Modeling Extractive Sentence Intersection via Subtree Entailment

Omer Levy* **Ido Dagan*** **Gabriel Stanovsky*** **Judith Eckle-Kohler†** **Iryna Gurevych†‡**

*Computer Science Department, Bar-Ilan University, Israel
†Ubiquitous Knowledge Processing Lab, Technische Universität Darmstadt, Germany
‡Ubiquitous Knowledge Processing Lab, German Institute for Educational Research, Germany

Abstract

Sentence intersection captures the semantic overlap of two texts, generalizing over paradigms such as textual entailment and semantic text similarity. Despite its modeling power, it has received little attention because it is difficult for non-experts to annotate. We analyze 200 pairs of similar sentences and identify several underlying properties of sentence intersection. We leverage these insights to design an algorithm that decomposes the sentence intersection task into several simpler annotation tasks, facilitating the construction of a high quality dataset via crowdsourcing. We implement this approach and provide an annotated dataset of 1,764 sentence intersections.

1 Introduction

Various paradigms exist for comparing the meanings of two texts and modeling their semantic overlap. Paraphrase detection (Dolan et al., 2004), for example, tries to identify whether two texts express the same information. It cannot, however, capture cases where there is only partial information overlap. One paradigm that addresses this issue is textual entailment (Dagan et al., 2006), which asks whether one text can be inferred from another. While textual entailment models when the information of one text is subsumed by another, it falls short when neither text strictly subsumes the other, yet a significant amount of information is common to both. A more recent paradigm, semantic text similarity (Agirre et al., 2012), allows for these cases by measuring "how much" similar two sentences are.

In this paper, we consider an additional paradigm for modeling the semantic overlap between two texts. The task of *sentence intersection*, as defined in several variants (Marsi and Krahmer, 2005; McKeown et al., 2010; Thadani and McKeown, 2011), is to construct a sentence containing all the information shared by the two input sentences (Figure 1). While sentence intersection was originally proposed for abstractive summarization, we argue that it is an interesting generic paradigm for modeling information overlap, which generalizes over several previous approaches. In particular, it is expressive enough to capture partial semantic overlaps that are not modeled by textual entailment or even partial textual entailment (Nielsen et al., 2009; Levy et al., 2013). Furthermore, rather than quantifying the *amount* of shared information as in semantic text similarity, sentence intersection captures *what* this shared information is.

Although sentence intersection has existed for over a decade, it has received little attention due to a lack of annotated data. Previous annotation attempts have either used experts, which did not scale, or crowdsourcing, which yielded unreliable annotations (McKeown et al., 2010). We also observe that annotating sentence intersection is difficult for non-experts. We hypothesize that this difficulty stems from the task's requirement that the annotator indicate *all* the shared information when constructing the intersection, necessitating a high level of attention to fine details in two different sentences simultaneously.

This paper addresses the issue of annotating sentence intersection. We clarify some ambiguous aspects of sentence intersection by defining *extractive sentence intersection* (§3): the *set* of all intersections that can be composed only from words in the input sentences. Though less ambiguous, this task is still challenging for non-experts to annotate correctly (§4). Our main observation is that extractive intersections have a common underlying structure – *an aligned partition* (§6). In a nutshell, the dependency tree of

This work is licensed under a Creative Commons Attribution 4.0 International License. License details: http://creativecommons.org/licenses/by/4.0/

Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers, pages 2891–2901, Osaka, Japan, December 11-17 2016.

s_1	Obama visited Canada yesterday.
s_2	The president flew to Ottawa to meet Trudeau.
$\cap$	The president visited Canada.
$\cup$	Obama flew to Ottawa yesterday to meet Trudeau.

Figure 1: Sentence intersection ($\cap$) versus union ($\cup$). Union captures all the information in the input sentences, whereas intersection captures all the *shared* information.

s_1	Sanders disagrees with HRC on Super PACs.
s_2	Bernie criticizes Hillary on campaign funding.
$\cap$	Sanders disagrees with HRC on campaign funding. Sanders disagrees with Hillary on campaign funding. Bernie disagrees with HRC on campaign funding. Bernie disagrees with Hillary on campaign funding.

Figure 2: Extractive intersection is a combinatorial problem, because the common information between two sentences can often be expressed in various equivalent ways.

an intersection sentence can be partitioned into subtrees, each one originating from one of the input sentences. The interesting property is that for every subtree originating from one input sentence, there exists another subtree in the other input sentence, which entails the first. For this purpose, we define *subtree entailment* (§5), an extension of the well-studied inference rules paradigm (Lin and Pantel, 2001).

We then design a semi-automatic algorithm that helps the annotator construct an aligned partition and eventually the extractive intersection set (§7). Our process decomposes the complex task of extractive sentence intersection into a collection of simpler local subtree entailment queries. The entailment queries (which are easy for humans but hard for machines) are annotated manually, while the combinatorial aspect of generating intersections is completely automated. Finally, we apply our process and annotate 1,764 examples via crowdsourcing (§8). These annotations are superior to direct manual crowdsourced annotations.[1]

2 Background: Sentence Intersection

The ability to combine two (or more) semantically-similar sentences into one is a core requirement of abstractive summarization. Various flavors of this combination task, dubbed *sentence fusion*, have been studied. Barzilay et al. (1999; 2003; 2005) focused on generating a single sentence from the most important parts of the input sentences. The importance criterion was argued to be too vague for consistent human annotation (Daume III and Marcu, 2004), giving rise to the more accurately-defined notions of sentence *union* and *intersection* (Marsi and Krahmer, 2005). Sentence union aspires to create a single sentence containing *all* the information in the input sentences, while sentence intersection tries to isolate all the information that they have *in common* – i.e. the consensus. Figure 1 demonstrates this difference.

Annotation Efforts Perhaps the most pressing issue in making sentence intersection an appealing semantic task is creating enough high-quality annotated data. However, prior art appears to lack a method for annotating sentence intersection that is both efficient (scalable) and accurate. The previous attempt to annotate sentence intersection via crowdsourcing (McKeown et al., 2010) resulted in unreliable annotations. Annotators were presented with two similar sentences, and asked to combine them "into a single sentence conveying only the information they have in common". Each pair of input sentences was given to 5 different workers, and the most agreed-upon (centroid) annotation was selected. Results showed that only 54% of the annotation were indeed valid intersections. On the other hand, sentence *union* annotations over the same sentence pairs had 95% accuracy. Krahmer et al. (2008) also observed intersection annotations to be less consistent than unions while studying query-based sentence fusion.

More recently, Thadani and McKeown (2013) proposed a different dataset, which relied on pre-existing pyramid annotated data (Nenkova and Passonneau, 2004). This dataset is more consistent, but does not comply with the definition of sentence intersection. In practice, it models a simpler task: generating a sentence that contains only shared information, not necessarily *all* of the shared information. It seems that this distinction – the need to identify *every* piece of common information at once – is what makes sentence intersection a combinatorial problem, explaining why it is so difficult for humans to perform this task consistently.

Algorithms The sentence fusion literature typically takes an alignment-based algorithmic approach (Barzilay and McKeown, 2005; Filippova and Strube, 2008), which was also adopted by Thadani and

[1] Our code and dataset are available online: `bitbucket.org/omerlevy/extractive_intersection`

s_1	Bill traveled back in time.
s_2	Ted traveled back in time.
	A person traveled back in time.
$\cap$	A man traveled back in time.
	Someone traveled back in time.

Figure 3: Possible lexical abstractions. The best choice of words is not always obvious.

s_1	Akiko ate cake.
s_2	Akiko ate pudding.
$\cap$	Akiko ate **[something]**.

Figure 4: An intersection containing a placeholder.

McKeown (2011) for sentence intersection. In general, they (lexically) align the input sentences, and then select a subset of the aligned components when generating the fused sentence. Our annotation method is inspired by this approach, though it differs in various ways, such as the kind of alignments it creates (subtree entailments), the output it generates (a *set* of sentences), and the fact that it is interactive.

3 Extractive Sentence Intersection

We define a new variant of sentence intersection, *extractive sentence intersection*, which departs from the original task in two aspects: specifying a set of intersection sentences, while limiting their scope to be based on the lexical elements in the input sentences.

First, we observe that several different output sentences may satisfy the original definition of sentence intersection. In Figure 2, we see two input sentences conveying very similar information that are expressed by completely different words. Since many of the terms are synonymous, choosing one over another is arbitrary (e.g. "Bernie" = "Sanders" in this context). Combining different words from the input sentences creates several intersections – each one as valid as the other. Therefore, instead of defining sentence intersection as a single sentence that contains all the information shared by the input sentences, we define it as the *set* of all such sentences.

Second, we observe that, according to its original definition, an intersection sentence might include words that do not appear in any of the two input sentences. This may happen in two situations: (a) lexical variability – when introducing an unmentioned synonym (e.g. using "Clinton" instead of "HRC" or "Hillary"); (b) lexical abstraction – where creating an intersection requires introducing a new word that subsumes two terms from the input sentences (Figure 3). Intersections that require lexical abstraction are difficult to define and annotate consistently because the desired level of abstraction is often unclear. We avoid the issues of lexical variability and abstraction by focusing on intersections that are *extractive*, i.e. intersections that *only contain lexical elements from the input sentences*.

To maintain grammaticality, we make an exception to this rule. Placeholders, such as *[something]*, *[somewhere]*, and *[sometime]*, can be used when the input sentences provide different information on an entity that cannot be omitted for syntactic reasons (e.g. subjects). In many cases, there is a specific word that could fit instead of a placeholder, but this word does not appear in the input sentences (Figure 4). The task of finding such words requires lexical abstraction, which is beyond our current scope.

Combining both these criteria – output set and extractiveness – we define *extractive sentence intersection* as the set of all sentences that each contains all the information common to the input sentences, while being composed only of words that appeared in the input sentences and placeholders.[2]

The entire set of extractive intersections allows for more accurate automatic evaluation, since the evaluation mechanism does not need to overcome issues in lexical variability; instead, it can simply select the most similar expert-crafted sentence from the set using simple metrics such as BLEU (Papineni et al., 2002) or ROUGE (Lin, 2004). Having multiple possible solutions is not a foreign concept to NLP, and is widely used in translation and summarization.

[2]While this paper discusses intersections between two input sentences, one can theoretically extend this setting to multiple input sentences by consecutively applying the intersection operation. For example, if we have three sentences s_1, s_2, s_3, we could first find the intersection between s_1 and s_2, and then for each sentence s' in $s_1 \cap s_2$, intersect that with s_3. We would essentially take the union of the latter set of intersection sets, i.e. $s_1 \cap s_2 \cap s_3 = (s_1 \cap s_2) \cap s_3 = \bigcup\{s' \cap s_3 | s' \in s_1 \cap s_2\}$.

s_1	It was Rehnquist who presided over the swearing-in ceremony when Roberts took his seat on the U.S. Court of Appeals for the District of Columbia.
s_2	Roberts, nominated to succeed retiring justice Sandra Day O'Connor, easily won senate confirmation to the U.S. Court of Appeals for the District of Columbia two years ago.
$\cap$	Roberts won senate confirmation to the U.S. Court of Appeals for the District of Columbia.

Figure 5: An example pair of sentences from our dataset and their (expert-annotated) intersection.

4 Annotation Feasibility

We wish to create a large gold-standard annotated dataset of extractive intersection via crowdsourcing. Since the previous crowdsourced intersection dataset was of mixed quality (McKeown et al., 2010), we annotate the same data with both experts and crowdsourcing to determine the annotation's feasibility.

Evaluation Metrics We present two complementary methods for evaluating extractive sentence intersection: a strict method based on sentence equality, and a softer measure similar to ROUGE (Lin, 2004). The methods essentially compare a candidate set of sentences C to a gold-standard set G, yielding precision and recall scores that are averaged across the dataset.

The sentence equality method follows directly from the task's definition: $Precision = |C \cap G|/|C|$, $Recall = |C \cap G|/|G|$. Note that the underlying sentence equality measure is simple string equality.

Sentence equality can sometimes be too harsh for measuring intersection. For example, if an intersection sentence contains exactly all the shared information except for one minor detail, we would want to partially penalize it rather than completely rejecting it. We therefore adopted a more lenient evaluation measure, based on ROUGE (Lin, 2004). In automatic summarization, ROUGE measures the n-gram overlap between a single candidate summary and a set of several gold-standard summaries. Whereas ROUGE tries to maximize agreement with *all* gold summaries, we are satisfied if there exists even one gold intersection sentence that is sufficiently similar to the given candidate sentence. Therefore, to calculate the precision of a given candidate intersection sentence $c \in C$, we take the best-matching gold sentence g, and calculate the portion of n-grams in c that are covered by g. Similarly, a gold sentence's recall is measured versus the best-matching $c \in C$. We define an intersection set's precision and recall by taking the averages. Formally: $Precision = \frac{1}{|C|}\sum_{c \in C} \max_{g \in G} (Cover\,(c, g))$, $Recall = \frac{1}{|G|}\sum_{g \in G} \max_{c \in C} (Cover\,(g, c))$, where the portion of n-grams covered is $Cover\,(a, b) = |ngrams(a) \cap ngrams(b)|\,/|ngrams(a)|$. To avoid deviating too much from the sentence equality measure, we select a conservative definition of n-grams ($n = 4$). Note that replacing $Cover$ with equality in the equations above is exactly the sentence equality method.

Annotation We annotated a random sample of 200 sentence pairs, half from Columbia's sentence fusion dataset (McKeown et al., 2010) and half from MSR's paraphrase dataset (Dolan et al., 2004). Columbia's dataset contains 297 pairs of related sentences, annotated for sentence intersection and union. MSR's dataset contains 1,500 pairs of related sentences, originally annotated for a paraphrase detection task, and later with semantic text similarity scores (Agirre et al., 2012). Existing annotations are ignored, and replaced by our new annotations (see Figure 5).

The first author annotated the data (as an expert), and Mechanical Turk workers were hired to annotate the same sentence pairs. Workers were told to "extract" a new sentence by selecting words from the input sentences or a bank of placeholders, and encouraged to create as many intersections as possible. Each sentence pair was given to 5 workers, of which the most consistent 3 were selected to reduce the effect of poor annotations (agreement between two workers was measured using the ROUGE-like F_1). We also used the set of both input sentences as a simple annotation baseline (the "input" baseline), which captures (with some error) the cases in which one sentence is completely entailed by another.

Results Table 1 shows very poor crowd-expert agreement rates when measured using sentence equality. The crowdsourced annotation is slightly better than the input baseline (precision-wise), and does not seem to improve coverage. Even using the softer ROUGE-like metric, the crowdsourced data does not seem to provide enough added value beyond the simple input baseline (its F_1 is only 0.014 higher).

	Sentence Equality			ROUGE-like		
	P	R	F_1	P	R	F_1
Input	0.080	0.124	0.097	0.611	0.952	0.744
Direct	0.130	0.124	0.127	0.735	0.781	0.758

Table 1: The agreement between crowd ("Direct") and expert annotations. "Input" is a baseline where the two input sentences are also the output.

s	Oswald killed JFK.
$\sigma(s, \text{JFK}, \text{Kennedy})$	Oswald killed Kennedy.
$\sigma(s, \text{killed}, \text{assassinated})$	Oswald assassinated JFK.
$\sigma(s, \text{killed}, \text{was shot by})$	JFK was shot by Oswald.
$\sigma(s, \text{killed}, \text{died})$	JFK died.

Figure 6: Four examples of the substitution function.

Extractive intersection is clearly a difficult task for non-expert annotators. We suggest that this stems from the many nuanced pieces of information in each sentence (precision) and the many possible combinations that reflect valid extractive intersections (recall). From our own annotation effort, we noticed that these two difficulties pertain to an underlying structure that consistently appears in extractive intersections. In the following sections, we describe this structure (§6) and propose a way of exploiting it to generate high-quality annotations (§7).

5 Subtree Entailment in Context

In this section, we describe a new textual relation, *subtree entailment in context*, which will assist in defining the underlying structure of extractive intersection. Subtree entailment in context models inference between rich lexical-syntactic patterns (subtrees of syntactic dependency trees), while considering internal context (instantiated arguments) and external context (substitution in a complete sentence). As such, it generalizes over previous ideas presented separately in prior art; subtrees were used in TEASE (Szpektor et al., 2015) and PPDB (Pavlick et al., 2015), internal context was considered in context-sensitive relation inference (Zeichner et al., 2012; Melamud et al., 2013; Levy and Dagan, 2016), and external context is studied in the lexical substitution task (McCarthy and Navigli, 2007; Biemann, 2013; Kremer et al., 2014). Subtree entailment in context is the first notion that combines all these traits. In addition to these advantages, we also introduce a new mechanism for handling changes in arity, in case one subtree contains more/less arguments than another.

The Substitution Function To define subtree entailment in context, we must first define an auxiliary operation – subtree substitution. The substitution function σ is given a sentence tree s, a subtree within that sentence t, and another subtree t', which is not necessarily part of s. It creates a new sentence s' by replacing t with t' (see Figure 6): $s' = \sigma(s, t, t')$.

Slot Assignments The substitution function does not specify how the child nodes of t connect to t'. This is not always trivial; e.g. in the third example in Figure 6, the *subject* of "killed" is attached as the *object* of "was shot by". Moreover, the arity of t (the number of expected child nodes) might be different from that of t', as in the fourth example.

The missing piece of the puzzle is *slot assignments*. A *slot* is a dangling edge whose head is part of a subtree (either t or t') but its modifier is not. Slot assignments match these outgoing edges, and are especially useful when going from passive to active and when there is a change in arity. We denote slot assignment as a function α from the slots of t' to those of t, and extend the definition of the substitution function to include it: $s' = \sigma(s, t, t', \alpha)$.

The fact that α is a function means that not all of t's child nodes will necessarily appear in the new sentence s'. In addition, the slots of t' can be assigned a null value ($\emptyset$), which means that none of t's child nodes will fill this slot, leaving it empty. When generating a natural language expression with an empty slot, it is often the case that a placeholder will fill it (see §3). For example, let us invert the fourth example in Figure 6: $s' = \sigma(\text{"JFK died."}, \text{died}, \text{killed}, \alpha)$. If $\alpha(subject) = \emptyset$, then s' will be "[someone] killed JFK", where "[someone]" is a placeholder.

Definition Given a sentence s, we can say that t_p (p for premise), a subtree of s, *entails* some other subtree t_h (h for hypothesis) in the context of s, if: $s \vDash \sigma(s, t_p, t_h, \alpha)$, where α assigns the slots of t_h, and $\vDash$ is textual entailment. In other words, if s is true, then replacing t_p with t_h in s should create a new sentence that is also true. Considering the second example from Figure 6, "killed" entails "assassinated"

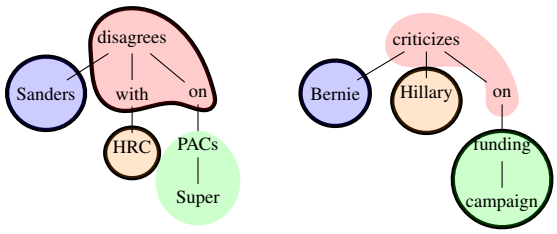

Figure 7: The aligned partition of the example in Figure 2. Subtrees marked by the same color are aligned, and those surrounded by dark borders are entailed by their counterpart.

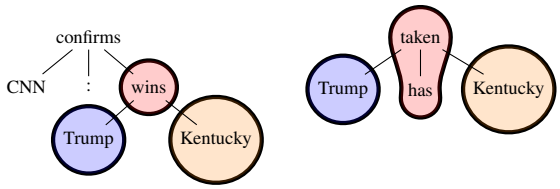

Figure 8: Two input sentences whose aligned partition does not include s_1's root.

(in the context of s) if given that "Oswald killed JFK", it is also true that "Oswald assassinated JFK". Subtree entailment is always in the context of a full premise s; we therefore use the shorthand notation $t_p \vDash_s t_h$.

Notice that, in fact, we are reducing the subtree entailment question to a textual entailment question, where both premise and hypothesis are full sentences (instantiated predicates). This is essential, since one cannot ask "given that 'killed' is true, is 'assassinated' true too?"; an uninstantiated predicate (or worse, a noun) does not yield a truth value.

6 Properties of Extractive Intersections

When examining the dependency trees of sentences created by extractive intersection, we observe that almost all of them (93%–99.5%, see footnote 5 in §8) exhibit three particular properties, which we define using subtree entailment in context.

Aligned Partition The dependency tree of a sentence created by extractive intersection can be partitioned into subtrees, each one originating from one of the input sentences. Interestingly, for every subtree t_1 originating from s_1 (without loss of generality), there exists another subtree t_2 in the other input sentence s_2, which entails t_1 ($t_2 \vDash_{s_2} t_1$). The entailment relation between t_1 and t_2 can be mutual or asymmetric, but it is always one-to-one, i.e. no other subtree within s_2 entails t_1, and vice versa. This induces a partition of the input sentences' dependency trees (seen in Figure 7), where each subtree t_1 is either unaligned or in an entailment relation with exactly one subtree from s_2. This also means that each aligned subtree (except for the root) has exactly one parent, which we denote as $\pi(t, s)$ (t being a subtree in sentence s).

We observe that the vast majority of output sentences can be easily created from such an aligned partition of the input sentences. This is not surprising, because every intersection sentence is entailed by both input sentences (Marsi and Krahmer, 2005). It makes sense that the same quality exists at the subtree level, which we model with aligned partitions.

Inverse Inheritance of Alignment The second property, *inverse inheritance of alignment*, describes how subtrees in the output sentence connect. Given a subtree t_1, its parent in the output sentence's tree is either its original parent $\pi(t_1, s_1)$ or its "parent-in-law" $\pi(t_2, s_2)$ (the parent of its aligned subtree). Looking back at the input sentences, this means that every pair of aligned subtrees also has aligned parents, i.e. the parents "inherit" the alignment from their children. However, this does not mean that the direction of entailment needs to be the same; in Figure 7, "Super PACs" entails "campaign funding" ($t_1 \vDash_{s_1} t_2$), while "X criticizes Y on Z" entails "X disagrees with Y over Z" ($\pi(t_2, s_2) \vDash_{s_2} \pi(t_1, s_1)$).

Entailed Embedded Clauses Figure 8 shows that the input sentences are not necessarily aligned at the root, and that the common information might be contained in an embedded clause. In such cases, if there is a valid (non-empty) intersection, we typically observe that the embedded clause is entailed by the original sentence. In terms of subtree entailment, this means that the subtree wrapping the embedded clause entails the empty subtree (a single slot with no nodes); e.g. if "CNN confirms: X", then "X" is typically true. Conversely, if the embedded clause is not entailed by the original sentence (e.g. s_3 ="Carson will

quit if Trump wins Kentucky") then there is no intersection ($s_1 \cap s_3 = s_2 \cap s_3 = \emptyset$, assuming s_1 and s_2 from Figure 8).

7 A Semi-Automatic Algorithm

Based on the properties discussed in §6, we describe a semi-automatic algorithm for annotating extractive intersection. It requires relatively simple human interaction throughout the process, while automating the combinatorial aspect, which is difficult for humans. Our algorithm also masks the underlying syntactic tree from the annotator, making the entire process crowdsourcable. The core of our algorithm is creating an aligned partition of the input sentences. Once the aligned partition is obtained, we can deterministically create the set of output dependency trees, from which natural language sentences are generated. Some minor technical details are omitted from this description, and will be made available with our code and its documentation upon publication.

7.1 Creating the Aligned Partition

Given the dependency trees[3] of two sentences, we create their aligned partition over 4 steps. We first elicit lexical alignments from the annotators; these are used to detect the aligned partition's root in each sentence, and then as a seed for automatically deriving subtree alignments. The final step validates that the subtrees are indeed in an entailment relation, and also annotates the entailment's direction.

Step 1: Lexical Alignment We present both sentences in natural language, and ask the annotators to mark words (or phrases) that appear in different sentences yet have similar meanings. This phase is recall-oriented, and its goal is to capture as many potential entailments by annotating a slightly looser notion. We limit the number of alignments to one per token; this constraint is essential for inducing an aligned *partition*. Despite detailed guidelines and examples, non-expert annotators mainly mark nouns, typically ignoring verbs and other predicates unless they are identical. We address this issue in Step 3.

Step 2: Root Detection As discussed in §6, the root of an aligned partition does not always contain the roots of both input sentences. We therefore apply a heuristic, on each sentence separately, which finds the root of the embedded clause participating in the aligned partition: the lowest common ancestor of nodes that have a lexical alignment. This technique is motivated by our observation that, for intersection purposes, the original sentence can be replaced with the embedded clause since it contains all the aligned components. In Figure 8, for example, if an annotator were to only align "Trump" and "Kentucky" with themselves, the heuristic would detect "wins" and "taken" as the roots of s_1 and s_2, respectively.

Step 3: Subtree Alignment Since annotators rarely align predicates, we introduce a heuristic method for automatically deriving subtree alignments from the lexical ones provided manually (Step 1). Figure 9 shows a typical lexical alignment. Our heuristic makes greedy local changes until the following four constraints are satisfied:

Inverse Inheritance of Alignment: In §6 we saw that if two subtrees are aligned, then so are their parents. To enforce this property, we create new alignments between parents of already-aligned subtrees. In our example (Figure 9), this yields two new alignments: "primary" – "wins" and "to" – "wins".

Include Function Modifiers: Annotators also tend to ignore function words. We observe that for certain dependency types[4] that connect function words to content words as modifiers, the modifiers should always be part of the same alignment as their heads. Therefore, if the head participates in an alignment, the modifier is added as well; e.g. "the" modifies "primary" and is therefore added to its alignment.

No Overlapping Alignments: Since aligned partitions induce a *partition* of each input sentence, each node take part in one alignment at most. Therefore, if a node participates in more than one alignment, those two alignments are merged. In our example, "wins" participates in two alignments. After merging, we are left with a single alignment (excluding the initial ones): "the", "primary", "to" – "wins".

[3] We used the Stanford converter on top of the Berkeley parser (Petrov and Klein, 2007), which produced correct dependency trees for most of our data.

[4] The following Stanford dependencies: `det`, `aux`, `auxpass`, `neg`, `prt`, `attr`.

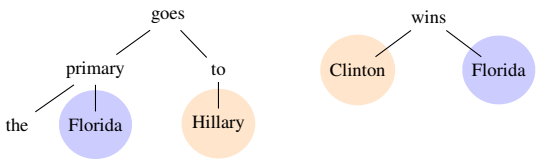

Figure 9: The information provided by a typical annotator in the lexical alignment phase is usually limited to nouns.

	Sentence Equality			ROUGE-like		
	P	R	F_1	P	R	F_1
Input	0.080	0.124	0.097	0.611	0.952	0.744
Direct	0.130	0.124	0.127	0.735	0.781	0.758
Semi-Auto	0.360	0.305	0.330	0.820	0.902	0.859

Table 2: The agreement between crowd and expert annotations. "Semi-Auto" is the semi-automatic annotation process, "Direct" is the direct annotation process described in §4, and "Input" is a baseline where the two input sentences are also the output.

Subtrees Only: Subtree entailment in context (§5) requires every alignment to be between two subtrees. If an aligned component is not a subtree, (i.e. is non-contiguous in the dependency tree) that subgraph will be turned into a subtree by adding the nodes all the way up to the lowest common ancestor. In our case, ("the", "primary", "to") is non-contiguous; adding the lowest common ancestor to this alignment ("goes") yields the subtree alignment: "the", "primary", "goes", "to" – "wins".

Step 4: Entailment Questions The previous step yields a *potential* aligned partition, where aligned subtrees have not yet been tested for entailment. Based on this structure, we generate a set of yes/no subtree entailment questions. Starting at the root, we traverse the aligned partition, and for each alignment (t_1, t_2), two subtree entailment questions are generated: $t_1 \vDash_{s_1} t_2$ and $t_2 \vDash_{s_2} t_1$.

Reusing the example from Figure 9, let t_1 = "the", "primary", "goes", "to" and t_2 = "wins". In the entailment question $t_1 \vDash_{s_1} t_2$, the premise is the original sentence in which t_1 appeared ("The Florida primary goes to Hillary."). The hypothesis is created by substituting t_1 with t_2: $\sigma(s_1, t_1, t_2, \alpha)$. The slot assignment α is deduced directly from the existing subtree alignments; if a child node of t_2 is aligned to a child node of t_1, their slots will be assigned accordingly. If no such alignment exists for a given slot, it will be assigned $\emptyset$ and manifest as a placeholder. In our example, this yields the assignment: "the X primary goes to Y" – "Y wins X", because (X) "Florida" is aligned to itself, and (Y) "Hillary" is aligned to "Clinton". Therefore, the hypothesis we generate is: "Hillary wins Florida."

Annotators are asked if given that the premise is true, the hypothesis is true too. This step determines both the validity and direction of each alignment, creating the final aligned partition.

7.2 Creating the Intersection Set

The aligned partition contains all the information necessary for constructing the intersection set. We create this set by traversing the aligned partition and generating every combination of subtrees, in which only one subtree from each alignment is chosen. Given an alignment between two subtrees, if only subtree is entailed, this subtree is always chosen. If the subtrees are mutually entailing, the number of generated trees is doubled, each one containing a different subtree. If there is no entailment, the subtrees are not aligned, and will not participate in any of the generated sentences.

Overall, this process creates 2^n sentences, where n is the number of mutually entailing subtrees in the aligned partition. For example, the aligned partition in Figure 7 contains 2 mutually-entailing subtrees ("Bernie" = "Sanders", "Hillary" = "HRC"), resulting in 4 intersection sentences (Figure 2). Finally, we generate natural language sentences from the intersection sentences' trees.

8 Annotated Dataset

We hired Mechanical Turk workers to annotate the entire set of 1,800 examples from the sentence fusion (McKeown et al., 2010) and paraphrase (Dolan et al., 2004) datasets presented in §4 using our semi-automatic algorithm. After removing spam annotations, we had an annotated dataset of 1,764 extractive sentence intersections. Again, we retained the 3 most consistent annotations out of 5, per example. For evaluation, we compared the same 200 examples we analyzed in §4. We call this annotation "Semi-Auto", the expert annotation "Expert", and the manual annotation in §4 "Direct".

In terms of measured agreement with Expert, Table 2 shows that Semi-Auto substantially improves over Direct in both the sentence equality and ROUGE-like metrics. Semi-Auto has much higher precision

than both Direct and the input baseline. We hypothesize that explicitly decomposing the intersection task into several local subtree entailment queries allows the annotators to focus on the finer details and nuances of each sentence. Semi-Auto's recall is also better than Direct's because the combinatorial component of creating the entire intersection set is automated.

Although Semi-Auto significantly increases agreement with experts, the agreement is still not perfect. To determine whether these errors stem from annotator quality or algorithmic issues, we sampled 25 input sentence pairs and examined the intersection sentences constructed by Semi-Auto that were not produced by Expert. From this sample of false-positives, 42.2% were actually valid intersections, which were not covered by Expert because of the task's subjective nature. While we used only a single expert, multiple expert annotators could potentially reduce the subjectivity factor. An additional 51.1% stem from real annotation errors by Mechanical Turk workers. A deeper look reveals that these errors result from low-quality annotators who did not follow our instructions. Having said that, the vast majority of these errors manifest as a single missing or superfluous word (in some cases, it is even a function word), while getting the core aspects of the intersection correct. Only 6.7% of the cases were traced to algorithmic errors, most of which are caused by the way English dependency parsers represent coordinations.[5] Overall, it appears that the major cause of error is a portion of Mechanical Turk annotators who failed to follow our instructions accurately. Our semi-automatic algorithm, on the other hand, seems relatively robust.

9 Conclusions

This work resurrects the task of sentence intersection, and addresses the main reason it has remained dormant: lack of annotated data. We show that our new variant of the task, extractive sentence intersection, can be decomposed into several simpler tasks, allowing for high-quality annotation via crowdsourcing. We hope our insights, data, and algorithmic framework provide a foundation for future work on sentence intersection and semantic overlap.

Acknowledgements

This work was supported by the German Research Foundation via the German-Israeli Project Cooperation (grant DA 1600/1-1), the Israel Science Foundation grant 880/12, and by grants from the MAGNET program of the Israeli Office of the Chief Scientist (OCS).

References

Eneko Agirre, Daniel Cer, Mona Diab, and Aitor Gonzalez-Agirre. 2012. Semeval-2012 task 6: A pilot on semantic textual similarity. In **SEM 2012: The First Joint Conference on Lexical and Computational Semantics – Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation (SemEval 2012)*, pages 385–393, Montréal, Canada, 7-8 June. Association for Computational Linguistics.

Regina Barzilay and Kathleen R McKeown. 2005. Sentence fusion for multidocument news summarization. *Computational Linguistics*, 31(3):297–328.

Regina Barzilay, Kathleen R. McKeown, and Michael Elhadad. 1999. Information fusion in the context of multi-document summarization. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pages 550–557, College Park, Maryland, USA, June. Association for Computational Linguistics.

Regina Barzilay. 2003. *Information Fusion for Multidocument Summarization: Paraphrasing and Generation*. Ph.D. thesis, Columbia University.

Chris Biemann. 2013. Creating a system for lexical substitutions from scratch using crowdsourcing. *Language Resources and Evaluation*, 47(1):97–122.

[5] A word on coordinations: English dependency parsers typically represent coordinations ("Bill and Ted") with the first conjunct ("Bill") as the head, and the other conjuncts ("Ted") and coordinators ("and") as its modifiers. This prevents us from modeling entailments between reflexive verbs, such as "X met Y"="X and Y met", using subtrees. To correctly capture the pattern "X and Y met", we need to use Prague-style coordinations (Popel et al., 2013), which place the coordinator ("and") at the head of the structure. This modification, which we did not apply to Semi-Auto, fixes 13 out of 14 cases in which we did not observe a perfect aligned partition during our expert annotation, leaving only 1 exception in 200 examples.

Ido Dagan, Oren Glickman, and Bernardo Magnini. 2006. The PASCAL recognising textual entailment challenge. In Joaquin Quiñonero Candela, Ido Dagan, Bernardo Magnini, and Florence d'Alché Buc, editors, *Machine Learning Challenges*, volume 3944 of *Lecture Notes in Computer Science*, pages 177–190. Springer.

Hal Daume III and Daniel Marcu. 2004. Generic sentence fusion is an ill-defined summarization task. In Stan Szpakowicz Marie-Francine Moens, editor, *Text Summarization Branches Out: Proceedings of the ACL-04 Workshop*, pages 96–103, Barcelona, Spain, July. Association for Computational Linguistics.

Bill Dolan, Chris Quirk, and Chris Brockett. 2004. Unsupervised construction of large paraphrase corpora: Exploiting massively parallel news sources. In *Proceedings of Coling 2004*, pages 350–356, Geneva, Switzerland, Aug 23–Aug 27. COLING.

Katja Filippova and Michael Strube. 2008. Sentence fusion via dependency graph compression. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 177–185, Honolulu, Hawaii, October. Association for Computational Linguistics.

Emiel Krahmer, Erwin Marsi, and Paul van Pelt. 2008. Query-based sentence fusion is better defined and leads to more preferred results than generic sentence fusion. In *Proceedings of ACL-08: HLT, Short Papers*, pages 193–196, Columbus, Ohio, June. Association for Computational Linguistics.

Gerhard Kremer, Katrin Erk, Sebastian Padó, and Stefan Thater. 2014. What substitutes tell us - analysis of an "all-words" lexical substitution corpus. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 540–549, Gothenburg, Sweden, April. Association for Computational Linguistics.

Omer Levy and Ido Dagan. 2016. Annotating relation inference in context via question answering. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, Berlin, Germany, August. Association for Computational Linguistics.

Omer Levy, Torsten Zesch, Ido Dagan, and Iryna Gurevych. 2013. Recognizing partial textual entailment. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 451–455, Sofia, Bulgaria, August. Association for Computational Linguistics.

Dekang Lin and Patrick Pantel. 2001. Dirt: Discovery of inference rules from text. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 323–328. ACM.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In Stan Szpakowicz Marie-Francine Moens, editor, *Text Summarization Branches Out: Proceedings of the ACL-04 Workshop*, pages 74–81, Barcelona, Spain, July. Association for Computational Linguistics.

Erwin Marsi and Emiel Krahmer. 2005. Explorations in sentence fusion. In *Proceedings of the European Workshop on Natural Language Generation*, pages 109–117. Citeseer.

Diana McCarthy and Roberto Navigli. 2007. Semeval-2007 task 10: English lexical substitution task. In *Proceedings of the Fourth International Workshop on Semantic Evaluations (SemEval-2007)*, pages 48–53, Prague, Czech Republic, June. Association for Computational Linguistics.

Kathleen McKeown, Sara Rosenthal, Kapil Thadani, and Coleman Moore. 2010. Time-efficient creation of an accurate sentence fusion corpus. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 317–320, Los Angeles, California, June. Association for Computational Linguistics.

Oren Melamud, Jonathan Berant, Ido Dagan, Jacob Goldberger, and Idan Szpektor. 2013. A two level model for context sensitive inference rules. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1331–1340, Sofia, Bulgaria, August. Association for Computational Linguistics.

Ani Nenkova and Rebecca Passonneau. 2004. Evaluating content selection in summarization: The pyramid method. In Daniel Marcu Susan Dumais and Salim Roukos, editors, *HLT-NAACL 2004: Main Proceedings*, pages 145–152, Boston, Massachusetts, USA, May 2 - May 7. Association for Computational Linguistics.

Rodney D Nielsen, Wayne Ward, and James H Martin. 2009. Recognizing entailment in intelligent tutoring systems. *Natural Language Engineering*, 15(04):479–501.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA, July. Association for Computational Linguistics.

Ellie Pavlick, Pushpendre Rastogi, Juri Ganitkevitch, Benjamin Van Durme, and Chris Callison-Burch. 2015. Ppdb 2.0: Better paraphrase ranking, fine-grained entailment relations, word embeddings, and style classification. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 425–430, Beijing, China, July. Association for Computational Linguistics.

Slav Petrov and Dan Klein. 2007. Improved inference for unlexicalized parsing. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*, pages 404–411, Rochester, New York, April. Association for Computational Linguistics.

Martin Popel, David Mareček, Jan Št?pánek, Daniel Zeman, and Zd?n?k Žabokrtský. 2013. Coordination structures in dependency treebanks. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 517–527, Sofia, Bulgaria, August. Association for Computational Linguistics.

Idan Szpektor, Hristo Tanev, Ido Dagan, Bonaventura Coppola, and Milen Kouylekov. 2015. Unsupervised acquisition of entailment relations from the web. *Natural Language Engineering*, 21:3–47, 1.

Kapil Thadani and Kathleen McKeown. 2011. Towards strict sentence intersection: Decoding and evaluation strategies. In *Proceedings of the Workshop on Monolingual Text-To-Text Generation*, pages 43–53, Portland, Oregon, June. Association for Computational Linguistics.

Kapil Thadani and Kathleen McKeown. 2013. Supervised sentence fusion with single-stage inference. In *Proceedings of the Sixth International Joint Conference on Natural Language Processing*, pages 1410–1418, Nagoya, Japan, October. Asian Federation of Natural Language Processing.

Naomi Zeichner, Jonathan Berant, and Ido Dagan. 2012. Crowdsourcing inference-rule evaluation. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 156–160, Jeju Island, Korea, July. Association for Computational Linguistics.