Proceedings of the Federated Conference on
Computer Science and Information Systems pp. 11–18

ISBN 978-83-60810-51-4

Dependency Tree Matching with Extended Tree Edit Distance with Subtrees for Textual Entailment

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

***Abstract*—A lot of natural language processing (NLP) applications require the computation of similarities between pairs of syntactic or semantic trees. Tree edit distance (TED), in this context, is considered to be one of the most effective techniques. However, its main drawback is that it deals with single node operations only. We therefore extended TED to deal with subtree transformation operations as well as single nodes. This makes the extended TED with subtree operations more effective and flexible than the standard 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). The preliminary results of extended TED with subtree operations were encouraging compared with the standard one when tested on different examples of dependency trees.**

I. Introduction

TREE edit distance has been been widely used as a component of NLP systems that attempt to determine whether one sentence entails another, with the distance between pairs of dependency trees being taken as a measure of the likelihood that one entails the other. We extend the standard algorithm for calculating the distance between two trees by allowing operations to apply to subtrees, rather than just to single nodes. This extension improves the performance of a textual entailment system for Arabic by around 5%.

Tree edit distance is a generalization of the edit distance for two strings, which measures the similarity between two strings. Tree edit distance (TED) has offered different solutions for several NLP applications such as information extraction, information retrieval and textual entailment. TED between two trees is defined as the minimum cost set of edit operations to transform one tree to another. There have been numerous approaches to calculate edit distance between trees (e.g. [1], [2], [3], [4]). The most popular of these is Zhang-Shasha's algorithm [3].

Our ultimate goal is to develop a *textual entailment* (TE) system for Arabic [5]. Modern standard Arabic (MSA) is the Arabic language version which we are concerned with in the current work. When we refer to Arabic throughout this paper, we mean MSA. Dagan et al. [6] describe *recognising textual entailment* (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 the authors, entailment holds if the truth of H, as interpreted by a typical language user, can be inferred from the meaning of T. One efficient technique that has been used in recent years to check entailment between two sentences is by using Zhang-Shasha's TED method to match dependency trees for both sentences [7], [8]. Approximate tree matching allows users to match a tree with solely some parts of another tree not a whole. However, one of the main drawbacks of TED is that transformation operations are applied solely on single nodes [7]. Kouylekov and Magnini [9] used the standard TED, which uses transformation operations (insert, delete and change) solely on single nodes, to check the entailment between two dependency trees. On the other hand, Heilman and Smith [8] extended the available operations in standard TED to `INSERT-CHILD`, `INSERT-PARENT`, `DELETE-LEAF`, `DELETE-&-MERGE`, `RELABEL-NODE` and `RELABEL-EDGE`. These authors also identify three new operations, `MOVE-SUBTREE`, which means move a node X in a tree T to be the last child on the left/right side of a node Y in T (s.t. Y is not a descendant of X), `NEW-ROOT` and `MOVE-SIBLING`, to enable succinct edit sequences for complex transformation. This extended set of edit operations allows certain combinations of the basic operations to be treated as single steps, and hence provides shorter (and therefore cheaper) derivations. The fine-grained distinctions between, for instance, different kinds of insertions also make it possible to assign different weights to different variations on the same operation. Nonetheless, these operations continue to operate on individual nodes rather than on subtrees (despite its name, even `MOVE-SUBTREE` appears to be defined as an operation on nodes rather than on subtrees). We have solved this problem by extending the basic version of the TED algorithm so that operations that insert/delete/exchange subtrees cost less than the sum of the costs of inserting/deleting/exchanging their parts. This makes the extended TED with subtree operations more effective and flexible than the standard one, especially for applications that pay attention to relations among nodes (e.g deleting a modifier subtree, in linguistic trees, should be

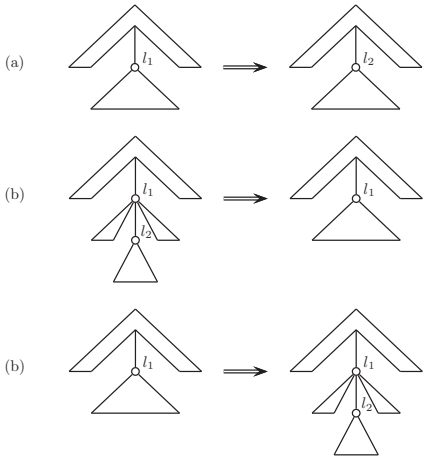

Fig. 1. (a) A relabeling of the node label ($l_1 \rightarrow l_2$). (b) Deleting the node labeled ($l_2 \rightarrow \wedge$). (c) Inserting a node labeled l_2 as the child of the node labeled l_1 ($\wedge \rightarrow l_2$) [10].

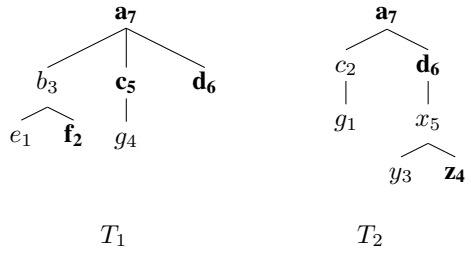

Fig. 2. Two trees T_1 and T_2 with their postorder traversal (the subscript for each node is considered the order of this node in the postorder of the tree). The postorder of T_1 is *e,f,b,g,c,d,a* and the postorder for T_2 is *g,c,y,z,x,d,a*. The leftmost leaf descendant of the subtrees of T_1 headed by the nodes *e,f,b,g,c,d,a* are 1,2,1,4,4,6,1 respectively, and similarly the leftmost leaf descendant of *g,c,y,z,x,d,a* in T_2 are 1,1,3,4,3,3,1. The bold items in each tree represent the keyroot items for this tree.

cheaper than the sum of deleting its components individually).

The rest of the paper is organised as follows: the Zhang-Shasha's TED is explained in Section II. Section III presents the extended TED with subtree operations. Section IV describes dependency trees matching. The experimental results are discussed in Section V. Conclusions are given in Section VI.

II. Zhang-Shasha's TED

Zhang-Shasha's TED is considered an efficient technique based on dynamic programming to calculate the approximate tree matching for two rooted ordered trees. Ordered trees are trees in which the left-to-right order among siblings is significant. There are three operations, namely deleting, inserting and changing a node, which can transform one ordered tree to another. Deleting a node *x* means attaching its children to the parent of *x*. Insertion is the inverse of deletion. This means an inserted node becomes a parent of a consecutive subsequence in the left to right order of its parent. Changing a node alters its label. All these editing operations are illustrated in Fig. 1.

Each operation is associated with a cost and is allowed on single nodes only. Selecting a good set of costs for these operations 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 [11].

In the TED algorithm, tree nodes are compared using a postorder traversal, which visits the nodes of a tree starting with the leftmost leaf descendant of the root and proceeding to the leftmost descendant of the right sibling of that leaf, the right siblings, and then the parent of the leaf and so on up the tree to the root. The last node visited will always be the root. An example of the postorder traversal and the leftmost leaf descendant of a tree is shown in Fig. 2.

For all the descendants of each node, the least cost mapping has to be calculated before the node is encountered, in order that the least cost mapping can be selected right away. To achieve this, the algorithm pursues the keyroots of the tree, which are defined as a set that contains the root of the tree plus all nodes having a left sibling. The number of keyroots must be equal to the number of leaves in the tree. So, the keyroots of the trees T_1 and T_2 in the Fig. 2 are equal to $\{2, 5, 6, 7\}$ and $\{4, 6, 7\}$ respectively. The keyroots of a tree are decided in advance, permitting the algorithm to distinguish between *tree distance* (the distance between two nodes when considered in the context of their left siblings in the trees T_1 and T_2) and *forest distance* (the distance between two nodes considered separately from their siblings and ancestors but not from their descendants)[7].

For each node, the computation to find out the least cost mapping (the tree distance) between a node in the first tree and one in the second depends solely on mapping the nodes and their children. To find the least cost mapping of a node, then, one needs to recognize the least cost mapping from all the keyroots among its children, plus the cost of its leftmost child. Because the postorder way is used in the nodes numbering, the algorithm proceeds in the following steps [7]: (i) the mappings from all leaf keyroots are determined; (ii) the mappings for all keyroot at the next higher level are decided recursively; and (iii) the root mapping is found. Algorithm 1 shows the pseudo code of TED algorithm.

III. Extended TED with Subtree Operations

The main weakness of the TED algorithm is that it is not able to do transformations on subtrees (i.e. delete subtree, insert subtree and change subtree). The output of the standard TED algorithm is the distance between the two trees only. In order to make the TED deal with subtree operations, we need first to extend it to find a sequence of single edit operations that transforms the first tree into the other with minimal cost, because it measures the distance between two trees only. Then, according to this sequence we can decide if there are subtree operations or not, as explained below.

A. Find a sequence of edit operations

In order to find the sequence of edit operations that transforms the first tree into another, the computation proceeds as

Algorithm 1 Pseudo code of Zhang-Shasha's TED algorithm [3]

T[i]	the i_{th} node of T, labeled in postorder
l(i)	the leftmost leaf descendant of the subtree rooted at i
K(T)	the keyroots of tree T, K(T) = {k $\in$T|$\neg\exists$ k_1 >k with l(k_1) = l(k)}
Ø	a null tree
$FD[T_1[i,i_1],T_2[j,j_1]]$	the forest distance from nodes i to i_1 in T_1 to nodes j to j_1 in T_2, if i< i_1 then $T_1[i,i_1]$ = Ø (temporary array).
D[i,j]	the tree distance between two nodes $T_1[i]$ and $T_2[j]$ (permanent array) .
$\gamma(T_1[i] \longrightarrow \wedge)$	delete the i_{th} node from T_1
$\gamma(\wedge \longrightarrow T_2[j])$	insert the j_{th} node of T_2 into T_1
$\gamma(T_1[i] \longrightarrow T_2[j])$	change the i_{th} node of T_1 with the j_{th} node of T_2
n and m	the number of nodes in T_1 and T_2 respectively
$\|X\|$	the length of X
min	function return minimum item among three items.

1: **compute** $l_1(n), l_2(m), K_1(T_1), K_2(T_2)$
2: **for** $x \leftarrow 1$ **to** $|K_1(T_1)|$ **do**
3: **for** $y \leftarrow 1$ **to** $|K_2(T_2)|$ **do**
4: $FD[,] \leftarrow 0$
5: **for** $i \leftarrow l_1(x)$ **to** x **do**
6: $FD[T_1[l_1(x),i],] \leftarrow FD[T_1[l_1(x),i-1],] + \gamma(T_1[i] \longrightarrow \wedge)$
7: **end for**
8: **for** $j \leftarrow l_2(y)$ **to** y **do**
9: $FD[,T_2[l_2(y),j]] \leftarrow FD[,T_2[l_2(y),y-1]] + \gamma(\wedge \longrightarrow T_2[j])$
10: **end for**
11: **for** $i \leftarrow l_1(x)$ **to** x **do**
12: **for** $j \leftarrow l_2(y)$ **to** y **do**
13: **if** $(l_1(i) == l_1(x)$ **and** $l_2(j) == l_2(y))$ **then**
14: $FD[T_1[l1(x),i],T_2[l_1(y),j]] \leftarrow min($
15: $FD[T_1[l_1(x),i-1],T_2[l_2(y),j]] + \gamma(T_1[i] \longrightarrow \wedge),$
16: $FD[T_1[l_1(x),i],T_2[l_2(y),j-1]] + \gamma(\wedge \longrightarrow T_2[j]),$
17: $FD[T_1[l_1(x),i-1],T_2[l_2(y),j-1]] + \gamma(T_1[i] \longrightarrow T_2[j]))$
18: $D[i,j] \leftarrow FD[T_1[l_1(x),i],T_2[l_2(y),j]]$
19: **else**
20: $FD[T_1[l_1(x),i],T_2[l_1(y),j]] \leftarrow min($
21: $FD[T_1[l_1(x),i-1],T_2[l_2(y),j]] + \gamma(T_1[i] \longrightarrow \wedge),$
22: $FD[T_1[l_1(x),i],T_2[l_2(y),j-1]] + \gamma(\wedge \longrightarrow T_2[j]),$
23: $FD[T_1[l_1(x),i-1],T_2[l_2(y),j-1]] + D[i,j])$
24: **end if**
25: **end for**
26: **end for**
27: **end for**
28: **end for**
29: **return** $D[n,m]$

follows: create new matrices called PATH matrix, which has the same dimensions of FD matrix, and DPATH matrix, which has the same dimensions of D matrix, to store the sequence of edit operations as a list. In particular, when the values of FD and D in Algorithm 1 are computed, the values of PATH and DPATH are computed, by using the edit operation labels: "i" for an insertion, "d" for deletion, "x" for changing and "m" for no operation (matching), as explained in the following rules:

1) Set PATH[i,j] = PATH[i-1,j]+ "d," if FD[i,j] = FD[i-1,j] + delete cost. (i.e. lines 6,14 and 20 in Algorithm 1)
2) Set PATH[i,j] = PATH[i,j-1]+ "i," if FD[i,j] = FD[i,j-1] + insert cost. (i.e. lines 9,14 and 20 in Algorithm 1)
3) Set PATH[i,j] = PATH[i-1,j-1]+ "m" (when change cost=0) or "x" (otherwise), if FD[i,j] = FD[i-1,j-1] + change cost. (i.e. line 14 in Algorithm 1)
4) Set DPATH[i,j] = PATH[i,j], if D[i,j] = FD[i,j]. (i.e. line 18 in Algorithm 1)
5) Set PATH[i,j] = PATH[i-1,j-1]+ DPATH[i,j] (i.e. the path of D[i,j]), if FD[i,j] = FD[i-1,j-1] + D[i,j]. (i.e. line 20 in Algorithm 1)

The first and second rules are applied to cells in the first column and row respectively, while the first cell in the matrix is empty. Therefore, for most objective functions, each cell in the first row equals the cell to its left appended with insert operation "i" (rule 2), and each cell in the first column equals the cell just above it appended with delete operation "d" (rule 1). For the other cells, it is possible (and common) that FD[i,j] resulted from more than one of the previous three neighboring cells. In this case, one of the values of FD[i,j] is selected according to the priority of edit operations specified by the user. Fig. 3 shows the direction of the computed FD[i,j] cell. A horizontal arc specifies that the j_{th} node of the second tree is to be inserted (rule 2), while a vertical arc indicates that the i_{th} node of the first tree to be deleted (rule 1). A diagonal arc specifies that the j_{th} node in the second tree is to be changed with the i_{th} node of the first tree if they are not equal, otherwise no operation is performed (rule 3 except rule 5, which it is append the value of DPATH[i,j] in rule 4 instead of either "x" or "m"). After the PATH matrix becomes full, the optimal path is the last cell (at final row and column) in PATH matrix.

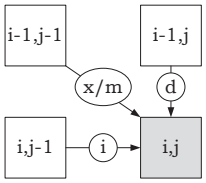

Fig. 3. The edit operation direction used in our algorithm. Each arc that implies an edit operation is labeled: "i" for an insertion, "d" for deletion, "x" for changing and "m" for no operation (matching).

Fig. 4 illustrates the intuition of how to compute this optimal path for T_1 and T_2 trees in Fig. 2. In this figure, the cells representing concordance of the optimal sequence of edit operations that transform T_1 into T_2 are highlighted in bold, whereas the final optimal path is the last cell (at final row and column).

The mapping between two trees can be found from the final sequence of edit operations by mapping the nodes corresponding to match operation "m" only.

The final distance is 6 which represents the final values (at final row and column) in the FD matrix.[1] The last value in the PATH matrix represents the final sequence of edit operations, which is: **dddmmiiimm**. According to this path, we can define an *alignment* between two postorder trees. The alignment between two trees T_1 and T_2 is obtained by inserting a *gap symbol* (i.e. "_") into either T_1 or T_2, according to the type of edit operation, so that the resulting strings S^1 and S^2 are the same length as the sequence of edit operations. The gap symbol is inserted into S^2 when the edit operation is delete ("d"), whereas it is inserted in S^1 when the edit operation is insert ("i"). Otherwise, the node of T_1 and T_2 are inserted into S^1 and S^2 respectively. The following is an optimal alignment between T_1 and T_2:

S^1:	*e*	*f*	*b*	*g*	*c*	_	_	_	*d*	*a*
	d	**d**	**d**	**m**	**m**	**i**	**i**	**i**	**m**	**m**
S^2:	_	_	_	*g*	*c*	*y*	*z*	*x*	*d*	*a*

This means,

d: Delete (e) from T_1
d: Delete (*f*) from T_1
d: Delete (*b*) from T_1
m: Leave (*g*) without change
m: Leave (*c*) without change
i: Insert (*y*) into T_1
i: Insert (*z*) into T_1
i: Insert (*x*) into T_1
m: Leave (*d*) without change
m: Leave (*a*) without change

The final mapping between T_1 and T_2 is shown in Fig. 5. For each mapping figure the insertion, deletion, matching and changing operations are shown with single, double, single dashed and double dashed outline respectively. The matching nodes (or subtrees) are linked with dashed arrows.

[1]Here, the cost of each single operation is considered 1 except matching is equal to 0.

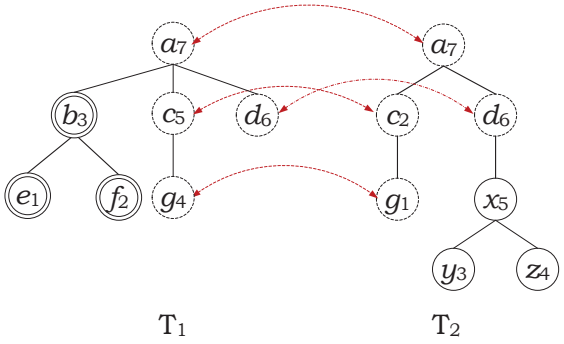

Fig. 5. Standard TED, mapping between T_1 and T_2.

B. Find a sequence of subtree edit 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 change) to apply both single nodes *and subtrees*.

All largest subtrees for each *same* sequence of edit operations are checked if they are already subtree(s). Each sequence of nodes in postorder is considered a subtree, which is itself a tree, if the following conditions are satisfied: (i) the first node is a leaf; and (ii) the leftmost of the last node in the sequence (i.e. the root of a subtree) is the same as the first node in the sequence. Therefore, the sequence of the nodes *e,f,b* in tree T_1 in Fig. 2 is a subtree because *e* is a leaf and the leftmost of the last node *b* is 1, which represents the first node *e*. On the other hand, the sequence of nodes *g,c,d* in the same tree is not a subtree because *g* is a leaf, but the leftmost of the last node *d* is 6, which represents itself, not the first node *g*.

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 III-A. 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 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 $h0 = h$,

	T_2	*g*	*c*	*y*	*z*	*x*	*d*	*a*
T_1	-	i	ii	iii	iiii	iiiii	iiiiii	iiiiiii
e	**d**	x	xi	iix	iiix	iiixi	iiixii	iiixiii
f	**dd**	xd	xid	xix	iixx	xiiix	xiiiix	xiiiixi
b	**ddd**	xdd	xdx	xdxi	iixxd	iixxx	iixxxi	iixxxii
g	dddd	**dddm**	dddmi	xdxx	xdxxi	xdxixi	xdxixii	iixxxiid
c	ddddd	dddmd	**dddmm**	**dddmmi**	**dddmmii**	**dddmmiii**	xdxixix	xdxixixi
d	dddddd	dddmdd	dddmmd	dddmmx	dddmmxi	dddmmxii	**dddmmiiim**	dddmmiiimi
a	ddddddd	dddmddd	dddmmdd	dddmmxd	dddmmxid	dddmmxiid	dddmmiiimd	**dddmmiiimm**

PATH matrix

Fig. 4. Compute the optimal path for the trees in Fig. 2.

we firstly check nodes $S_h^1, ..., S_t^1$ and $S_h^2, ..., S_t^2$ if they are subtree or not. In case of E_k is *"d,"* the nodes $S_h^1, ..., S_t^1$ are checked, whereas the nodes $S_h^2, ..., S_t^2$ are checked when E_k is *"i."* Otherwise, the nodes $S_h^1, ..., S_t^1$ and $S_h^2, ..., S_t^2$ 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 $h = t$.

The cost of a subtree edit operation is taken to be half the sum of the costs of its parts[2] (i.e. subtree deletion is cheaper than individual nodes deletion), whereas the cost of change identical subtree is equal to 0.

To explain how the subtree operations are applied, let us consider the two trees T_1 and T_2 in Fig. 2.

According to extended TED with subtree operations, the cost is 4 and the sequence of operation is as follows: there is a sequence of "**d,**" "**m**" and "**i**" in the result. These sequences consist of three subtrees (i.e. the three deleted nodes, the first two matched nodes and the three inserted nodes): **ddd mm iii mm**. So, the final result is: **++d +m ++i mm**. This means:

++d: Delete subtree (*e,f,b*) from T_1
+m: Leave subtree (*g,c*) without change
++i: Insert subtree (*y,z,x*) into T_1
m: Leave (*d*) without change
m: Leave (*a*) without change

The final mapping between T_1 and T_2 is shown in Fig. 6, the extended TED with subtree operations.

IV. Dependency Trees Matching

As we mentioned before, our main goal is to check entailment between a pair of Arabic sentences (i.e. text and hypothesis) using TED algorithm. To match text:hypothesis dependency tree pairs effectively, we use the extended TED with subtree operations. It enables us to find the minimum edit operations to transform one tree to another. Also, it allows us

[2]These costs are changed to match the requirements of specific applications.

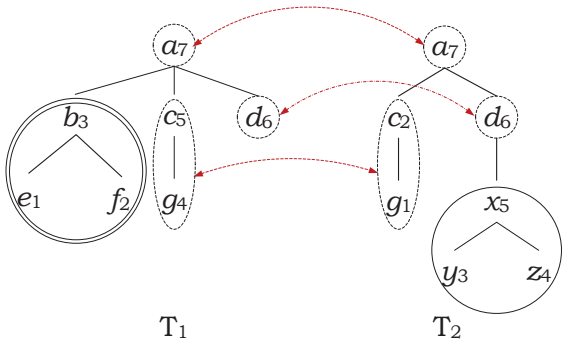

Fig. 6. Extended TED with subtree operations, mapping between T_1 and T_2.

to be sensitive to the fact that the links in a dependency tree carry linguistic information about relations between complex units, and hence to ensure that when we compare two trees we are paying attention to these relations. For instance, this enables us to be sensitive to the fact that operations involving modifiers, in particular, should be applied to the subtree as a whole rather than to its individual elements. Thus, we transform tree D_1 to tree D_2 in Fig. 7 by deleting "in the park" in a single operation, removing the modifier as a whole, rather than three operations removing "in," "the" and "park" one by one. We have applied the current technique to mapping different dependency trees by using the costs in Fig. 8 as initial test for edit operations in our experiments. These costs are an updating version for the costs that used by Punyakanok et al. [12]. These authors found that using TED gives better results than bag-of-word scoring methods, when they applied them for question answering task. The stop word here is a list that contains some of most common Arabic words (e.g. the particle إن *Ăn* "indeed").[3] For instance, إن المدير مشغول *Ăn Almdyr mšγwl* "The director is **indeed** busy" entails المدير مشغول *Almdyr mšγwl* "The director is busy." Dependency trees have also relations between each two nodes (e.g. SUBJ, OBJ, MOD, etc.), whereas TED does not consider them in its processing. We solved this problem by embedding a dependency relation with its connected node (i.e. node(POS-tag,relation)).

By using the costs in Fig. 8, the cost of transfering D_1 into D_2 according to the standard TED is 19 (i.e. one stop

[3]The transcription of Arabic examples in this document follows Habash-Soudi-Buckwalter (HSB) transliteration scheme [13] for transcribing Arabic symbols.

Cost	Single node	Subtree
Delete:	if a node X is a stop word =5, else =7	half the sum of the costs of its parts
Insert:	if a node Y is a stop word =5, else =100	half the sum of the costs of its parts
Change:	if a node X is subsume of a node Y=0, if X is a stop word =5, else =100	if a subtree S1 is identical to a subtree S2=0 else half the sum of the costs of its parts

Fig. 8. Edit operation costs.

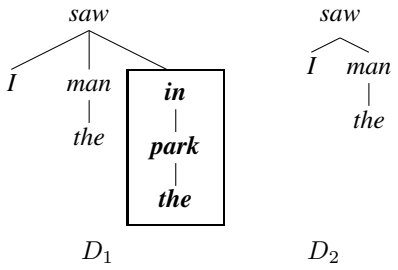

Fig. 7. Two dependency trees, D_1 and D_2.

word "the" (5) and two words (14)), whereas according to the extended TED with subtree operations it is 10. Therefore, it is easy to decide that D_1 entails D_2, whereas the reverse is not. We also exploited the subset/superset relations encoded by Arabic WordNet (AWN) [14] when comparing items in a tree. Roughly speaking, if comparing one tree to another requires us to swap two lexical items, we will be happier doing so if the item in the source tree is a hyponym of the one in the target tree. Doing this will allow us to delay making decisions about potentially ambiguous lexical items: it is reasonably safe to assume that if W_1 has a sense which is a hyponym of some sense of W_2 then a sentence involving W_1 will entail a similar sentence involving W_2 as shown in (1) (since "cat" is hyponym of "animal," (1a) entails (1b), whereas the reverse must be not). This will definitely be quicker, and may be more reliable, than trying to disambiguate the two from first principles and then looking for entailment relationships.

(1) a. I saw a *cat*.
b. I saw an *animal*.

This reflects the widely accepted view that contextual information is the key to lexical disambiguation. Within the RTE task, the text provides the context for disambiguation of the hypothesis, and the hypothesis provides the context for disambiguation of the text. Almost any human reader would, for instance, accept that (2a) entails (2b), despite the potential ambiguity of the word "bank."

(2) a. My money is all tied up at the bank.
b. I cannot easily spend my money.

TABLE I
TESTING DATASET TEXT'S RANGE ANNOTATION.

Text length		Entails	Not entails
<20	90	51	39
20-29	226	118	108
30-39	86	55	31
>39	9	6	3
Total:	**411**	**225**	**186**

V. EXPERIMENTS

In order to check the effectiveness of the extended TED with subtree operations, we used it to check the entailment between T-H Arabic pairs of sentences and compared its results with standard TED on the same set of pairs. Checking whether if one Arabic sentence entails another, however, is particularly challenging because Arabic is more ambiguous than most languages, such as English. For instance, Arabic is written without diacritics (short vowels), often leading to multiple ambiguities. This matter makes the morphological analysis very difficult (i.e. a single written form corresponding to as many as ten different lexemes [15], [16], [17]). The preliminary testing dataset contains 411 pairs, annotated as 'Entails' and 'Notentails.' This dataset was collected automatically using the 'headline-lead paragraph' technique [18] from newspaper websites via Google-API, pairing the first paragraph of article (as text) with its headline (as hypothesis). This is based on the observation that a news article's headline is very often a partial paraphrase of the first paragraph of this article, conveying thus a comparable meaning. The annotation is performed manually by eight expert and nonexpert human annotators to identify the different pairs as positive 'Entails' or negative 'Notentails' in our dataset. Each pair was annotated by three inter-annotators and the results is the overall agreement among them. The distribution of these pairs over the text length is summarised in Table I, when the hypothesis average length is around 10 words and the average of common words between text and hypothesis is around 4 words. The average length of sentence in this dataset is 27 words per sentence, with some sentences containing 40+ words. The inter-annotator agreement (where all annotators agree) is around 74% compared with 89% where each annotator agrees with at least one co-annotator. This suggests that the annotators found this is difficult task.

In order to check the entailment between T-H pairs, we follow three steps:

- Representing both T-H pair to dependency trees. Dependency tree is a tree where words are vertices and

syntactic relations are dependency relations. Each vertex therefore has a single parent, except the root of the tree. A dependency relation holds between *dependent*, i.e. a syntactically subordinate vertex, and *head*, i.e. another vertex which it dependent. So, the dependency structure represents as head-dependent relation between vertices that are classified by dependency types such as `SBJ` "subject," `OBJ` "object," `ATT` "attribute," etc.
We have carried out a number of experiments with state-of-the-art taggers (i.e. AMIRA [19], MADA [20] and an in-house maximum-likelihood (MXL) tagger [21]) and parsers (i.e. MALTParser [22] and MSTParser [23]).[4] These experiments show in particular that merging MADA with MSTParser gives better result (around 80%) than the other merging tagger:parser [26]. We therefore use MADA+MSTParser in the current experiments.

- Using standard TED or extended TED with subtree operations with edit operations costs in Fig. 8 to find the cost of matching between T-H pair. In this step, we use AWN as a lexical resource in order to take account of synonymy and hyponymy relations when calculating the cost of an edit.
- A text entails a hypothesis when the cost of matching is less than some specific threshold.

The extended TED with subtree operations gives better results (around a 5% overall increase in accuracy) than the standard TED. For instance, in (3) if there are "Fifty thousand tourists visited Lebanon and Syria last September," then (3a) entails (3b). The reason is that if "Fifty thousand tourists visited Lebanon and Syria last September" is true, then there is no way to avoid the conclusion that "Fifty thousand tourists visited Lebanon and Syria" as a matter of fact, because (3b) did not specify the time of visiting. Hence deleting this subtree as a whole removes a single piece of information. The reverse does not hold (i.e. (3b) does not entail (3a)) because if "Fifty thousand tourists visited Lebanon and Syria" is true, then we cannot conclude that the visiting happened in "last September." In general, the first sentence, the entailing expression, is more informative than (or sometime a paraphrase of) the second one, what is entailed, because the information that the second sentence carries is included in the information that the first sentence carries. For this reason, we define the cost of deleting to be cheaper than the cost of inserting.

[4]These parser are data-driven dependency parsers. For Arabic they are usually trained on an Arabic dependency treebank, such as Prague Arabic Dependency Treebank (PADT) [24], or on some version of the Penn Arabic Treebank (PATB) [25] that has been converted to dependency trees: scoring of such parsers is a matter of counting dependency links.

(3) a. خمسون الف سائح زاروا لبنان وسوريا في ايلول الماضي
xmswn Alf sA'H zArwA lbnAn w+swryA fy Aylwl AlmaDy
"Fifty thousand tourists visited Lebanon and Syria last September"

b. خمسون الف سائح زاروا لبنان وسوريا
xmswn Alf sA'H zArwA lbnAn w+swryA
"Fifty thousand tourists visited Lebanon and Syria"

VI. Conclusion

We have presented here an extended version of tree edit distance (TED) that solved one of the main drawbacks of standard TED, which is that it only supports transformation operations (i.e. delete, insert and change) on single nodes. The extended TED deals with subtree transformation operations as well as single ones. It has the advantage of being efficient and more appropriate to find the minimum edit operations to transform one tree to another compared with standard TED.

Here, the lead-paragraph in the Arabic articles that are returned by this process typically contain very long sentences (upwards of 100 words), where only a small part has a direct relationship to the headline. With very long sentences of this kind, it commonly happens that only the part of lead-paragraph is relevant to headline. This is typical of Arabic text, which is often written with very little punctuation, with elements of the text linked by conjunctions rather than being broken into implicit segments by punctuation marks such as full stops and question marks. For instance, the lead-paragraph has the form $\texttt{CONJ}\ S_1\ \texttt{CONJ}\ S_2\ \texttt{CONJ}\ \ldots\ \texttt{CONJ}\ S_n$ entails the headline, S_i, i=1..n. Thus the extended TED with subtree operations surpasses the standard TED in such type of sentences.

The current findings, while preliminary, are quite encouraging. We are currently experimenting with extended TED with subtree operations within a textual entailment system for Arabic with different costs and more sentences pairs.

We plan to use other Arabic lexical resources, such as OpenOffice Arabic dictionary and MS Word Arabic dictionary, to provide us with more information about relations between words, because the information in AWN, while very useful, is sparse in comparison to the Princeton WordNet (PWN), i.e. English WordNet [27].

Acknowledgments

We would like to thank anonymous reviewers for their valuable comments. We would like to extend our thanks to our volunteers' annotator for their time and effort they have annotating our experimental dataset. 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).

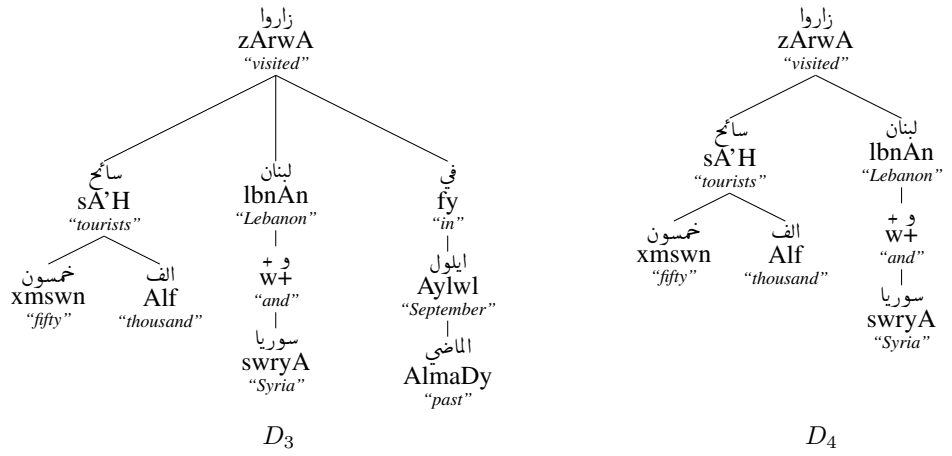

Fig. 9. Arabic dependency trees, D_3 and D_4 for sentences (3b) and (3a) respectively.

REFERENCES

[1] S. Selkow, "The tree-to-tree editing problem," *Information processing letters,* vol. 6, no. 6, 1977, pp. 184–186.

[2] K. Tai, "The tree-to-tree correction problem," *Journal of the ACM (JACM),* vol. 26, no. 3, 1979, pp. 422–433.

[3] K. Zhang and D. Shasha, "Simple fast algorithms for the editing distance between trees and related problems," *SIAM Journal of Computing,* vol. 18, no. 6, 1989, pp. 1245–1262.

[4] P. Klein, S. Tirthapura, D. Sharvit, and B. Kimia, "A tree-edit-distance algorithm for comparing simple, closed shapes," *in Proceedings of the 11th annual ACM-SIAM symposium on Discrete algorithms.* Society for Industrial and Applied Mathematics, 2000, pp. 696–704.

[5] 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.

[6] I. Dagan and O. Glickman, "Probabilistic textual entailment: generic applied modeling of language variability," *Learning Methods for Text Understanding and Mining,* 2004, pp. 26–29.

[7] M. Kouylekov, "Recognizing textual entailment with tree edit distance: Application to question answering and information extraction," PhD dissertation, DIT-University of Trento, 2006.

[8] M. Heilman and N. Smith, "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.* Association for Computational Linguistics, 2010, pp. 1011–1019.

[9] M. Kouylekov and B. Magnini, "Recognizing textual entailment with tree edit distance algorithms," *in Proceedings of the 1st Challenge Workshop Recognising Textual Entailment,* Southampton, UK, 2005, pp. 17–20.

[10] P. Bille, "A survey on tree edit distance and related problems," *Theoretical computer science,* vol. 337, no. 1-3, 2005, pp. 217–239.

[11] Y. Mehdad and B. Magnini, "Optimizing textual entailment recognition using particle swarm optimization," *in Proceedings of the 2009 Workshop on Applied Textual Inference.* Suntec, Singapore: Association for Computational Linguistics, 2009, pp. 36–43.

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

[13] N. Habash, A. Soudi, and T. Buckwalter, "On Arabic transliteration," *Arabic Computational Morphology,* 2007, pp. 15–22.

[14] W. Black, S. Elkateb, H. Rodriguez, and M. Alkhalifa, "Introducing the Arabic WordNet project," *in Proceedings of the 3rd International WordNet Conference (GWC-06),* 2006.

[15] 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.

[16] 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.

[17] 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.

[18] S. Bayer, J. Burger, L. Ferro, J. Henderson, and E. Yeh, "MITRE's Submissions to the EU Pascal RTE Challenge," *in The 1st PASCAL Recognising Textual Entailment Challenge (RTE I),* 2005, pp. 41–44.

[19] 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, Eygpt: The MEDAR Consortium, 2009, pp. 285–288.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] O. Smrž, V. Bielicky, I. Kouřilová, J. Kráčmar, J. Hajič, and P. Zemánek, "Prague Arabic dependency treebank: a word on the million words," *in Proceedings of the Workshop on Arabic and Local Languages (LREC 2008),* Morocco, 2008, pp. 16–23.

[25] 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.

[26] M. Alabbas and A. Ramsay, "Combining black-box taggers and parsers for modern standard Arabic," *in Proceedings of the Federated Conference on Computer Science and Information Systems (FedCSIS 2012),* Wroclaw, Poland: IEEE, 2012, pp. 19–26.

[27] N. Habash, *Introduction to Arabic natural language processing,* Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers, 2010.