Efficient Search for Transformation-based Inference

Asher Stern[§], **Roni Stern**[‡], **Ido Dagan**[§], **Ariel Felner**[‡]
§ Computer Science Department, Bar-Ilan University
‡ Information Systems Engineering, Ben Gurion University
astern7@gmail.com
roni.stern@gmail.com
dagan@cs.biu.ac.il
felner@bgu.ac.il

Abstract

This paper addresses the search problem in textual inference, where systems need to infer one piece of text from another. A prominent approach to this task is attempts to transform one text into the other through a sequence of inference-preserving transformations, a.k.a. a proof, while estimating the proof's validity. This raises a search challenge of finding the best possible proof. We explore this challenge through a comprehensive investigation of prominent search algorithms and propose two novel algorithmic components specifically designed for textual inference: a gradient-style evaluation function, and a local-lookahead node expansion method. Evaluations, using the open-source system, BIUTEE, show the contribution of these ideas to search efficiency and proof quality.

1 Introduction

In many NLP settings it is necessary to identify that a certain semantic inference relation holds between two pieces of text. For example, in *paraphrase recognition* it is necessary to identify that the meanings of two text fragments are roughly equivalent. In *passage retrieval* for question answering, it is needed to detect text passages from which a satisfying answer can be inferred. A generic formulation for the inference relation between two texts is given by the *Recognizing Textual Entailment (RTE)* paradigm (Dagan et al., 2005), which is adapted here for our investigation. In this setting, a system is given two text fragments, termed "text" (T) and "hypothesis" (H), and has to recognize whether the hypothesis is entailed by (inferred from) the text.

An appealing approach to such textual inferences is to explicitly transform T into H, using a sequence of transformations (Bar-Haim et al., 2007; Harmeling, 2009; Mehdad, 2009; Wang and Manning, 2010; Heilman and Smith, 2010; Stern and Dagan, 2011). Examples of such possible transformations are lexical substitutions (e.g. "letter" → "message") and predicate-template substitutions (e.g. "X [verb-active] Y" → "Y [verb-passive] by X"), which are based on available knowledge resources. Another example is coreference substitutions, such as replacing "he" with "the employee" if a coreference resolver has detected that these two expressions corefer. Table 1 exemplifies this approach for a particular *T-H* pair. The rationale behind this approach is that each transformation step should preserve inference validity, such that each text generated along this process is indeed inferred from the preceding one.

An inherent aspect in transformation-based inference is modeling the certainty that each inference step is valid. This is usually achieved by a cost-based or probabilistic model, which quantifies confidence in the validity of each individual transformation and consequently of the complete chain of inference.

Given a set of possible transformations, there may be many transformation sequences that would transform T to H. This creates a very large search space, where systems have to find the "best" transformation sequence – the one of lowest cost, or of highest probability. To the best of our knowledge, this search challenge has not been investigated yet in a substan-

Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics, pages 283–291,
Jeju, Republic of Korea, 8-14 July 2012.

#	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.

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.

tial manner: each of the above-cited works described the search method they used, but none of them tried alternative methods while evaluating search performance. Furthermore, while experimenting with our own open-source inference system, BIUTEE[1], we observed that search efficiency is a major issue, often yielding practically unsatisfactory run-times.

This paper investigates the search problem in transformation-based textual inference, naturally falling within the framework of heuristic AI (Artificial Intelligence) search. To facilitate such investigation, we formulate a generic search scheme which incorporates many search variants as special cases and enable a meaningful comparison between the algorithms. Under this framework, we identify special characteristics of the textual inference search space, that lead to the development of two novel algorithmic components: a special lookahead method for node expansion, named *local lookahead*, and a gradient-based evaluation function. Together, they yield a new search algorithm, which achieved substantially superior search performance in our evaluations.

The remainder of this paper is organized as follows. Section 2 provides an overview of transformation-based inference systems, AI search algorithms, and search methods realized in prior inference systems. Section 3 formulates the generic search scheme that we have investigated, which covers a broad range of known algorithms, and presents our own algorithmic contributions. These new algorithmic contributions were implemented in our system, BIUTEE. In Section 4 we evaluate them empirically, and show that they improve search efficiency as well as solution's quality. Search performance is evaluated on two recent RTE benchmarks, in terms of runtime, ability to find lower-cost transformation chains and impact on overall inference.

[1] www.cs.biu.ac.il/~nlp/downloads/biutee

2 Background

Applying sequences of transformations to recognize textual inference was suggested by several works. Such a sequence may be referred to as a *proof*, in the sense that it is used to "prove" the hypothesis from the text. Although various works along this line differ from each other in several respects, many of them share the common challenge of *finding* an optimal proof. The following paragraphs review the major research approaches in this direction. We focus on methods that perform transformations over parse trees, and highlight the search challenge with which they are faced.

2.1 Transformation-based textual inference

Several researchers suggested using various types of transformations in order to derive H from T. Some suggested a set of predefined transformations, for example, *insertion*, *deletion* and *substitution* of parse-tree nodes, by which any tree can be transformed to any other tree. These transformations were used by the open-source system *EDITS* (Mehdad, 2009), and by (Wang and Manning, 2010). Since the above mentioned transformations are limited in capturing certain interesting and prevalent semantic phenomena, an extended set of tree edit operations (e.g., relabel-edge, move-sibling, etc.) was proposed by Heilman and Smith (2010). Similarly, Harmeling (2009) suggested a heuristic set of 28 transformations, which include various types of node-substitutions as well as restructuring of the entire parse-tree.

In contrast to such predefined sets of transformations, knowledge oriented approaches were sug-

gested by Bar-Haim et al. (2007) and de Salvo Braz et al. (2005). Their transformations are defined by knowledge resources that contain a large amount of *entailment rules*, or *rewrite rules*, which are pairs of parse-tree fragments that entail one another. Typical examples for knowledge resources of such rules are *DIRT* (Lin and Pantel, 2001), and *TEASE* (Szpektor et al., 2004), as well as syntactic transformations constructed manually. In addition, they used knowledge-based lexical substitutions.

However, when only knowledge-based transformations are allowed, transforming the text into the hypothesis is impossible in many cases. This limitation is dealt by our open-source integrated framework, BIUTEE (Stern and Dagan, 2011), which incorporates knowledge-based transformations (entailment rules) with a set of predefined tree-edits. Motivated by the richer structure and search space provided by BIUTEE, we adopted it for our empirical investigations.

The semantic validity of transformation-based inference is usually modeled by defining a *cost* or a *probability estimation* for each transformation. Costs may be defined manually (Kouylekov and Magnini, 2005), but are usually learned automatically (Harmeling, 2009; Mehdad, 2009; Wang and Manning, 2010; Heilman and Smith, 2010; Stern and Dagan, 2011). A global cost (or probability estimation) for a complete sequence of transformations is typically defined as the sum of the costs of the involved transformations.

Finding the lowest cost proof, as needed for determining inference validity, is the focus of our research. Textual inference systems limited to the standard tree-edit operations (insertion, deletion, substitution) can use an exact algorithm that finds the optimal solution in polynomial time under certain constraints (Bille, 2005). Nevertheless, for the extended set of transformations it is unlikely that efficient exact algorithms for finding lowest-cost sequences are available (Heilman and Smith, 2010).

In this harder case, the problem can be viewed as an AI search problem. Each state in the search space is a parse-tree, where the *initial state* is the text parse-tree, the *goal state* is the hypothesis parse-tree, and we search for the shortest (in terms of costs) *path* of transformations from the initial state to the goal state. Next we briefly review major concepts from the field of AI search and summarize some relevant proposed solutions.

2.2 Search Algorithms

Search algorithms find a path from an initial state to a goal state by *expanding* and *generating* states in a search space. The term *generating* a state refers to creating a data structure that represents it, while *expanding* a state means generating all its immediate derivations. In our domain, each state is a parse tree, which is expanded by performing all applicable transformations.

Best-first search is a common search framework. It maintains an *open list* (denoted hereafter as OPEN) containing all the generated states that have not been expanded yet. States in OPEN are prioritized by an *evaluation function*, $f(s)$. A best-first search algorithm iteratively removes the *best* state (according to $f(s)$) from OPEN, and inserts new states being generated by expanding this best state. The evaluation function is usually a linear combination of the shortest path found from the start state to state s, denoted by $g(s)$, and a heuristic function, denoted by $h(s)$, which estimates the cost of reaching a goal state from s.

Many search algorithms can be viewed as special cases or variations of best-first search. The well-known A* (Hart et al., 1968). algorithm is a best-first search that uses an evaluation function $f(s) = g(s) + h(s)$. Weighted A* (Pohl, 1970) uses an evaluation function $f(s) = w \cdot g(s) + h(s)$, where w is a parameter, while pure heuristic search uses $f(s) = h(s)$. K-BFS (Felner et al., 2003) expands k states in each iteration. Beam search (Furcy and Koenig, 2005; Zhou and Hansen, 2005) limits the number of states stored in OPEN, while Greedy search limits OPEN to contain only the single best state generated in the current iteration.

The search algorithm has crucial impact on the quality of proof found by a textual inference system, as well as on its efficiency. Next, we describe search strategies used in prior works for textual inference.

2.3 Search in prior inference models

In spite of being a fundamental problem, prior solutions to the search challenge in textual inference were mostly ad-hoc. Furthermore, there was no investigation of alternative search methods, and no

evaluation of search efficiency and quality was reported. For example, in (Harmeling, 2009) the order by which the transformations are performed is predetermined, and in addition many possible derivations are discarded, to prevent exponential explosion. Handling the search problem in (Heilman and Smith, 2010) was by a variant of greedy search, driven by a similarity measure between the current parse-tree and the hypothesis, while ignoring the cost already paid. In addition, several constraints on the search space were implemented. In the earlier version of BIUTEE (Stern and Dagan, 2011)[2], a version of beam search was incorporated, named hereafter BIUTEE-orig. This algorithm uses the evaluation function $f(s) = g(s) + w_i \cdot h(s)$, where in each iteration (i) the value of w is increased, to ensure successful termination of the search. Nevertheless, its efficiency and quality were not investigated.

In this paper we consider several prominent search algorithms and evaluate their quality. The evaluation concentrates on two measures: the runtime required to find a proof, and proof quality (measured by its cost). In addition to evaluating standard search algorithms we propose two novel components specifically designed for proof-based textual-inference and evaluate their contribution.

3 Search for Textual Inference

In this section we formalize our search problem and specify a unifying search scheme by which we test several search algorithms in a systematic manner. Then we propose two novel algorithmic components specifically designed for our problem. We conclude by presenting our new search algorithm which combines these two ideas.

3.1 Inference and search space formalization

Let t be a parse tree, and let o be a transformation. Applying o on t, yielding t', is denoted by $t \vdash_o t'$. If the underlying meaning of t' can indeed be inferred from the underlying meaning of t, then we refer to the application of o as *valid*. Let $O = (o_1, o_2, \ldots o_n)$ be a sequence of transformations, such that $t_0 \vdash_{o_1} t_1 \vdash_{o_2} t_2 \ldots \vdash_{o_n} t_n$. We write $t_0 \vdash_O t_n$, and say that t_n can be *proven* from t_0 by applying the sequence O. The *proof* might be valid, if all the transformations involved are valid, or invalid otherwise.

An inference system specifies a *cost*, $C(o)$, for each transformation o. In most systems the costs are automatically learned. The interpretation of a high cost is that it is unlikely that applying o will be valid. The cost of a sequence $O = (o_1, o_2, \ldots o_n)$ is defined as $\sum_{i=1}^{n} C(o)$ (or ,in some systems, $\prod_{i=1}^{n} C(o)$). Denoting by t_T and t_H the text parse tree and the hypothesis parse tree, a proof system has to find a sequence O with minimal cost such that $t_\mathrm{T} \vdash_O t_\mathrm{H}$. This forms a search problem of finding the lowest-cost proof among all possible proofs.

The search space is defined as follows. A *state* s is a parse-tree. The *start state* is t_T and the *goal state* is t_H. In some systems any state s in which t_H is embedded is considered as goal as well.

Given a state s, let $\{o^{(1)}, o^{(2)} \ldots o^{(m)}\}$ be m transformations that can be applied on it. *Expanding* s means generating m new states, $s^{(j)}$, $j = 1 \ldots m$, such that $s \vdash_{o^{(j)}} s^{(j)}$. The number m is called *branching factor*. Our empirical observations on BIUTEE showed that its branching factor ranges from 2-3 for some states to about 30 for other states.

3.2 Search Scheme

Our empirical investigation compares a range prominent search algorithms, described in Section 2. To facilitate such investigation, we formulate them in the following unifying scheme (Algorithm 1).

Algorithm 1 Unified Search Scheme

Parameters: $f(\cdot)$: state evaluation function
$expand(\cdot)$: state generation function

Input: k_{expand}: # states expanded in each iteration
k_{maintain}: # states in OPEN in each iteration
s_{init}: initial state

1: OPEN $\leftarrow \{s_{\text{init}}\}$
2: **repeat**
3: BEST $\leftarrow k_{\text{expand}}$ best (according to f) states in OPEN
4: GENERATED $\leftarrow \bigcup_{s \in \text{BEST}} expand(s)$
5: OPEN $\leftarrow$ (OPEN $\setminus$ Best) $\cup$ GENERATED
6: OPEN $\leftarrow k_{\text{maintain}}$ best (according to f) states in OPEN
7: **until** BEST contains the goal state

Initially, the open list, OPEN contains the initial state. Then, the *best* k_{expand} states from OPEN are chosen, according to the evaluation function $f(s)$

[2]More details in www.cs.biu.ac.il/~nlp/downloads/biutee/search_ranlp_2011.pdf

Algorithm	$\mathbf{f}()$	$\mathbf{expand}()$	$\mathbf{k}_{\text{maintain}}$	$\mathbf{k}_{\text{expand}}$
A*	$g+h$	regular	∞	1
Weighted A*	$g+w\cdot h$	regular	∞	1
K-Weighted A*	$g+w\cdot h$	regular	∞	$k>1$
Pure Heuristic	h	regular	∞	1
Greedy	$g+w\cdot h$	regular	1	1
Beam	$g+h$	regular	$k>1$	$k>1$
BIUTEE-orig	$g+w_i\cdot h$	regular	$k>1$	$k>1$
LLGS	$\frac{\Delta g}{\Delta h}$	local-lookahead	1	1

Table 2: Search algorithm mapped to the unified search scheme. "Regular" means generating all the states which can be generated by applying a single transformation. Alternative greedy implementations use $f=h$.

(line 3), and expanded using the *expansion function* $expand(s)$. In classical search algorithms, $expand(s)$ means generating a set of states by applying all the possible state transition operators to s. Next, we remove from OPEN the states which were expanded, and add the newly generated states. Finally, we keep in OPEN only the best k_{maintain} states, according to the evaluation function $f(s)$ (line 6). This process repeats until the goal state is found in BEST (line 7). Table 2 specifies how known search algorithms, described in Section 2, fit into the unified search scheme.

Since runtime efficiency is crucial in our domain, we focused on improving one of the simple but fast algorithms, namely, greedy search. To improve the quality of the proof found by greedy search, we introduce new algorithmic components for the expansion and evaluation functions, as described in the next two subsections, while maintaining efficiency by keeping k_{maintain}=k_{expand}= 1

3.3 Evaluation function

In most domains, the heuristic function $h(s)$ estimates the cost of the minimal-cost path from a current state, s, to a goal state. Having such a function, the value $g(s)+h(s)$ estimates the expected total cost of a search path containing s. In our domain, it is yet unclear how to calculate such a heuristic function. Given a state s, systems typically estimate the difference (the gap) between s and the hypothesis t_{H} (the goal state). In BIUTEE this is quantified by the number of parse-tree nodes and edges of t_{H} that do not exist in s. However, this does not give an estimation for the expected cost of the path (the sequence of transformations) from s to the goal state. This is because the number of nodes and edges that can be changed by a single transformation can vary from a single node to several nodes (e.g., by a lexical syntactic entailment rule). Moreover, even if two transformations change the same number of nodes and edges, their costs might be significantly different. Consequently, the measurement of the cost accumulated so far ($g(s)$) and the remaining gap to t_{H} ($h(s)$) are unrelated. We note that a more sophisticated heuristic function was suggested by Heilman and Smith (2010), based on tree-kernels. Nevertheless, this heuristic function, serving as $h(s)$, is still unrelated to the transformation costs ($g(s)$).

We therefore propose a novel gradient-style function to overcome this difficulty. Our function is designed for a greedy search in which OPEN always contains a single state, s. Let s^j be a state generated from s, the cost of deriving s^j from s is $\Delta_g(s^j) \equiv g(s^j)-g(s)$. Similarly, the reduction in the value of the heuristic function is defined $\Delta_h(s^j) \equiv h(s)-h(s^j)$. Now, we define $f_\Delta(s^j) \equiv \frac{\Delta_g(s^j)}{\Delta_h(s^j)}$. Informally, this function measures how costly it is to derive s^j relative to the obtained decrease in the remaining gap to the goal state. For the edge case in which $h(s)-h(s^j)\leq 0$, we define $f_\Delta(s^j)=\infty$. Empirically, we show in our experiments that the function $f_\Delta(s)$ performs better than the traditional functions $f(s)=g(s)+h(s)$ and $f_w(s)=g(s)+w\cdot h(s)$ in our domain.

3.4 Node expansion method

When examining the proofs produced by the above mentioned algorithms, we observed that in many cases a human could construct proofs that exhibit some internal structure, but were not revealed by the algorithms. Observe, for example, the proof in Table 1. It can be seen that transformations 2,3 and 4 strongly depend on each other. Applying transformation 3 requires first applying transformation 2, and similarly 4 could not be applied unless 2 and 3 are first applied. Moreover, there is no gain in applying transformations 2 and 3, unless transformation 4 is applied as well. On the other hand, transformation 1 does not depend on any other transformation. It may be performed at any point along the proof, and

moreover, changing all other transformations would not affect it.

Carefully examining many examples, we generalized this phenomenon as follows. Often, a sequence of transformations can be decomposed into a set of *coherent subsequences* of transformations, where in each subsequence the transformations strongly depend on each other, while different subsequences are independent. This phenomenon can be utilized in the following way: instead of searching for a complete sequence of transformations that transform t_{T} into t_{H}, we can iteratively search for independent coherent subsequences of transformations, such that a combination of these subsequences will transform t_{T} into t_{H}. This is somewhat similar to the technique of applying *macro* operators, which is used in automated planning (Botea et al., 2005) and puzzle solving (Korf, 1985).

One technique for finding such subsequences is to perform, for each state being expanded, a brute-force depth-limited search, also known as *lookahead* (Russell and Norvig, 2010; Bulitko and Lustrek, 2006; Korf, 1990; Stern et al., 2010). However, performing such lookahead might be slow if the branching factor is large. Fortunately, in our domain, coherent subsequences have the following characteristic which can be leveraged: typically, a transformation depends on a previous one only if it is performed over some nodes which were affected by the previous transformation. Accordingly, our proposed algorithm searches for coherent subsequences, in which each subsequent transformation must be applied to nodes that were affected by the previous transformation.

Formally, let o be a transformation that has been applied on a tree t, yielding t'. $\sigma_{\text{affected}}(o, t')$ denotes the subset of nodes in t' which were affected (modified or created) by the application of o.

Next, for a transformation o, applied on a parse tree t, we define $\sigma_{\text{required}}(t, o)$ as the subset of t's nodes required for applying o (i.e., in the absence of these nodes, o could not be applied).

Finally, let t be a parse-tree and σ be a subset of its nodes. $\mathrm{enabled\_ops}(t, \sigma)$ is a function that returns the set of the transformations that can be applied on t, which require at least one of the nodes in σ. Formally, $\mathrm{enabled\_ops}(t, \sigma) \equiv \{o \in \mathcal{O} : \sigma \cap \sigma_{\text{required}}(t, o) \neq \emptyset\}$, where $\mathcal{O}$ is the set of transformations that can be applied on t. In our algorithm, σ is the set of nodes that were affected by the preceding transformation of the constructed subsequence.

The recursive procedure described in Algorithm 2 generates all coherent subsequences of lengths up to d. It should be initially invoked with t - the current state (parse tree) being expanded, σ - the set of all its nodes, d - the maximal required length, and $\emptyset$ as an empty initial sequence. We use $O{\cdot}o$ as concatenation of an operation o to a subsequence O.

Algorithm 2 local-lookahead (t,σ,d,O)

1: **if** $d = 0$ **then**
2: **return** $\emptyset$ (empty-set)
3: **end if**
4: SUBSEQUENCES $\leftarrow \emptyset$
5: **for all** $o \in \mathrm{enabled\_ops}(t, \sigma)$ **do**
6: Let $t \vdash_o t'$
7: Add $\{O{\cdot}o\} \cup$local-lookahead$(t', \sigma_{\text{affected}}(o, t'), d-1, O{\cdot}o)$ to SUBSEQUENCES
8: **end for**
9: **return** SUBSEQUENCES

The loop in lines 5 - 8 iterates over transformations that can be applied on the input tree, t, requiring the same nodes that were affected by the previous transformation of the subsequence being constructed. Note that in the first call $\mathrm{enabled\_ops}(t, \sigma)$ contain all operations that can be applied on t, with no restriction. Applying an operation o results in a new subsequence $O \cdot o$. This subsequence will be part of the set of subsequences found by the procedure. In addition, it will be used in the next recursive call as the prefix of additional (longer) subsequences.

3.5 Local-lookahead gradient search

We are now ready to define our new algorithm LOCAL-LOOKAHEAD GRADIENT SEARCH (LLGS). In LLGS, like in greedy search, $k_{\text{maintain}}{=}k_{\text{expand}}{=}\ 1$. $expand(s)$ is defined to return all states generated by subsequences found by the *local-lookahead* procedure, while the evaluation function is defined as $f = f_{\Delta}$ (see last row of Table 2).

4 Evaluation

In this section we first evaluate the search performance in terms of efficiency (run time), the quality

of the found proofs (as measured by proof cost), and overall inference performance achieved through various search algorithms. Finally we analyze the contribution of our two novel components.

4.1 Evaluation settings

We performed our experiments on the last two published RTE datasets: RTE-5 (2009) and RTE-6 (2010). The RTE-5 dataset is composed of a training and test corpora, each containing 600 text-hypothesis pairs, where in half of them the text entails the hypothesis and in the other half it does not. In RTE-6, each of the training and test corpora consists of 10 topics, where each topic contains 10 documents. Each corpus contains a set of hypotheses (211 in the training dataset, and 243 in the test dataset), along with a set of candidate entailing sentences for each hypothesis. The system has to find for each hypothesis which candidate sentences entail it. To improve speed and results, we used the filtering mechanism suggested by (Mirkin et al., 2009), which filters the candidate sentences by the Lucene IR engine[3]. Thus, only top 20 candidates per hypothesis were tested

Evaluation of each of the algorithms was performed by running BIUTEE while replacing BIUTEE-orig with this algorithm. We employed a comprehensive set of knowledge resources (available in BIUTEE's web site): WordNet (Fellbaum, 1998), Directional similarity (Kotlerman et al., 2010), DIRT (Lin and Pantel, 2001) and generic syntactic rules. In addition, we used coreference substitutions, detected by ArkRef[4].

We evaluated several known algorithms, described in Table 2 above, as well as BIUTEE-orig. The latter is a strong baseline, which outperforms known search algorithms in generating low cost proofs. We compared all the above mentioned algorithms to our novel one, LLGS.

We used the training dataset for parameter tuning, which controls the trade-off between speed and quality. For weighted A*, as well as for greedy search, we used $w = 6.0$, since, for a few instances, lower values of w resulted in prohibitive runtime. For beam search we used $k = 150$, since higher values of k did not improve the proof cost on the training dataset. The value of d in LLGS was set to 3. $d = 4$ yielded the same proof costs, but was about 3 times slower.

Since lower values of w could be used by weighted A* for most instances, we also ran experiments where we varied the value of w according to the dovetailing method suggested in (Valenzano et al., 2010) (denoted *dovetailing WA**) as follows. When weighted A* has found a solution, we reran it with a new value of w, set to half of the previous value. The idea is to guide the search for lower cost solutions. This process was halted when the total number of states generated by all weighted A* instances exceeded a predefined constant (set to $10,000$).

4.2 Search performance

This experiment evaluates the search algorithms in both efficiency (run-time) and proof quality. Efficiency is measured by the average CPU (Intel Xeon 2.5 GHz) run-time (in seconds) for finding a complete proof for a text-hypothesis instance, and by the average number of generated states along the search. Proof quality is measured by its cost.

The comparison of costs requires that all experiments are performed on the same model which was learned during training. Thus, in the training phase we used the original search of BIUTEE, and then ran the test phase with each algorithm separately. The results, presented in Table 3, show that our novel algorithm, LLGS, outperforms all other algorithms in finding lower cost proofs. The second best is BIUTEE-orig which is much slower by a factor of 3 (on RTE-5) to 8 (on RTE-6)[5]. While inherently fast algorithms, particularly greedy and pure heuristic, achieve faster running times, they achieve lower proof quality, as well as lower overall inference performance (see next subsection).

4.3 Overall inference performance

In this experiment we test whether, and how much, finding better proofs, by a better search algorithm, improves overall success rate of the RTE system. Table 4 summarizes the results (accuracy in RTE-5

[3] `http://lucene.apache.org`

[4] `www.ark.cs.cmu.edu/ARKref/` See (Haghighi and Klein, 2009)

[5] Calculating T-test, we found that runtime improvement is statistically significant with $p < 0.01$, and $p < 0.052$ for cost improvement over BIUTEE-orig.

Algorithm	Avg. time	Avg. generated	Avg. cost
Weighted A*	0.22 / **0.09**	301 / 143	1.11 / 10.52
Dovetailing WA*	7.85 / 8.53	9797 / 9979	1.05 / 10.28
Greedy	0.20 / 0.10	468 / **158**	1.10 / 10.55
Pure heuristic	**0.09** / 0.10	**123** / 167	1.35 / 12.51
Beam search	20.53 / 9.48	43925 / 18992	1.08 / 10.52
BIUTEE-orig	7.86 / 14.61	14749 / 22795	1.03 / 10.28
LLGS	2.76 / 1.72	1722 / 842	**0.95 / 10.14**

Table 3: Comparison of algorithms on RTE-5 / RTE-6

and F1 in RTE-6). We see that in RTE-5 LLGS outperforms all other algorithms, and BIUTEE-orig is the second best. This result is statistically significant with $p < 0.02$ according to *McNemar* test. In RTE-6 we see that although LLGS tends to finds lower cost proofs, as shown in Table 3, BIUTEE obtains slightly lower results when utilizing this algorithm.

Algorithm	RTE-5 accuracy %	RTE-6 F1 %
Weighted A*	59.50	48.20
Dovetailing WA*	60.83	49.01
Greedy	60.50	48.56
Pure heuristic	60.83	45.70
Beam search	61.33	48.58
BIUTEE-orig	60.67	**49.25**
LLGS	**64.00**	49.09

Table 4: Impact of algorithms on system success rate

4.4 Component evaluation

In this experiment we examine separately our two novel components. We examined f_Δ by running LLGS with alternative evaluation functions. The results, displayed in Table 5, show that using f_Δ yields better proofs and also improves run time.

f	Avg. time	Avg. cost	Accuracy %
$f = g + h$	3.28	1.06	61.50
$f = g + w \cdot h$	3.30	1.07	61.33
$f = f_\Delta$	**2.76**	**0.95**	**64.0**

Table 5: Impact of f_Δ on RTE-5. $w = 6.0$. Accuracy obtained by retraining with corresponding f.

Our *local-lookahead* (Subsection 3.4) was examined by running LLGS with alternative node expansion methods. One alternative to local-lookahead is standard expansion by generating all immediate derivations. Another alternative is to use the standard lookahead, in which a brute-force depth-limited search is performed in each iteration, termed here "exhaustive lookahead". The results, presented in Table 6, show that by avoiding any type of lookahead one can achieve fast runtime, while compromising proof quality. On the other hand, both exhaustive and local lookahead yield better proofs and accuracy, while local lookahead is more than 4 times faster than exhaustive lookahead.

lookahead	Avg. time	Avg. cost	Accuracy (%)
exhaustive	13.22	**0.95**	**64.0**
local	2.76	**0.95**	**64.0**
none	**0.24**	0.97	62.0

Table 6: Impact of local and global lookahead on RTE-5. Accuracy obtained by retraining with the corresponding lookahead method.

5 Conclusion

In this paper we investigated the efficiency and proof quality obtained by various search algorithms. Consequently, we observed special phenomena of the search space in textual inference and proposed two novel components yielding a new search algorithm, targeted for our domain. We have shown empirically that (1) this algorithm improves run time by factors of 3-8 relative to BIUTEE-orig, and by similar factors relative to standard AI-search algorithms that achieve similar proof quality; and (2) outperforms all other algorithms in finding low cost proofs.

In future work we plan to investigate other search paradigms, e.g., Monte-Carlo style approaches (Kocsis and Szepesvári, 2006), which do not fall under the AI search scheme covered in this paper. In addition, while our novel components were motivated by the search space of textual inference, we foresee their potential utility in other application areas for search, such as automated planning and scheduling.

Acknowledgments

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

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

Philip Bille. 2005. A survey on tree edit distance and related problems. *Theoretical Computer Science*.

Adi Botea, Markus Enzenberger, Martin Müller, and Jonathan Schaeffer. 2005. Macro-FF: Improving ai planning with automatically learned macro-operators. *J. Artif. Intell. Res. (JAIR)*, 24:581–621.

Vadim Bulitko and Mitja Lustrek. 2006. Lookahead pathology in real-time path-finding. In *proceedings of AAAI*.

Ido Dagan, Oren Glickman, and Bernardo Magnini. 2005. The pascal recognising textual entailment challenge. In *Proceedings of MLCW*.

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

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

Ariel Felner, Sarit Kraus, and Richard E. Korf. 2003. KBFS: K-best-first search. *Ann. Math. Artif. Intell.*, 39(1-2):19–39.

David Furcy and Sven Koenig. 2005. Limited discrepancy beam search. In *proceedings of IJCAI*.

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

Stefan Harmeling. 2009. Inferring textual entailment with a probabilistically sound calculus. *Natural Language Engineering*.

Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, SSC-4(2):100–107.

Michael Heilman and Noah A. Smith. 2010. Tree edit models for recognizing textual entailments, paraphrases, and answers to questions. In *Proceedings of NAACL*.

Levente Kocsis and Csaba Szepesvári. 2006. Bandit based monte-carlo planning. In *proceedings of ECML*.

Richard E. Korf. 1985. Macro-operators: A weak method for learning. *Artif. Intell.*, 26(1):35–77.

Richard E. Korf. 1990. Real-time heuristic search. *Artif. Intell.*, 42(2-3):189–211.

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

Milen Kouylekov and Bernardo Magnini. 2005. Recognizing textual entailment with tree edit distance algorithms. In *Proceedings of Pascal Challenges Workshop on Recognising Textual Entailment*.

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

Yashar Mehdad. 2009. Automatic cost estimation for tree edit distance using particle swarm optimization. In *Proceedings of the ACL-IJCNLP*.

Shachar Mirkin, 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*.

Ira Pohl. 1970. Heuristic search viewed as path finding in a graph. *Artificial Intelligence*, 1(3-4):193 – 204.

Stuart Russell and Peter Norvig. 2010. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 3rd edition edition.

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

Roni Stern, Tamar Kulberis, Ariel Felner, and Robert Holte. 2010. Using lookaheads with optimal best-first search. In *proceedings of AAAI*.

Idan Szpektor, Hristo Tanev, Ido Dagan, and Bonaventura Coppola. 2004. Scaling web-based acquisition of entailment relations. In *Proceedings of EMNLP*.

Richard Anthony Valenzano, Nathan R. Sturtevant, Jonathan Schaeffer, Karen Buro, and Akihiro Kishimoto. 2010. Simultaneously searching with multiple settings: An alternative to parameter tuning for suboptimal single-agent search algorithms. In *proceedings of ICAPS*.

Mengqiu Wang and Christopher D. Manning. 2010. Probabilistic tree-edit models with structured latent variables for textual entailment and question answering. In *Proceedings of COLING*.

Rong Zhou and Eric A. Hansen. 2005. Beam-stack search: Integrating backtracking with beam search. In *proceedings of ICAPS*.