Proceedings of the Federated Conference on
Computer Science and Information Systems pp. 19–26

ISBN 978-83-60810-51-4

Combining Black-box Taggers and Parsers for Modern Standard Arabic

Maytham Alabbas, Allan Ramsay
School of Computer Science
University of Manchester
Manchester, M13 9PL, UK
Email: {alabbasm,ramsay}@cs.man.ac.uk

***Abstract*—A number of trainable dependency parsers have been presented in the literature. These parsers require tagged input: this may potentially cause a problem, because taggers are not in general 100% accurate, and any errors in tagging are likely to lead to errors in the output of the parsers. The current paper investigates the relationship between tagging errors and parsing errors. The investigation is carried out on Arabic text, using specific taggers and parsers, but the lessons that can be learned are applicable to other languages and other tools of the same kind.**

I. Introduction

THE problem: a number of trainable parsers have been proposed, particularly for dependency grammars. These parsers are trained on treebanks, where the words that appear at the nodes of the trees have been part-of-speech tagged (POS tagged, tagged). They can then be used to parse unseen texts *which have also been tagged*. In general, parsers and taggers can make mistakes. It seems likely that mistakes in tagging the target unseen texts will lead to extra errors being introduced into the output of the parsers. The aim of the current paper is to investigate the extent of this problem. How much worse does a data-driven parser perform if the input is incorrectly tagged?

We report on a number of experiments using different taggers, different parsers, and different sizes of training corpus. The specific taggers we use are AMIRA [1], MADA [2] and an in-house maximum-likelihood (MXL) Arabic tagger [3], which achieves similar accuracy: we will say more about the advantages of the in-house tagger for the current investigations below. The parsers are MALTParser [4], [5], [6] and MSTParser [7], [8]. The training corpus is the Penn Arabic Treebank (PATB) [9]. The trees in the PATB are phrase-structure trees: we used a standard algorithm for converting phrase-structure trees to dependency trees. We will discuss this conversion when we consider the PATB in more detail below.

We carried out two sets of experiments:

- We took all possible combinations of taggers and parsers and calculated the accuracy, taking the output of the given parser with the tags included in the PATB as the reference. If the errors introduced by the tagger and the mistakes made by the parser are independent, you would expect that combining a tagger whose accuracy was T with a parser whose accuracy was P when trained on a perfectly tagged corpus to be $T \times P$. The result of these experiments (to save the reader the suspense of waiting to the end of the paper) was that in general the accuracy is greater than $T \times P$–the problems introduced by having a less than perfect tagger are to some extent compensated for by the parsers.
- The mistakes made by one combination of parser and tagger were not in general the same as the mistakes made by another combination. We therefore looked at the effects of merging tagger:parser pairs–of using say MADA+MSTParser and AMIRA+MALTParser, accepting only those analyses where the two agreed. Unsurprisingly, the precision of this approach was markedly higher than the accuracy of either of the contributing combinations, but in general the F-score for the merged version was very close to their average: this is roughly what you would expect–where they disagree at least one of them must be wrong, so eliminating cases where they disagree is bound to remove some errors. Unfortunately, there does not seem to be any compensating factor of the kind that comes into play when combining taggers and parsers. In the first set of experiments the results are better than might have been expected. In the second set they are almost exactly what you would expect. For tasks where precision is important it is worth doing the merge, but you do lose almost the same level of recall as you gain in precision.

II. From the PATB to dependency trees

We used PATB (Part 1 v3 using '`without-vowel`' set of trees)[1] to train the two parsers we used in our experiments. The trees in PATB are phrase-structure trees (it contains just over 5000 trees, at an average length of 28 words per tree, with some trees containing 100+ words), and hence are not directly useable for training dependency parsers. We therefore need to convert PATB trees to dependency trees.

[1]Catalog number LDC2005T02 from the Linguistic Data Consortium (LDC). Available at: http://www.ldc.upenn.edu/Catalog/catalogEntry.jsp?catalogId=LDC2005T02

Converting phrase-structure trees to dependency trees is a straightforward task, so long as (i) the item in each subtree which contains the head can be identified; and (ii) there are no constructions with zero-heads. Assuming that these two conditions hold, the following algorithm will produce a dependency tree from a phrase-structure tree [10], [11]:

1) given a leaf node, turn it into a tree with no daughters;
2) a) otherwise, choose the subtree which contains the head: turn it into a dependency tree: call this D;
   b) turn all the other subtrees into dependency trees, and add them as daughters of D.

The only difficult part of this algorithm, which has been discussed by Xia and Palmer [12], is the selection of the subtree which contains the head. The approach we have taken to this task is to look for all the trees headed by a given label, and find all the labels for their subtrees. This gives us a list of labels for potential head daughters for each non-terminal label (a 'head-percolation table' [13]). We then order these in terms of candidacy for being the head daughter, in terms of what we believe to be the correct dependency structure, and we use this preference order for 'choose the subtree which contains the head: turn it into a dependency tree' in the above algorithm.

There are, however, a number of problems with the PATB.

- Very large numbers of Arabic sentences begin with the conjunction +و *w+* "and." The most plausible reading is that this item implicitly conjoins the first clause in the current sentence to the previous sentence, and hence should be taken as its head. This clashes with the treatment in the PATB, where the conjunction is taken to be the head of the whole sentence. This makes a difference in cases where the sentence has the general form `CONJ` S_1 `CONJ` S_2 `CONJ ... CONJ` S_n, where we bracket it as `CONJ (`S_1 `CONJ` S_2 `CONJ ... CONJ` S_n`)` and the PATB brackets it as `(CONJ` S_1`) CONJ` S_2 `CONJ ... CONJ` S_n. There are a surprising number of such cases, and the rebracketing makes a noticeable difference to parsing accuracy. We therefore have to restructure these trees, and we also mark sentence-initial +و *w+* "and" with a special tag (i.e. `ICONJ`) to prevent the parsers confusing it with more normal conjunctive uses of this item [14].
- The PATB deals with free word-order by using traces, where the function of an extraposed item is co-indexed with an item whose label indicates whether it is a subject or object, as shown in Fig. 1, where the relative pronoun اللذان *All⋆An*[2] "who" is coindexed with a trace which is itself taken to be the subject of يساهمان *y+sAhm+An* "contribute to."

[2]The transcription of Arabic examples follows Buckwalter's system for transcribing Arabic symbols. Available at: http://www.qamus.org/transliteration.htm

```
[S [S [CONJ w-]
      [VP [VERB -gyr]
          [NP-SBJ [NP [DET+NOUN Al+mSrf+An]]
                  [PUNC ,]
                  [SBAR [WHNP-1 [REL_PRON All⋆An]]
                        [S [VP [VERB y+sAhm+An]
                               [NP-SBJ-1 [-NONE- ⋆T⋆]]
                               ...]]]]]]]
```

Fig. 1. Phrase-structure tree with trace.

This does not really make sense within a dependency-based framework. The use of traces is antithetical to the basic idea of dependency grammar, namely that syntactic structure is determined by relations between *words*: a trace is not a word, and as such has no place in dependency grammar, at least as strictly conceived (e.g. [15]). We therefore systematically transform the PATB so that traces are eliminated, with the topicalised NP treated as a proper constituent of the sentence, e.g. in Fig. 1 the relative pronoun اللذان *All⋆An* "who" is taken to be a dependent (as subject) of يساهمان *y+sAhm+An* "contribute to" despite appearing higher in the original phrase structure tree.

- Arabic allows 'verbless' or 'equational' sentences, consisting of a noun phrase (`NP`) and some kind of predication (another `NP`, a prepositional phrase (`PP`), an adjective) [16], [17]. It is tempting to think of these constructions as containing a zero-copula (the fact that their negated forms do include an explicit copula supports this analysis). As noted above, we prefer to eliminate empty items, especially heads.
  The standard treatment of such sentences assumes that the predication is the head, largely on semantic grounds. This is almost certainly the right thing to do, but given that the parsers we are using exploit clues extracted from the local context to guide their decisions, it seemed worth investigating the effects of choosing either the subject or the predication as the head–if it turns out the parsers can more reliably assign labels when the subject is the head, we can easily transform the resulting dependency tree to the more normal form once it has been constructed. Over a number of experiments, it turns out that there is around a 2% overall increase in accuracy if the subject is taken to be the head [14]. We therefore make this unintuitive switch–if we can identify the constituents of a verbless sentence more easily by taking the subject as the head, then the extra cost of inverting this move seems well worth paying.
- The PATB uses a very fine-grained set of tags, which carry a great deal of syntactically relevant information (particularly case-marking). This tagset contains 306 tags, with for instance 47 tags for different kinds of verb and 44 for different kinds of noun. In particular, case-marking, and to a lesser extent number and gender marking, in Arabic is carried by diacritics which are unwritten in normal text. Thus the only way to extract this information is by making guesses based on the syntactic context.

Given that the task of the parser is to determine the syntactic context, it is hard to see how reliable guesses about it can be made prior to parsing. Marton et al. [18], for instance, note that adding the case-marking tags supplied by MADA actually decreases the accuracy, because the parser gives considerable weight to them, but the tagger only manages to assign them correctly in 86% of cases. A feature which is both significant and hard to determine is not something a reliable cue.

We therefore collapsed this set to a coarse-grained set with 39 distinct tags (e.g. PATB has 305 fine-grained tags which become 39 coarse-grained tags, for instance, the fine-grained tags `ADJ+CASE_DEF_GEN`, `ADJ+CASE_INDEF_GEN`, `ADJ+CASE_INDEF_ACC`, `ADJ+NSUFF_FEM_SG+CASE_INDEF_ACC` are grouped to `ADJ`). We can tag this set to just over 96% accuracy, which is comparable to the performance of other taggers [19], [20], [21], [22] on this kind of tagset, whereas we only achieve 91% with the full 306 tags (similarly MADA obtains 96.6% on the coarse-grained tagset but only 93.6% on the fine-grained (356-element) one).

The dependency trees to be used by the parsers also have to be labelled with functional relations. Since dependency grammar is entirely concerned with relations between words, any information beyond simple constituency has to be encoded in the labels on the relations. Such relations tend not be explicitly marked in phrase-structure trees, since they are often implied by the label of the local tree. We therefore have to *impose* a set of functional relations. We cannot use a very fine-grained set of labels here, because the information in the PATB simply does not provide enough information. The only labels we can assign with any degree of confidence relate to conjunctions, and to the subject and object of the verb (the PATB assigns distinct labels to the subject and object in verb-initial sentences, so we *can* use these to determine the subject and object).

III. The taggers

We are interested in the effects of incorrect tagging on the behaviour of the parsers. In order to ensure that the results are not just an artefact of the mistakes made by a specific tagger, we have carried out all our experiments using four tagged versions of the corpus. In each experiment, we have used the same tagger for both training and testing.

A. Gold-standard tags

The words in the PATB are already tagged. This provides us with a benchmark to evaluate the consequences of using taggers that do not provide 100% accuracy: we are unlikely to achieve higher accuracy when we tag the test sets using one of the taggers below than we obtain when we use the original tags of the PATB. In subsequent discussion we refer to the tags we obtain this way as gold-standard tags. Even these tags are not guaranteed to be 100% accurate–they have been obtained by some mixture of automatic and manual tagging, and both of these are liable to error. However, this is the most accurate available set of tags, and furthermore any systematic errors will also appear in the training set, and hence may be compensated for when the parsers are trained. We use this set for reference–if for some experiment we obtain N accuracy with the gold tags and N' using one of the taggers then we know that the tagger has introduced an error of $N - N'$.

ABBREV	DET+NOUN_PROP	LATIN	PUNC
ADJ	DET+NUM	NEG_PART	PV
ADV	EMPH_PART	NOUN	PVSUFF_DO
CONJ	EXCEPT_PART	NOUN_PROP	RC_PART
CV	FOCUS_PART	NO_FUNC	REL_ADV
CVSUFF_DO	FUT+IV	NUM	REL_PRON
DEM_PRON	INTERJ	PART	SUB
DET	INTERROG_PART	POSS_PRON	SUB_CONJ
DET+ADJ	IV	PREP	VERB_PART
DET+NOUN	IVSUFF_DO	PRON	

Fig. 2. Our coarse-grained tagset.

The tags in the PATB itself are very fine-grained, since the tags assigned to nouns include gender, number and case markers, and verb tags are also marked for number and gender. We follow common practice in discarding this very fine-grained information, reducing the 306 tags that appear in the PATB to a more coarse-grained set with 39 tags that are shown in Fig. 2 (reducing the granularity of the PATB tagset is fairly common practice, since some of the most fine-grained tags are very hard indeed to distinguish simply on the basis of local clues, which are what most taggers depend on. Unfortunately there is no universally accepted coarse-grained version, so we simply made what seemed like a reasonable compromise).[3] We will return to this in Section V.

B. AMIRA

The first tagger we use is AMIRA 2.0 [1]. This tagger is reported to achieve around 97% accuracy, which is about as good as any reported system.

Using AMIRA, however, highlights one of the problems that arise when you try to connect black-boxes together. The parsers require training data, for which we are using the PATB. The PATB is tagged, but with different tags from the ones used by AMIRA. In order to use AMIRA to tag the text before inputting it to the parsers, we will have to translate between the two tagsets.

This is a non-trivial task. The two tagsets have different numbers of tags (AMIRA has 29 tags, whereas

[3]We used the extended reduced tagset (ERTS) setting for AMIRA and then removed inflectional markers. This produced a set of tags that is very similar to the 25 tags in the Bies/RTS tagset, but with distinctions between nouns, adjectives and cardinal numbers. Although the Bies/RTS tagset is taken as a norm, there are numerous minor variants in use: Marton et al. [18], for instance collapse a number of POS tags but add other distinctions. This is perhaps unsurprising. Given the influence that using a specific tagset has on system performance, people will want to try out different tagsets. It does, however, make it difficult to maintain strict comparability.

our coarse-grained version of the PATB tagset has 39), and more importantly they make different kinds of distinctions. The AMIRA tagset, for instance, uses one tag (RP) to cover a range of particles which are subdivided into eight subclasses (EMPH_PART, EXCEPT_PART, FOCUS_PART, INTERROG_PART, RC_PART, NEG_PART, PART, VERB_PART) in the PATB; and it uses several tags to describe different kinds of verbs (VB, VBG, VBD, VBN, VBP) where the PATB just uses three (IV, PV, CV).

Places where the AMIRA tagset is coarser than the PATB tagset are particularly problematic. It seems plausible, for instance, that the different kinds of particle enter into different syntactic relationships. If we cannot tell the parsers which kind of particle some word is, then they are not going to be able to be very intelligent about guessing what its relationships with other words are going to be (this might have knock-on effects, so that it may not just be this word that is affected). Thus tagging using AMIRA is bound to introduce problems that would not arise if AMIRA used the same tagset as the PATB.

Places where the AMIRA tagset is finer than the PATB tagset are also awkward, because it may be that distinctions that are missing from the PATB tagset would have been useful.

In order to overcome these problems, we use transformation-based retagging (TBR) [23], [24] to recover from the mismatches between the two tagsets. TBR collects statistics about the local context in which erroneous tags have been assigned, and attempts to find rules based on this information to apply *after* the original tagger has been run. This technique will produce a small improvement in the performance of almost any tagger. Typically, taggers that achieve scores in the mid 90s are boosted by 2-3%–the lower the original accuracy, the greater the typical improvement. When we used it for comparing the original tags produced by AMIRA and the gold tags the score improved from around 87% to just over 97%. Some of this improvement arises simply from rules that spot that the two tagsets use different names for the same things (e.g. that things that are called JJ are called ADJ in the PATB) but some of it comes from learning how to split coarse-grained AMIRA tags into fine-grained PATB ones.

There is a further problem with using AMIRA. The fact that Arabic allows a range of items to be cliticised (conjunctions, prepositions, pronouns) makes it difficult even to tokenise text reliably. This means that not only does AMIRA sometimes assign different tags from the PATB, it sometimes even splits the text into different numbers of tokens. That makes it even harder to use AMIRA to tag texts in order to use parsers that have been trained using the PATB, since the tagged versions of the texts do not even contain the same numbers of tokens.

Here, we used AMIRA v2.0 with Yamcha toolkit v0.33.[4]

[4]http://chasen.org/~taku/software/yamcha/.

C. MADA

MADA [2] uses a slightly extended version of the PATB (Part 1 v3) tagset, with some extra classification of nouns (e.g. NOUN_QUANT and NOUN_NUM). The fine-grained MADA tagset contains 352 tags, compared to the normal PATB set of 306, and the coarse version has 57, compared to the 39 that we obtain by omitting inflectional markers (case, number, gender). Fortunately the MADA tags are a strict superset of the standard PATB set, and hence can be reduced to either the standard fine-grained version or our coarse version by omitting the extra classification of nouns, so we do not have the same problems using MADA with the PATB as we have with AMIRA.

We also applied TBR to the output of MADA, because although we were not faced with mapping incompatible tagsets in the same way as with AMIRA, using TBR nearly always provides a small improvement, amounting in this case to an increase from 94% to 96.6%. When we refer to AMIRA and MADA below, we will always be talking about the results of using these taggers with TBR.

Here, we used MADA 3.1 with SVMTools v1.3.1[5] and SRI's Language Modeling Toolkit (SRLIM) v1.5.12[6] and Standard Arabic Morphological Analyser (SAMA) v3.1 (catalog number LDC2010L01) from the LDC.[7]

D. Maximum-likelihood tagger

We also use an in-house maximum-likelihood (MXL) Arabic tagger [3]. We will simply outline the basic principles that it is based on and note its accuracy here.

MXL operates in two stages, as follows:

- In the first stage we use two simple kinds of statistic: (i) the conditional likelihood that a word which starts with the same three letters or ends with the same three letters as the one we are trying to tag has a given tag, and (ii) the transition probabilities between tags. We use a weighted combination of these to produce a maximum-likelihood guess at the current tag. This process produces about 91% accuracy.
- We then use TBR to refine the original set of hypotheses, leading to a final accuracy of 96.4%. It is noteworthy that adding TBR to a tagger whose initial accuracy is significantly lower than that of MADA and AMIRA brings it up to about the same level.

The advantage of this tagger is that because it was trained on the PATB, the tags it uses are exactly the PATB tags. We therefore do not need to overcome problems associated with mismatches between tagsets.

[5]http://www.lsi.upc.edu/~nlp/SVMTool/

[6]http://www.speech.sri.com/projects/srilm/download.html

[7]http://www.ldc.upenn.edu/Catalog/catalogEntry.jsp?catalogId=LDC2010L01

IV. The parsers

The parsers we use are data-driven dependency parsers. The parsers are given a representation of a set of dependency trees, and they infer rules that will enable them to parse unseen texts. The key advantages of these parsers are that they are robust (they will do something no matter what the input) and fairly fast. They are not perfectly accurate, but then no parser that is perfectly accurate has yet been developed.

A number of such tools are available. We have chosen MALTParser v1.5.2[8] and MSTParser v0.2[9] as examples of different approaches that are reported as achieving impressive results. The details of how these parsers work are outside the scope of the current paper. For us they are simply black-boxes which require tagged text as input, and whose performance is likely to worsen if the text is incorrectly tagged. The key question is: if a tagger assigns tags with accuracy T and a parser assigns roles to words with accuracy P, will the combination of the tagger and parser achieve $T \times P$ or more or less?

V. Experiments

A. Individual combinations of parsers and taggers

We are principally concerned with the interactions between the parsers and taggers. However, given that the parsers have to be trained on a corpus of trees, it is also of interest to see how their accuracy varies as the size of the dataset varies. In general, the performance of data-driven systems improves as the amount of data available to them increases.

Collecting large training sets, however, is very time-consuming, and the time to train a system also goes up, often non-linearly, as the amount of training data increases. Looking at the way the performance changes with the size of the training set can provide clues about the maximum that can be attained by a given tool and about the amount of data required to achieve a specified accuracy. In most of the experiments below, then, we plot accuracy against size of training data. This also provides a number of insights into how the taggers and parsers interact.

For the experiments described below, both parsers were trained on 16 datasets, starting with the first 250 sentences and incrementing by 250 sentences up to a maximum of 4000. In the testing step, the last 1000 sentences (from 5000 sentences in PATB) are used to test both parsers after each training step on one of the training datasets. Also for all the experiments described below the *label attachment score* (LA), i.e. the percentage of tokens with correct head and dependency relation, and the *unlabelled attachment score* (UA), i.e. the percentage of tokens with correct head, are used. The first experiment shows the results when we used the gold tags: this provides an upper bound for each parser–introducing mistakes into the tagging will almost certainly lead to a decrease in accuracy.

[8]http://www.maltparser.org/download.html
[9]http://www.seas.upenn.edu/~strctlrn/MSTParser/MSTParser.html

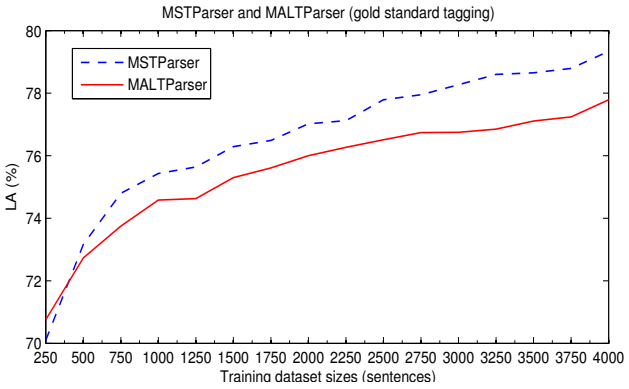

Fig. 3. Both parsers, labelled accuracy (LA) for gold standard tagging.

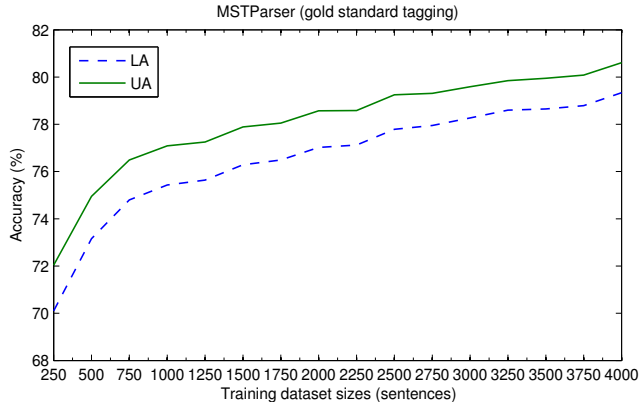

Fig. 4. Labelled (LA) and unlabelled (UA) accuracies of MSTPArser, gold standard tagging.

Fig. 3 shows exactly what you would expect: the accuracy of the parsers increases as the size of the training set increases. There is an initial sharp improvement, as the training set increases to about 1000 sentences, and then in both cases the improvement looks roughly linear in the size of the training set. This clearly cannot continue indefinitely–the accuracy must be capped at 100%, and presumably the actual limit is somewhere below that. However, since neither of the plots has become non-linear at the point where we were forced to stop training, it is impossible to estimate the asymptotic accuracy. The important thing to note is that the accuracy of the two parsers is of the same order of magnitude, *but that the errors they make are not identical.*

As noted above, the parsers both work with labelled dependency trees. However, for a number of tasks the constituency structure is all that is required. We have therefore compared the results when simply looking at whether the right head-dependent relations have been found ('UA' in Fig. 4) and at whether the right labels have been assigned to these relations ('LA'). The results for this comparison are shown in Fig. 4 for MSTParser with the gold tags–the results are extremely similar for all the tagger:parser combinations, so we have not repeated them for other combinations.

TABLE I
MALTPARSER AND MSTPARSER ACCURACIES, MULTIPLE TAGGERS COMPARED WITH GOLD STANDARD TAGGING.

Parser	Accuracy	gold	AMIRA	MXL	MADA
MSTParser	LA	79.3%	77.3%	78.2%	**78.5%**
	UL	80.6%	78.7%	**80%**	79.8%
MALTParser	LA	77.8%	75%	76%	**76.9%**
	UL	78.8%	76.3%	77.1%	**78%**

In the remaining experiments we will concentrate on scores for labelled trees. In every case the scores for unlabelled ones are about 2% higher, for all tagger:parser combinations and training sets.

Table I shows the effects of combining the taggers and parsers. In both cases we have included the results for the gold tags as a benchmark.

Both MALTParser and MSTParser do better when trained and tested with the corpus tagged by MXL or MADA than when we used AMIRA, despite the fact that the three taggers achieve very similar scores when viewed simply as taggers (AMIRA 97%, MXL 96.4% and MADA 96.5%).

We considered two possible causes for this difference: that it arises because of the difference between the tagsets used by the taggers, or that it arises from the differences in tokenisation.

Different tagsets: although the *sizes* of the two tagsets are similar, the nature of the tags themselves is different. The MXL tags, which are a coarse-grained variant of the tags used in the PATB itself, seem to provide more information about syntactic relations than the AMIRA set. In particular, the fine-grained distinctions between singular and plural nouns, and between various verb forms, that AMIRA provides are not actually very useful when trying to see whether two items are related.

It may be that the information about particles that MXL is sensitive to but AMIRA is not is likely to be useful for parsing: knowing, for instance, that some particle expects the next item to be a verb-initial sentence, whereas another expects a subject-initial sentence, may well be useful.

In order to see whether this was the cause of the difference, we constructed a version of the corpus where we replaced PATB tags by a coarse copy of the AMIRA tags, using hand-coded substitution rules, and then replaced these by fine-grained AMIRA tags where the substituted tags were compatible with tags actually assigned by AMIRA. Thus if the PATB assigned a word the tag `ADJ` we replaced it by the AMIRA tag `JJ`. We then inspected the tags assigned by AMIRA itself: if the tag assigned to this word was one of AMIRA's fine-grained adjective tags, e.g. `JJR`, then we used this instead. If, on the other hand, the hand-coded replacement for the PATB tag was incompatible with the one assigned by AMIRA then we retained the hand-coded one. This gave us a version of the corpus where all the tags were compatible with the tags in the PATB, but where some were AMIRA refinements of the originals.

Using the gold standard tags obtained directly from the PATB scores 78%, using the AMIRA tagset scores 76.5%. The only differences are that where the PATB tag translates to an AMIRA tag, and the tag assigned by AMIRA is compatible with the translation but is finer-grained, we have used the one assigned by AMIRA; and that where several PATB tags translate to the same AMIRA tag (e.g. particles) we have simply used the commonest translation. These results strongly suggest that the choice of tags is significant, since in this experiment the two sets are compatible at every point, but the LA-AMIRA tags include fine-grained distinctions that are made by AMIRA but not ones that are made in the PATB. Thus the decrease in accuracy of the parser must be due solely to the loss of information that arises when we merge PATB tags.

Tokenisation: the other potential problem is that AMIRA's tokeniser segments the text differently from the way that it is segmented in the PATB. In the PATB, for instance, the string لذلك *l⋆lk* "therefore" is treated as a single subordinating conjunction, whereas AMIRA breaks it into a preposition ل *l* "for" and a determiner ذلك *⋆lk* "that." Similarly, in the PATB the string لي *ly* "mine" is split into two tokens (a preposition ل *l* "for," and a pronoun ي *y* "me"), whereas AMIRA treats it as a single proper noun. This caused us problems when trying to use AMIRA to tag the treebank, since the leaves in the trees did not always correspond to tokens in the output of AMIRA. The difference in tokenisation affected around 2% of tokens, so since the parser performed around 2% worse with AMIRA than with the other taggers it seemed plausible that this was the source of the discrepancy.

In order to investigate the effect of this problem, we produced a version of the corpus where we replaced sequences where the PATB had a single token and AMIRA had several by the hand-coded AMIRA equivalent of the PATB tag, and likewise where the PATB had several tokens and AMIRA had one by the sequence of hand-coded AMIRA equivalents of the PATB tags.

This gave us a version of the treebank that had the same number of tokens as the original PATB, with as many items as possible given the tags assigned by AMIRA and the others given hand-coded AMIRA equivalents of the original PATB tags. The results suggest that this is not the source of the problem, since AMIRA produces almost identical results (no difference to three significant figures) no matter whether its own tokeniser or the tokenisation used in the PATB is used.

B. Merging combinations

A given tagger:parser combination will make errors. If two such combinations produce different parents for some word, then at least one of them must be wrong. So if we take the output of two combinations and reject all instances of words where the two suggest different parents for a word, we must improve the precision, because we will be throwing away items where we know that one of the combinations has made a

TABLE II
PRECISION (P), RECALL (R) AND F-SCORE (F) FOR DIFFERENT TAGGER$_1$:PARSER$_1$+TAGGER$_2$:PARSER$_2$ COMBINATIONS

MSTParser	MALTParser	Precision (P)	Recall (R)	F-score (F)
gold	gold	88%	71%	0.79
	AMIRA	86%	60%	0.71
	MXL	88%	59%	0.71
	MADA	89%	57%	0.69
AMIRA	gold	88%	57%	0.70
	AMIRA	86%	68%	**0.76**
	MXL	**88%**	67%	**0.76**
	MADA	**88%**	65%	0.75
MXL	gold	88%	59%	0.70
	AMIRA	83%	72%	**0.77**
	MXL	87%	56%	0.68
	MADA	**88%**	67%	0.76
MADA	gold	88%	57%	0.69
	AMIRA	84%	69%	0.76
	MXL	**87%**	67%	0.76
	MADA	**87%**	69%	**0.77**

mistake. The other one may, of course, have got the parent for this word right, but since we cannot tell which has got it wrong and which has got it right, we have to distrust the output of each. We will also decrease the recall, because there will now be items for which we have discarded the parents suggested by the parsers, and hence we will end up with no parent assigned to them. The question is: does the increase in precision compensate for the decrease in recall?

If the sets of mistakes that were made by the two combinations were complementary, then the precision of the merged output would be 1 and the recall would be $1-(w_1+w_2)$, where w_i is the error rate of combination i. The F-score for the merge if the combinations have complementary distributions would thus be $(1-(w_1+w_2))/(1-(w_1+w_2)/2)$. If, on the other hand, the two combinations made exactly the same mistakes then the F-score for the merge would be $(1-w_1)$. In other words, for combinations with similar accuracy, the highest precision will come if the errors they make are complementary, and the highest F-score will come if they make identical errors.

Table II shows the precision, recall and F-score for all possible pairs of combinations of tagger and parser, e.g. that if you use MSTParser having tagged using AMIRA and MALTParser having tagged using MADA then the precision is 88%, the recall is 65% and the F-score is 75%. This table includes cases where one or both the parsers were combined with the gold tags. These cases are greyed out, because in any real situation the gold tags would not be available.

Table II roughly bears out the observations above. The best precision is generally obtained by using different taggers for each part of the combination (so the best precision for AMIRA+MSTParser comes from merging it with MXL+MALTParser or MADA+MALTParser, the best for MXL+MSTParser comes from merging it with MADA+MALTParser, and the best for MADA+MSTParser from merging it with MXL+MALTParser (or MADA+MALTParser)); and the best F-score is generally obtained by using the same tagger with the two parsers.

VI. CONCLUSIONS

The impetus for the current paper arose from a simple query: what happens if you combine a tagger whose accuracy is T with a parser whose accuracy is P? It turns out that the answer is often better than $T \times P$. When, for instance, we combine MSTParser, for which we get an accuracy of 79.3% when using the gold tags, and MXL, whose accuracy is 96.4%, we get 78.7%, which is noticeably greater than the 76.1% that you would expect if the errors simply compounded one another. It seems as though the fact that the training set contains the same pattern of errors as the test set automatically provides a degree of compensation.

Closer inspection shows that the nature of the tagsets has a substantial effect. Because MXL uses the PATB tagset, which was presumably chosen because it carried the kind of information that is required for parsing, it interacted better with both the parsers than AMIRA, which uses a general purpose tagset which is not tuned to this task. The mismatch between AMIRA's built-in tokeniser and the tokenisation in the treebank caused us some technical problems, but does not seem to be a critical factor in the accuracy of the combination of AMIRA with the two parsers.

We have also investigated the effects of merging the outputs of pairs of tagger:parser combinations. The results here are broadly as expected–the precision of the combination is always better than the accuracy of either of the contributing pairs (which is inevitable), and using a different tagger with each parser in the combination produces the greatest improvement in precision (which is what we expected, but it needed confirmation).

The results above arise from investigating combinations of specific tools–three taggers $\times$ two parsers. Are these results compromised by the fact that we chose these specific tools, or are there general lessons to be learnt? The taggers all use different mechanisms and different information, and the parsers also use substantially different approaches. The fact that nonetheless we get fairly consistent results in Section V-B suggests that there may be some robustness about the conclusions. It seems that no matter which tagger and which parser we use we get results that are better than you would expect just by looking at the individual performance of the components. The main result of the second set of experiments–that taking the combined output of two different tagger:parser combinations improves the precision–is almost inevitable, but there is a tendency for combinations that involve different taggers to achieve greater precision than ones where the two parsers are combined with a single tagger.

It seems likely that this pattern would be repeated if other parsers or taggers were used: mistakes made by a combination of T and P could be caused either by T or by P. If T is combined with some other parser P', then the mistakes caused by T will almost certainly arise again, *and will not be spotted when the output of the two combinations is merged*, whereas a combination of another tagger T' with P' will not repeat the mistakes introduced by T. Thus although our experiments

were limited to a specific set of tools and a specific language (as any experiments must be) we believe that the results are likely to be applicable if other tools are used. The critical issue is that the tools should be distinct, either using different principles or different underlying data (training sets, rule sets), so that they do indeed make mistakes in different places. This observation is likely to transfer to other languages, not just other tools for handling Arabic. If you have multiple taggers and parsers which make distinct mistakes (which is likely to happen if they are based on different principles or use different features) then combining them will inevitably improve the precision and is likely to also improve the F-score.

ACKNOWLEDGMENTS

We would like to thank Dr. Yasser Sabtan (Al-Azhar University, Egypt) for important suggestions and for helpful discussions. We would like to extend our thanks to anonymous reviewers for their valuable comments. Maytham Alabbas owes his deepest gratitude to Iraqi Ministry of Higher Education and Scientific Research for financial support in his PhD study. Allan Ramsay's contribution to this work was partially supported by the Qatar National Research Fund (grant NPRP 09 - 046 - 6 - 001).

REFERENCES

[1] M. Diab, "Second generation tools (AMIRA 2.0): fast and robust tokenization, POS tagging, and base phrase chunking," *in Proceedings of the 2nd International Conference on Arabic Language Resources and Tools,* Cairo, 2009, pp. 285–288.

[2] N. Habash, O. Rambow, and R. Roth, "MADA+TOKAN: a toolkit for Arabic tokenization, diacritization, morphological disambiguation, POS tagging, stemming and lemmatization," *in Proceedings of the 2nd International Conference on Arabic Language Resources and Tools.* Cairo: The MEDAR Consortium, 2009.

[3] A. Ramsay and Y. Sabtan, "Bootstrapping a lexicon-free tagger for Arabic," *in Proceedings of the 9th Conference on Language Engineering ESOLEC'2009,* Cairo, Egypt, 2009, pp. 202–215.

[4] J. Nivre, J. Hall, and J. Nilsson, "MaltParser: a data-driven parser-generator for dependency parsing," *in Proceedings of the International Conference on Language Resources and Evaluation (LREC),* vol. 6, 2006, pp. 2216–2219.

[5] J. Nivre, J. Hall, J. Nilsson, A. Chanev, G. Eryigit, S. Kübler, S. Marinov, and E. Marsi, "MaltParser: a language-independent system for data-driven dependency parsing," *Natural Language Engineering,* vol. 13, no. 02, 2007, pp. 95–135.

[6] J. Nivre, L. Rimell, R. McDonald, and C. Gómez-Rodríguez, "Evaluation of dependency parsers on unbounded dependencies," *in Proceedings of the 23rd International Conference on Computational Linguistics (COLING 2010),* Beijing: Association for Computational Linguistics, 2010, pp. 833–841.

[7] R. McDonald, K. Lerman, and F. Pereira, "Multilingual dependency parsing with a two-stage discriminative parser," *in Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL-X),* New York, 2006.

[8] R. McDonald, K. Lerman, and F. Pereira, "Multilingual dependency parsing with a two-stage discriminative parser," *in 10th Conference on Computational Natural Language Learning (CoNLL-X),* New York, 2006.

[9] M. Maamouri and A. Bies, "Developing an Arabic treebank: methods, guidelines, procedures, and tools," *in Proceedings of the Workshop on Computational Approaches to Arabic Script-based Languages,* Geneva, 2004, pp. 2–9.

[10] M. Alabbas and A. Ramsay, "Evaluation of dependency parsers for long Arabic sentences," *in Proceeding of International Conference on Semantic Technology and Information Retrieval (STAIR'11).* Putrajaya, Malaysia: IEEE, 2011, pp. 243–248.

[11] M. Alabbas and A. Ramsay, "Evaluation of combining data-driven dependency parsers for Arabic," *in Proceeding of 5th Language & Technology Conference: Human Language Technologies (LTC'11),* Poznań, Poland, 2011, pp. 546–550.

[12] F. Xia and M. Palmer, "Converting dependency structures to phrase structures," *in Proceedings of the 1st International Conference on Human language technology research (HLT-2001),* San Diego, 2001, pp. 1–5.

[13] M. Collins, "Three generative, lexicalised models for statistical parsing," *in Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics,* Madrid, 1997, pp. 16–23.

[14] M. Alabbas and A. Ramsay, "Arabic treebank: from phrase-structure trees to dependency trees," *in Proceedings of the META-RESEARCH Workshop on Advanced Treebanking at the 8th International Conference on Language Resources and Evaluation (LREC),* Istanbul, Turkey, 2012, pp. 61–68.

[15] R. Hudson, *Word grammar.* Oxford: Basil Blackwell, 1984.

[16] M. Alabbas, "ArbTE: Arabic textual entailment," *in Proceedings of the 2nd Student Research Workshop associated with RANLP 2011.* Hissar, Bulgaria: RANLP 2011 Organising Committee, 2011, pp. 48–53.

[17] M. Attia, *Ambiguity in Arabic computational morphology and syntax: a study within the lexical functional grammar framework.* LAP Lambert Academic Publishing, 2012.

[18] Y. Marton, N. Habash, and O. Rambow, "Improving Arabic dependency parsing with lexical and inflectional morphological features," *in Proceedings of the NAACL HLT 2010 1st Workshop on Statistical Parsing of Morphologically-Rich Languages (SPMRL 2010).* Association for Computational Linguistics, 2010, pp. 13–21.

[19] F. Al Shamsi and A. Guessoum, "A hidden Markov model-based POS tagger for Arabic," *in Des Journées internationales d'Analyse statistique des Données Textuelles (JADT-8),* Besançon, 2006, pp. 31–42.

[20] S. AlGahtani, W. Black, and J. McNaught, "Arabic part-of-speech tagging using transformation-based learning," *in Proceedings of the 2nd International Conference on Arabic Language Resources and Tools,* Cairo, 2009, pp. 66–70.

[21] Y. Hadj, I. Al-Sughayeir, and A. Al-Ansari, "Arabic part-of-speech tagging using the sentence structure," *in Proceedings of the 2nd International Conference on Arabic Language Resources and Tools.* Cairo: The MEDAR Consortium, 2009, pp. 241–245.

[22] M. Diab, K. Hacioglu, and D. Jurafsky, "Automatic tagging of Arabic text: from raw text to base phrase chunks," *in Proceedings of NAACL-HLT 2004,* Boston, 2004, pp. 149–152.

[23] E. Brill, "Transformation-based error-driven learning and natural language processing: a case study in part of speech tagging," *Computational Linguistics,* vol. 23, no. 4, 1995, pp. 543–565.

[24] T. Lager, "μ-TBL lite: a small, extendible transformation-based learner," *in Proceedings of the 9th European Conference on Computational Linguistics (EACL-99).* Bergen: Association for Computational Linguistics, 1999, pp. 279–280.