The BIUTEE Research Platform for Transformation-based Textual Entailment Recognition

Asher Stern and Ido Dagan[1]

Recent progress in research of the Recognizing Textual Entailment (RTE) task shows a constantly-increasing level of complexity in this research field. A way to avoid having this complexity becoming a barrier for researchers, especially for new-comers in the field, is to provide a freely available RTE system with a high level of flexibility and extensibility. In this paper, we introduce our RTE system, BiuTee[2], and suggest it as an effective research framework for RTE. In particular, BiuTee follows the prominent transformation-based paradigm for RTE, and offers an accessible platform for research within this approach. We describe each of BiuTee's components and point out the mechanisms and properties which directly support adaptations and integration of new components. In addition, we describe BiuTee's visual tracing tool, which provides notable assistance for researchers in refining and "debugging" their knowledge resources and inference components.

1 Introduction and Background

Textual inference (the ability to automatically find conclusions that can be inferred from a natural language text) is a capability required for many tasks at the semantic level of *Natural Language Processing (NLP)*. For example, a typical *Information Extraction (IE)* task may be to extract, from a natural language text, the employer-employee

[1]Bar-Ilan University, Ramat-Gan, Israel

[2]BiuTee: **B**ar **I**lan **U**niversity **T**extual **E**ntailment **E**ngine. It is freely available at `http://www.cs.biu.ac.il/~nlp/downloads/biutee/`

LiLT Volume 9
Perspectives on Semantic Representations for Textual Inference.

relationship. Such a task can be formalized as the task of identifying text fragments from which it can be concluded that "X is employed by Y" for some entities X and Y. Similarly, a typical *Question Answering (QA)* task might be the task of finding answers to the question "By whom X is employed", for some entity X (a person, in this case). Addressing both these examples requires a mechanism that recognizes that the text fragment "X is employed by Y" can be inferred from a given text. *Recognizing Textual Entailment (RTE)* unifies this concept, and aims to serve as a common paradigm for textual inference. By adopting the RTE paradigm, the efforts required to solve the problem of textual inference need not to be duplicated for multiple NLP tasks which require this capability. Rather, the aim is to develop a generic solver for the RTE task, which can then be used as an inference component in task-specific systems.

The formal definition of the *Recognizing Textual Entailment* task is as follows. Given two text fragments, one termed *text* and the other *hypothesis*, the task is to recognize whether the hypothesis can be inferred from the text (Dagan et al. 2006a).

Since first introduced, several approaches have been proposed for this task, ranging from shallow lexical similarity methods (e.g., Clark and Harrison 2010; MacKinlay and Baldwin 2009), to complex linguistically-motivated methods, which incorporate extensive linguistic analysis (syntactic parsing, coreference resolution, semantic role labelling, etc.) and a rich inventory of linguistic and world-knowledge resources (e.g., Iftene 2008; de Salvo Braz et al. 2005; Bar-Haim et al. 2007). The latter methods convert the text and the hypothesis into rich representation levels, like syntactic parse-trees, semantic-role graph, or even a logical representation in which the text is converted into a collection of logical formulas. The next step is the entailment recognition itself in which the available knowledge resources are utilized.

Building such complex systems requires substantial development efforts, which might become a barrier for new-comers to RTE research. Thus, flexible and extensible publicly available RTE systems are expected to significantly facilitate research in this field. More concretely, two major research communities would benefit from a publicly available RTE system:

1. End application developers, who would use an RTE system to solve inference tasks within their application. RTE systems utilized by this type of researchers should be adaptable for the application specific data: they should be configurable, trainable, and extensible with inference knowledge that captures application-

specific phenomena.

2. Researchers in the RTE community, who would not need to build from scratch a complete RTE system for their research, but could integrate their novel research components into an existing open-source system. Such research efforts might include developing knowledge resources, developing inference components for specific phenomena such as temporal inference (see, for example, (Wang and Zhang 2008)), or extending RTE to different languages. A flexible and extensible RTE system is expected to encourage researchers to create and share their textual-inference components. A good example from another research area is the *Moses* system for *Statistical Machine Translation (SMT)* (Koehn et al. 2007), which provides the core SMT components while being extended with new research components by a large scientific community.

Until now rather few and quite limited RTE systems were made publicly available. These systems are quite restricted in the types of knowledge resources which they can utilize, and in the scope of their inference algorithms. For example, *EDITS*[3] (Kouylekov and Negri 2010) is a distance-based RTE system, which can exploit only lexical knowledge resources. *NutCracker*[4] (Bos and Markert 2005) is a system based on logical representation and automatic theorem proving, but utilizes only WordNet (Fellbaum 1998) as a lexical knowledge resource.

To address the above needs, we provide our open-source textual-entailment system, BIUTEE.[5] Our system provides state-of-the-art linguistic analysis tools and exploits various types of manually built and automatically acquired knowledge resources, including lexical, lexical-syntactic and syntactic rewrite rules. Furthermore, the system components, including pre-processing utilities, knowledge resources, and even the steps of the inference algorithm, are modular, and can be replaced or extended easily with new components. Extensibility and flexibility are also supported by a *plug-in mechanism*, by which new inference components can be integrated without changing existing code.

Notable support for researchers is provided by *a visual tracing tool*, *Tracer*, which visualizes every step of the inference process as shown in Figures 5 and 6, in Section 4.

This paper is organized as follows: A review of the main algorithmic components of BIUTEE is given in Section 2, followed by the system architecture description in Section 3. The visual tracing tool and its

[3] `http://edits.fbk.eu/`

[4] `http://svn.ask.it.usyd.edu.au/trac/candc/wiki/nutcracker`

[5] See footnote 2 above.

TABLE 1: A sequence of transformations that transform the text *"He received the letter from the secretary."* into the hypothesis *"The secretary delivered the message to the employee."*. The knowledge required for such transformations is often obtained from available knowledge resources and NLP tools.

#	Operation	Generated text
0	-	He received the letter from the secretary.
1	Coreference substitution	The employee received the letter from the secretary.
2	X received Y from Z → Y was sent to X by Z	The letter was sent to the employee by the secretary.
3	Y [verb-passive] by X → X [verb-active] Y	The secretary sent the letter to the employee.
4	X send Y → X deliver Y	The secretary delivered the letter to the employee.
5	letter → message	The secretary delivered the message to the employee.

typical use cases are described in Section 4, while in Section 5 we present experimental results. Conclusions, as well as suggestions for future work are given in Section 6.

2 Algorithms and Components

In this section we describe BiuTee's main algorithms and components. This description is given at a high-level, while more details of individual components are available in the papers cited along this section.

BiuTee (Stern and Dagan 2011) is a *transformation-based* inference system, in the sense that it transforms the text, T, into the hypothesis, H. Transforming T into H is done by applying a *sequence* of transformations, such that after applying the last transformation, the resulting text is identical to the hypothesis.[6] In this paper we use the term *proof* to refer to such a sequence of transformations. Table 1 demonstrates a proof for a typical (T,H) pair. The transformation-based paradigm requires three main design decisions:

1. How to represent the text and the hypothesis?
2. Which transformations to apply?
3. How to estimate whether a sequence of transformation preserves entailment?

In the following subsections we discuss each of these aspects. Finally, we deal with another crucial issue, namely:

[6] In practice, this goal is heuristically relaxed in the current version of BiuTee to having the hypothesis embedded in the obtained transformed text.

4. How to *find* automatically an "optimal" sequence of transformations that transforms T into H?

2.1 Representation level

There are several levels on which the text and the hypothesis can be represented. The simplest level of representation is the lexical level (e.g. bag of words) (see, for example, (Shnarch et al. 2011; Clark and Harrison 2010)). While this level has some advantages, for example, it can be easily implemented for languages that lack linguistic processing tools, it cannot handle structural differences of T and H. Consider, for example, the text "The first chapter of the book was written yesterday" and the hypothesis "The book was written yesterday". Though the hypothesis words are embedded in the text in the same order, the text does not entail the hypothesis. A common representation level of sentence structure is the syntactic representation, given as parse trees (See Figure 1). This level of representation was adopted by the vast majority of RTE systems (e.g., Iftene 2008; Cabrio et al. 2008; Wang and Neumann 2008). A deeper representation is the logical form level, which represents T and H as logical clauses, extracted from the syntactic representation. Typical examples are Tatu and Moldovan (2006), Raina et al. (2005) and Clark and Harrison (2010)

While deep logical representations may capture additional aspects of the meaning of the text, their much higher complexity makes them more vulnerable to inaccuracies and errors involved in their generation. Moreover, most of the structural information required for inference can be found in the syntactic representation, which was therefore chosen for BiuTee. However, since syntax does not capture some key semantic properties (e.g., the truth-value of predicates), we enrich the syntactic parse trees with additional annotations, as described later (Subsection 2.3).

2.2 Transformations

The second aspect in transformation-based inference is the type of transformations that can be applied by the system. Our goal is to apply transformations, such that each of them preserves the meaning of the text, as follows. When a transformation is applied on a text t, it transforms it into a new text, t'. The goal is that the meaning of t' will be entailed from t.

We apply many types of transformations, that are derived from many knowledge and linguistic resources. These transformations are relatively reliable, in most cases. In addition, we allow less reliable transformations, to be utilized when no prior knowledge of reliable transformations

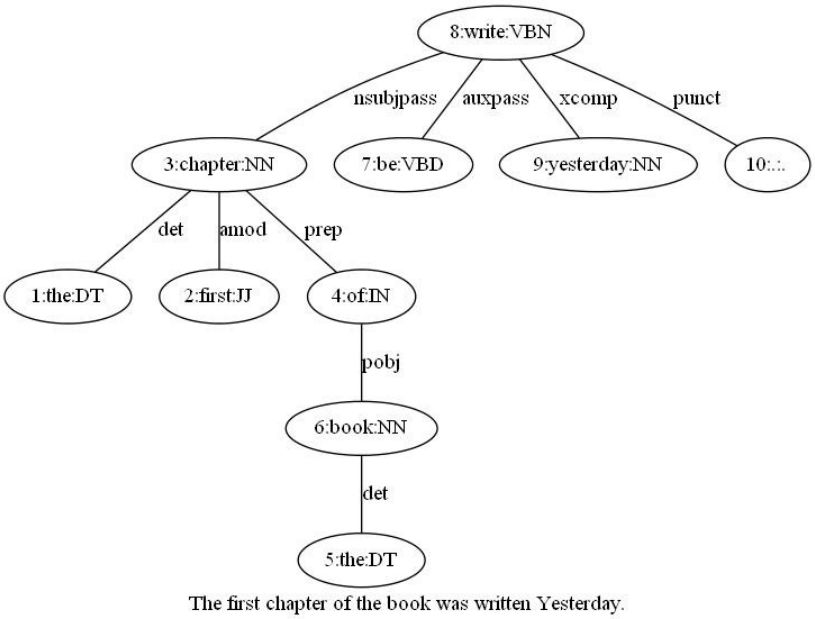

FIGURE 1: A parse tree, parsed by Easy-First parser (Goldberg and Elhadad 2010).

is available. Then, we estimate the validity of every transformation, and consequently of every *sequence* of transformations, and decide whether the text entails the hypothesis based on this estimation.

In the rest of this subsection we describe the transformations allowed by BiuTee, and in the next subsection we describe the model by which transformation reliability is estimated.

Entailment rules

The main type of transformations, and the most reliable one, is the application of *entailment-rules* (Bar-Haim et al. 2007), which are available from various knowledge resources. An entailment rule is composed of two sub-trees, termed *left-hand-side* and *right-hand-side*. It is *applied* on a parse-tree fragment that matches its left-hand-side, by substituting the left-hand-side with the right-hand-side. Figure 2 demonstrates a rule and its application. The complete formalism of entailment rules, adopted by our system, is described in (Bar-Haim et al. 2007; Stern and Dagan 2011).

The entailment rules formalism is simple yet powerful, and captures many types of knowledge. The simplest type of rules is *lexical rules*, like

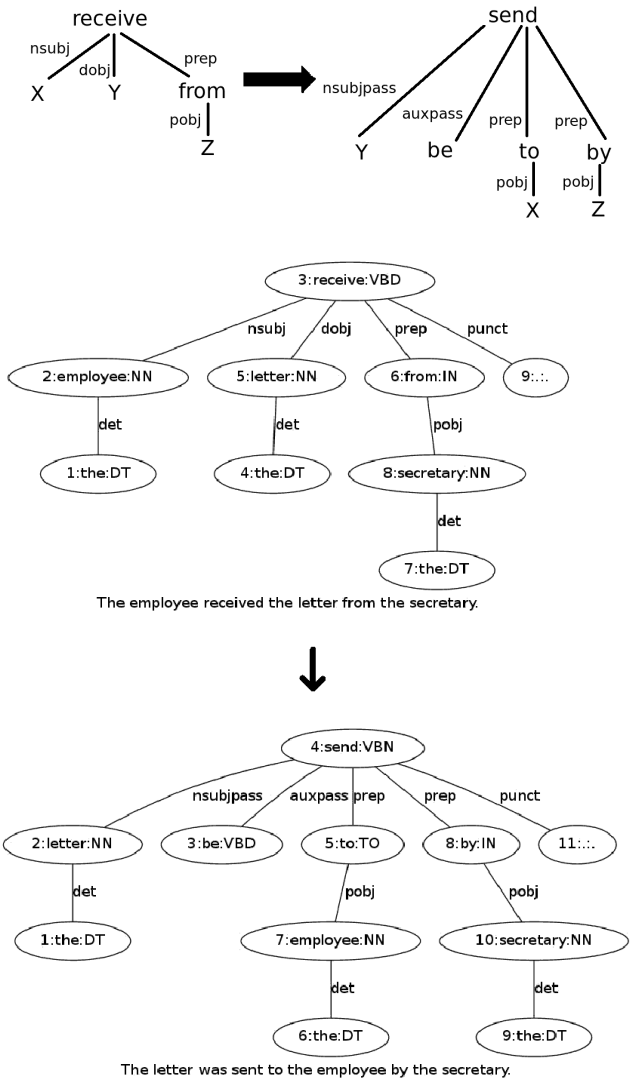

FIGURE 2: An entailment rule and its application. The rule "X received Y from Z $\rightarrow$ Y was sent to X by Z" is applied on the sentence "The employee received the letter from the secretary.", resulting in "The letter was sent to the employee by the secretary."

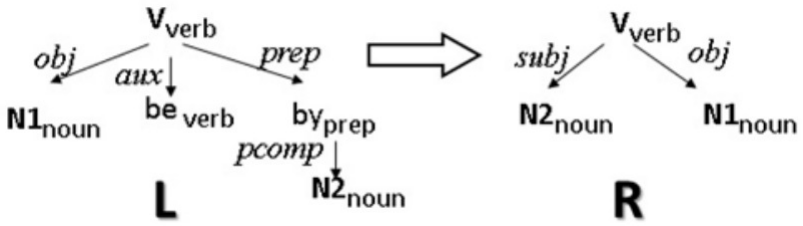

FIGURE 3: A syntactic entailment rule that converts passive form into active form.

`car` → `vehicle`. More complicated rules capture the entailment relation between predicate-argument structures, like `X receive Y from Z` → `Y was sent to X by Z`.

Entailment rules can also encode syntactic phenomena like the semantic equivalence of active and passive structures (`X Verb[active] Y` ↔ `Y is Verb[passive] by X`). See illustration in Figure 3.

BiuTee incorporates a comprehensive set of knowledge resources, which are represented as entailment-rules. Lexical knowledge resources within BiuTee include:

- WordNet (Fellbaum 1998) – a large database of English words, which are interconnected by means of conceptual-semantic and lexical relations. We used the following WordNet relations: *synonym*, *derivationally-related*, *hypernym*, *instance-hypernym*, *part-holonym*, *member-holonym*, *substance-meronym* and *verb-entailment*.
- Wikipedia – Wikipedia-based entailment-rules (Shnarch et al. 2009) which contain background knowledge about various entities and concepts, e.g., Einstein is a Scientist.
- GEO – a geographical knowledge resource (Mirkin et al. 2009) which contains information like New-York is in United States.
- CatVar – English derivations (Habash and Dorr 2003). For example motivation (noun) is derived from motivate (verb).
- DIRECT – directional distributional similarity (Kotlerman et al. 2010).
- Distributional-similarity-based entailment rules by Lin (1998a), as well as a reimplementation of the Lin-Similarity algorithm on the Reuters-corpus.[7]

[7] `http://trec.nist.gov/data/reuters/reuters.html`

- VerbOcean (Chklovski and Pantel 2004). – A broad-coverage semantic network of verbs. The used VerbOcean relations are configurable in BiuTee's configuration file. We achieved the best results when using only the "stronger than" relation.

Lexical-Syntactic knowledge resources, which capture entailment between predicate-argument structures within BiuTee include:

- DIRT (Lin and Pantel 2001)
- REVERB (Berant 2012)
- FrameNet-based entailment-rules (Ben-Aharon et al. 2010)

As for entailment-rules that capture syntactic phenomena, we use the freely available collection of rules by Lotan (2012).

Discourse phenomena

Other phenomena that must be handled by textual entailment systems are discourse phenomena, by which the desired information, expressed within a single sentence hypothesis, can be spread over several sentences of the text. The importance of this type of phenomena was investigated in (Mirkin et al. 2010), along with several proposals for ways to handle it by utilizing coreference information. In BiuTee, we utilize coference resolution information by defining two types of transformations: coreference substitution and Is-A coreference. We obtain the coreference information from an off-the-shelf coreference resolution system, ArkRef (see Subsection 3.1).

Coreference substitution is implemented as follows: one mention of an entity is replaced by another mention of the same entity, based on a coreference relation between them. Consider, for example, the following text-hypothesis pair:
T*ext*: ... Obasanjo invited him to step down as president ... and accept political asylum in Nigeria.
H*ypothesis*: Charles G. Taylor was offered asylum in Nigeria.
In this example it is required to apply such a transformation to substitute "him" with "Charles G. Taylor" (which is mentioned earlier in the document).

The Is-A coreference transformation creates a new parse-tree of the form "X is Y" for each two coreferring mentions X and Y. Consider the following hypothesis:

> The Irish Republican Army is a paramilitary group.

and consider the following text:

> Northern Ireland's peace process would receive a major boost if the Irish Republican Army responds positively to appeals to embrace political methods, a leading figure in the paramilitary group's political wing said Friday.

Given that "Irish Republican Army" corefers to "paramilitary group", the Is-A coreference transformation can construct the desired hypothesis.

On the fly transformations

Since applications of entailment rules from available knowledge resources and coreference substitutions are, in most cases, insufficient for completely transforming T into H, our system allows *on-the-fly* parse-tree transformations. These transformations include insertions of missing nodes, flipping parts-of-speech, moving sub-trees, etc. (see (Stern and Dagan 2011) for a complete list of these transformations). Since these transformations are not justified by given knowledge resources, we use linguistically-motivated features to estimate their validity. For example, for on-the-fly lexical insertions we consider as features the named-entity annotation of the inserted word, and its probability estimation according to a unigram language model, which yields lower costs (higher confidence) for more frequent words (with similar rationale to that of the Inverse Document Frequency (IDF) heuristic in Information Retrieval). The usage of such features is described in Subsection 2.4.

Plug-ins

BiuTee is designed as an open system, which can serve as a research platform for the RTE community. As such, it is not limited to the aforementioned transformations. Easy integration of additional types of transformations is supported by BiuTee's *plug-in* mechanism, described in Subsection 3.2.

2.3 Truth-value annotations

As mentioned above, BiuTee transforms T's parse tree into H's parse tree. This requires a well defined criterion for determining that nodes and edges in T are identical to corresponding nodes and edges in H. The natural criterion is that nodes are identical if they contain the same lemma and the same part-of-speech, while edges are identical if they have the same relation (edge label). This criterion, however, might lead to an error when a predicate is negated in T, but not in H, or vice versa. More generally, such inconsistencies might arise not only due to negations, but also due to other natural language constructions that change the *truth-value* of a predicate. For example, given a text "`The computer failed to calculate the equation.`", the truth-value of "calculate" is negative, since it can be inferred that the computer did not calculate the equation (See (Karttunen 1971)). Thus, we should not treat the predicate "calculate" in a hypothesis "`The computer calculated the equation.`" as identical to "calculate" in the text.

Handling such cases is performed by the integration of *Truth-Teller* (Lotan 2012; Lotan et al. 2013), which annotates truth-values of all the predicates in a given sentence, by utilizing several mechanisms. Given a sentence, *Truth-Teller* begins by annotating some of its clauses with a polarity of positive, negative, or unknown. This is done by identifying pre-suppositions, which are marked as positive-polarity clauses, as well as the main clause of the sentence, which is always marked as positive. In addition, it annotates the predicates of the annotated clauses with a positive, negative or unknown polarity. This annotation is based on the clause annotation along with identification of negation and modality expressions (e.g. "not", "never"). Then, the annotation process proceeds to annotate all remaining clauses and predicates by utilizing a recursive algebra which makes use of the implicativity and factivity signatures of the predicates.

A parse-tree node that corresponds to a predicate in the text is considered identical to a corresponding node in the hypothesis only if these two nodes have the same predicate-truth annotation, as well as the same lemma and part-of-speech. A new transformation, *change predicate-truth value*, has been added to BIUTEE. This transformation flips the truth-value annotation of a predicate in the text, such that it becomes identical to a predicate in the hypothesis. Like all other transformations (see Subsection 2.4), a cost is learned for this transformation. Following the high reliability of Truth-Teller, the learned cost is usually high. Hence, proofs which require a flip in a truth-value annotation are usually considered incorrect proofs.

2.4 Cost model

Given a (T,H) pair, the system has to find a sequence of transformations (a *proof*) that transforms T into H. Next, the system calculates how likely it is that the proof preserves entailment. This calculation is performed by a *cost model*, similar to the one defined in (Raina et al. 2005), as follows. First, we define features which characterize transformations (see below). A single transformation, o, can then be characterized by a *feature-vector*, $f(o)$, which contains values for all of these features. We use this feature vector to define a *cost* for each transformation, such that reliable transformations are assigned low costs, while transformations that are less likely to be reliable are assigned high costs. This cost is calculated using a *weight vector*, w, learned automatically over a training-data. Formally, the cost of a transformation o is defined as:

$$c(o) = w \cdot f(o) = \sum_{i=1}^{m} w_i \cdot f_i(o) \tag{1}$$

where m is the number of features, and $f_i(o)$ is the value of of the ith feature of transformation o.

Next, we define the cost of the proof as the sum of the costs of all its transformations. Formally, a proof $O = (o_1, o_2 \dots o_n)$ is assigned the cost $c(O) = \sum_{j=1}^{n} c(o_j)$. By defining $f(O) = \sum_{j=1}^{n} f(o_j)$, the last equation can be algebraically manipulated as follows:

$$(2) \quad c(O) = \sum_{j=1}^{n} c(o_j) = \sum_{j=1}^{n} \sum_{i=1}^{m} w_i \cdot f_i(o_j) = \sum_{i=1}^{m} w_i \cdot f_i(O) = w \cdot f(O)$$

The last equation suggests a typical linear learning paradigm, in which the training data is represented as a collection of feature vectors, and the goal is to find two parameters: a weight vector, w, and a threshold b, for which $w \cdot f(O) \leq b$ for positive examples (i.e. T entails H), while $w \cdot f(O) > b$ for negative ones. The learning scheme details are described in (Stern and Dagan 2011).

For the transformations described above (Subsection 2.2) we defined the features as follows. First, a feature has been defined for each knowledge resource. For some knowledge resources, the feature is assigned a constant value of 1 for each rule-application based on this resources. For example, let us assume that feature number 10 is the WordNet feature, and that we are given a transformation which substitutes "dog" by "pet" based on a WordNet hypernym relation. The feature-vector for this transformation then has 1 as the value of the 10th feature, while all other features are assigned 0. Other knowledge resources (e.g. DIRT) provide a score for each rule. For these knowledge resources we use the log of that score as the value for the corresponding feature.

For on-the-fly insertion transformations we defined several features that describe whether the inserted word is a content word, whether it is a named entity, etc. In all cases, the feature value is the probability of the inserted word to appear in an English text, according to a Unigram language model based on the Reuters corpus[8].

For on-the-fly move transformations we defined features that quantify how much the move transformation changed the context of the moved node. The feature-value is the path-length between the original and the new parents of the moved node.

We also defined features for other on-the-fly transformations, like flip-part-of-speech, splitting a multi-word-expression, etc.

As for plug-ins, the plug-in mechanism allows every plug-in to define its own features, which get their values by the plug-in when applied, as described below (Subsection 3.2).

[8] `http://trec.nist.gov/data/reuters/reuters.html`

2.5 Search challenge

Given a text-hypothesis pair, there might be many proofs by which the text can be transformed into the hypothesis. For example, a syntactic manipulation can be performed by applying a syntactic rule. However, the same manipulation might be achieved by applying several "on-the-fly move sub-tree" transformations. Another example is when the same knowledge exists in two different knowledge-resources, but one of them is more reliable than the other.

Since the decision whether the text entails the hypothesis is determined by the cost of the proof, the system has to find the proof with the lowest cost. Finding that proof is a non-trivial challenge since, in general, there are many transformations that can be applied to a given parse-tree, and thus the number of possible *sequences* of transformation is exponential in the size of the proof.

BiuTee provides implementations of several search algorithms, which differ from one another in their speed and proof quality (measured by the cost of the proofs they find). A novel improved search algorithm that directly utilizes characteristics of the textual-inference domain was recently developed and integrated into BiuTee. This algorithm iteratively generates limited-length subsequences of transformations. In each iteration it measures the quality-cost ratio of each generated subsequence, and chooses the one with the best ratio. The full details are described in (Stern et al. 2012).

3 System Architecture

The input of BiuTee is a collection of (T,H) pairs, and the output is an entailment / non-entailment classification for each pair. To determine these classifications, each (T,H) pair is pipe-lined in several processing phases which (1) generate the appropriate representations, (2) find a sequence of transformations (a proof) which transforms the text into the hypothesis, (3) classify these proofs. During training, the last step, *classification* of proofs, is replaced by *learning* a classification model.

We note that in phase (2) we start with inference-related calculations that are not part of the proof construction, but add additional annotations which are required for entailment recognition, and are used during the proof construction. Currently, the *Truth-Teller* annotations (see Subsection 2.3) take place here. In the future we plan to add other types of annotations, like recognizing temporal relations between T and H (see, for example, (Wang and Zhang 2008)).

BiuTee's processing flow is illustrated in Figure 4, and is described in the reminder of this section.

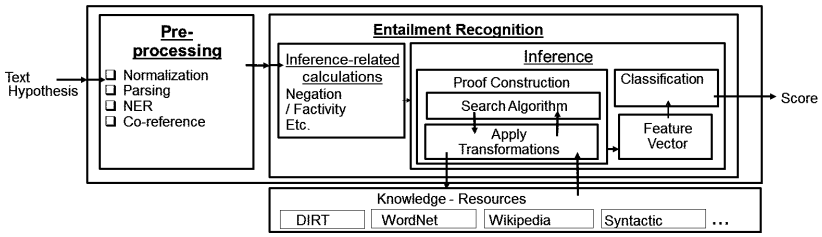

FIGURE 4: System architecture

3.1 Linguistic analysis processing

The target representation of T and H is a parse-tree representation. We generate dependency parse trees, in which each node corresponds to exactly one token of a sentence, and contains its lemma and part-of-speech. In addition, it contains named-entity annotation, integrated into parse-tree nodes. In addition, we build a data-structure for coreference information. This data-structure is a collection of coreference chains of the text's parse trees nodes.

To build this representation, each text fragment (which may be either T or H) is processed through the following steps.

1. Text normalization
2. Sentence splitter
3. Tokenizer
4. Part-of-speech tagger
5. Parser
6. Coreference resolver

Text normalization is done through a collection of text modifications that are performed over the raw text, prior to any linguistic analysis. Mainly, we perform number normalization,[9] and heuristically add missing punctuations when necessary.

Then, BiuTee proceeds to linguistic processing, using state-of-the-art utilities: Tokenization and part-of-speech-tagging are performed by *Stanford* utilities (Toutanova et al. 2003), parsing is performed by *Easy-First parser* (Goldberg and Elhadad 2010), named-entity recognition is done by *Stanford named-entity-recognizer* (Finkel et al. 2005) and coreference resolution is performed by *ArkRef coreference resolver*.[10]

[9] Available to download as a stand-alone utility at `http://www.cs.biu.ac.il/~nlp/downloads/normalizer.html`

[10] This tool is a reimplementation of (Haghighi and Klein 2009), and is downloadable from `http://www.ark.cs.cmu.edu/ARKref/`.

As a flexible system, BiuTee provides simple interfaces for each type of linguistic analysis utility. This way, replacing the above-mentioned utilities can be done by merely implementing the relevant interfaces.

3.2 Proof construction

Entailment recognition begins with *inference-related calculations*. As mentioned above, the Truth-Teller annotations are added at this phase.

Next, the system constructs a proof comprising of a sequence of transformations that transform the text into the hypothesis. Finding such a proof is a sequential process, conducted by the search algorithm (see Subsection 2.5). In each step of the proof construction the system examines all possible transformations that can be applied, generates new trees by applying these transformations, and calculates their accumulated proof costs by constructing appropriate feature-vectors for them.

BiuTee provides unified interfaces for all types of transformations, making the addition of new types of transformations straightforward. The interfaces are:

1. *Finder* which gets a parse tree, and finds the transformations that can be applied on that tree.
2. *Operation* which gets a parse tree and a transformation (found by the *Finder*), and applies this transformation on the given parse tree. The result is a new parse tree.
3. *Feature-Vector Updater* which gets a parse-tree, a feature-vector and a transformation, and updates this feature-vector with new values that correspond to the given transformation. This updated feature-vector represents the new parse-tree that was generated by applying the given transformation.

BiuTee contains a collection of implementation steps for these interfaces for each type of transformation and for each knowledge resource. For example, for the WordNet lexical resource that is used with lexical entailment rules, the implementation is as follows: the *Finder* gets a parse-tree, and finds for each parse-tree node (when applicable) a lexical rule from WordNet which changes its lemma to another lemma, entailed by the original one.[11]

The *Operation* then takes the parse-tree and generates a new parse tree in which the original lemma is replaced by the entailed lemma. The *Feature-Vector Updater* adds 1 to the WordNet feature in the feature vector.

[11] This *Finder* is actually shared among all the lexical resources, as they all share the same interface for querying lexical entailment rules.

When a (T, H) pair is given to the system, it is first processed in the inference-related calculations phase. Then, the system iteratively runs all of the implementation steps of the above mentioned interfaces (*Finder*, *Operation* and *Feature-Vector Updater*). Thus, in each iteration several new parse-trees, along with their corresponding feature-vectors, are generated. Since the number of generated trees increases exponentially, the system must prune the set of the generated trees after few iterations, and maintain only those which are most likely to be part of the best (i.e., cheapest) proof. The policy of which trees to prune is determined by the search algorithm.

Extension mechanism

The goal of BiuTee is not only to apply its own built-in transformations, but also to be a framework for transformation-based inference, into which additional types of transformations can be integrated and applied. This is achieved by the *plug-in* mechanism. Using plug-ins, new types of transformations can be integrated and applied, without the need to change the current system code. For example, imagine a researcher applying BiuTee in the medical domain, say, over prescription texts. There might be some well-known domain knowledge and some re-writing rules that every medical person knows. Integrating such new rules is directly supported by the plug-in mechanism.

Basically, adding a new type of transformation can be done by implementing the above mentioned interfaces. Thus, the plug-in developer has to implement *Finder*, *Operation* and *Feature-Vector Updater*. A factory of these implementations, called *Plugin*, has to be developed as well, and be integrated into the system. BiuTee has a smart mechanism, which safely utilizes Java reflection capabilities, by which this integration is performed with no changes of existing code.

In addition, BiuTee provides a more advanced type of plug-in that is able to perform its own calculations in the *inference-related calculations phase*, in addition to integrating the new transformations. Such plug-ins have to implement a method which gets the parsed text and the hypothesis, as well as other auxiliary parameters, as input. This method is called by the system for each (T, H) pair right before starting the proof construction phase. The results of a plug-in's global calculation can be stored in the plug-in's internal data-structures, and be forwarded to the actual instances of its corresponding *Finder*, *Operation* and *Feature-Vector Updater*. By implementing such a plug-in, the user can incorporate new types of inference-oriented annotations, which can then be utilized by new types of transformations.

3.3 Learning and classification

The final processing phase is to classify the constructed proof as reliable or not. The proof, represented as a feature vector, is assigned a cost, as described in Subsection 2.4. The proof is classified as reliable if its cost is lower than a threshold. The system also provides the result as a numerical value in the $[0, 1]$ range by using the *sigmoid* function: the return value is $\frac{1}{1+e^{-z}}$ where z is the cost minus the threshold. This value is interpreted as "positive" (T entails H) if it is higher than 0.5, and "negative" otherwise.

This final phase has a different role during training. In training we are not interested in classifying the proof, but in learning the cost model parameters: the weight vector, w, and the threshold b. These parameters can be learned by any linear learning algorithm. The input for the learning algorithm is a collection of feature-vectors that represent the proofs of a collection of (T, H) pairs (i.e., an RTE dataset). We use a *Logistic-Regression* learning algorithm, but, similar to other components, alternative learning-algorithms can be integrated easily by implementing an appropriate interface.

Note that for the last two RTE datasets of 2010 and 2011 (RTE-6 and RTE-7), the goal is to optimize the F1 measure[12] of the positive entailments, while in the older datasets, the goal is to optimize the accuracy measure. We implemented the logistic regression classifier to optimize each of these measures. The accuracy-optimized classifier is the standard logistic-regression classifier, while the F1-optimized classifier was implemented according to (Jansche 2005). The appropriate classifier is automatically chosen by the system, based on the given dataset.

We note that the weight vector, w, has a dual role. One is to assign a cost to the constructed proof. The other role is to be used by the search algorithm. As described above, the search algorithm prunes some of the intermediate trees that are generated during the proof construction, and it favours trees that are more likely to be part of the best (cheapest) proof. One of the parameters by which the search algorithm estimates this likelihood is the current cost of the generated tree. If that cost is higher than that of other generated trees, it is less likely that it will be part of the best proof. This dual role raises a new problem: the only way to learn the weight vector is by considering the proofs that

[12] F1 measure is a success-rate estimation which takes both the recall (how many positive instances have been detected by the system) and precision (how many of the positively classified instances are indeed positive) into consideration. Formally, it is defined as $2 \cdot \frac{\text{Recall} \cdot \text{Precision}}{\text{Recall} + \text{Rrecision}}$

have already been constructed for a complete (T, H) pairs dataset. However, to construct these proofs, a weight vector is needed for the search algorithm.

We solve this problem by an *iterative learning* scheme. In this scheme we initialize w with a reasonable guess vector, and iteratively process the whole dataset, such that in each iteration w is improved. More information about the learning scheme can be found in (Stern and Dagan 2011).

3.4 Configuration, adaptation and extension

BiuTee supports all of the RTE datasets that have been published so far (Dagan et al. 2006b; Bar-Haim et al. 2006; Giampiccolo et al. 2007, 2008; Bentivogli et al. 2009, 2010, 2011).

Controlling the system behaviour, adapting it to specific needs as well as extending it, can be done at various levels. First, many of the system parameters are controlled by a configuration file, as follows:

1. Parser: the user can choose to use either easy-first parser (Goldberg and Elhadad 2010) or Minipar parser (Lin 1998b).
2. Coreference resolver: The user can choose the coreference resolver to be either ArkRef[13] or Bart (Versley et al. 2008). A third option is to configure the system to skip coreference resolution.
3. Knowledge Resources: The user can provide a list of knowledge resources to be used for the proof construction. Using many knowledge resources often increases the system performance, but also increases its runtime.
4. Multi-threading: The number of concurrent threads used by the system is a configurable parameter.
5. Plug-ins: If the user writes a plug-in, it can be integrated into the system through configuration file parameters.

Beyond configuration, the system can be extended by the plug-in mechanism, described in Subsection 3.2.

More advanced adaptations and extensions can be performed by changing the system's source code, as all of BiuTee's source code is freely available. The code is modular so that changing one module does not affect the others. Further support is given by an extensive documentation, both code-level documentation as well as a developer guide that describes the details of the system flow and its components.

[13] See footnote 10.

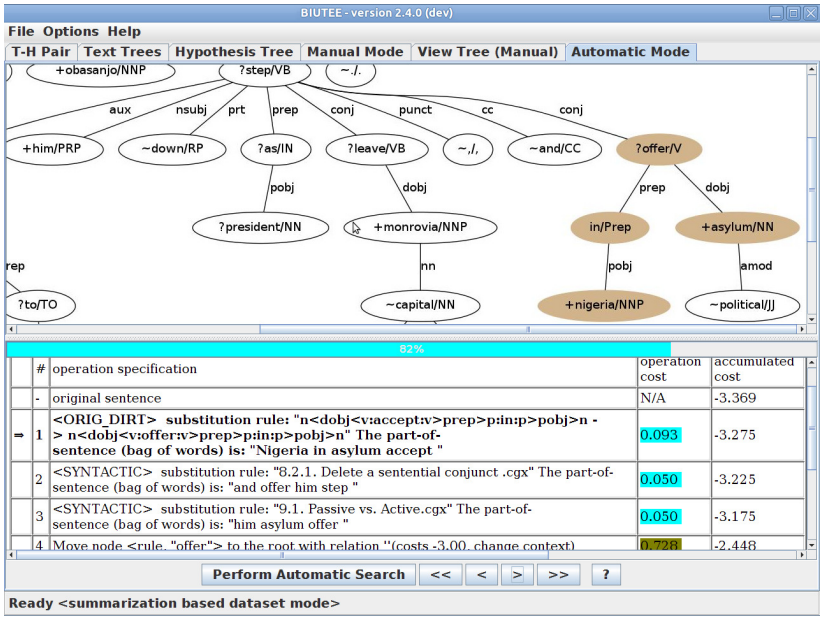

FIGURE 5: Entailment Rule application visualized in the tracing tool. The upper pane displays the parse-tree generated by applying the rule. The rule description, highlighted in bold, is the first transformation of the proof, shown in the lower pane. The rule is "X (is) accepted in Y" → "X (is) offered in Y", and captures, for example, that if someone accepted asylum in Nigeria, than it is also known that he was offered asylum in Nigeria. This rule application is followed by transformations 2 and 3, which are syntactic rewrite rules.

4 Visual Tracing Tool

The final score provided as output, as well as the system's detailed logging information, do not expose all the decisions and calculations performed by the system. In particular, they do not show all the potential transformations that could have been applied, but were rejected by the search algorithm. However, such information is crucial for researchers, who need to observe the use and the potential impact of each component of the system.

We address this need by providing an interactive *visual tracing tool*, *Tracer*, which presents detailed information for each proof step, including the ability to force potential inference steps that were not included in the automatically-constructed proof.

4.1 Modes

Tracer provides two modes for tracing proof construction: *automatic mode* and *manual mode*. In the automatic mode, shown in Figure 5, the tool presents the complete process of inference, as conducted by the system's search: the parse trees, the proof steps, the cost of each step and the final score. For each transformation the tool presents the parse tree before and after applying the transformation, highlighting the impact of this transformation.

In manual mode, the user can invoke specific transformations proactively, including transformations rejected by the search algorithm. As shown in Figure 6, the tool provides a list of transformations that match the given parse-tree, from which the user can choose to apply a single transformation at each step. Similar to automatic mode, the impact on the parse tree is shown visually.

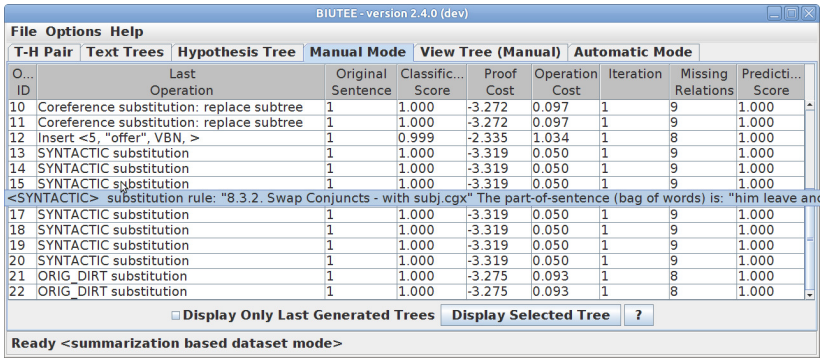

FIGURE 6: List of available transformations, provided by *Tracer* in the manual mode. The user can manually choose and apply each of these transformations, and observe their impact on the parse-tree.

4.2 Typical Use cases

Developers of knowledge resources and other types of transformations can use *Tracer* as follows. Applying an entailment rule is a process of first *matching* the rule's left-hand-side to the text parse-tree (or to any tree along the proof), and then substituting it by the rule's right-hand-side. To test a rule, the user can provide a text for which the rule is supposed to match, examine the list of potential transformations that can be performed on the text's parse tree, as shown in Figure 6, and verify that the examined rule has been matched as expected. Next, the user can apply the rule, visually examine its impact on the parse-tree,

table 2: Performance (accuracy) of an earlier version of BiuTee on RTE challenges, compared to other systems participated in these challenges. *Median* and *Best* indicate the median score and the highest score of all submissions, respectively.

RTE challenge	Median %	Best %	BiuTee %
RTE-1	55.20	58.60	57.13
RTE-2	58.13	75.30	61.63
RTE-3	61.75	80.00	67.13
RTE-5	61.00	73.50	63.50

as in Figure 5, and validate that it operates as intended and does not cause undesired side-effects.

Researchers of proof construction and classification algorithms can also make use of *Tracer*. As described above, the complete inference process depends on the parameters learned in the training phase, as well as on the search algorithm which looks for a lowest-cost proof from T to H. For a given (T,H) pair, the automatic mode provides the complete proof found by the system. Then, in the manual mode the researcher can try to construct alternative proofs. If a proof with a lower cost can be constructed manually it signals a limitation of the search algorithm. In contrast, if the user can manually construct a better linguistically motivated proof, but it turns out that this proof has higher cost than the one found by the system, it signals a limitation of the learning phase which may be caused either by a limitation of the learning method, or due to insufficient training data.

5 Experimental Results

In this section we briefly overview BiuTee's performance on the RTE challenges. We omit RTE-4 (2008) since it does not contain training data. Table 2 shows results of an earlier version of BiuTee on RTE 1,2,3 and 5 (Dagan et al. 2006b; Bar-Haim et al. 2006; Giampiccolo et al. 2007; Bentivogli et al. 2009). This earlier version does not contain many components that were described in this paper, e.g., it does not contain the syntactic rules, the Truth-Teller, the logistic-regression classifier, and uses an older parser, an older coreference resolver and an older scheme for the search algorithm. It can be seen, however, that even the earlier version achieved better results than the median of all submitted results to these challenges.

BiuTee's performance on the more recent RTE-6 and RTE-7 (Bentivogli et al. 2010, 2011) challenges is presented in Table 3: BiuTee is better than the median of all submitted results, and in RTE-6 it

table 3: Performance (F1) of BiuTee on RTE challenges, compared to other systems participated in these challenges. *Median* and *Best* indicate the median score and the highest score of all submissions, respectively.

RTE challenge	Median %	Best %	BiuTee %
RTE-6	33.72	48.01	49.09
RTE-7	39.89	48.00	42.93

outperforms all other systems.

6 Conclusions and Future Work

In this paper we described BiuTee, an open-source textual-inference system, and suggested it as a research platform in the RTE field in general, and within the transformation-based paradigm in particular. Such a platform is needed in this complex research area, in which a lot of linguistic analysis utilities, extensive usage of knowledge resources, and sophisticated entailment recognition components are needed.

From this perspective, we highlighted the following key advantages of our system: (a) modularity and extensibility, (b) a plug-in mechanism, (c) utilization of entailment rules that can capture diverse types of knowledge, and (d) a tracing tool that visualizes every detail of the inference process.

We suggest several directions for future work. First, as described above, BiuTee can be extended with additional types of transformations. One example is arithmetic transformations, as in the following example:

Text: *Two men and three women were wounded in the accident.*

Hypothesis: *Three people were wounded in the accident.*

Such example requires a complex transformation which first identifies that men, as well as women, entail people, and then makes the arithmetic calculation of $2 + 3 = 5$.

Another suggested extension is temporal entailment, like:

Text: *In 2004 the Olympic games took place in Athens, and four years later in Beijing*

Hypothesis: *In 2008 the Olympic games took place in Beijing.*

This example requires a transformation of "four years later" into "2008", followed by the introduction of the implicit sentence "the Olympic games took place in Beijing" by generic-syntactic rewrite rules.

Another direction for future work is in improving the on-the-fly transformations and their features. For example, the current feature-values for on-the-fly insertion are based on a Unigram language model. However, this model is limited, as it does not consider the context of the given text. A better model would calculate for each inserted word how likely it is that this word is entailed by the given text, thus improving the overall cost estimation of proofs that contain the insertion on-the-fly transformation.

More broadly, an appealing property of BiuTee's inference model and architecture is the ability to easily integrate various types of linguistically-motivated inferences. For example, one can easily integrate additional linguistic processing levels, such as semantic role labelling,discourse analysis and temporal analysis as additional annotation levels, which can then be leveraged by appropriate transformations with their reliability being assessed by corresponding features. Thus, we believe that BiuTee provides a suitable platform for investigating the incorporation of various advanced annotations and inference types within textual entailment recognition.

Acknowledgments

We thank Amnon Lotan for contributing and integrating the Truth-Teller component, and for enhancing the visual tracing tool. This work was partially supported by the Israel Science Foundation grant 1112/08, the PASCAL-2 Network of Excellence of the European Community FP7-ICT-2007-1-216886, and the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 287923 (EXCITEMENT).

References

Bar-Haim, Roy, Ido Dagan, Bill Dolan, Lisa Ferro, Danilo Giampiccolo, Bernardo Magnini, and Idan Szpektor. 2006. The second PASCAL recognising textual entailment challenge. In *Proceedings of the Second PASCAL Challenges Workshop on Recognising Textual Entailment*.

Bar-Haim, Roy, Ido Dagan, Iddo Greental, and Eyal Shnarch. 2007. Semantic inference at the lexical-syntactic level. In *Proceedings of AAAI*.

Ben-Aharon, Roni, Idan Szpektor, and Ido Dagan. 2010. Generating entailment rules from framenet. In *Proceedings of ACL*.

Bentivogli, Luisa, Peter Clark, Ido Dagan, Hoa Trang Dang, and Danilo Giampiccolo. 2010. The sixth PASCAL recognizing textual entailment challenge. In *Proceedings of TAC*.

Bentivogli, Luisa, Peter Clark, Ido Dagan, Hoa Trang Dang, and Danilo Giampiccolo. 2011. The seventh PASCAL recognizing textual entailment challenge. In *Proceedings of TAC*.

Bentivogli, Luisa, Ido Dagan, Hoa Trang Dang, Danilo Giampiccolo, and Bernardo Magnini. 2009. The fifth PASCAL recognizing textual entailment challenge. In *Proceedings of TAC*.

Berant, Jonathan. 2012. *Global Learning of Textual Entailment Graphs (submitted)*. Ph.D. thesis, Tel Aviv University.

Bos, Johan and Katja Markert. 2005. Recognising textual entailment with logical inference. In *Proceedings of EMNLP*.

Cabrio, Elena, Milen Kouylekov, and Bernardo Magnini. 2008. Combining specialized entailment engines for RTE-4. In *Proceedings of TAC*. Gaithersburg, Maryland.

Chklovski, Timothy and Patrick Pantel. 2004. Verbocean: Mining the web for fine-grained semantic verb relations. In *Proceedings of EMNLP*.

Clark, Peter and Phil Harrison. 2010. Blue-lite: a knowledge-based lexical entailment system for rte6. In *Proceedings of TAC*.

Dagan, Ido, Oren Glickman, and Bernardo Magnini. 2006a. The PASCAL recognising textual entailment challenge. In *Quiñonero-Candela, J.; Dagan, I.; Magnini, B.; d'Alché-Buc, F. (Eds.) Machine Learning Challenges. Lecture Notes in Computer Science*.

Dagan, I., O. Glickman, and B. Magnini. 2006b. The pascal recognising textual entailment challenge. *Machine Learning Challenges. Evaluating Predictive Uncertainty, Visual Object Classification, and Recognising Tectual Entailment* .

de Salvo Braz, Rodrigo, Roxana Girju, Vasin Punyakanok, Dan Roth, and Mark Sammons. 2005. An inference model for semantic entailment in natural language. In *Proceedings of AAAI*.

Fellbaum, Christiane, ed. 1998. *WordNet An Electronic Lexical Database*. The MIT Press.

Finkel, Jenny Rose, Trond Grenager, and Christopher Manning. 2005. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of ACL*.

Giampiccolo, Danilo, Hoa Trang Dang, Bernardo Magnini, Ido Dagan, Elena Cabrio, and Bill Dolan. 2008. The fourth PASCAL recognizing textual entailment challenge. In *Proceedings of TAC*.

Giampiccolo, Danilo, Bernardo Magnini, Ido Dagan, and Bill Dolan. 2007. The third PASCAL recognizing textual entailment challenge. In *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*.

Goldberg, Yoav and Michael Elhadad. 2010. An efficient algorithm for easy-first non-directional dependency parsing. In *Proceedings of NAACL*.

Habash, Nizar and Bonnie Dorr. 2003. A categorial variation database for english. In *Proceedings of NAACL*.

Haghighi, Aria and Dan Klein. 2009. Simple coreference resolution with rich syntactic and semantic features. In *Proceedings of EMNLP*.

Iftene, Adrian. 2008. Uaic participation at RTE4. In *Proceedings of TAC*.

Jansche, Martin. 2005. Maximum expected f-measure training of logistic regression models. In *Proceedings of EMNLP*.

Karttunen, Lauri. 1971. Implicative verbs. *Language* .

Koehn, Philipp, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondrej Bojar, Alexandra Constantin, and Evan Herbst. 2007. Moses: Open source toolkit for statistical machine translation. In *Proceedings of ACL*.

Kotlerman, Lili, Ido Dagan, Idan Szpektor, and Maayan Zhitomirsky-geffet. 2010. Directional distributional similarity for lexical inference. *Natural Language Engineering* .

Kouylekov, Milen and Matteo Negri. 2010. An open-source package for recognizing textual entailment. In *Proceedings of ACL Demo*.

Lin, Dekang. 1998a. Automatic retrieval and clustering of similar words. In *Proceedings of COLING*.

Lin, Dekang. 1998b. Dependency-based evaluation of minipar. In *Proceedings of the Workshop on Evaluation of Parsing Systems at LREC 1998*.

Lin, Dekang and Patrick Pantel. 2001. DIRT - discovery of inference rules from text. In *Proceedings of ACM SIGKDD Conference on Knowledge Discovery and Data Mining*.

Lotan, Amnon. 2012. *A Syntax-based Rule-base for Textual Entailment and a Semantic Truth Value Annotator*. Master's thesis, Bar Ilan University.

Lotan, Amnon, Asher Stern, and Ido Dagan. 2013. Truthteller: Annotating predicate truth. In *Proceedings of NAACL*.

MacKinlay, Andrew and Timothy Baldwin. 2009. A baseline approach to the RTE5 search pilot. In *Proceedings of TAC*.

Mirkin, Shachar, Roy Bar-Haim, Jonathan Berant, Ido Dagan, Eyal Shnarch, Asher Stern, and Idan Szpektor. 2009. Addressing discourse and document structure in the RTE search task. In *Proceedings of TAC*. Gaithersburg, Maryland.

Mirkin, Shachar, Ido Dagan, and Sebastian Pado. 2010. Assessing the role of discourse references in entailment inference. In *Proceedings of ACL*.

Raina, Rajat, Andrew Y. Ng, and Christopher D. Manning. 2005. Robust textual inference via learning and abductive reasoning. In *Proceedings of AAAI*.

Shnarch, Eyal, Libby Barak, and Ido Dagan. 2009. Extracting lexical reference rules from Wikipedia. In *Proceedings of ACL-IJCNLP*.

Shnarch, Eyal, Jacob Goldberger, and Ido Dagan. 2011. A probabilistic modeling framework for lexical entailment. In *Proceedings of ACL*.

Stern, Asher and Ido Dagan. 2011. A confidence model for syntactically-motivated entailment proofs. In *Proceedings of RANLP*.

Stern, Asher, Roni Stern, Ido Dagan, and Ariel Felner. 2012. Efficient search for transformation-based inference. In *Proceedings of ACL*.

Tatu, Marta and Dan Moldovan. 2006. A logic-based semantic approach to recognizing textual entailment. In *Proceedings of ACL*.

Toutanova, Kristina, Dan Klein, Christopher Manning, and Yoram Singer. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of NAACL*.

Versley, Yannick, Simone Paolo Ponzetto, Massimo Poesio, Vladimir Eidelman, Alan Jern, Jason Smith, Xiaofeng Yang, and Ro Moschitti. 2008. Bart: A modular toolkit for coreference resolution. In *Proceedings of ACL, Demo Session*.

Wang, Rui and Guenter Neumann. 2008. An divide-and-conquer strategy for recognizing textual entailment. In *Proceedings of TAC*. Gaithersburg, Maryland.

Wang, Rui and Yajing Zhang. 2008. Recognizing textual entailment with temporal expressions in natural language texts. In *Proceedings of IEEE*.