is efficient and has also been extended to non-projective trees (McDonald et al., 2005b). It provides a discriminative online learning algorithm which when combined with a rich feature set reaches state-of-the-art performance across multiple languages.

However, within this framework one can only define features over single attachment decisions. This leads to cases where basic linguistic constraints are not satisfied (e.g. verbs with two subjects or incompatible coordination arguments). An example of this for Dutch is illustrated in Figure 2 which was produced by the parser of McDonald et al. (2005b). Here the parse contains a coordination of incompatible word classes (a preposition and a verb).

Our approach is able to include additional constraints which forbid configurations such as those in Figure 2. While McDonald and Pereira (2006) address the issue of local attachment decisions by defining scores over attachment pairs, our solution is more general. Furthermore, it is complementary in the sense that we could formulate their model using ILP and then add constraints.

The method we present is not the only one that can take global constraints into account. Deterministic dependency parsing (Nivre et al., 2004; Yamada and Matsumoto, 2003) can apply global constraints by conditioning attachment decisions on the intermediate parse built. However, for efficiency a greedy search is used which may produce sub-optimal solutions. This is not the case when using ILP.

3 Model

Our underlying model is a modified labelled version[2] of McDonald et al. (2005b):

$$\begin{aligned} s(\mathbf{x}, \mathbf{y}) &= \sum_{(i,j,l) \in \mathbf{y}} s(i,j,l) \\ &= \sum_{(i,j,l) \in \mathbf{y}} \mathbf{w} \cdot \mathbf{f}(i,j,l) \end{aligned}$$

[2]Note that this is not described in the McDonald papers but implemented in his software.

where $\mathbf{x}$ is a sentence, $\mathbf{y}$ is a set of labelled dependencies, $\mathbf{f}(i, j, l)$ is a multidimensional feature vector representation of the edge from token i to token j with label l and $\mathbf{w}$ the corresponding weight vector. For example, a feature f_{101} in $\mathbf{f}$ could be:

$$f_{101}(i,j,l) = \begin{cases} 1 & \text{if } t(i) = \text{“en”} \wedge p(j) = \text{V} \\ & \wedge l = \text{“coordination”} \\ 0 & \text{otherwise} \end{cases}$$

where $t(i)$ is the word at token i and $p(j)$ the part-of-speech tag at token j.

Decoding in this model amounts to finding the $\mathbf{y}$ for a given $\mathbf{x}$ that maximises $s(\mathbf{x}, \mathbf{y})$:

$$y' = \arg\max_{y} s(\mathbf{x}, \mathbf{y})$$

while fulfilling the following constraints:

T1 For every non-root token in $\mathbf{x}$ there exists exactly one head; the root token has no head.

T2 There are no cycles.

Thus far, the formulation follows McDonald et al. (2005b) and corresponds to the Maximum Spanning Tree (MST) problem. In addition to **T1** and **T2**, we include a set of linguistically motivated constraints:

A1 Heads are not allowed to have more than one outgoing edge labelled l for all l in a set of labels U.

C1 In a symmetric coordination there is exactly one argument to the right of the conjunction and at least one argument to the left.

C2 In an asymmetric coordination there are no arguments to the left of the conjunction and at least two arguments to the right.

C3 There must be at least one comma between subsequent arguments to the left of a symmetric coordination.

C4 Arguments of a coordination must have compatible word classes.

P1 Two dependencies must not cross if one of their labels is in a set of labels P.

A1 covers constraints such as “there can only be one subject” if U contains “subject” (see Section 4.4 for more details of U). **C1** applies to configurations which contain conjunctions such as “en”,“of” or “maar” (“and”, “or” and “but”). **C2** will rule-out settings where a conjunction such as “zowel” (translates as “both”) having arguments to its left. **C3** forces coordination arguments to the left of a conjunction to have commas in between. **C4** avoids parses in which incompatible word classes are coordinated, such as nouns and verbs. Finally, **P1** allows *selective projective* parsing: we can, for instance, forbid the crossing of “Noun-Determiner” dependencies if we add the corresponding label type to P(see Section 4.4 for more details of P) . If we extend P to contain all labels we forbid any type of crossing dependencies. This corresponds to projective parsing.

3.1 Decoding

McDonald et al. (2005b) use the Chu-Liu-Edmonds (CLE) algorithm to solve the maximum spanning tree problem. However, global constraints cannot be incorporated into the CLE algorithm (McDonald et al., 2005b). We alleviate this problem by presenting an equivalent Integer Linear Programming formulation which allows us to incorporate global constraints naturally.

Before giving full details of our formulation we first introduce some of the concepts of linear and integer linear programming (for a more thorough introduction see Winston and Venkataramanan (2003)).

Linear Programming (LP) is a tool for solving optimisation problems in which the aim is to maximise (or minimise) a given linear function with respect to a set of linear *constraints*. The function to be maximised (or minimised) is referred to as the *objective function*. A number of *decision variables* are under our control which exert influence on the objective function. Specifically, they have to be optimised in order to maximise (or minimise) the objective function. Finally, a set of constraints restrict the values that the decision variables can take. Integer Linear Programming is an extension of linear programming where all decision variables must take integer values.

There are several explicit formulations of the MST problem as an integer linear program in the literature (Williams, 2002). They are based on the concept of eliminating subtours (cycles), cuts (disconnections) or requiring intervertex flows (paths). However, in practice these formulations cause long solve times — as the first two meth-

Algorithm 1 Incremental Integer Linear Programming

$C \leftarrow B_{\mathbf{x}}$
repeat
  $\mathbf{y} \leftarrow solve(C, O_{\mathbf{x}}, V_{\mathbf{x}})$
  $W \leftarrow violated(\mathbf{y}, I_{\mathbf{x}})$
  $C \leftarrow C \cup W$
until $V = \emptyset$
return $\mathbf{y}$

ods yield an exponential number of constraints. Although the latter scales cubically, it produces non-fractional solutions in its relaxed version; this causes long runtimes for the branch and bound algorithm (Williams, 2002) commonly used in integer linear programming. We found out experimentally that dependency parsing models of this form do not converge on a solution after multiple hours of solving, even for small sentences.

As a workaround for this problem we follow an incremental approach akin to the work of Warme (1998). Instead of adding constraints which forbid all possible cycles in advance (this would result in an exponential number of constraints) we first solve the problem without any cycle constraints. The solution is then examined for cycles, and if cycles are found we add constraints to forbid these cycles; the solver is then run again. This process is repeated until no more violated constraints are found. The same procedure is used for other types of constraints which are too expensive to add in advance (e.g. the constraints of **P1**).

Algorithm 1 outlines our approach. For a sentence $\mathbf{x}$, $B_{\mathbf{x}}$ is the set of constraints that we add in advance and $I_{\mathbf{x}}$ are the constraints we add incrementally. $O_{\mathbf{x}}$ is the objective function and $V_{\mathbf{x}}$ is a set of variables including integer declarations. $solve(C, O, V)$ maximises the objective function O with respect to the set of constraints C and variables V. $violated(\mathbf{y}, I)$ inspects the proposed solution ($\mathbf{y}$) and returns all constraints in I which are violated.

The number of iterations required in this approach is at most polynomial with respect to the number of variables (Grötschel et al., 1981). In practice, this technique converges quickly (less than 20 iterations in 99% of approximately 12,000 sentences), yielding average solve times of less than 0.5 seconds.

Our approach converges quickly due to the quality of the scoring function. Its weights have been trained on cycle free data, thus it is more likely to guide the search to a cycle free solution.

In the following section we present the objective function $O_{\mathbf{x}}$, variables $V_{\mathbf{x}}$ and linear constraints $B_{\mathbf{x}}$ and $I_{\mathbf{x}}$ needed for parsing $\mathbf{x}$ using Algorithm 1.

3.1.1 Variables

$V_{\mathbf{x}}$ contains a set of binary variables to represent labelled edges:

$$e_{i,j,l} \quad \forall i \in 0..n, j \in 1..n, \\ l \in best_k(i,j)$$

where n is the number of tokens and the index 0 represents the root token. $best_k(i,j)$ is the set of k labels with highest $s(i,j,l)$. $e_{i,j,l}$ equals 1 if there is a edge (i.e., a dependency) with the label l between token i (head) and j (child), 0 otherwise. k depends on the type of constraints we want to add. For the plain MST problem it is sufficient to set $k = 1$ and only take the best scoring label for each token pair. However, if we want a constraint which forbids duplicate subjects we need to provide additional labels to fall back on.

$V_{\mathbf{x}}$ also contains a set of binary auxiliary variables:

$$d_{i,j} \quad \forall i \in 0..n, j \in 1..n$$

which represent the existence of a dependency between tokens i and j. We connect these to the $e_{i,j,l}$ variables by the constraint:

$$d_{i,j} = \sum_{l \in best_k(i,j)} e_{i,j,l}$$

3.1.2 Objective Function

Given the above variables our objective function $O_{\mathbf{x}}$ can be represented as (using a suitable k):

$$\sum_{i,j} \sum_{l \in best_k(i,j)} s(i,j,l) \cdot e_{i,j,l}$$

3.1.3 Base Constraints

We first introduce a set of base constraints $B_{\mathbf{x}}$ which we add in advance.

Only One Head (T1) Every token has exactly one head:

$$\sum_i d_{i,j} = 1$$

for non-root tokens $j > 0$ in $\mathbf{x}$. An exception is made for the artificial root node:

$$\sum_i d_{i,0} = 0$$

Label Uniqueness (A1) To enforce uniqueness of children with labels in U we augment our model with the constraint:

$$\sum_j e_{i,j,l} \leq 1$$

for every token i in $\mathbf{x}$ and label l in U.

Symmetric Coordination (C1) For each conjunction token i which forms a symmetric coordination we add:

$$\sum_{j<i} d_{i,j} \geq 1$$

and

$$\sum_{j>i} d_{i,j} = 1$$

Asymmetric Coordination (C2) For each conjunction token i which forms an asymmetric coordination we add:

$$\sum_{j<i} d_{i,j} = 0$$

and

$$\sum_{j>i} d_{i,j} \geq 2$$

3.1.4 Incremental Constraints

Now we present the set of constraints $I_{\mathbf{x}}$ we add incrementally. The constraints are chosen based on the two criteria: (1) adding them to the base constraints (those added in advance) would result in an extremely large program, and (2) it must be efficient to detect whether the constraint is violated in $\mathbf{y}$.

No Cycles (T2) For every possible cycle c for the sentence $\mathbf{x}$ we have a constraint which forbids the case where all edges in c are active simultaneously:

$$\sum_{(i,j)\in c} d_{i,j} \leq |c| - 1$$

Comma Coordination (C3) For each symmetric conjunction token i which forms a symmetric coordination and each set of tokens A in $\mathbf{x}$ to the left of i with no comma between each pair of successive tokens we add:

$$\sum_{a\in A} d_{i,a} \leq |A| - 1$$

which forbids configurations where i has the argument tokens A.

Compatible Coordination Arguments (C4) For each conjunction token i and each set of tokens A in $\mathbf{x}$ with incompatible POS tags, we add a constraint to forbid configurations where i has the argument tokens A.

$$\sum_{a\in A} d_{i,a} \leq |A| - 1$$

Selective Projective Parsing (P1) For each pair of triplets (i, j, l_1) and (m, n, l_2) we add the constraint:

$$e_{i,j,l_1} + e_{m,n,l_2} \leq 1$$

if l_1 or l_2 is in P.

3.2 Training

For training we use single-best MIRA (McDonald et al., 2005a). This is an online algorithm that learns by parsing each sentence and comparing the result with a gold standard. Training is complete after multiple passes through the whole corpus. Thus we decode using the Chu-Liu-Edmonds (CLE) algorithm due to its speed advantage over ILP (see Section 5.2 for a detailed comparison of runtimes).

The fact that we decode differently during training (using CLE) and testing (using ILP) may degrade performance. In the presence of additional constraints weights may be able to capture other aspects of the data.

4 Experimental Set-up

Our experiments were designed to answer the following questions:

1. How much do our additional constraints help improve accuracy?

2. How fast is our generic inference method in comparison with the Chu-Liu-Edmonds algorithm?

3. Can approximations be used to increase the speed of our method while remaining accurate?

Before we try to answer these questions we briefly describe our data, features used, settings for U and P in our parametric constraints, our working environment and our implementation.

4.1 Data

We use the Alpino treebank (van der Beek et al., 2002), taken from the CoNLL shared task of multilingual dependency parsing[3]. The CoNLL data differs slightly from the original Alpino treebank as the corpus has been part-of-speech tagged using a Memory-Based-Tagger (Daelemans et al., 1996). It consists of 13,300 sentences with an average length of 14.6 tokens. The data is non-projective, more specifically 5.4% of all dependencies are crossed by at least one other dependency. It contains approximately 6000 sentences more than the Alpino corpus used by Malouf and van Noord (2004).

The training set was divided into a 10% development set (*dev*) while the remaining 90% is used as a training and cross-validation set (*cross*). Feature sets, constraints and training parameters were selected through training on *cross* and optimising against *dev*.

Our final accuracy scores and runtime evaluations were acquired using a nine-fold cross-validation on *cross*

4.2 Environment and Implementation

All our experiments were conducted on a Intel Xeon with 3.8 Ghz and 4Gb of RAM. We used the open source Mixed Integer Programming library *lp_solve*[4] to solve the Integer Linear Programs. Our code ran in Java and called the JNI-wrapper around the *lp_solve* library.

4.3 Feature Sets

Our feature set was determined through experimentation with the development set. The features are based upon the data provided within the Alpino treebank. Along with POS tags the corpus contains several additional attributes such as gender, number and case.

Our best results on the development set were achieved using the feature set of McDonald et al. (2005a) and a set of features based on the additional attributes. These features combine the attributes of the head with those of the child. For example, if token i has the attributes a_1 and a_2, and token j has the attribute a_3 then we created the features $(a_1 \wedge a_3)$ and $(a_2 \wedge a_3)$.

[3]For details see `http://nextens.uvt.nl/~conll`.

[4]The software is available from `http://www.geocities.com/lpsolve`.

4.4 Constraints

All the constraints presented in Section 3 were used in our model. The set U of unique labels constraints contained *su, obj1, obj2, sup, ld, vc, predc, predm, pc, pobj1, obcomp* and *body*. Here *su* stands for subject and *obj1* for direct object (for full details see Moortgat et al. (2000)).

The set of projective labels P contained *cnj*, for coordination dependencies; and *det*, for determiner dependencies. One exception was added for the coordination constraint: dependencies can cross when coordinated arguments are verbs.

One drawback of hard deterministic constraints is the undesirable effect noisy data can cause. We see this most prominently with coordination argument compatibility. Words ending in "en" are typically wrongly tagged and cause our coordination argument constraint to discard correct coordinations. As a workaround we assigned words ending in "en" a wildcard POS tag which is compatible with all POS tags.

5 Results

In this section we report our results. We not only present our accuracy but also provide an empirical evaluation of the runtime behaviour of this approach and show how parsing can be accelerated using a simple approximation.

5.1 Accuracy

An important question to answer when using global constraints is: How much of a performance boost is gained when using global constraints?

We ran the system without any linguistic constraints as a baseline (*bl*) and compared it to a system with the additional constraints (*cnstr*). To evaluate our systems we use the accuracy over labelled attachment decisions:

$$LAC = \frac{N_l}{N_t}$$

where N_l is the number of tokens with correct head and label and N_t is the total number of tokens. For completeness we also report the unlabelled accuracy:

$$UAC = \frac{N_u}{N_t}$$

where N_u is the number of tokens with correct head.

	LAC	UAC	LC	UC
bl	84.6%	88.9%	27.7%	42.2%
cnstr	85.1%	89.4%	29.7%	43.8%

Table 1: Labelled (*LAC*) and unlabelled (*UAC*) accuracy using nine-fold cross-validation on $cross$ for baseline (*bl*) and constraint-based (*constr*) system. *LC* and *UC* are the percentages of sentences with 100% labelled and unlabelled accuracy, respectively.

Table 1 shows our results using nine-fold cross-validation on the *cross* set. The baseline system (no additional constraints) gives an unlabelled accuracy of 84.6% and labelled accuracy of 88.9%. When we add our linguistic constraints the performance increases by 0.5% for both labelled and unlabelled accuracy. This increase is significant ($p < 0.001$) according to Dan Bikel's parse comparison script and using the Sign test ($p < 0.001$).

Now we give a little insight into how our results compare with the rest of the community. The reported state-of-the-art parser of Malouf and van Noord (2004) achieves 84.4% labelled accuracy which is very close numerically to our baseline. However, they use a subset of the CoNLL Alpino treebank with a higher average number of tokens per sentences and also evaluate control relations, thus results are not directly comparable. We have also run our parser on the relatively small (approximately 400 sentences) CoNNL test data. The best performing system (McDonald et al. 2006; note: this system is different to our baseline) achieves 79.2% labelled accuracy while our baseline system achieves 78.6% and our constrained version 79.8%. However, a significant difference is only observed between our baseline and our constraint-based system.

Examining the errors produced using the *dev* set highlight two key reasons why we do not see a greater improvement using our constraint-based system. Firstly, we cannot improve on coordinations that include words ending with "en" based on the workaround present in Section 4.4. This problem can only be solved by improving POS taggers for Dutch or by performing POS tagging within the dependency parsing framework.

Secondly, our system suffers from poor next best solutions. That is, if the best solution violates some constraints, then we find the next best solution is typically worse than the best solution with violated constraints. This appears to be a consequence of inaccurate local score distributions (as opposed to inaccurate best local scores). For example, suppose we attach two subjects, t_1 and t_2, to a verb, where t_1 is the actual subject while t_2 is meant to be labelled as object. If we forbid this configuration (two subjects) and if the score of labelling t_1 object is higher than that for t_2 being labelled subject, then the next best solution will label t_1 incorrectly as object and t_2 incorrectly as subject. This is often the case, and thus results in a drop of accuracy.

5.2 Runtime Evaluation

We now concentrate on the runtime of our method. While we expect a longer runtime than using the Chu-Liu-Edmonds as in previous work (McDonald et al., 2005b), we are interested in how large the increase is.

Table 2 shows the average solve time (*ST*) for sentences with respect to the number of tokens in each sentence for our system with constraints (*cnstr*) and the Chu-Liu-Edmonds (*CLE*) algorithm. All solve times do not include feature extraction as this is identical for all systems. For *cnstr* we also show the number of sentences that could not be parsed after two minutes, the average number of iterations and the average duration of the first iteration.

The results show that parsing using our generic approach is still reasonably fast although significantly slower than using the Chu-Liu-Edmonds algorithm. Also, only a small number of sentences take longer than two minutes to parse. Thus, in practice we would not see a significant degradation in performance if we were to fall back on the CLE algorithm after two minutes of solving.

When we examine the average duration of the first iteration it appears that the majority of the solve time is spent within this iteration. This could be used to justify using the CLE algorithm to find a initial solution as starting point for the ILP solver (see Section 6).

5.3 Approximation

Despite the fact that our parser can parse all sentences in a reasonable amount of time, it is still significantly slower than the CLE algorithm. While this is not crucial during decoding, it does make discriminative online training difficult as training requires several iterations of parsing the whole corpus.

Tokens	1-10	11-20	21-30	31-40	41-50	>50
Count	5242	4037	1835	650	191	60
Avg. ST (CLE)	0.27ms	0.98ms	3.2ms	7.5ms	14ms	23ms
Avg. ST (cnstr)	5.6ms	52ms	460ms	1.5s	7.2s	33s
ST > 120s (cnstr)	0	0	0	0	3	3
Avg. # iter. (cnstr)	2.08	2.87	4.48	5.82	8.40	15.17
Avg. ST 1st iter. (cnstr)	4.2ms	37ms	180ms	540ms	1.3s	2.6s

Table 2: Runtime evaluation for different sentence lengths. Average solve time (*ST*) for our system with constraints (*constr*), the Chu-Liu-Edmonds algorithm (*CLE*), number of sentences with solve times greater than 120 seconds, average number of iterations and first iteration solve time.

	q=5	q=10	all	CLE
LAC	84.90%	85.11%	85.14%	85.14%
ST	351s	760s	3640s	20s

Table 3: Labelled accuracy (*LAC*) and total solve time (*ST*) for the *cross* dataset using varying q values and the Chu-Liu-Edmonds algorithm (*CLE*)

Thus we investigate if it is possible to speed up our inference using a simple approximation. For each token we now only consider the q variables in $V_{\mathbf{x}}$ with the highest scoring edges. For example, if we set $q = 2$ the set of variables for a token j will contain two variables, either both for the same head i but with different labels (variables e_{i,j,l_1} and e_{i,j,l_2}) or two distinct heads i_1 and i_2 (variables e_{i_1,j,l_1} and e_{i_2,j,l_2}) where labels l_1 and l_2 may be identical.

Table 3 shows the effect of different q values on solve time for the full corpus *cross* (roughly 12,000 sentences) and overall accuracy. We see that solve time can be reduced by 80% while only losing a marginal amount of accuracy when we set q to 10. However, we are unable to reach the 20 seconds solve time of the CLE algorithm. Despite this, when we add the time for preprocessing and feature extraction, the CLE system parses a corpus in around 15 minutes whereas our system with $q = 10$ takes approximately 25 minutes[5]. When we view the total runtime of each system we see our system is more competitive.

6 Discussion

While we have presented significant improvements using additional constraints, one may wonder whether the improvements are large enough to justify further research in this direction; especially since McDonald and Pereira (2006) present an approximate algorithm which also makes more global decisions. However, we believe that our approach is complementary to their model. We can model higher order features by using an extended set of variables and a modified objective function. Although this is likely to increase runtime, it may still be fast enough for real world applications. In addition, it will allow exact inference, even in the case of non-projective parsing. Also, we argue that this approach has potential for interesting extensions and applications.

For example, during our runtime evaluations we find that a large fraction of solve time is spent in the first iteration of our incremental algorithm. After the first iteration the solver uses its last state to efficiently search for solutions in the presence of new constraints. Some solvers allow the specification of an initial solution as a starting point, thus it is expected that significant improvements in terms of speed can be made by using the CLE algorithm to provide an initial solution.

Our approach uses a generic algorithm to solve a complex task. Thus other applications may benefit from it. For instance, Germann et al. (2001) present an ILP formulation of the Machine Translation (MT) decoding task in order to conduct exact inference. However, their model suffers from the same type of exponential blow-up we observe when we add all our cycle constraints in advance. In fact, the constraints which cause the exponential explosion in their graphically formulation are of the same nature as our cycle constraints. We hope that the incremental approach will allow exact MT decoding for longer sentences.

[5]Even when caching feature extraction during training McDonald et al. (2005a) still takes approximately 10 minutes to train.

7 Conclusion

In this paper we have presented a novel approach for inference using ILP. While previous approaches which use ILP for decoding have solved each integer linear program in one run, we incrementally add constraints and solve the resulting program until no more constraints are violated. This allows us to efficiently use ILP for dependency parsing and add constraints which provide a significant improvement over the current state-of-the-art parser (McDonald et al., 2005b) on the Dutch Alpino corpus (see *bl* row in Table 1).

Although slower than the baseline approach, our method can still parse large sentences (more than 50 tokens) in a reasonable amount of time (less than a minute). We have shown that parsing time can be significantly reduced using a simple approximation which only marginally degrades performance. Furthermore, we believe that the method has potential for further extensions and applications.

Acknowledgements

Thanks to Ivan Meza-Ruiz, Ruken Çakıcı, Beata Kouchnir and Abhishek Arun for their contribution during the CoNLL shared task and to Mirella Lapata for helpful comments and suggestions.

References

Culotta, Aron and Jeffery Sorensen. 2004. Dependency tree kernels for relation extraction. In *42nd Annual Meeting of the Association for Computational Linguistics*. Barcelona, Spain, pages 423–429.

Daelemans, W., J. Zavrel, and S. Berck. 1996. MBT: A memory-based part of speech tagger-generator. In *Proceedings of the Fourth Workshop on Very Large Corpora*. pages 14–27.

Ding, Yuan and Martha Palmer. 2005. Machine translation using probabilistic synchronous dependency insertion grammars. In *The 43rd Annual Meeting of the Association of Computational Linguistics*. Ann Arbor, MI, USA, pages 541–548.

Germann, Ulrich, Michael Jahr, Kevin Knight, Daniel Marcu, and Kenji Yamada. 2001. Fast decoding and optimal decoding for machine translation. In *Meeting of the Association for Computational Linguistics*. Toulouse, France, pages 228–235.

Grötschel, M., L. Lovász, and A. Schrijver. 1981. The ellipsoid method and its consequences in combina- torial optimization. *Combinatorica* I:169– 197.

Malouf, Robert and Gertjan van Noord. 2004. Wide coverage parsing with stochastic attribute value grammars. In *Proc. of IJCNLP-04 Workshop "Beyond Shallow Analyses"*. Sanya City, Hainan Island, China.

McDonald, R., K. Crammer, and F. Pereira. 2005a. Online large-margin training of dependency parsers. In *43rd Annual Meeting of the Association for Computational Linguistics*. Ann Arbor, MI, USA, pages 91–98.

McDonald, R. and F. Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *11th Conference of the European Chapter of the Association for Computational Linguistics*. Trento, Italy, pages 81–88.

McDonald, R., F. Pereira, K. Ribarov, and J. Hajic. 2005b. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Vancouver, British Columbia, Canada, pages 523–530.

McDonald, Ryan, Kevin Lerman, and Fernando Pereira. 2006. Multilingual dependency analysis with a two-stage discriminative parser. In *Proceedings of CoNLL-2006*. New York, USA.

Moortgat, M., I. Schuurman, and T. van der Wouden. 2000. Cgn syntactische annotatie. Internal report Corpus Gesproken Nederlands.

Nivre, J., J. Hall, and J. Nilsson. 2004. Memory-based dependency parsing. In *Proceedings of CoNLL-2004*. Boston, MA, USA, pages 49–56.

Roth, D. and W. Yih. 2004. A linear programming formulation for global inference in natural language tasks. In *Proceedings of CoNLL-2004,*. Boston, MA, USA, pages 1–8.

van der Beek, L., G. Bouma, R. Malouf, G. van Noord, Leonoor van der Beek, Gosse Bouma, Robert Malouf, and Gertjan van Noord. 2002. The Alpino dependency treebank. In *Computational Linguistics in the Netherlands (CLIN)*. Rodopi.

Warme, David Michael. 1998. *Spanning Trees in Hypergraphs with Application to Steiner Trees*. Ph.D. thesis, University of Virginia.

Williams, Justin C. 2002. A linear-size zero - one programming model for the minimum spanning tree problem in planar graphs. *Networks* 39:53–60.

Winston, Wayne L. and Munirpallam Venkataramanan. 2003. *Introduction to Mathematical Programming*. Brooks/Cole.

Yamada, Hiroyasu and Yuji Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proceedings of IWPT*. pages 195–206.