Towards Developing Generation Algorithms for Text-to-Text Applications

Radu Soricut and Daniel Marcu
Information Sciences Institute
University of Southern California
4676 Admiralty Way, Suite 1001
Marina del Rey, CA 90292
{radu, marcu}@isi.edu

Abstract

We describe a new sentence realization framework for text-to-text applications. This framework uses IDL-expressions as a representation formalism, and a generation mechanism based on algorithms for intersecting IDL-expressions with probabilistic language models. We present both theoretical and empirical results concerning the correctness and efficiency of these algorithms.

1 Introduction

Many of today's most popular natural language applications – Machine Translation, Summarization, Question Answering – are text-to-text applications. That is, they produce textual outputs from inputs that are also textual. Because these applications need to produce well-formed text, it would appear natural that they are the favorite testbed for generic generation components developed within the Natural Language Generation (NLG) community. Over the years, several proposals of generic NLG systems have been made: Penman (Matthiessen and Bateman, 1991), FUF (Elhadad, 1991), Nitrogen (Knight and Hatzivassiloglou, 1995), Fergus (Bangalore and Rambow, 2000), HALogen (Langkilde-Geary, 2002), Amalgam (Corston-Oliver et al., 2002), etc. Instead of relying on such generic NLG systems, however, most of the current text-to-text applications use other means to address the generation need. In Machine Translation, for example, sentences are produced using application-specific "decoders", inspired by work on speech recognition (Brown et al., 1993), whereas in Summarization, summaries are produced as either extracts or using task-specific strategies (Barzilay, 2003). The main reason for which text-to-text applications do not usually involve generic NLG systems is that such applications do not have access to the kind of information that the input representation formalisms of current NLG systems require. A machine translation or summarization system does not usually have access to deep subject-verb or verb-object relations (such as ACTOR, AGENT, PATIENT, POSSESSOR, etc.) as needed by Penman or FUF, or even shallower syntactic relations (such as `subject`, `object`, `premod`, etc.) as needed by HALogen.

In this paper, following the recent proposal made by Nederhof and Satta (2004), we argue for the use of IDL-expressions as an application-independent, information-slim representation language for text-to-text natural language generation. IDL-expressions are created from strings using four operators: concatenation ($\cdot$), interleave ($\|$), disjunction ($\vee$), and lock ($\times$). We claim that the IDL formalism is appropriate for text-to-text generation, as it encodes meaning only via words and phrases, combined using a set of formally defined operators. Appropriate words and phrases can be, and usually are, produced by the applications mentioned above. The IDL operators have been specifically designed to handle natural constraints such as word choice and precedence, constructions such as phrasal combination, and underspecifications such as free word order.

Proceedings of the 43rd Annual Meeting of the ACL, pages 66–74,
Ann Arbor, June 2005. ©2005 Association for Computational Linguistics

NLG System	Representation (formalism)	Representation (computational)	Generation (mechanism)	Generation (computational)
FUF, PENMAN	$\ominus$ Semantic, few meanings	$\oplus$ Linear	$\ominus$ Deterministic	$\oplus$ Linear
Nitrogen, HALogen	$\ominus$ Syntactically/ Semantically grounded	$\ominus$ Exponential	$\oplus$ Non−deterministic via intersection with probabilistic LMs	$\oplus$ Efficient Run−time Optimal Solution
Fergus, Amalgam	$\ominus$ Syntactic dependencies	$\oplus$ Linear	$\oplus$ Non−deterministic via intersection with probabilistic LMs	$\oplus$ Efficient Run−time Optimal Solution
IDL (Nederhof&Satta 2004)	$\oplus$ Word/Phrase based	$\oplus$ Linear	$\ominus$ Deterministic via intersection with CFGs	$\ominus$ Efficient Run−time All Solutions
IDL (this paper)	$\oplus$ Word/Phrase based	$\oplus$ Linear	$\oplus$ Non−deterministic via intersection with probabilistic LMs	$\oplus$ Efficient Run−time Optimal Solution

Table 1: Comparison of the present proposal with current NLG systems.

In Table 1, we present a summary of the representation and generation characteristics of current NLG systems. We mark by $\oplus$ characteristics that are needed/desirable in a generation component for text-to-text applications, and by $\ominus$ characteristics that make the proposal inapplicable or problematic. For instance, as already argued, the representation formalism of all previous proposals except for IDL is problematic ($\ominus$) for text-to-text applications. The IDL formalism, while applicable to text-to-text applications, has the additional desirable property that it is a compact representation, while formalisms such as word-lattices and non-recursive CFGs can have exponential size in the number of words available for generation (Nederhof and Satta, 2004).

While the IDL representational properties are all desirable, the generation mechanism proposed for IDL by Nederhof and Satta (2004) is problematic ($\ominus$), because it does not allow for scoring and ranking of candidate realizations. Their generation mechanism, while computationally efficient, involves intersection with context free grammars, and therefore works by excluding all realizations that are not accepted by a CFG and including (without ranking) all realizations that are accepted.

The approach to generation taken in this paper is presented in the last row in Table 1, and can be summarized as a $\oplus$ tiling of generation characteristics of previous proposals (see the shaded area in Table 1). Our goal is to provide an optimal generation framework for text-to-text applications, in which the representation formalism, the generation mechanism, and the computational properties are all needed and desirable ($\oplus$). Toward this goal, we present a new generation mechanism that intersects IDL-expressions with probabilistic language models. The generation mechanism implements new algorithms, which cover a wide spectrum of run-time behaviors (from linear to exponential), depending on the complexity of the input. We also present theoretical results concerning the correctness and the efficiency input IDL-expression) of our algorithms.

We evaluate these algorithms by performing experiments on a challenging word-ordering task. These experiments are carried out under a high-complexity generation scenario: find the most probable sentence realization under an n-gram language model for IDL-expressions encoding bags-of-words of size up to 25 (up to 10^{25} possible realizations!). Our evaluation shows that the proposed algorithms are able to cope well with such orders of complexity, while maintaining high levels of accuracy.

2 The IDL Language for NLG

2.1 IDL-expressions

IDL-expressions have been proposed by Nederhof & Satta (2004) (henceforth N&S) as a representation for finite languages, and are created from strings using four operators: concatenation ($\cdot$), interleave ($\|$), disjunction ($\vee$), and lock ($\times$). The semantics of IDL-expressions is given in terms of sets of strings.

The concatenation ($\cdot$) operator takes two arguments, and uses the strings encoded by its argument expressions to obtain concatenated strings that respect the order of the arguments; e.g., $\mathrm{a} \cdot \mathrm{b}$ encodes the singleton set $\{\mathrm{ab}\}$. The $\mathcal{I}$nterleave ($\|$) operator interleaves the strings encoded by its argument expressions; e.g., $\|(\mathrm{a} \cdot \mathrm{b}, \mathrm{c})$ encodes the set $\{\mathrm{cab}, \mathrm{acb}, \mathrm{abc}\}$. The $\mathcal{D}$isjunction ($\vee$) operator allows a choice among the strings encoded by its argument expressions; e.g., $\vee(\mathrm{a}, \mathrm{b})$ encodes the set $\{\mathrm{a}, \mathrm{b}\}$. The $\mathcal{L}$ock ($\times$) operator takes only one argument, and "locks-in" the strings encoded by its argument expression, such that no additional material can be interleaved; e.g., $\|(\times(\mathrm{a} \cdot \mathrm{b}), \mathrm{c})$ encodes the set $\{\mathrm{cab}, \mathrm{abc}\}$.

Consider the following IDL-expression:

$$\|(\mathit{finally}, \vee(\times(\mathit{the} \cdot \mathit{prisoners}), \times(\mathit{the} \cdot \mathit{captives})) \cdot \mathit{were} \cdot \mathit{released}) \qquad (1)$$

The concatenation ($\cdot$) operator captures precedence constraints, such as the fact that a determiner like *the* appears before the noun it determines. The lock ($\times$) operator enforces phrase-encoding constraints, such as the fact that *the captives* is a phrase which should be used as a whole. The disjunction ($\vee$) operator allows for multiple word/phrase choice (e.g., *the prisoners* versus *the captives*), and the interleave ($\|$) operator allows for word-order freedom, i.e., word order underspecification at meaning representation level. Among the strings encoded by IDL-expression 1 are the following:

finally the prisoners were released
the captives finally were released
the prisoners were finally released

The following strings, however, are not part of the language defined by IDL-expression 1:

the finally captives were released
the prisoners were released
finally the captives released were

The first string is disallowed because the $\times$ operator locks the phrase *the captives*. The second string is not allowed because the $\|$ operator requires all its arguments to be represented. The last string violates the order imposed by the precedence operator between *were* and *released*.

2.2 IDL-graphs

IDL-expressions are a convenient way to compactly represent finite languages. However, IDL-expressions do not directly allow formulations of algorithms to process them. For this purpose, an equivalent representation is introduced by N&S, called IDL-graphs. We refer the interested reader to the formal definition provided by N&S, and provide here only an intuitive description of IDL-graphs.

We illustrate in Figure 1 the IDL-graph corresponding to IDL-expression 1. In this graph, vertices v_s and v_e are called initial and final, respectively. Vertices v_0, v_2 with in-going $\vdash$-labeled edges, and v_1, v_{20} with out-going $\dashv$-labeled edges, for example, result from the expansion of the $\|$ operator, while vertices v_3, v_4 with in-going ϵ-labeled edges, and v_9, v_{14} with out-going ϵ-labeled edges result from the expansion of the $\vee$ operator. Vertices v_5 to v_8 and v_{10} to v_{13} result from the expansion of the two $\times$ operators, respectively. These latter vertices are also shown to have rank 1, as opposed to rank 0 (not shown) assigned to all other vertices. The ranking of vertices in an IDL-graph is needed to enforce a higher priority on the processing of the higher-ranked vertices, such that the desired semantics for the lock operator is preserved.

With each IDL-graph $G(\pi)$ we can associate a finite language: the set of strings that can be generated by an IDL-specific traversal of $G(\pi)$, starting from v_s and ending in v_e. An IDL-expression π and its corresponding IDL-graph $G(\pi)$ are said to be equivalent because they generate the same finite language, denoted $L(\pi)$.

2.3 IDL-graphs and Finite-State Acceptors

To make the connection with the formulation of our algorithms, in this section we link the IDL formalism with the more classical formalism of finite-state acceptors (FSA) (Hopcroft and Ullman, 1979). The FSA representation can naturally encode precedence and multiple choice, but it lacks primitives corresponding to the interleave ($\|$) and lock ($\times$) operators. As such, an FSA representation must explicitly enumerate all possible interleavings, which are implicitly captured in an IDL representation. This correspondence between implicit and explicit interleavings is naturally handled by the notion of a cut of an IDL-graph $G(\pi)$.

Intuitively, a cut through $G(\pi)$ is a set of vertices that can be reached *simultaneously* when traversing $G(\pi)$ from the initial node to the final node, following the branches as prescribed by the encoded $\mathcal{I}$, $\mathcal{D}$, and $\mathcal{L}$ operators, in an attempt to produce a string in $L(\pi)$. More precisely, the initial vertex v_s is considered a cut (Figure 2 (a)). For each vertex in a given cut, we create a new cut by replacing the start vertex of some edge with the end vertex of that edge, observing the following rules:

- the vertex that is the start of several edges labeled using the special symbol $\vdash$ is replaced by a sequence of all the end vertices of these edges (for example, v_0v_2 is a cut derived from v_s (Figure 2 (b))); a mirror rule handles the special symbol $\dashv$;

- the vertex that is the start of an edge labeled using vocabulary items or ϵ is replaced by the end vertex of that edge (for example, v_1v_2, v_0v_3, v_0v_5, v_0v_6 are cuts derived from v_0v_2, v_0v_2,

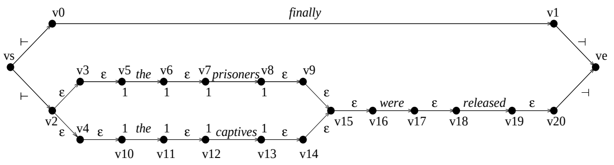

Figure 1: The IDL-graph corresponding to the IDL-expression $\|(\mathit{finally}, \vee(\times(\mathit{the} \cdot \mathit{prisoners}), \times(\mathit{the} \cdot \mathit{captives})) \cdot \mathit{were} \cdot \mathit{released})$.

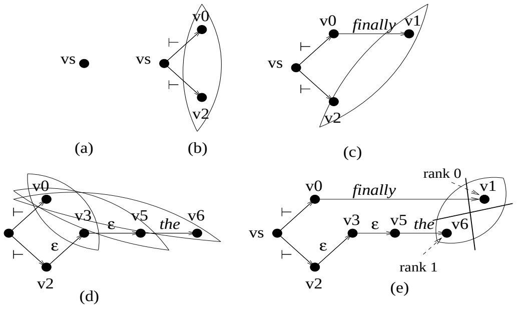

Figure 2: Cuts of the IDL-graph in Figure 1 (a-d). A non-cut is presented in (e).

v_0v_3, and v_0v_5, respectively, see Figure 2 (c-d)), only if the end vertex is not lower ranked than any of the vertices already present in the cut (for example, v_1v_6 is *not* a cut that can be derived from v_0v_6, see Figure 2 (e)).

Note the last part of the second rule, which restricts the set of cuts by using the ranking mechanism. If one would allow v_1v_6 to be a cut, one would imply that *finally* may appear inserted between the words of the locked phrase *the prisoners*.

We now link the IDL formalism with the FSA formalism by providing a mapping from an IDL-graph $G(\pi)$ to an acyclic finite-state acceptor $A(\pi)$. Because both formalisms are used for representing finite languages, they have equivalent representational power. The IDL representation is much more compact, however, as one can observe by comparing the IDL-graph in Figure 1 with the equivalent finite-state acceptor $A(\pi)$ in Figure 3. The set of states of $A(\pi)$ is the set of cuts of $G(\pi)$. The initial state of the finite-state acceptor is the state corresponding to cut v_s, and the final states of the finite-state acceptor are the state corresponding to cuts that contain v_e. In what follows, we denote a state of $A(\pi)$ by the name of the cut to which it corresponds. A transition labeled α in $A(\pi)$ between state $[v'_i \ldots v'_k \ldots v'_j]$ and state $[v''_i \ldots v''_k \ldots v''_j]$ occurs if there is an edge (v'_k, α, v''_k) in $G(\pi)$. For the example in Figure 3, the transition labeled *were* between states $[v_0v_{16}]$ and $[v_0v_{17}]$ occurs because of the edge labeled *were* between nodes v_{16} and v_{17} (Figure 1), whereas the transition labeled *finally* between states $[v_0v_{16}]$ and $[v_1v_{16}]$ occurs because of the edge labeled *finally* between nodes v_0 and v_1 (Figure 1). The two representations $G(\pi)$ and $A(\pi)$ are equivalent in the sense that the language generated by IDL-graph $G(\pi)$ is the same as the language accepted by FSA $A(\pi)$.

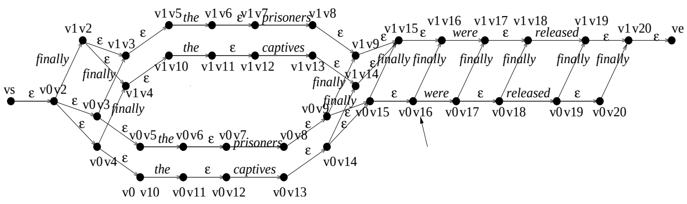

Figure 3: The finite-state acceptor corresponding to the IDL-graph in Figure 1.

It is not hard to see that the conversion from the IDL representation to the FSA representation destroys the compactness property of the IDL formalism, because of the explicit enumeration of all possible interleavings, which causes certain labels to appear repeatedly in transitions. For example, a transition labeled *finally* appears 11 times in the finite-state acceptor in Figure 3, whereas an edge labeled *finally* appears only once in the IDL-graph in Figure 1.

3 Computational Properties of IDL-expressions

3.1 IDL-graphs and Weighted Finite-State Acceptors

As mentioned in Section 1, the generation mechanism we propose performs an intersection of IDL-expressions with n-gram language models. Following (Mohri et al., 2002; Knight and Graehl, 1998), we implement language models using weighted finite-state acceptors (wFSA). In Section 2.3, we presented a mapping from an IDL-graph $G(\pi)$ to a finite-state acceptor $A(\pi)$. From such a finite-state acceptor $A(\pi)$, we arrive at a weighted finite-state acceptor $W(\pi)$, by splitting the states of $A(\pi)$ according to the information needed by the language model to assign weights to transitions. For example, under a bigram language model LM, state $[v_1v_{16}]$ in Figure 3 must be split into three different states, $[prisoners, v_1v_{16}]$, $[captives, v_1v_{16}]$, and $[\mathit{finally}, v_1v_{16}]$, according to which (non-epsilon) transition was last used to reach this state. The transitions leaving these states have the same labels as those leaving state $[v_1v_{16}]$, and are now weighted using the language model probability distributions $p_{LM}(\cdot|prisoners)$, $p_{LM}(\cdot|captives)$, and $p_{LM}(\cdot|\mathit{finally})$, respectively.

Note that, at this point, we already have a naïve algorithm for intersecting IDL-expressions with n-gram language models. From an IDL-expression π, following the mapping $\pi \rightarrow G(\pi) \rightarrow A(\pi) \rightarrow W(\pi)$, we arrive at a weighted finite-state acceptor, on which we can use a single-source shortest-path algorithm for directed acyclic graphs (Cormen et al., 2001) to extract the realization corresponding to the most probable path. The problem with this algorithm, however, is that the premature unfolding of the IDL-graph into a finite-state acceptor destroys the representation compactness of the IDL representation. For this reason, we devise algorithms that, although similar in spirit with the single-source shortest-path algorithm for directed acyclic graphs, perform on-the-fly unfolding of the IDL-graph, with a mechanism to control the unfolding based on the scores of the paths already unfolded. Such an approach has the advantage that prefixes that are extremely unlikely under the language model may be regarded as not so promising, and parts of the IDL-expression that contain them may not be unfolded, leading to significant savings.

3.2 Generation via Intersection of IDL-expressions with Language Models

Algorithm IDL-NGLM-BFS The first algorithm that we propose is algorithm IDL-NGLM-BFS in Figure 4. The algorithm builds a weighted finite-state acceptor W corresponding to an IDL-graph G incrementally, by keeping track of a set of active states, called $active$. The incrementality comes from creating new transitions and states in W originating in these active states, by unfolding the IDL-graph G; the set of newly unfolded states is called $unfold$. The new transitions in W are weighted according to the language model. If a final state of W is not yet reached, the while loop is closed by making the $unfold$ set of states to be the next set of $active$ states. Note that this is actually a breadth-first search (BFS) with incremental unfolding. This algorithm still unfolds the IDL-graph completely, and therefore suffers from the same drawback as the naïve algorithm.

```
IDL-NGLM-BFS$(G, LM)$
1  $active \leftarrow \{[vs^G]\}$
2  $flag \leftarrow 1$
3  while $flag$
4      do $unfold \leftarrow$ UNFOLDIDLG$(active, G)$
5         EVALUATENGLM$(unfold, LM)$
6         if FINALIDLG$(unfold, G)$
7           then $flag \leftarrow 0$
8         $active \leftarrow unfold$
9  return $active$
```

Figure 4: Pseudo-code for intersecting an IDL-graph G with an n-gram language model LM using incremental unfolding and breadth-first search.

The interesting contribution of algorithm IDL-NGLM-BFS, however, is the incremental unfolding. If, instead of line 8 in Figure 4, we introduce mechanisms to control which $unfold$ states become part of the $active$ state set for the next unfolding iteration, we obtain a series of more effective algorithms.

Algorithm IDL-NGLM-A* We arrive at algorithm IDL-NGLM-A* by modifying line 8 in Figure 4, thus obtaining the algorithm in Figure 5. We use as control mechanism a priority queue, $astarQ$, in which the states from $unfold$ are PUSH-ed, sorted according to an admissible heuristic function (Russell and Norvig, 1995). In the next iteration, $active$ is a singleton set containing the state POP-ed out from the top of the priority queue.

Algorithm IDL-NGLM-BEAM We arrive at algorithm IDL-NGLM-BEAM by again modifying line 8 in Figure 4, thus obtaining the algorithm in Figure 6. We control the unfolding using a probabilistic beam $beam$, which, via the BEAMSTATES function, selects as $active$ states only the states in

```
IDL-NGLM-A$^*(G, LM)$
$active \leftarrow \{[vs^G]\}$
$flag \leftarrow 1$
while $flag$
    do $unfold \leftarrow$ UNFOLDIDLG$(active, G)$
       EVALUATENGLM$(unfold, LM)$
       if FINALIDLG$(unfold, G)$
         then $flag \leftarrow 0$
       for each $state$ in $unfold$
           do PUSH$(astarQ, state)$
       $active \leftarrow$ POP$(astarQ)$
return $active$
```

Figure 5: Pseudo-code for intersecting an IDL-graph G with an n-gram language model LM using incremental unfolding and A* search.

```
IDL-NGLM-BEAM$(G, LM, beam)$
$active \leftarrow \{[vs^G]\}$
$flag \leftarrow 1$
while $flag$
    do $unfold \leftarrow$ UNFOLDIDLG$(active, G)$
       EVALUATENGLM$(unfold, LM)$
       if FINALIDLG$(unfold, G)$
         then $flag \leftarrow 0$
       $active \leftarrow$ BEAMSTATES$(unfold, beam)$
return $active$
```

Figure 6: Pseudo-code for intersecting an IDL-graph G with an n-gram language model LM using incremental unfolding and probabilistic beam search.

$unfold$ reachable with a probability higher or equal to the current maximum probability times the probability beam $beam$.

3.3 Computing Admissible Heuristics for IDL-expressions

The IDL representation is ideally suited for computing accurate admissible heuristics under language models. These heuristics are needed by the IDL-NGLM-A* algorithm, and are also employed for pruning by the IDL-NGLM-BEAM algorithm.

For each state S in a weighted finite-state acceptor W corresponding to an IDL-graph G, one can efficiently extract from G – without further unfolding – the set[1] of all edge labels that can be used to reach the final states of W. This set of labels, denoted FE_S^{all}, is an overestimation of the set of future events reachable from S, because the labels under the $\vee$ operators are all considered. From FE_S^{all} and the n-1 labels (when using an n-gram language model) recorded in state S we obtain the set of label sequences of length n-1. This set, denoted FCE_S, is an (over)estimated set of possible future conditioning events for state S, guaranteed to contain the most cost-efficient future conditioning events for state S. Using FCE_S, one needs to extract from FE_S^{all} the set of most cost-efficient future events from under each $\vee$ operator. We use this set, denoted FE_S, to arrive at an admissible heuristic for state S under a language model LM, using Equation 2:

$$h(S) = \Sigma_{e \in FE_S} - \log(\max_{ce \in FCE_S} p_{LM}(e|ce)) \qquad (2)$$

If $h^*(S)$ is the true future cost for state S, we guarantee that $h(S) \leq h^*(S)$ from the way FE_S and FCE_S are constructed. Note that, as it usually happens with admissible heuristics, we can make $h(S)$ come arbitrarily close to $h^*(S)$, by computing increasingly better approximations FCE_S of FCE_S^*. Such approximations, however, require increasingly advanced unfoldings of the IDL-graph G (a complete unfolding of G for state S gives $FCE_S = FCE_S^*$, and consequently $h(S) = h^*(S)$). It follows that arbitrarily accurate admissible heuristics exist for IDL-expressions, but computing them on-the-fly requires finding a balance between the time and space requirements for computing better heuristics and the speed-up obtained by using them in the search algorithms.

3.4 Formal Properties of IDL-NGLM algorithms

The following theorem states the correctness of our algorithms, in the sense that they find the maximum probability path encoded by an IDL-graph under an n-gram language model.

Theorem 1 *Let π be an IDL-expression, G(π) its IDL-graph, and W(π) its wFSA under an n-gram language model LM. Algorithms* IDL-NGLM-BFS *and* IDL-NGLM-A* *find the*

[1] Actually, these are multisets, as we treat multiply-occurring labels as separate items.

path of maximum probability under LM. Algorithm IDL-NGLM-BEAM *finds the path of maximum probability under LM, if all states in W(π) along this path are selected by its* BEAMSTATES *function.*

The proof of the theorem follows directly from the correctness of the BFS and A* search, and from the condition imposed on the beam search.

The next theorem characterizes the run-time complexity of these algorithms, in terms of an input IDL-expression π and its corresponding IDL-graph $G(\pi)$ complexity. There are three factors that linearly influence the run-time complexity of our algorithms: a is the maximum number of nodes in $G(\pi)$ needed to represent a state in $A(\pi)$ – a depends solely on π; w is the maximum number of nodes in $G(\pi)$ needed to represent a state in $W(\pi)$ – w depends on π and n, the length of the context used by the n-gram language model; and K is the number of states of $W(\pi)$ – K also depends on π and n. Of these three factors, K is by far the predominant one, and we simply call K the complexity of an IDL-expression.

Theorem 2 *Let π be an IDL-expression, $G(\pi)$ its IDL-graph, $A(\pi)$ its FSA, and $W(\pi)$ its wFSA under an n-gram language model. Let $V[A(\pi)]$ be the set of states of $A(\pi)$, and $V[W(\pi)]$ the set of states of $W(\pi)$. Let also $a = \max_{c \in V[A(\pi)]}|c|$, $w = \max_{c \in V[W(\pi)]}|c|$, and $K = |V[W(\pi)]|$. Algorithms* IDL-NGLM-BFS *and* IDL-NGLM-BEAM *have run-time complexity* $\mathrm{O}(awK)$. *Algorithm* IDL-NGLM-A* *has run-time complexity* $\mathrm{O}(awK \log K)$.

We omit the proof here due to space constraints. The fact that the run-time behavior of our algorithms is linear in the complexity of the input IDL-expression (with an additional log factor in the case of A* search due to priority queue management) allows us to say that our algorithms are efficient with respect to the task they accomplish.

We note here, however, that depending on the input IDL-expression, the task addressed can vary in complexity from linear to exponential. That is, for the intersection of an IDL-expression $\pi = \|(w_1, \ldots, w_n)$ (bag of n words) with a trigram language model, we have $a(\pi) = n$, $w(\pi) = n + 2$, $K = c^n, c > 1$, and therefore a $\mathrm{O}(n^2c^n)$ complexity. This exponential complexity comes as no surprise given that the problem of intersecting an n-gram language model with a bag of words is known to be NP-complete (Knight, 1999). On the other hand, for intersecting an IDL-expression $\pi = w_1 \cdot \ldots \cdot w_n$ (sequence of n words) with a trigram language model, we have $a(\pi) = 1$, $w(\pi) = 3$, and $K = n$, and therefore an $\mathrm{O}(n)$ generation algorithm.

In general, for IDL-expressions for which a is bounded, which we expect to be the case for most practical problems, *our algorithms perform in polynomial time in the number of words available for generation.*

4 Evaluation of IDL-NGLM Algorithms

In this section, we present results concerning the performance of our algorithms on a word-ordering task. This task can be easily defined as follows: from a bag of words originating from some sentence, reconstruct the original sentence as faithfully as possible. In our case, from an original sentence such as *"the gifts are donated by american companies"*, we create the IDL-expression $\langle s \rangle \cdot \|(\mathit{the}, \mathit{gifts}, \mathit{donated}, \mathit{companies}, \mathit{by}, \mathit{are}, \mathit{american}) \cdot \langle /s \rangle$, from which some algorithm realizes a sentence such as *"donated by the american companies are gifts"*. Note the natural way we represent in an IDL-expression beginning and end of sentence constraints, using the $\cdot$ operator. Since this is generation from bag-of-words, the task is known to be at the high-complexity extreme of the run-time behavior of our algorithms. As such, we consider it a good test for the ability of our algorithms to scale up to increasingly complex inputs.

We use a state-of-the-art, publicly available toolkit[2] to train a trigram language model using Kneser-Ney smoothing, on 10 million sentences (170 million words) from the Wall Street Journal (WSJ), lower case and no final punctuation. The test data is also lower case (such that upper-case words cannot be hypothesized as first words), with final punctuation removed (such that periods cannot be hypothesized as final words), and consists of 2000 unseen WSJ sentences of length 3-7, and 2000 unseen WSJ sentences of length 10-25.

The algorithms we tested in this experiments were the ones presented in Section 3.2, plus two baseline algorithms. The first baseline algorithm, L, uses an

[2] http://www.speech.sri.com/projects/srilm/

inverse-lexicographic order for the bag items as its output, in order to get the word *the* on sentence initial position. The second baseline algorithm, G, is a greedy algorithm that realizes sentences by maximizing the probability of joining any two word sequences until only one sequence is left.

For the A* algorithm, an admissible cost is computed for each state S in a weighted finite-state automaton, as the sum (over all unused words) of the minimum language model cost (i.e., maximum probability) of each unused word when conditioning over all sequences of two words available at that particular state for future conditioning (see Equation 2, with $FE_S = FE_S^{all}$). These estimates are also used by the beam algorithm for deciding which IDL-graph nodes are not unfolded. We also test a greedy version of the A* algorithm, denoted A$^*_k$, which considers for unfolding only the nodes extracted from the priority queue which already unfolded a path of length greater than or equal to the maximum length already unfolded minus k (in this notation, the A* algorithm would be denoted A$^*_\infty$). For the beam algorithms, we use the notation B$_p$ to specify a probabilistic beam of size p, i.e., an algorithm that beams out the states reachable with probability less than the current maximum probability times p.

Our first batch of experiments concerns bags-of-words of size 3-7, for which exhaustive search is possible. In Table 2, we present the results on the word-ordering task achieved by various algorithms. We evaluate accuracy performance using two automatic metrics: an identity metric, ID, which measures the percent of sentences recreated exactly, and BLEU (Papineni et al., 2002), which gives the geometric average of the number of uni-, bi-, tri-, and four-grams recreated exactly. We evaluate the search performance by the percent of Search Errors made by our algorithms, as well as a percent figure of Estimated Search Errors, computed as the percent of searches that result in a string with a lower probability than the probability of the original sentence. To measure the impact of using IDL-expressions for this task, we also measure the percent of unfolding of an IDL graph with respect to a full unfolding. We report speed results as the average number of seconds per bag-of-words, when using a 3.0GHz CPU machine under a Linux OS.

ALG	ID	BLEU	Search	Unfold	Speed
	(%)		Errors (%)	(%)	(sec./bag)
L	2.5	9.5	97.2 (95.8)	N/A	.000
G	30.9	51.0	67.5 (57.6)	N/A	.000
BFS	67.1	79.2	0.0 (0.0)	100.0	.072
A*	67.1	79.2	0.0 (0.0)	12.0	.010
A$^*_1$	60.5	74.8	21.1 (11.9)	3.2	.004
A$^*_2$	64.3	77.2	8.5 (4.0)	5.3	.005
B$_{0.2}$	65.0	78.0	9.2 (5.0)	7.2	.006
B$_{0.1}$	66.6	78.8	3.2 (1.7)	13.2	.011

Table 2: Bags-of-words of size 3-7: accuracy (ID, BLEU), Search Errors (and Estimated Search Errors), space savings (Unfold), and speed results.

The first notable result in Table 2 is the savings achieved by the A* algorithm under the IDL representation. At no cost in accuracy, it unfolds only 12% of the edges, and achieves a 7 times speed-up, compared to the BFS algorithm. The savings achieved by not unfolding are especially important, since the exponential complexity of the problem is hidden by the IDL representation via the folding mechanism of the $\|$ operator. The algorithms that find sub-optimal solutions also perform well. While maintaining high accuracy, the A$^*_2$ and B$_{0.2}$ algorithms unfold only about 5-7% of the edges, at 12-14 times speed-up.

Our second batch of experiments concerns bag-of-words of size 10-25, for which exhaustive search is no longer possible (Table 3). Not only exhaustive search, but also full A* search is too expensive in terms of memory (we were limited to 2GiB of RAM for our experiments) and speed. Only the greedy versions A$^*_1$ and A$^*_2$, and the beam search using tight probability beams (0.2-0.1) scale up to these bag sizes. Because we no longer have access to the string of maximum probability, we report only the percent of Estimated Search Errors. Note that, in terms of accuracy, we get around 20% Estimated Search Errors for the best performing algorithms (A$^*_2$ and B$_{0.1}$), which means that 80% of the time the algorithms are able to find sentences of equal or better probability than the original sentences.

5 Conclusions

In this paper, we advocate that IDL expressions can provide an adequate framework for develop-

ALG	ID	BLEU	Est. Search	Speed
	(%)		Errors (%)	(sec./bag)
L	0.0	1.4	99.9	0.0
G	1.2	31.6	83.6	0.0
A_1^*	5.8	47.7	34.0	0.7
A_2^*	7.4	51.2	21.4	9.5
$B_{0.2}$	9.0	52.1	23.3	7.1
$B_{0.1}$	12.2	52.6	19.9	36.7

Table 3: Bags-of-words of size 10-25: accuracy (ID, BLEU), Estimated Search Errors, and speed results.

ing text-to-text generation capabilities. Our contribution concerns a new generation mechanism that implements intersection between an IDL expression and a probabilistic language model. The IDL formalism is ideally suited for our approach, due to its efficient representation and, as we show in this paper, efficient algorithms for intersecting, scoring, and ranking sentence realizations using probabilistic language models.

We present theoretical results concerning the correctness and efficiency of the proposed algorithms, and also present empirical results that show that our algorithms scale up to handling IDL-expressions of high complexity. Real-world text-to-text generation tasks, such as headline generation and machine translation, are likely to be handled graciously in this framework, as the complexity of IDL-expressions for these tasks tends to be lower than the complexity of the IDL-expressions we worked with in our experiments.

Acknowledgment

This work was supported by DARPA-ITO grant NN66001-00-1-9814.

References

Srinivas Bangalore and Owen Rambow. 2000. Using TAG, a tree model, and a language model for generation. In *Proceedings of the 1st International Natural Language Generation Conference*.

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

Peter F. Brown, Stephen A. Della Pietra, Vincent J. Della Pietra, and Robert L. Mercer. 1993. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263–311.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2001. *Introduction to Algorithms*. The MIT Press and McGraw-Hill. Second Edition.

Simon Corston-Oliver, Michael Gamon, Eric K. Ringger, and Robert Moore. 2002. An overview of Amalgam: A machine-learned generation module. In *Proceedings of the International Natural Language Generation Conference*.

Michael Elhadad. 1991. FUF User manual — version 5.0. Technical Report CUCS-038-91, Department of Computer Science, Columbia University.

John E. Hopcroft and Jeffrey D. Ullman. 1979. *Introduction to automata theory, languages, and computation*. Addison-Wesley.

Kevin Knight and Jonathan Graehl. 1998. Machine transliteration. *Computational Linguistics*, 24(4):599–612.

Kevin Knight and Vasileios Hatzivassiloglou. 1995. Two level, many-path generation. In *Proceedings of the Association of Computational Linguistics*.

Kevin Knight. 1999. Decoding complexity in word-replacement translation models. *Computational Linguistics*, 25(4):607–615.

Irene Langkilde-Geary. 2002. *A foundation for general-purpose natural language generation: sentence realization using probabilistic models of language*. Ph.D. thesis, University of Southern California.

Christian Matthiessen and John Bateman. 1991. *Text Generation and Systemic-Functional Linguistic*. Pinter Publishers, London.

Mehryar Mohri, Fernando Pereira, and Michael Riley. 2002. Weighted finite-state transducers in speech recognition. *Computer Speech and Language*, 16(1):69–88.

Mark-Jan Nederhof and Giorgio Satta. 2004. IDL-expressions: a formalism for representing and parsing finite languages in natural language processing. *Journal of Artificial Intelligence Research*, 21:287–317.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the Association for Computational Linguistics (ACL-2002)*, pages 311–318, Philadelphia, PA, July 7-12.

Stuart Russell and Peter Norvig. 1995. *Artificial Intelligence. A Modern Approach*. Prentice Hall, Englewood Cliffs, New Jersey.