Efficient Semantic Deduction and Approximate Matching over Compact Parse Forests

Roy Bar-Haim[1], **Jonathan Berant**[2], **Ido Dagan**[1],**Iddo Greental**[3], **Shachar Mirkin**[1], **Eyal Shnarch**[1] **and Idan Szpektor**[1]
[1]Computer Science Department, Bar-Ilan University, Ramat Gan 52900, Israel
[2]School of Computer Science, Tel-Aviv University, Ramat Aviv 69978, Israel
[3]Department of Linguistics, Tel-Aviv University, Ramat Aviv 69978, Israel

Abstract

Semantic inference is often modeled as application of *entailment rules*, which specify generation of entailed sentences from a source sentence. Efficient generation and representation of entailed consequents is a fundamental problem common to such inference methods. We present a new data structure, termed *compact forest*, which allows efficient generation and representation of entailed consequents, each represented as a parse tree. Rule-based inference is complemented with a new approximate matching measure inspired by tree kernels, which is computed efficiently over compact forests. Our system also makes use of novel large-scale entailment rule bases, derived from Wikipedia as well as from information about predicates and their argument mapping, gathered from available lexicons and complemented by unsupervised learning.

1 Introduction

Textual entailment is concerned with inferring one textual form (the *hypothesis*) from another (the *text*). Most commonly, such inferences are made by applying some kind of transformations or substitutions to the text representation. Transformations are often represented as *entailment rules* (or *inference rules*), which capture semantic knowledge about paraphrases, synonyms, syntactic variations etc., e.g. (Lin and Pantel, 2001; de Salvo Braz et al., 2005; Romano et al., 2006).

While many text understanding applications employ transformations and inference rules, their use is typically limited, application-specific, and quite heuristic. Such practices lack a clear formalism specifying how inference knowledge should be represented and applied. Our long term research goal is development of generic, robust semantic inference engines, based on principled and clear formulation of knowledge representation and inference mechanisms. Such inference engines could be plugged into the various text understanding applications, encapsulating all the required inferences. In (Bar-Haim et al., 2007a) we made a step in this direction by introducing a generic formalism for semantic inference over parse trees. We defined a proof system, based on application of entailment rules, which provides a principled and uniform mechanism for incorporating a wide variety of inference knowledge.

Formally, application of an entailment rule corresponds to generation of a new sentence, a consequent, semantically entailed by the source sentence. The inferred consequent itself may be the source of further rule applications and so on. However, explicitly generating a new sentence (or parse tree) for each rule application (as in (Bar-Haim et al., 2007a)) may quickly lead to exponential explosion. Consider, for example, a sentence containing five content words, each one having two synonyms. The number of derivable sentences would be 3^5 (including the source sentence). Thus, representing each entailed sentence explicitly leads to severe efficiency problems. Intuitively, we would like to add for each rule application just the entailed part (e.g. the synonymous words) to the source sentence representation. However, we still want the inference process to be formulated over individual sentences, rather than over some superposition of sentences whose seman-

tics is unclear.

The current work proposes a solution to this inherent problem. We present a novel data structure, termed *compact forest*, which allows efficient generation and representation of entailed consequents, where each consequent is represented by a dependency tree. We show how all inference operations defined in our framework, including an extension to handle co-reference, can be implemented over compact forests.

In addition to knowledge-based deduction, textual entailment systems usually employ some form of approximate matching mechanisms aiming to bridge over inevitable knowledge gaps. We present a novel approximate matching measure inspired by tree kernels, and show how it can be efficiently calculated over compact forests.

With the compact forest, large-scale rule application based on massive rule bases becomes feasible. We present two novel sources for entailment rules, each containing millions of rules: one is a resource of lexical rules, derived from Wikipedia. The other is a resource for entailment between predicates (both verbal and nominal), including mapping of their arguments. This resource is derived by integrating information from WordNet and Nomlex-plus, which is complemented by corpus-based unsupervised rule learning.

2 System Overview

Our system consists of three stages: (a) Preprocessing of text and hypothesis (b) Knowledge-based inference, and (c) Feature extraction and entailment classification. Preprocessing includes dependency parsing using Minipar (Lin, 1998), named-entity recognition using the Stanford NER (Finkel et al., 2005) and co-reference resolution using OpenNLP[1]. In addition, we have developed a normalization module for numbers, which is applied first. The result is a set of dependency trees for the text (and a single tree for the hypothesis) with additional annotations of co-reference and named-entities. Next, our inference engine applies to the text trees entailment rules derived from diverse knowledge sources, aiming to bring it closer to the hypothesis. The set of inferred trees is efficiently represented as a *compact forest* (see next section). A small set of 'canonization' entailment rules (such as passive-to-active transformation) is also applied to the hypothesis. The resulting pair $(\mathcal{F}, \mathcal{H})$ of the compact forest inferred from the text and the transformed hypothesis tree is passed to the final stage - feature extraction and entailment classification. The features measure coverage of $\mathcal{H}$ by $\mathcal{F}$, and detect various types of mismatches between $\mathcal{H}$ and $\mathcal{F}$.

[1] http://opennlp.sourceforge.net/

3 Inference

This section starts with briefly (and somewhat informally) reviewing the inference formalism introduced in (Bar-Haim et al., 2007a) (Section 3.1), continues with presenting its new efficient implementation using compact forests (Section 3.2) and concludes with additional enhancements for handling co-reference (Section 3.3).

3.1 Inference Framework Review

Given two syntactically parsed text fragments, termed *text* and *hypothesis*, the inference system tries to *generate* the hypothesis from the text by applying a sequence of *entailment rules*. Entailment rules represent tree transformations, and the inference system aims to transform the text into the hypothesis through a sequence of intermediate parse trees, similar to a proof process in logic.

More specifically, text and hypothesis are represented as dependency trees, where nodes are annotated with lemma and part-of-speech, and edges are annotated with dependency relation. A rule '$L \rightarrow R$' is primarily composed of two templates, termed *left-hand-side* (L), and *right-hand-side* (R). *Templates* are dependency subtrees which may contain *variables*, matching any lemma. Figure 1(a) shows a sample rule, representing passive-to-active transformation. Applying this rule to the sentence *'Beautiful Mary was seen by John yesterday'* generates the sentence *'John saw beautiful Mary yesterday'*.

Rule application first matches L in the source tree s. A successful match *binds* L's variables to nodes or subtrees in s. In our example, the variable V binds to the verb *see*, $N1$ binds to *Beautiful Mary* and $N2$ binds to *John*. Based on this binding, R's variables are instantiated. In addition, a rule may specify *alignments* between nodes in L and

R, indicating that any modifiers of the source node which are not part of the rule structure should also be copied to the target node. In our example, the alignment between the V nodes indicates that *yesterday* (modifying *see*) should be copied to the generated sentence.

The final step in rule application is generation of the derived tree d. Let r be the instantiated R, and l be the subtree matched by L. Our formalism has two methods for generating the derived tree: *substitution* and *introduction*, as specified by the rule type. With *substitution* rules, the derived tree d is obtained by making a local modification to the source tree s. Except for this modification s and d are identical (a typical example is a lexical rule, such as *buy* $\rightarrow$ *purchase*). For this rule type, d is formed by copying s while replacing l (and the descendants of l's nodes) with r. This is the case for the passive rule. By contrast, *introduction* rules are used to make inferences from a subtree of s, while the other parts of s are ignored and do not affect d. A typical example is inference of a proposition embedded as a relative clause in s. In this case, the derived tree d is simply taken to be r.

3.2 Efficient Inference over Compact Forests

Generating explicitly a new dependency tree at each inference step provides an intuitively clear formulation for inference at the lexical-syntactic level. However, as shown in the introduction, it leads to exponential explosion. Thus, while we would like to keep our inference formalism, it is crucial to improve its implementation efficiency.

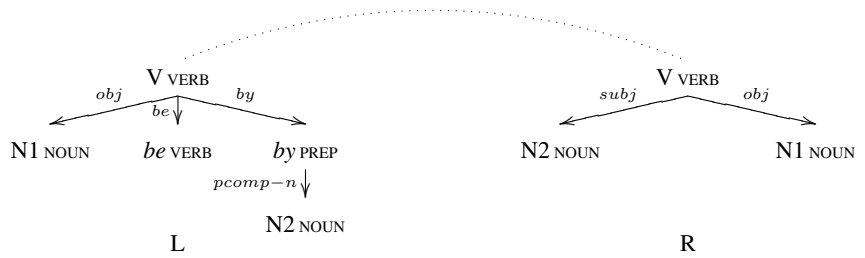
In our new implementation, each rule application generates only the right-hand side, R. In this respect, we follow the general idea of Braz et al. (2005), who "augmented" the text representation only with the right-hand-side of the applied entailment rule. However, in their work, both rule application and the semantics of the resulting "augmented" structure were not fully specified. By contrast, our method is fully formalized, and the semantics of the resulting structure, termed *compact forest* is the same as before: it represents a collection of trees, including the original text trees as well as inferred trees (consequents). Although common subtrees are shared, the distinction between individual consequents is kept. Matching a subtree against the forest only succeeds if the subtree is fully matched within a particular consequent (rather than spread over multiple consequents). Independent rules are applied in parallel, reducing complexity from exponential to linear.

The *Compact Forest* data structure. A compact forest $\mathcal{F}$ represents a set of dependency trees. Figure 1(b) shows an example of a compact forest, containing both the source sentence and the derived sentence, after applying the passive-to-active transformation. Formally, $\mathcal{F} = (\mathcal{N}, \mathcal{E})$ is composed of a set $\mathcal{N}$ of nodes, and a set $\mathcal{E}$ of *disjunction edges* (*d-edges* in short), an extension of dependency edges.

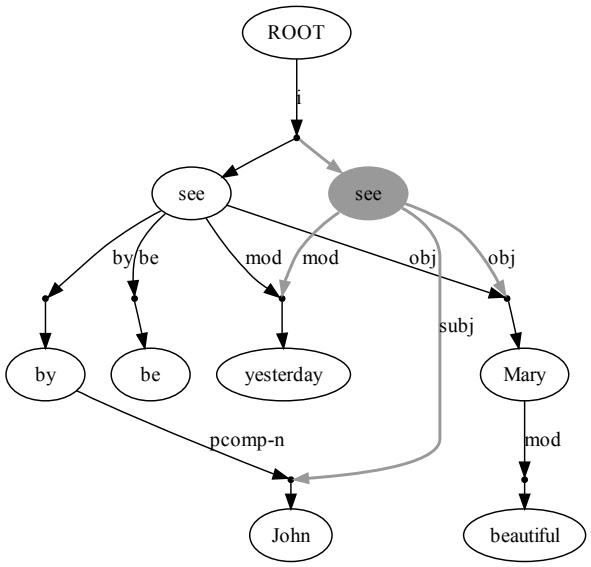
d-edges specify choice among multiple source nodes, as well as choice among multiple target nodes. In our example, we may choose to connect the root to the left *see*, resulting in the source passive sentence, or to the right *see*, resulting in the derived active sentence. Similarly, since *John* appears in both sentences, its incoming edge specifies a choice between two possible parents. It is important to note that in both cases the alternatives are *disjoint*, namely, belong to different trees in the forest. More formally, A *d-edge* d has a set of source nodes, S_d and a set of target nodes, T_d. Each source node n within d is annotated with a dependency relation, $rel_d(n)$. A disjunction edge represents a set of disjoint dependency edges, whose endpoints are given by the cartesian product $S_d \times T_d$, and the dependency relation of each edge is determined by the source node (given by the rel function).

$\mathcal{F}$ is assumed to be acyclic, and has an (artificial) root node with a single outgoing d-edge, where the target nodes are the roots of the forest trees. Except for the root node, all nodes have a single incoming d-edge. These properties are guaranteed by the initialization of the compact forest, and maintained by the inference operations applied to it, as we discuss next.

Since a dependency edge is a special case of a d-edge (with a single source and target), transforming a dependency tree into a compact forest is trivial. The set of individual trees encoded by the forest can be recovered by traversing $\mathcal{F}$ starting from the root, and for each outgoing d-edge d choosing one of the target nodes in T_d. Since the compact forest is acyclic, this procedure is guaranteed to terminate,



(a) Passive to active transformation (substitution rule). The dotted arc represents alignment.



(b) A compact forest containing both the source sentence "beautiful Mary was seen by John yesterday", and the inferred sentence "John saw beautiful Mary yesterday" resulting from the application of the rule in (a). d-edges are shown as point nodes, connected by incoming edges to source nodes and by outgoing edges to target nodes. Nodes and d-edge parts added to the compact forest by this rule application are shown in gray. Parts of speech are omitted.

Figure 1: Applying an entailment rule to a compact forest.

and each resulting extracted structure will be a tree. Each such sequence of choices leads to a different tree, and the set of all derivable trees makes up the forest. As mentioned above, the choice between the two "see" nodes leads to two different trees.

Forest initialization $\mathcal{F}$ is initialized with the set of dependency trees generated from the text at preprocessing, with their roots connected to the forest root.

Rule application Application of a rule '$L \rightarrow R$' to $\mathcal{F}$ begins by matching L in $\mathcal{F}$. A match m of a tree (L in this case) in a compact forest is a 1-1 function mapping each node in the tree to a compatible node in the forest, so that for each edge $s \rightarrow t$ in the tree with dependency relation dep , there is a d-edge d in the forest such that $m(s) \in S_d$, $m(t) \in T_d$ and $rel_d(m(s)) = dep$. As noted above, a tree should be matched only as a subtree of a particular tree in the forest. In other words, the match must not include disjoint edges. This requirement is realized by not allowing multiple edges to be matched in the same d-edge. We denote by l the subtree in $\mathcal{F}$ matched by L.

Next, a copy r of R is added to $\mathcal{F}$. We now need to instantiate each variable X in r. For *leaf variables*, defined as variables which are leaf nodes in both L and R, the subtree they bind to can simply be shared with r. Let t be the subtree we would like to share, and d be the incoming d-edge of its root. Then we simply add the parent p of X in r to set of source nodes in d, and set $rel_d(p)$ to the relation between p and X (in r). Then X itself can be removed from r. In our example, since $N1$ and $N2$ are leaf variables,

beautiful Mary and *John* are shared this way. Modifiers of aligned nodes such as *yesterday* are shared in an analogous fashion.

If X is not a leaf variable (e.g. V in the example), it cannot be shared this way, because it would typically have different modifiers in L and R. Therefore, X is instantiated with a copy of the source node (as with the duplication of *see*). However, defining alignment between the X's occurrences in L and R allows sharing of modifiers which are not part of the rule structure, such as *yesterday*. Such alignments are implicitly assumed for non-leaf variables.

Finally, r is set as an alternative to either l (for substitution rules), or to the other trees in the forest (for introduction rules). In the former case, r's root is added to the target node set of l's root incoming d-edge. In the latter case, it is added to the target node set of the forest root outgoing d-edge.

It can be verified that if the forest is acyclic (as with our initialization), it will remain acyclic after each rule application. Applying the rule in our example added only a single node and linked it to four d-edges, compared to duplicating the whole tree if all trees are represented explicitly, as in our RTE-3 system.

3.3 Handling Co-Reference

We extended our framework to handle co-reference. Co-reference is defined as an equivalence relation between complete subtrees, which corresponds to a co-reference chain. Each subtree is represented by its root node. To allow inference based on co-reference relation, we introduced *co-reference substitution*, an inference mechanism similar to our substitution rules, which allows for each pair of co-referring subtrees to replace each other. Given co-reference information obtained from preprocessing, all substitutions in which the substituting subtree is not a pronoun are performed prior to any rule application. Similar to application of substitution rules, in co-reference substitution the substituting subtree is cloned and its root is added as a target node of the incoming d-edge of the replaced tree.

4 Entailment Rules

Our inference engine applies diverse types of entailment rules, derived from various sources. After a brief review of rule types retained from our RTE-3 system, this section describes three novel types of entailment rules added to this year's system: lexical rules from Wikipedia, lexical-syntactic rules from WordNet, and new annotation rules for polarity.

4.1 RTE-3 Rules

The following rule types were retained from our RTE-3 system (Bar-Haim et al., 2007b): (a) *Syntactic rules:* These rules capture entailment inferences associated with common syntactic structures, such as conjunction, relative clause, apposition, etc. (b) *WordNet lexical rules:* Lexical rules were derived from WordNet[2] using the *synonym*, *hyponym*, *hyponym instance* and *derivation* relations. (c) *DIRT rules:* The DIRT algorithm (Lin and Pantel, 2001) learns entailment rules between binary predicates, e.g. 'X explain to Y→X talk to Y'. We used the version described in (Szpektor and Dagan, 2007), which learns canonical rule forms.

4.2 Lexical Entailment Rules from Wikipedia

Wikipedia is becoming a popular resource for extraction of semantic relationships. Recently, Shnarch (2008) created an extensive resource of lexical entailment rules from Wikipedia, using several extraction methods. This resource consists of 8 million rules, and was found to be fairly accurate. We used rules extracted by the following methods, which were found to be the most accurate: (a) *Be-Complement*: as the first sentence in Wikipedia typically consists of a definition of the title, this extraction method creates a rule between noun complements of the verb *be* in the sentence and the article title, e.g. *'Adi Shamir is an Israeli Cryptographer'*; (b) *Title Parenthesis*: a common convention in Wikipedia for ambiguous titles is adding a descriptive term in parenthesis at the end of the title, like *Graph (Mathematics)*. In such cases a rule is created between the main part of the title and its parenthesized description; (c) *Redirect*: a bidirectional rule based on Wikipedia's redirections between user queries and the title they are redirected to (e.g. *Yankee land* is redirected to *United States*); (d) *Link*: a rule between a link in the text and the article title it links to. For example,

[2]We used the MIT Java WordNet Interface, http://projects.csail.mit.edu/jwi/

Gestaltist ⇒ *Gestalt psychology*. These methods extracted about 5.5 million rules, with precision of about 80%.

4.3 Lexical Syntactic Rules from WordNet

WordNet (Fellbaum, 1998) is commonly used for inducing lexical entailment rules from relations such as hypernyms and synonyms. However, for rules between predicates, e.g. between verbs, there are benefits to using lexical-syntactic rules instead of lexical rules. First, some rules need also argument mapping (not just lexical substitution), e.g. '*buy Y for Z* ⇒ *pay Z for Y*'. Second, verbs connected by WordNet may have different allowed subcategorization frames, which may result in incorrect inferences due to changes of the argument roles. For example, from the text "*The car crashed the wooden fence*" we can incorrectly infer that "*The car broke*" using WordNet lexical rule '*crash* ⇒ *break*'.

Thus, where possible, we want to consider information about arguments as well when inducing rules between predicates from WordNet. We extract argument mapping between verbs from automatically generated corpus-based rule-set of entailment rules between templates. We use unary-DIRT as our learning method for such a rule-set (Szpektor and Dagan, 2008).

Predicates can be expressed not only by verbs, but also by nouns (nominalizations), e.g. *acquisition* and *employment*. WordNet contains relations between verbs and their related nominalizations, e.g. '*acquire* ↔ *acquisition*'. However, to use these relations we need argument mapping between the verb and its nominalizations. We extract these mappings from Nomlex-plus (Meyers et al., 2004), a database of about 6000 English nominalizations, together with the allowed sub-categorization frames for each nominalization. We thus generate rules from nouns to verbs and vice versa, e.g. '*X acquire Y* ⇔ *X's acquisition of Y*'. We also use hypernym relations between nouns to generate more rules between nouns and verbs and between verbs and verbs, e.g. '*homicide of X by Y* ⇒ *killing of X by Y* ⇔ *Y kill X*'.

Finally, we add allowed sub-categorization frames for each verb and nominalization, and require for lexical substitution rules between verbs or between a verb and its nominalization that both have the same frame. For example, 'break' can substitute 'crash' only when either both appear as transitive or both are intransitive. We obtain allowed subcategorization frames for verbs from WordNet frames and allowed frames for nominalizations from Nomlex-plus.

4.4 Annotation Rules for Polarity

In addition to inference rules, our framework includes annotation rules, which do not generate new consequents, but add features to parse tree nodes. Annotation rules do not have an R, but rather each node of L may contain annotation features. If L is matched in a tree then the annotations are copied to the matched nodes. We use annotation rules to handle negation and modality phenomena. We achieve this by marking the *polarity* status on verbs and nouns, which may take the value of *positive*(+), *negative*(-) or *unknown(?)*. As section 6.1 explains further, we utilize polarity mismatch between $\mathcal{H}$ and $\mathcal{F}$ to detect non-entailment. Some examples of polarity annotation are shown below:

John called$^{(+)}$ Mary.
John *didn't* call$^{(-)}$ Mary.
John *forgot* to call$^{(-)}$ Mary.
John *wanted* to call$^{(?)}$ Mary.

Annotation rules capture *negative* polarity that is expressed by verbal negation in its full or contracted form, as well as implied by certain determiners and nouns. In addition, based on the PARC polarity lexicon (Nairn et al., 2006), we created rules for special classes of verbs such as *forget* which induce negative polarity context, as in the example. *Unknown* polarity is expressed by the use of modal verbs, as well as conditional sentences and modal adverbials. We also compiled from VerbNet a list of verbs such as *want*, *suspect*, and *attempt* which indicate unknown polarity of their complement predicate, as in the above example. Annotation rules are applied to both $\mathcal{H}$ and the initial $\mathcal{F}$ prior to any inference rule application, and the polarity feature is copied through the application of inference rules either as part of node copying or due to node alignment.

5 Search

Finding a sequence of entailment rules that derives the hypothesis from the text is a search problem, which we addressed in our system as follows. Sev-

eral iterations of rule application are performed, where at each iteration we first find all rule matches, and then apply all matched rules. To avoid repeated applications, we mark newly added nodes at each iteration, and in the next iteration consider only matches containing new nodes. We define two types of rule matches: *L-match*, where L is matched in $\mathcal{F}$ and *LR-match*, where L is matched in $\mathcal{F}$ and R is matched in $\mathcal{H}$. LR-matches attempt to bridge between $\mathcal{F}$ and $\mathcal{H}$ in a single step, while L-matches allow chaining of rule applications, but substantially increase the search space. For our RTE-4 submission we first applied three iterations of L-matches using only syntactic rules[3], followed by one iteration of LR-matches using all other rule types. In the future we plan to further experiment with other search strategies.

6 Feature Extraction and Classification

After entailment rules from various knowledge sources have been applied to $\mathcal{F}$, features are extracted from the resulting $(\mathcal{F}, \mathcal{H})$ pair. Features can be broadly categorized into two subsets: (a) lexical features that solely depend on the lexical items in $\mathcal{F}$ and $\mathcal{H}$, and (b) lexical-syntactic features that also take into account coverage of dependency relations in $\mathcal{H}$.

6.1 Lexical Features

Coverage Features The following features check if the words in $\mathcal{H}$ are present in $\mathcal{F}$. We assume that good lexical coverage should correlate well with entailment. The first five features measure the proportion of uncovered distinct verbs[4]/nouns/adjectives and adverbs/Named Entities/numbers in $\mathcal{H}$. The last coverage feature measures the proportion of covered distinct words[5] in $\mathcal{H}$.

Polarity Features We conjecture that mismatched polarity is indicative of non-entailment. Three binary features are thus extracted: (1) Whether there exists a noun/verb in $\mathcal{H}$ such that all its matches in $\mathcal{F}$ have mismatching polarity. (2) Whether there exists a noun/verb in $\mathcal{H}$, which has a match in $\mathcal{F}$ with opposite polarity (i.e. *positive* vs. *negative*), but doesn't have a match with the same polarity. (3) Whether there exists a noun/verb in $\mathcal{H}$, which has a match in $\mathcal{F}$ with incompatible but non-opposite polarity (i.e. the polarity of one of the nouns/verbs is *unkonwn*), but doesn't have a match with the same polarity. Note that the first feature is the disjunction of the second and third features.

6.2 Local Lexical-Syntactic Features

Two types of local lexical-syntactic features are extracted: (a) predicate-argument features (b) edge coverage features.

Predicate-argument features If $\mathcal{F}$ entails $\mathcal{H}$, then the predicates in $\mathcal{H}$ should be matched in $\mathcal{F}$ along with their arguments. Predicates include verbs (aside for the verb "be") or subject complements in copular sentences (For example, *smart* in *Joseph is smart*). Arguments are the daughters of the predicate node in $\mathcal{H}$.[6] Four features are computed for each $(\mathcal{F}, \mathcal{H})$ pair. We categorize every predicate in $\mathcal{H}$ that has a match in $\mathcal{F}$ to one or more of four possible categories:

1. *complete match* - a matching predicate exists in $\mathcal{F}$ with matching arguments and dependency relations.
2. *partial match* - a matching predicate exists in $\mathcal{F}$ with some matching arguments and dependency relations.
3. *opposite match* - a matching predicate exists in $\mathcal{F}$ with some matching arguments but incorrect dependency relations.
4. *no match* - no matching predicate in $\mathcal{F}$ has any matching arguments.

If some predicate is categorized as a *complete match* it will not be in any other category. Finally, we compute the four features for the $(\mathcal{F}, \mathcal{H})$ pair: the proportion of predicates in $\mathcal{H}$ that have a *complete match* in $\mathcal{F}$, and three binary features checking if there is any predicate in $\mathcal{H}$ categorized as a *partial match/opposite match/no match*. Since the *subject* and *object* arguments are crucial for textual entailment, we compute four similar features only for the subset of predicates which have these arguments (while ignoring other arguments).

[3] cf. section 4.1

[4] The verb 'be' is discarded during features extarction

[5] Only the following Minipar parts-of-speech are taken into account: Verb, Noun, Adverb, Adjective and Other

[6] When the dependent is a preposition or a clause we take the complement of the preposition or the head of the clause respectively as the dependent.

Edge coverage features We say that an edge in $\mathcal{H}$ is *matched* in $\mathcal{F}$ if there is an edge in $\mathcal{F}$ with matching *relation*, *source* node and *target* node. We say an edge in $\mathcal{H}$ is *loosely-matched* if there is some path in $\mathcal{F}$ from a matching *source* node to a matching *target* node. Based on these definitions we extract two features: the proportion of $\mathcal{H}$ edges *matched/loosely matched* in $\mathcal{F}$.[7]

6.3 Tree-Kernel Inspired Global Feature

All the lexical-syntactic features hitherto presented focus on local syntactic dependencies. We would like to compute the coverage of $\mathcal{H}$ by $\mathcal{F}$ for larger and more complex substructures. Kernel functions are often used to compute the similarity between complex structures such as dependency trees. Collins and Duffy (2001) introduced a dependency tree kernel, which measures the similarity between two trees by counting their common subtrees.

While the general idea of finding common subtrees seems attractive as an approximate syntactic measure, applying the dependency tree kernel measure as-is for textual entailment has several drawbacks: First, this measure, like any kernel function, is symmetric, while entailment is a directional relation: we want all $\mathcal{H}$ subtrees to be matched in $\mathcal{F}$, but the opposite is not required. Second, the dependency tree kernel counts the number of times each $\mathcal{H}$ subtree appears in $\mathcal{F}$ while we only wish to know if it appears in $\mathcal{F}$ or not. This is important since $\mathcal{F}$ may contain redundant structures resulting from rule application. Third, Collins and Duffy assign equal weight to all nodes in the dependency tree, while it is likely that the closer the node to the root, the more significant it is for entailment. Finally, we need to adapt the tree kernel measure to operate on a compact forest and a tree, rather than on a tree pair.

To address these issues, we have adapted the dependency tree kernel to our needs. We measure how well the subtrees in $\mathcal{H}$ are covered by $\mathcal{F}$, weighted according to the proximity to the root of $\mathcal{H}$. The feature is computed in two steps. First, we find for each pair of nodes $n_H \in \mathcal{H}$ and $n_F \in \mathcal{F}$ the maximal-weight subtree rooted at n_H and n_F, with respect to some node weighting function, and store its weight in a table. Note that unlike tree kernels, we do not count the number of common substructures[8], but only consider the maximal-weight subtree. In the second step, we compute an overall structural coverage score based on these weights.

Table Construction For each node pair $n_H \in \mathcal{H}$ and $n_F \in \mathcal{F}$, we would like to compute $MaxWght(n_H, n_F)$, the maximal weight of a common subtree rooted at n_H and n_F. The weight of a subtree is the sum of its node weights, given by some node weighting function $wght(n)$. Specifically, we define $wght(n)$ as 1+the height of $\mathcal{H}$ minus the distance of n from the root.

$MaxWght$ can be defined recursively as:

$$MaxWght(n_H, n_F) = \begin{cases} 0 & \text{if } n_H \neq n_F \\ wght(n_H) + & \\ BM(n_H, n_F) & \text{otherwise} \end{cases}$$

where $BM(n_H, n_F)$ is the weight of the best match of n_H daughter subrees in the daughter subtrees of n_F, formally defined as:

$$BM(n_H, n_F) = \max_{m \in matches(n_H, n_F)} \sum_{(h,f):m(h)=f, h \neq n_H} MaxWght(h, f)$$

$matches(n_H, n_F)$ is the set of all matches[9] in $\mathcal{F}$ of any subtree composed of n_H and some subset (possibly empty) of its daughters, where n_H iteslf is mapped to n_F.

We efficiently compute $MaxWght$ for each pair of nodes $n_H \in \mathcal{H}$ and $n_F \in \mathcal{F}$ bottom-up using dynamic programming. Notice that finding the best match cannot be done independently for each outgoing edge of n_H, otherwise multiple edges in $\mathcal{H}$ could be mapped to the same d-edge in $\mathcal{F}$. Nevertheless, computing the best match can still be done efficiently since this is an instance of the *assignment problem*: we look for a match from edges in $\mathcal{H}$ to d-edges in $\mathcal{F}$ that maximizes the weight of the match.

Feature Computation After computing the table, the global similarity GS can be computed. For each node n_H in $\mathcal{H}$, we compute the maximal weight of a subtree rooted in n_H that is matched in $\mathcal{F}$. GS is obtained by summing these weights over n_H nodes:

$$GS(\mathcal{H}, \mathcal{F}) = \sum_{n_H \in \mathcal{H}} \max_{n_F \in \mathcal{F}} MaxWght(n_H, n_F)$$

[7] We only look at a subset of the edges labeled with relevant dependency relations.

[8] Though this could be easily computed.

[9] cf. match definition in section 3.2

and the final feature is simply the normalized similarity $\frac{GS(\mathcal{H},\mathcal{F})}{GS(\mathcal{H},\mathcal{H})}$.

6.4 Feature Selection and Classification

We used 10-fold cross validation for feature selection and omitted the following four features from the final training set: We omitted the feature recording the proportion of uncovered distinct verbs from the coverage features. We omitted feature (1) and (3) from the polarity features. From the predicate-argument features we omitted the binary feature recording *no match* instances. We used the SVMperf package (Joachims, 2005; Joachims, 2006) to train a linear kernel SVM and the WEKA package (Witten and Frank, 2005) to train a decision tree (J48) on the training set.

7 Evaluation and Analysis

Our system accuracy results for two-way entailment classification are summarized in Table 1. The first three rows correspond to the submitted runs. The rest of the results reported in this section were obtained with our current version of the system, which includes some minor fixes and improvements, using all the features (no feature selection). The results show that adding more training data does not improve accuracy. The best results were obtained when training only on the RTE-3 development set (row 4), improving over our best submitted run by 2.1%. SVM was found to perform a little better than the J48 decision tree.

In order to better understand the impact of each inference type, including the entailment rule bases and co-reference substitution, we conducted additional experiments, summarized in Table 2. The results for ablation tests in which we excluded each of the inference types and measured accuracy loss are shown in rows 1-7. Each of the entailment rule bases contributed to the overall accuracy, while co-reference substitution somewhat degraded the accuracy, probably because the quality of the NP co-reference resolution tool we used is not sufficient yet. Without co-reference, accuracy is improved to 61.1%. Row 8 shows the accuracy drop resulting from skipping inference altogether, that is, classification is performed right after preprocessing. Thus, the overall contribution of rule-based inference (without co-reference) is 3.4%. Furthermore, we found that inference increased the average node coverage of $\mathcal{H}$ in $\mathcal{F}$ from 68% to 76%, and average edge coverage from 26% to 35%. Overall, the contribution of knowledge-based inference is substantial. We also tested our system on RTE-3, and obtained quite competitive results: compared to our 66.6%, only 3 teams out of the 26 who participated in RTE-3 scored higher than 67%, and three more systems scored around 67%.

Compared to our RTE-3 system, the RTE-4 system works faster by an order of magnitude, while applying substantially more entailment rules. In future work we plan to conduct further experiments to quantify the impact of the compact forest data structure on inference efficiency in terms of execution time and memory consumption.

#	Classifier	Training Set	Accuracy
1	SVM	D2+D3+T2+T3	58.3%
2	J48	D2+D3+T2+T3	57.3%
3	SVM	D3+T3	58.4%
4	**SVM**	**D3**	**60.5%**

Table 1: Results on RTE-4 test set (2-way entailment classification). D2 denotes RTE-2 development set, T3 is RTE-3 test set, etc.

#	Inference Type	ΔAccuracy
1	WordNet (Lexical)	0.8%
2	WordNet (Lex. Syn.)	0.7%
3	Wikipedia	1.0%
4	DIRT	0.9%
5	Syntactic	0.4%
6	Polarity	0.9%
7	Co-reference	-0.6%
8	All	2.8%

Table 2: Contribution of various inference types. Rows 1-7 show accuracy loss obtained for removing each inference type (ablation tests). Row 8 shows accuracy loss obtained for skipping inference altogether.

8 Conclusion

The main contribution of the current work is a new data structure for inference over parse trees, which is both efficient and semantically sound. Using this data structure, we were able to integrate many entailment rules from diverse sources, and show their

contribution to overall performance. In future work, we plan in-depth analysis of our system and its various rule sources, as well as adding new rule sources and experimenting with various search strategies.

Acknowledgements

This work was partially supported by the Negev Consortium of the Israeli Ministry of Industry, Trade and Labor, the PASCAL-2 Network of Excellence of the European Community FP7-ICT-2007-1-216886, the FIRB-Israel research project N. RBIN045PXH and the Israel Science Foundation grant 1095/05. Jonathan Berant is grateful to the Azrieli Foundation for the award of an Azrieli Fellowship.

References

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

Roy Bar-Haim, Ido Dagan, Iddo Greental, Idan Szpektor, and Moshe Friedman. 2007b. Semantic inference at the lexical-syntactic level for textual entailment recognition. In *ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*.

Michael Collins and Nigel Duffy. 2001. Convolution kernels for natural language. In *Advances in Neural Information Processing Systems 14*, pages 625–632. MIT Press.

Rodrigo de Salvo Braz, Roxana Girju, Vasin Punyakanok, Dan Roth, and Mark Sammons. 2005. An inference model for semantic entailment in natural language. In *AAAI*, pages 1043–1049.

Christiane Fellbaum, editor. 1998. *WordNet: An Electronic Lexical Database*. Language, Speech and Communication. MIT Press.

Jenny R. Finkel, Trond Grenager, and Christopher Manning. 2005. Incorporating non-local information into information extraction systems by Gibbs sampling. In *ACL '05: Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 363–370, Morristown, NJ, USA.

Thorsten Joachims. 2005. A support vector method for multivariate performance measures. In Luc De Raedt and Stefan Wrobel, editors, *Proceedings of the 22nd International Conference on Machine Learning (ICML 2005), August 7-11, 2005, Bonn, Germany*, pages 377–384. ACM Press, New York, NY, USA.

Thorsten Joachims. 2006. Training linear SVMs in linear time. In Tina Eliassi-Rad, Lyle H. Ungar, Mark Craven, and Dimitrios Gunopulos, editors, *KDD*, pages 217–226. ACM.

Dekang Lin and Patrick Pantel. 2001. Discovery of inference rules for question answering. *Natural Language Engineering*, 7(4):343–360.

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

A. Meyers, R. Reeves, Catherine Macleod, Rachel Szekeley, Veronkia Zielinska, and Brian Young. 2004. The cross-breeding of dictionaries. In *Proceedings of LREC*.

Rowan Nairn, Cleo Condoravdi, and Lauri Karttunen. 2006. Computing relative polarity for textual inference. In *Proceedings of International workshop on Inference in Computational Semantics (ICoS-5)*.

Lorenza Romano, Milen Kouylekov, Idan Szpektor, Ido Dagan, and Alberto Lavelli. 2006. Investigating a generic paraphrase-based approach for relation extraction. In *Proceedings of EACL 2006*.

Eyal Shnarch. 2008. Lexical entailment and its extraction from Wikipedia. Master's thesis, Computer Science Department, Bar-Ilan University, Israel.

Idan Szpektor and Ido Dagan. 2007. Learning canonical forms of entailment rules. In *Proceedings of RANLP*.

Idan Szpektor and Ido Dagan. 2008. Learning entailment rules for unary templates. In *Proceedings of COLING*.

Ian H. Witten and Eibe Frank. 2005. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, San Francisco, 2nd edition.