Optimising Tree Edit Distance with Subtrees for Textual Entailment

Maytham Alabbas
Department of Computer Science,
College of Science, Basrah University,
Basrah, Iraq
maytham.alabbas@gmail.com

Allan Ramsay
School of Computer Science,
University of Manchester,
Manchester, M13 9PL, UK
Allan.Ramsa@manchester.ac.uk

Abstract

This paper introduces a method for improving tree edit distance (TED) for textual entailment. We explore two ways of improving TED: we extend the standard TED to use edit operations that apply to subtrees as well as to single nodes; and we use the 'artificial bee colony' algorithm (ABC) to estimate the cost of edit operations for single nodes and subtrees and to determine thresholds. The preliminary results of the current work for checking entailment between two texts are encouraging compared with the common bag-of-words, string edit distance and standard TED algorithms.

1 Introduction

One key task for natural language systems is to determine whether one natural language sentence entails another. *Entailment* can be defined as a relationship between two sentences where the truth of one sentence, the entailing expression, forces the truth of another sentence, what is entailed. Many natural language processing (NLP) tasks such as information extraction and question answering have to cope with this notion.

An alternative formulation for the entailment between two texts is given by the *recognising textual entailment* (RTE) paradigm, which contrasts with the standard definition of entailment above. Dagan et al. (2005) describe RTE as a task of determining, for two sentences *text T* and *hypothesis H*, whether *"...typically, a human reading T would infer that H is most likely true."* According to these authors, entailment holds if the truth of *H*, *as interpreted by a typical language user*, can be inferred from the meaning of *T*. This notion of entailment is less rigorous, and less clearly defined, than the standard notion, but it can be useful for a number of tasks, and has been investigated very extensively in recent times.

Tree edit distance (TED), which models *T-H* pairs by explicitly transforming *T* into *H* via a minimal cost sequence of editing operations, has been widely used for this task. Using TED poses two challenges: the standard three operations (i.e. deletion, insertion and exchange) apply only to single nodes, rather than to subtrees; and estimating a combination of costs for these operations with threshold(s) is hard when dealing with complex problems. This is because alterations in these costs or choosing a different combination of them can lead to drastic changes in TED performance (Mehdad and Magnini, 2009).

In order to overcome these challenges, we have extended the standard TED to deal with subtree operations as well as operations on single nodes. This allows the algorithm to treat semantically coherent parts of the tree as single items, thus allowing for instance entire modifiers (such as prepositional phrase (PPs)) to be inserted or deleted as single units. We have also applied the artificial bee colony (ABC) algorithm (Akay and Karaboga, 2012) to estimate costs both of edit operations (single node and subtree) and of threshold(s).

The work was carried out as part of an attempt to build a textual entailment (TE) system for modern standard Arabic (MSA)(Alabbas, 2011). MSA poses a number of problems that, while familiar from other languages, make tasks such as TE particularly difficult for this language–the lack of diacritics in written MSA combines with the complex derivational and inflectional morphology of the language to produce worse levels of lexical ambiguity than occur in many other languages; the combination of free word-order, pro-drop, verbless sentences and complex nominals produces higher levels of syntactic ambiguity than occur in many other languages; and the combination of these combinations makes things even worse. We have tested our algorithms on a corpus of MSA *T*-*H* pairs. This corpus contains 600 pairs, binary annotated as 'yes' and 'no' (a 50%-50% split). The average length of sentence in this dataset is 25 words per sentence, with some sentences containing 40+ words (see (Alabbas, 2013) for further details of this dataset and description of the methodology used for collecting it). In order to maintain comparability with work on TE for English, in Section 4 we have replicated a number of standard techniques (bag-of-words, Levenshtein distance on strings, standard TED). These experiments show that the extended version of TED, ETED, improves the performance of our technique for Arabic by around 3% in f-score and around 2% in accuracy compared with a number of well-known techniques. The relative performance of the standard techniques on our Arabic testset replicates the results reported for these techniques for English testsets. We have also applied our ETED to the English RTE2 testset, where it again outperforms the standard version of TED.

2 TED for RTE

The idea here is to convert both *T* and *H* from natural language expressions into parse trees through parsing and then to explicitly transform *T*'s parse tree into *H*'s parse tree, using a sequence of edit operations (Kouylekov and Magnini, 2005; Bar-Haim et al., 2007; Harmeling, 2009; Mehdad and Magnini, 2009; Wang and Manning, 2010; Heilman and Smith, 2010; Stern et al., 2012). If a low-cost transformation sequence can be found then it may be that *T* entails *H*. Dependency parsers (Kübler et al., 2009) are popular for this task, as in other NLP areas in recent years, since they allow us to be sensitive to the fact that the links in a dependency tree carry linguistic information about relations between complex units.

Different sets of operations on trees, using various types of transformations in order to derive *H* from *T*, have been suggested. Herrera et al. (2005), for instance, used the notion of tree inclusion (Kilpeläinen, 1992), which obtained one tree from another by deleting nodes. Herrera et al. (2006) and Marsi et al. (2006) used a tree alignment algorithm (Meyers et al., 1996), which produces a multiple sequence alignment on a set of sequences over a fixed tree. TED (Zhang and Shasha, 1989; Klein et al., 2000; Pawlik and Augsten, 2011) is another example of a transformation-based model in that it computes the minimum cost sequence of transformations (e.g. insertion, deletion and exchange of nodes) that turns one tree into the other. To obtain more accurate predictions, it is important to define an appropriate inventory of edit operations and assign appropriate costs to the edit operations during a training stage (Kouylekov and Magnini, 2005; Harmeling, 2009). For instance, exchanging a noun with its synonyms or hypernyms should cost less than exchanging it with an unrelated word. Heilman and Smith (2010) extended the above mentioned operations (e.g. move-sibling, relabel-edge, move-subtree, etc.), since the available edit operations are limited in capturing certain interesting and prevalent semantic phenomena. Similarly, a heuristic set of 28 edit operations, which include numbers of node-exchanges and restructuring of the entire parse tree, is suggested (Harmeling, 2009).

TED-based inference requires the specification of a cost for each edit operation and a threshold for the total cost of the edit sequence. Selecting a best set of costs and a suitable threshold is challenging. Some researchers have defined costs manually (Kouylekov and Magnini, 2005), but they are usually learned automatically (Harmeling, 2009; Wang and Manning, 2010; Heilman and Smith, 2010; Stern and Dagan, 2011), e.g. Mehdad and Magnini (2009) have used particle swarm optimization (PSO), which is a stochastic technique that mimics the social behaviour of bird flocking and fish schooling (Russell and Cohn, 2012), for estimating and optimising the cost of each edit operation for TED.

2.1 Standard TED

In this paper we will use Zhang and Shasha (1989)'s TED algorithm (henceforth, ZS-TED), which is an efficient technique based on dynamic programming to calculate the approximate tree matching for two rooted ordered trees, as a starting point. Ordered trees are trees in which the left-to-right order among siblings is significant. Approximate tree matching allows us to match a complete tree with just some parts of another tree. There are three operations, namely deleting, inserting and exchanging a node, which can transform one ordered tree to another. A nonnegative real cost is associated with each edit operation. These costs are changed to match the requirements of specific applications. Deleting a node x means attaching its children to the parent of x. Insertion is the inverse of deletion, with an inserted node becoming a parent of a consecutive subsequence in the left-to-right order of its parent. Exchanging a node alters its label. Detailed presentation of ZS-TED can be found in (Bille, 2005): the main change that we make to the basic algorithm is to include extra tables for recording *which* operations were performed rather than simply recording their cost.

2.2 Extended TED

The main weakness of ZS-TED is that it is not able to perform transformations on subtrees (i.e. delete subtree, insert subtree and exchange subtree). In order to make ZS-TED deal with subtree operations, we need to follow two stages:

1. Run ZS-TED (without entire subtree operations) and compute the standard alignment from the results;
2. Go over the alignment and group subtrees operations (e.g. every consecutive k deletions that correspond to an entire subtree reduces the edit distance score by $\alpha \times k + \beta$ for any desired α and β in interval [0,1]).

We have applied this technique on Zhang and Shasha (1989)'s $O(n^4)$ algorithm but it will also work for Klein (1998)'s $O(n^3 log_n)$ algorithm, Demaine et al. (2009)'s $O(n^3)$ algorithm or Pawlik and Augsten (2011)'s $O(n^3)$ algorithm. The additional time cost of $O(n^2)$ can be ignored since it is less than the time cost for any available TED algorithm.

2.2.1 Find a sequence of single operations

In order to find the sequence of edit operations that transforms one tree into another, such as the pair shown in Figure 1, the computation proceeds as follows: create a new matrix called δ_2, which has the same dimensions as the matrix δ which is used to store the forest costs during ZS-TED to store the sequence of edit operations as a list. In particular, when the values of δ are computed, the values of δ_2 are computed, by using the edit operation labels: "i" for an insertion, "d" for deletion, "x" for exchange and "m" for no operation (matching). So, the final edit sequence to transform T_1 into T_2 in Figure 1 is **dddmmiiimm**.

The final mapping between T_1 and T_2 is shown in Figure 1. For each mapping figure the insertion, deletion, matching and exchange operations are shown with single, double, single dashed and double dashed outline respectively. The matching nodes (or subtrees) are linked with dashed arrows.

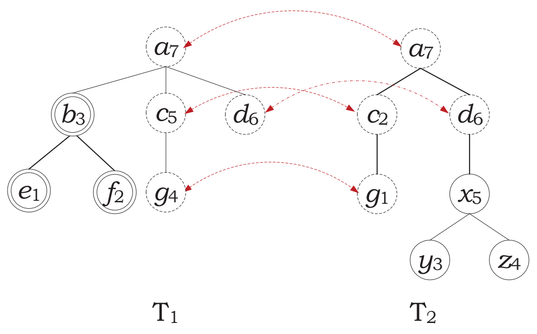

Figure 1: Standard TED, mapping between T_1 and T_2.

2.2.2 Find a sequence of subtree operations

Extending TED to cover subtree operations will give us more flexibility when comparing trees (especially linguistic trees). Thus, we have extended the TED algorithm to allow the standard edit operations (insert, delete and exchange) to apply both single nodes *and subtrees*.

Let $E_{p=1..L} \in$ {"d", "i", "x", "m"} be an edit operation sequence that transforms T_1 into T_2 by applying the technique in Section 2.2.1. Suppose that S^1 and S^2 are the optimal alignment for T_1 and T_2 respectively, when the length of $S^1 = S^2 = L$.

To find the optimal single and subtree edit operations sequence that transform T_1 into T_2, each largest sequence of same operation is checked to see whether it contains subtree(s) or not. Checking whether such a sequence corresponds to a subtree depends on the type of edit operation, according to the following rules: (i) if the operation is "d," the sequence is checked on the first tree; (ii) if the operation is "i," the sequence is checked on the second tree; and (iii) otherwise, the sequence is checked on both trees. After that, if the sequence of operations corresponds to a subtree, then all the symbols of the sequence are replaced by "+" except the last one (which represents the root of the subtree). Otherwise, checking starts from a new sequence as explained below. For instance, let us consider $E_h, ..., E_t$, where $1 \leq h < L$, $1 < t \leq L$, $h < t$, is a sequence of the *same* edit operation, i.e. $E_{k=h..t} \in$ {"d", "i", "x", "m"}. Let us consider $h0 = h$, we firstly check nodes $S^1_h, ..., S^1_t$ and $S^2_h, ..., S^2_t$ to see whether they or not are subtrees. If E_k is "*d*," the nodes $S^1_h, ..., S^1_t$ are checked, whereas the nodes $S^2_h, ..., S^2_t$ are checked when E_k is "*i*." Otherwise, the nodes $S^1_h, ..., S^1_t$ and $S^2_h, ..., S^2_t$ are checked. All edit operations $E_h, ..., E_{t-1}$ are replaced by "+" when this sequence is corresponding to a subtree. Then, we start checking from the beginning of another sequence from the left of the subtree $E_h, ..., E_t$, i.e. $t = h - 1$. Otherwise, the checking is applied with the sequence start from the next position, i.e. $h = h + 1$. The checking is continued until $h = t$. After that, when the $(t - h)$ sequences that start with different positions and end with t position do not contain a subtree, the checking starts from the beginning with the new sequence, i.e. $h = h0$ and $t = t - 1$. The process is repeated until $h = t$.

So, the final edit sequence to transform T_1 into T_2 in Figure 1 is **++d+m++imm**.

The final mapping between T_1 and T_2 according to the extended TED is shown in Figure 2.

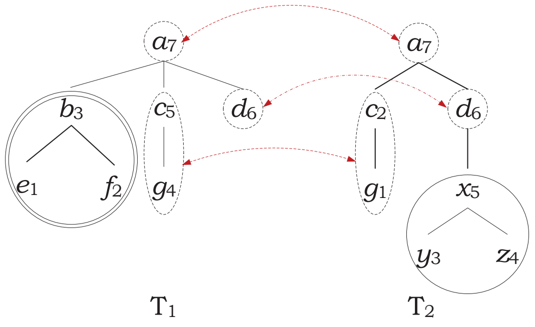

Figure 2: Extended TED with subtree operations, mapping between T_1 and T_2.

3 Optimisation algorithms

We used two optimisation algorithms, genetic algorithm (GA) and artificial bee colony (ABC), to estimate the cost of each edit operation (i.e. for single nodes and for subtrees) and threshold(s) based on application and type of system output.

3.1 GA

The GA starts with an initial population of solutions (known as chromosomes). In each generation, solutions from the current population are taken and used to form a new population by modifying the selected solutions' genome (recombined and possibly randomly mutated). This is motivated by a hope that the new population will be better than the old one. Solutions which are selected to form new solutions (offspring) are selected according to their fitness–the more suitable they are the more chances they have to reproduce. The algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population. The main steps of the algorithm are shown in Algorithm 1.

Algorithm 1 The basic algorithm for GA.

```
1: Initialise population;
2: repeat
3:     Evaluation;
4:     Reproduction;
5:     Crossover;
6:     Mutation;
7: until (termination conditions are met);
```

3.2 ABC algorithm

In the ABC algorithm, the colony of artificial bees consists of three groups. First, employed bees going to the food source (a possible solution to the problem to be optimised) that they have visited previously. Second, onlookers waiting to choose a food source. Third, scouts carrying out random search. The first half of the colony consists of the employed artificial bees and the second half includes the onlookers and scouts. The number of employed bees is equal to the number of food sources. The employed bee of an abandoned food source becomes a scout. The main steps of the algorithm are shown in Algorithm 2.

ABC follows three steps during each cycle: (i) moving both the employed and onlooker bees onto the food sources; (ii) calculating their nectar amounts (fitness value); and (iii) determining the scout bees and then moving them randomly onto the possible food sources.

The ABC algorithm has been widely used in many optimisation applications, since it is easy to implement and has fewer control parameters.

4 Experimental results

To check the effectiveness of the extended TED with subtree operations, ETED, we used it to check the entailment between *T-H* Arabic pairs of text snippets and compared its results with a simple bag-of-words, Levenshtein distance and ZS-TED on the same set of pairs.

4.1 Systems

We have investigated different approaches that can be divided into two groups as follow.

Algorithm 2 Pseudo-code of the ABC algorithm (Akay and Karaboga, 2012).

```
SN    size of population.
D     number of optimisation parameters.
$x_{ij}$   solution i,j, i = 1 ...SN, j = 1 ...D
1: Initialise the population of solutions $x_{i,j}$, $i$ = 1...SN, $j$ = 1...D, $trial_i$ = 0;
2: Evaluate the population;
3: cycle = 1;
4: repeat
5:     Produce new solutions $v_{ij}$ for the employed bees (using $v_{ij} = x_{ij} + \phi_{ij}(x_{ij} - x_{kj})$, where $k \in \{1, ..., SN\}$ and $\phi_{ij}$ is a random number between [-1,1]) and evaluate them;
6:     Apply the greedy selection process for the employed bees (if the new solution $v_{ij}$ has an equal or better nectar (fitness) than the old source, it is replaced with the old one in the memory. Otherwise, the old one is retained in the memory);
7:     Calculate the probability values $p_i = fit_i / \sum_{i=1}^{SN} fit_i$ for the solutions $x_i$;
8:     Produce the new solutions $v_{ij}$ for the onlookers from the solutions $x_i$ selected depending on $p_i$ and evaluate them;
9:     Apply the greedy selection process for the onlookers;
10:    Determine the abandoned solution for the scout, if exists, and replace it with a new randomly produced solution $x_i$ by $x_i^j = x_{min}^j$+rand[0,1]$(x_{max}^j - x_{min}^j)$;
11:    Memorise the best solution achieved so far;
12:    cycle = cycle+1;
13: until (cycle = Maximum Cycle Number);
```

Surface string similarity approaches

We tested the following approaches:

BoW: this approach uses the bag-of-words, which measures the similarity between *T* and *H* as a number of common words between them (either in surface forms or lemma forms), divided by the length of *H*, when the highest similarity is better.

LD$_1$: this approach uses the Levenshtein distance with 0.5, 1, 1.5 for cost of deleting, inserting and exchanging a word respectively.

LD$_2$: the same as for LD$_1$ except that the cost of exchanging non-identical words is the Levenshtein distance between the two words (with lower costs for vowels) divided by the length of the longer of the two words (derived and inflected forms of Arabic words tend to share the same consonants, at least in the root, so this provides a very approximate solution to the task of determining whether two forms correspond to the same lexical item).

Syntactic similarity approaches

These approaches follow three steps:

1. each sentence is preprocessed by a tagger and a parser in order to convert them to dependency trees, using a combination of taggers (i.e. AMIRA (Diab, 2009), MADA (Habash et al., 2009) and maximum-likelihood (MXL) tagger (Ramsay and Sabtan, 2009)) and parsers (i.e. MALTParser (Nivre et al., 2007) and MSTParser (McDonald et al., 2006)), which give around 85% for labelled accuracy (Alabbas and Ramsay, 2012; Alabbas and Ramsay, 2011), which is the best result we have seen for the Penn Arabic treebank (PATB). We use these combinations in series of experiments which involve;
2. pairs of dependency trees are matched using the ZS-TED/ETED to obtain a score for the pair;
3. either one threshold (for simple entails/fails-to-entail tests or two (for entails/unknown/fails-to-entail tests) are used to determine whether this score should lead to a particular judgement.

We tested the following approaches:

ZS-TED$_1$: this system uses ZS-TED with a manually determined set of fixed costs. The cost of deleting a node, inserting a node or exchanging a node are 0, 10 and 10 respectively.

ZS-TED$_2$: this system uses ZS-TED with a manually determined intuition-based set of costs that depend on a set of stopwords and on sets of synonyms and hypernyms, obtained from Arabic WordNet (AWN) (Black et al., 2006), as explained in Figure 3 (column A). These costs are an updated version of the costs used by Punyakanok et al. (2004).

ZS-TED+GA: this system uses a GA to estimate the costs of edit single operations and threshold(s) for ZS-TED. The chromosome for binary decision output is *{cost of deleting a node, cost of inserting a node, cost of exchanging a node, threshold}*, and the fitness is *a**f-score+*b**accuracy, where *a* and *b* are real numbers in the interval [0,1]. Providing different values for

a and b makes it possible to optimise the system for different applications–in the experiments below a is 0.6 and b is 0.4, which effectively puts more emphasis on precision than on recall, but for other tasks different values could be used. For three-way decisions, the chromosome is the same as for binary decisions except that we add a second threshold, and the fitness is simply the f-score. We used the steady state GA with the following settings: 40 chromosomes as population size, uniform crossover (UX), Gaussian mutation and maximum number of generations is 100.

ZS-TED+ABC: the same as ZS-TED+GA except using ABC instead of GA as the optimisation algorithm. We used the ABC algorithm with the following settings: 40 as the colony size and the maximum number of cycles for foraging is 100.

ETED$_1$: this system uses ETED with manually assigned costs. The costs for single nodes are the same for the ZS-TED$_1$ experiment and the costs for subtrees are half the sum of the costs of their parts.

ETED$_2$: this system uses ETED with the intuition-based costs for single nodes given in Figure 3 (column A) and the costs for subtrees given in Figure 3 (column B).

ETED+ABC: this system uses the ABC algorithm to estimate the costs of edit single operations and threshold(s) for ETED. For binary decision output, the chromosome is *{cost of deleting a node, cost of inserting a node, cost of exchanging a node, multiplier for the sum of the costs of the deletions in a deleted subtree, multiplier for the sum of the costs of the insertions in an inserted subtree, multiplier for the sum of the costs of the exchanges in an exchanged subtree, threshold}*. For three-way decisions the chromosome also contains the second threshold. For both cases the fitness is as for ZS-TED+GA. We do not include GA results for ETED, as extensive comparison of the standard GA algorithm and ABC on the ZS-TED experiments shows that ABC consistently produces better results for the same initial seeds and the same number of iterations.

The BoW algorithm and the basic string-edit algorithm are supplemented by the first two of the three procedures listed below and the others by all three, to ensure that we get the best possible performance at each stage:

- use AWN, OpenOffice Arabic dictionary and others as a lexical resource in order to take account of synonymy, antonym and hyponymy relations when comparing two words and when calculating the cost of an edit;
- take into consideration the POS tag when comparing two similar words (i.e. they should have the same POS tag);
- use a list of stopwords that contains some of the commonest Arabic words, which are treated specially when comparing words (e.g. by using different edit costs for them in distance-based approaches).

4.2 Results

We carried out experiments using the approaches above with two types of decisions as below.

Simple binary decision ('yes' and 'no'): *T* entails *H* when the cost of matching is less (more in case of bag-of-words) than a threshold. The results of this experiments, in terms of precision (P), recall (R) and f-score (F) for 'yes' class and accuracy (Acc.), are shown in Table 1. ETED shows a substantial improvement over bag-of-words and Levenshtein distance (around 19% in f-score and 6% in total accuracy) and over ZS-TED (around 2% in f-score and 2% in total accuracy).

Although we are primarily interested in Arabic, we have carried out parallel sets of experiments on the English RTE2 parsed testset,[1] using the Princeton WordNet (PWN) as a lexical resource, with the input text converted to dependency trees using Minipar (Lin, 1998). The pattern in Table 1 for English is similar to that for Arabic. ZS-TED is better than bag-of-words, ETED is a further improvement over ZS-TED.

Making a three-way decision ('yes,' 'unknown' and 'no' (not 'contradicts')): for this task we use two thresholds, one to trigger a positive answer if the cost of matching is lower than the lower threshold (exceeds the higher one for the bag-of-words algorithm) and the other to trigger a negative answer if the cost of matching exceeds the higher one (*mutatis mutandis* for bag-of-words). Otherwise, the result will be 'unknown.' The reason for making a three-way decision is to drive systems to make more precise distinctions. There is a difference between knowing that *H* does not

[1] `http://u.cs.biu.ac.il/~nlp/RTE2/Datasets/RTE-2\%20Preprocessed\%20Datasets.html`

Cost	(A) Single node	(B) Subtree (more than one node)
Delete	if X is a stop word =5, else =7	0
Insert	if Y is a stop word =5, else =100	double the sum of the costs of its parts
Exchange	if X subsumes Y =0, if X is a stop word =5, if Y subsumes or contradicts X=100 else =50	if a subtree S1 is identical to a subtree S2=0 else half the sum of the costs of its parts

Figure 3: Intuition-based edit operation costs for the systems ZS-TED2 and ETED1 (X in T, Y in H).

		Binary decision					**Three-way decision**		
Dataset	**Approach**	**P$_{yes}$**	**R$_{yes}$**	**F$_{yes}$**	**Acc.**	**F$_{yes}$× 0.6+ Acc.× 0.4**	**P**	**R**	**F**
ArbDS	BoW	63.6%	43.7%	0.518	59.3%	0.548	59.0 %	57.3%	0.581
	LD$_1$	64.7%	44%	0.524	60%	0.554	61.4%	58.0%	0.597
	LD$_2$	65%	47.7%	0.550	61%	0.574	62.9%	58.3 %	0.605
	ZS-TED$_1$	57.7%	64.7%	0.61	58.7%	0.601	64.3%	58.4%	0.612
	ZS-TED$_2$	61.6%	73.7%	0.671	63.8%	0.658	64.8%	58.3%	0.614
	ZS-TED+GA	59.2%	92%	0.721	64.3%	0.690	65.5 %	58.6 %	0.619
	ZS-TED+ABC	60.1%	91%	0.724	65.3%	0.696	67.8 %	58.2 %	0.626
	ETED$_1$	59%	65.7%	0.621	60%	0.613	65.3%	58.3%	0.616
	ETED$_2$	63.2%	75%	0.686	65.7%	0.674	66.7%	60%	0.632
	ETED+ABC	**61.5%**	**92.7%**	**0.739**	**67.3%**	**0.713**	**70.7%**	**62.4%**	**0.663**
RTE2	BoW	53.1%	49.9%	0.514	52.9%	0.520	50.8%	48.3%	0.495
	ZS-TED$_2$	52.9%	62.5%	0.573	53.5%	0.558	52.3%	50.2%	0.512
	ETED$_2$	54.2%	66.6%	0.598	55.2%	0.580	54.3%	52.7%	0.535
	ETED+ABC	**55.4%**	**70.1%**	**0.619**	**56.8%**	**0.599**	**55.7%**	**56.1%**	**0.559**

Table 1: Comparison between ETED, simple bag-of-words, Levenshtein distance and ZS-TED.

entail T and not knowing whether it does or not. Note that answering 'no' here means "I believe that H does not entail T", **not** "I believe that H contradicts T."

The results of this experiment, in terms of precision, recall and f-score for 'yes' class, are shown in Table 1. Again, ETED shows a worthwhile improvement bag-of-words and Levenshtein distance (around 6% in f-score) and over ZS-TED (around 4% in f-score).

5 Summary

We have described an extended version of tree edit distance (TED) that allows operations (i.e. delete, insert and exchange) both on single nodes and on subtrees. The extended TED with subtree operations, ETED, is more effective and flexible than the ZS-TED, especially for applications that pay attention to relations among nodes (e.g. in linguistic trees, deleting a modifier subtree should be cheaper than the sum of deleting its components individually).

We have also investigated the use of different optimisation algorithms, and have shown that using these produces better performance than setting the costs of edit operations by hand, and that using the ABC algorithm produces better results for the same amount of effort as traditional GAs.

The current findings, while preliminary, are quite encouraging. The fact that the results on our original testset, particularly the improvement in f-score, were replicated for a testset where we had no control over the parser that was used to produce dependency trees from the *T-H* pairs provides some evidence for the robustness of the approach. We anticipate that in both cases having a more accurate parser (our parser for Arabic attains around 85% accuracy on the PATB, Minipar is reported to attain about 80% on the Suzanne corpus) would improve the performance of both ZS-TED and ETED.

Acknowledgements

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

B. Akay and D. Karaboga. 2012. A modified artificial bee colony algorithm for real-parameter optimization. *Information Sciences*, 192(0):120 – 142.

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

M. Alabbas and A. Ramsay. 2012. Improved POS-tagging for Arabic by combining diverse taggers. In L. Iliadis, I. Maglogiannis, and H. Papadopoulos, editors, *Artificial Intelligence Applications and Innovations (AIAI)*, volume 381 of *IFIP Advances in Information and Communication Technology*, pages 107–116. Springer Berlin Heidelberg, Halkidiki, Thessaloniki, Greece.

M. Alabbas. 2011. ArbTE: Arabic textual entailment. In *Proceedings of the Second Student Research Workshop associated with RANLP 2011*, pages 48–53, Hissar, Bulgaria. RANLP 2011 Organising Committee.

M. Alabbas. 2013. A dataset for Arabic Textual Entailment. In *Proceedings of the Second Student Research Workshop associated with RANLP 2013*, Hissar, Bulgaria. RANLP 2013 Organising Committee.

R. Bar-Haim, I. Dagan, I. Greental, and E. Shnarch. 2007. Semantic inference at the lexical-syntactic level. In *Proceedings of 22nd AAAI Conference on Artificial Intelligence (AAAI-07)*, pages 871–876, Vancouver, British Columbia, Canada. The AAAI Press.

P. Bille. 2005. A survey on tree edit distance and related problems. *Theoretical Computer Science*, 337(1-3):217–239.

W. Black, S. Elkateb, H. Rodriguez, M. Alkhalifa, P. Vossen, A. Pease, and C. Fellbaum. 2006. Introducing the Arabic WordNet project. In *Proceedings of the 3rd International WordNet Conference (GWC-06)*, pages 295–299, Jeju Island, Korea.

I. Dagan, O. Glickman, and B. Magnini. 2005. The PASCAL recognising textual entailment challenge. In *Proceedings of the 1st PASCAL Recognising Textual Entailment Challenges*, pages 1–8, Southampton, UK.

E. Demaine, S. Mozes, B. Rossman, and O. Weimann. 2009. An optimal decomposition algorithm for tree edit distance. *ACM Transactions on Algorithms (TALG)*, 6(1):2:1–2:19.

M. Diab. 2009. 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*, pages 285–288, Cairo, Eygpt. The MEDAR Consortium.

N. Habash, O. Rambow, and R. Roth. 2009. 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, Eygpt. The MEDAR Consortium.

S. Harmeling. 2009. Inferring textual entailment with a probabilistically sound calculus. *Natural Language Engineering*, 15(4):459–477.

M. Heilman and N. Smith. 2010. Tree edit models for recognizing textual entailments, paraphrases, and answers to questions. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 1011–1019, Los Angeles, California, USA. Association for Computational Linguistics.

J. Herrera, A. Peñas, and F. Verdejo. 2005. Textual entailment recognition based on dependency analysis and WordNet. In *Proceedings of PASCAL Workshop on Recognizing Textual Entailment*, pages 21–24, Southampton, UK.

J. Herrera, A. Peñas, A. Rodrigo, and F. Verdejo. 2006. UNED at PASCAL RTE-2 challenge. In *Proceedings of the 2nd PASCAL Challenges Workshop on Recognising Textual Entailment*, pages 38–43, Venice, Italy.

P. Kilpeläinen. 1992. Tree matching problems with applications to structured text databases. Technical Report A-1992-6, Department of Computer Science, University of Helsinki, Helsinki, Finland.

P. Klein, S. Tirthapura, D. Sharvit, and B. Kimia. 2000. A tree-edit-distance algorithm for comparing simple, closed shapes. In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, pages 696–704. Society for Industrial and Applied Mathematics.

P. Klein. 1998. Computing the edit-distance between unrooted ordered trees. In *Proceedings of the 6th Annual European Symposium on Algorithms (ESA '98)*, pages 91–102, Venice, Italy. Springer-Verlag.

M. Kouylekov and B. Magnini. 2005. Recognizing textual entailment with tree edit distance algorithms. In *Proceedings of the 1st PASCAL Recognising Textual Entailment Challenge*, pages 17–20, Southampton, UK.

S. Kübler, R. McDonald, and J. Nivre. 2009. *Dependency Parsing*. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers.

D. Lin. 1998. Dependency-based evaluation of MINIPAR. In *LREC'98: Workshop on the Evaluation of Parsing systems*, pages 317–330, Granada, Spain.

E. Marsi, E. Krahmer, W. Bosma, and M. Theune. 2006. Normalized alignment of dependency trees for detecting textual entailment. In *Proceedings of the 2nd PASCAL Challenges Workshop on Recognising Textual Entailment*, pages 56–61, Venice, Italy.

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

Y. Mehdad and B. Magnini. 2009. Optimizing textual entailment recognition using particle swarm optimization. In *Proceedings of the 2009 Workshop on Applied Textual Inference (TextInfer '09)*, pages 36–43, Suntec, Singapore. Association for Computational Linguistics.

A. Meyers, R. Yangarber, and R. Grishman. 1996. Alignment of shared forests for bilingual corpora. In *Proceedings of the 16th Conference on Computational Linguistics*, volume 1, pages 460–465, Copenhagen, Denmark. Association for Computational Linguistics.

J. Nivre, J. Hall, J. Nilsson, A. Chanev, G. Eryigit, S. Kübler, S. Marinov, and E. Marsi. 2007. MaltParser: a language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(02):95–135.

M. Pawlik and N. Augsten. 2011. RTED: a robust algorithm for the tree edit distance. *Proceedings of the VLDB Endowment*, 5(4):334–345.

V. Punyakanok, D. Roth, and W. Yih. 2004. Natural language inference via dependency tree mapping: An application to question answering. *Computational Linguistics*, 6:1–10.

A. Ramsay and Y. Sabtan. 2009. Bootstrapping a lexicon-free tagger for Arabic. In *Proceedings of the 9th Conference on Language Engineering (ESOLEC'2009)*, pages 202–215, Cairo, Egypt.

J. Russell and R. Cohn. 2012. *Particle Swarm Optimization*. Book on Demand Ltd.

A. Stern and I. Dagan. 2011. A confidence model for syntactically-motivated entailment proofs. In *Proceedings of Recent Advances in Natural Language Processing (RANLP 2011)*, pages 455–462, Hissar, Bulgaria. RANLP 2011 Organising Committee.

A. Stern, R. Stern, I. Dagan, and A. Felner. 2012. Efficient search for transformation-based inference. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics*, volume 1, Jeju Island, Korea. The Association for Computer Linguistics.

M. Wang and C. Manning. 2010. Probabilistic tree-edit models with structured latent variables for textual entailment and question answering. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, pages 1164–1172, Beijing, China. Association for Computational Linguistics, Tsinghua University Press.

K. Zhang and D. Shasha. 1989. Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal of Computing*, 18(6):1245–1262.