Real-Time Stochastic Language Generation for Dialogue Systems

Nathanael Chambers
Florida Institute for Human and Machine Cognition
40 South Alcaniz Street
Pensacola, FL 32502
nchambers@ihmc.us

Abstract

This paper describes Acorn, a sentence planner and surface realizer for dialogue systems. Improvements to previous stochastic word-forest based approaches are described, countering recent criticism of this class of algorithms for their slow speed. An evaluation of the approach with semantic input shows runtimes of a fraction of a second and presents results that suggest it is also portable across domains.

1 Introduction

This paper describes Acorn, a real-time sentence planner and surface realizer for dialogue systems that is independent of a specific domain. Acorn is based on a two-phased grammar and stochastic approach, such as the HALogen system [Langkilde-Geary, 2002], but offers several improvements to make it more realistic for dialogue use. The first is to offer an algorithm for *trickle-down features* that passes head/foot features through the grammar as the initial word forest is created, allowing the grammar to broadly represent phenomena such as wh-movement. The second is to more tightly link the grammar to a lexicon and represent syntactic properties such as number, person, and tense to constrain the over-generation process. Lastly, efficiency improvements are described which further decrease the runtime of the system, allowing Acorn to be used in a real-time dialogue context. It is named Acorn, based on the word *forests* that are created and searched.

The task of Natural Language Generation is frequently split into three somewhat disjoint steps: document planning, microplanning (reference and sentence planning) and surface realization. Document planning is a more reduced task in dialogue, mainly involving *content determination* since there is no need for a document. Since the system follows a notion of discourse, content determination is typically performed by some reasoner external to generation, such as a Task Manager. This paper addresses the sentence planning and surface realization steps, assuming that content determination and referential generation has already occurred and is represented in a high-level semantics.

This stochastic approach involves two phases; the first uses a grammar to over-generate the possible realizations of an input form into a word forest, and the second uses a language model to choose the preferred path through the forest. This approach is attractive to dialogue systems because it offers flexibility and adaptivity that cannot be achieved through most symbolic systems. By over-generating possible utterances, the (sometimes dynamic) language models can decide which is more natural in the current context. Other advantages include domain independence and an under-specified input. The main disadvantages most often cited include a very slow runtime and the inability to capture complex linguistic constraints, such as wh-movement. The latter is a side effect of the word-forest creation algorithm and a solution to broaden the coverage of language is presented in this paper. The issue of runtime is critical to dialogue.

Slow runtime is a two-fold problem: the word-forest that is generated is extremely large and often not linguistically constrained, and second, the algorithm has not been efficiently implemented. These issues must be addressed before stochastic approaches can be suited for dialogue. Langkilde [Langkilde, 2000] provides an evaluation of coverage of HALogen and shows runtimes around 28 seconds for sentences with average lengths of 22 words. Callaway [Callaway, 2003] later commented on the runtime that *HALogen is anywhere from 6.5 to 16 times slower* than the symbolic realizer FUF/SURGE (which may also be too slow for dialogue). This paper shows that more work can be done in stochastic generation to reduce the runtime by constraining the grammar and making simple algorithm improvements. Runtimes of only a fraction of one second are presented.

The next section provides a brief background on stochastic generation, followed by a description of Acorn in section 3. The description presents several new grammar additions to broaden language coverage, including a mechanism, called trickle-down features, for representing head and foot features in the grammar. Section 4 describes the evaluation of Acorn, as well as the results concerning domain independence and the overall runtime. A brief discussion and related work follows the evaluation.

2 Background

The task of Content Determination is typically relegated to a module outside of the Generation component, such as with a Task Manager or other reasoning components. This leaves the tasks of Sentence Planning and Surface Realization as the

main steps in dialogue generation, and this paper is describing a module that performs both. The task of referential generation is not addressed, and it is assumed that each logical input is a single utterance, thus removing the need for multiple sentence generation.

Traditionally, surface realization has been performed through templates or more complex syntactic grammars, such as the FUF/SURGE system [Elhadad and Robin, 1996]. Template-based approaches produce inflexible output that must be changed in every new domain to which the system is ported. Symbolic approaches produce linguistically correct utterances, but require a syntactic input and typically have runtimes that are impractical for dialogue. Requiring word choice to be finished beforehand, including most syntactic decisions, puts a heavy burden on dialogue system designers.

Stochastic approaches have recently provided a new method of reducing the need for syntactic input and produce flexible generation in dialogue. HALogen [Langkilde-Geary, 2002] was one of the first stochastic generation systems, providing a two-phased approach that allowed the system designer to use an under-specified input. The first phase uses a hand written grammar that over-generates possible word orderings into a word forest. The second phase uses an n-gram language model to choose the highest probability path through the forest, returning this path as the generated sentence. This approach was first used in a dialogue system in [Chambers and Allen, 2004] as an attempt to create a domain independent surface realizer. A human evaluation showed a slight decline in naturalness when moved to a new domain. The stochastic approach was shown in [Langkilde, 2000] to produce good coverage of the Penn Treebank, but its runtime was significantly slow and others have suggested the stochastic approach is not feasible for dialogue.

3 Acorn: System Description

3.1 Input Form

The input to Acorn is a semantic feature-value form rooted on the *type* of speech act. On the top level, the :speechact feature gives the type (i.e. sa_tell, sa_yn-question, sa_accept, etc.), the :terms feature gives the list of semantic, content bearing terms, and the :root feature gives the variable of the root term in the utterance. Other features are allowed and often required, such as a :focus for wh-questions. Each term in the :terms list is a feature-value structure based on thematic roles, as used in many other representations (e.g. Verbnet [Kipper *et al.*, 2000]). This utterance input is a syntactically modified version of the domain independent Logical Form described in [Dzikovska *et al.*, 2003].

Each term is specified by the features: *:indicator, :class*, optional *:lex*, and any other relevant thematic roles (e.g. *:agent, :theme*, etc.). The *:indicator* indicates the type or function of the term and takes the values THE, A, F, PRO, and QUANTITY-TERM. THE represents a grounded object in the discourse, A represents an abstract object, F is a functional operator, PRO is used for references, and QUANTITY-TERM represents quantities expressed in various scales. There are other indicators, but the details are beyond the scope of this paper. The *:class* specifies the semantic class of the term, and the *:lex* is the root lexical item for the term. Lex is an optional feature and is created from the *:class* if it is not present in the input. Figure 1 gives the specification of the input, and figure 2 shows an example input to Acorn for the utterance, *'I want a 2.4 gigahertz computer'*. Appendix A provides further examples of both semantic and lexical inputs.

```
<UTTERANCE> ::=
 (utt :speechact <act> :root <variable>
      <FEATURE-VALUE>*
      :terms (<TERM>*))

<TERM> ::=
 (<variable> :indicator <indicator>
       :class <class> <FEATURE-VALUE>*)

<FEATURE-VALUE> ::=
 <keyword> <value>
```

Figure 1: BNF for the input to Acorn. A keyword is a symbol preceded by a colon, and a value is any valid symbol, variable, or list.

```
(utt :speechact sa_tell :root v8069 :terms
 ((v8324 :indicator speechact :class sa_tell :content v8069)
  (v8069 :indicator f :class want :lex want :theme v8173
         :experiencer v7970 :tense present)
  (v7970 :indicator pro :class person :lex i :context-rel i)
  (v8173 :indicator a :class computer :lex computer
         :assoc-with v8161)
  (v8161 :indicator quantity-term :class speed-unit
         :lex gigahertz :quantity v8225)
  (v8225 :indicator quantity-term :class number :value 2.4)))
```

Figure 2: Input to Acorn for the utterance, 'I want a 2.4 ghz computer.' This input provides the lexical items for the utterance, but these are typically absent in most cases.

3.2 Grammar Rules

The grammar rules in Acorn convert the input utterance into word orderings by matching keywords (features) in each *term*. A unique aspect of Acorn is that the utterance level features can also be matched at any time. It is often necessary to write a rule based on the current speech act type. The left-hand side (LHS) of a rule showing both options is given here:

```
(grule focus
    (:subject ?s)
    :g (:speechact ?act sa_tell)
  >>
    ...)
```

Each rule matches keywords in its LHS to the current *term* and binds the values of the keywords in the *term* to the variables in the LHS. In the above example, the variable *?s* would be bound to the subject of the term, and the variable *?act* is bound to the top-level :speechact value. A LHS element that is preceded by the *:g* symbol indicates a top-level (global)

feature. In this example, the value sa_tell is also specified as a requirement before the rule can match.

When matched, the right-hand side (RHS) offers several different options of processing. As in HALogen, the recasting (changing a keyword to a new keyword, such as converting a semantic role into a syntactic one), substitution (removing a keyword and its value, or just changing its value), and ordering rules (specifying phrasal and word-level ordering in the word forest) are supported. Two additional rules are supported in Acorn that are able to handle wh-movement and other head features. The first is called *empty-creation* and its complement is *filling*. In order to effectively use these rules, a method of passing head and/or foot features is needed. The following describes trickle-down features, followed by a description of the empty-creation and filling rules.

Trickle-Down Features

A drawback of the grammar phase is that all features in the terms must be explicitly coded in the rules, otherwise they are discarded when ordering rules are applied. Using a simple example of subject-object placement, the following *ordering rule* places the subject in front of the verb, and the object behind.

```
(grule subject
    (:subject ?s)
    (:object ?o)
  >>
    (-> (?s) (?rest) (?o)))
```

Three new branches are created in the forest, one each for (?s), (?rest), and (?o). This rule creates a branch in the word forest that is a conjunct of three non-terminal nodes:

```
N3 -> N4 N5 N6
```

Processing of the (?s) and (?o) branches is restarted at the top of the grammar, but they do not contain any features (the ?rest variable is a catch-all variable that represents all features not matched in the LHS). Indeed, it is possible to write rules with a list of optional features, and include them in the RHS:

```
(grule subject
    (:subject ?s)
    (:object ?o)
  &keep &optional
    (:gap ?g)
  >>
  (-> (?s :gap ?g) (?rest) (?o :gap ?g)))
```

However, this quickly leads to bloated rules and can slow the matching procedure considerably. It is very intuitive to keep features like head and foot features hidden from the grammar writer as much as possible. This is accomplished through what we are calling *trickle-down features*. The syntax for these special case features includes an asterisk before the name, as in *:*gap*. The result of using these features is to get the effect of the latter rule with the ease of use in the former rule. It essentially trickles down the features until their appropriate place in the input utterance is found. Figure 3 shows the feature 'searching' for its correct path. One use of this is shown in the following examples of the *empty-creation* and *filling* rules.

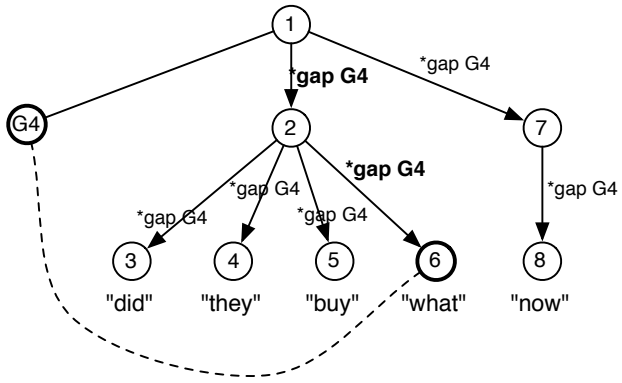

Figure 3: A graphical representation of trickle down features. The gap head feature can be seen percolating to each node, finding its true path (1->2->6) to the wh-term *what*, and linking the filler with the gap (6->G4).

Empty-Creation

When building the word forest, we often need to create a gap node that will be filled later by movement phenomena, such as in wh-questions. The content of the node may not be known, but through *empty-creation*, we can instantiate a variable and link it to the current location in the word forest. This variable can then be attached to a special *trickle-down feature* which is implicitly passed through the grammar. The following is an example of an empty-creation rule:

```
(grule wh-question
    (:root ?r +)  ;; root term
    :g (:speechact ?s sa_wh-question)
    ((gentemp "?GAP") ?wh-gap)
  >>
    (g-> ?wh-gap)
    (->  ?wh-gap (?rest :*gap ?wh-gap)))
```

The first half of the RHS (the g-> rule) creates a global variable and binds a new word forest node label to it. This label is then used in the second half of the RHS where the node is inserted into the word forest, and as of now, is empty. The variable is then passed as a trickle-down feature :*gap to the current term using the ?rest catch-all variable. This rule is applied to node 1 in figure 3, creating gap node G4 and the ?rest node 2, passing the :*gap through the forest.

Filling

Filling rules perform the wh-movement needed in wh-questions and many complement structures. Filling in the context of Acorn can be seen as binding a gap variable that has already been created through an empty-creation rule. The following is an example filling rule that completes the above wh-gap example.

```
(grule gap-fill
    (:indicator ?i wh-term)
    (:*gap ?gap)
  >>
    (b-> ?gap (?rest)))
```

This rule checks that the current term is a *wh-term* that has a gap head feature. The RHS (the b-> rule) binds the current term to the gap that has already been created, filling the empty node in the word forest. The Filling rule essentially grafts a branch onto a location in the word forest that has previously been created by an Empty-Creation rule. The dotted line in figure 3 is created by such a Filling rule.

3.3 Grammar Over-Generation

One of the main attractions of the two-phased approach is that the grammar in the first phase can be left linguistically unconstrained and over-generates many possibilities for an input utterance. However, the statistical second phase may then be over-burdened with the task of searching it. The converse problem arises when the first stage is too constrained and does not produce enough realizations to be natural and flexible, perhaps removing the need for a stochastic phase entirely. There needs to be a balance between the two stages. The processing time is also critical in that over-generation can take too much time to be useful for dialogue.

The grammar used in HALogen largely relied on the over-generation first phase to ensure full coverage of the output. It also reduced the number of rules in the grammar. Subject-verb agreement was loosely enforced, particularly with subject number. Also, singular and plural nouns were both generated when the input was unspecified, doubling the size of the noun phrase possibilities. One of the biggest over-generations was in morphology. HALogen has its own morphology generator that relies on over-generating algorithms rather than a lexicon to morph words. The typical word forest then contains many unknown words that are ignored during the stochastic search, but which explode the size of the word forest. Lastly, modifiers are over-generated to appear both in front of and behind the head words.

Our approach removes the above over-generation and links a lexicon to the grammar for morphology. Subject-verb agreement is enforced where possible without dramatically increasing the grammar size, nouns are only made plural when the input specifies so (under the assumption that the input would contain such semantically critical information), and modifiers are placed in specific locations on certain phrases (i.e. adjectives are always premodifiers for nouns, complements of infinitive verbs are postmodifiers, etc.).

These changes greatly reduce the runtime of the first phase and directly affect the runtime of the second phase by creating smaller word forests.

3.4 Algorithm

Forest Creation

Word forest creation begins with the input utterance, such as the one in figure 2. The top level utterance features are stored in a global feature list, easily accessed by the grammar rules if need be. The *:root* feature points to the root semantic term given in the list of *:terms*. This root term is then processed, beginning at the top of the grammar.

The grammar is pre-processed and each rule is indexed in a hash table of features according to the least popular feature in the rule. For example, if a rule has two features, *:theme* and *:agent*, and *:agent* only appears in 8 rules while *:theme* appears in 14, the rule will be added to the list of rules in the *:agent* bin. During processing of an input term, all of the term's features are extracted and the rules under each feature in the hash table are merged into an ordered subset of the full grammar. This process differs from HALogen and its successors by vastly limiting the number of rules that are checked against each input. Instead of checking 250 rules, we may only check the relevant 20 rules. After a grammar rule matches, the index is queried again with the new term(s) from the RHS of the rule. A new subset of the grammar is created and used to continue processing through the grammar.

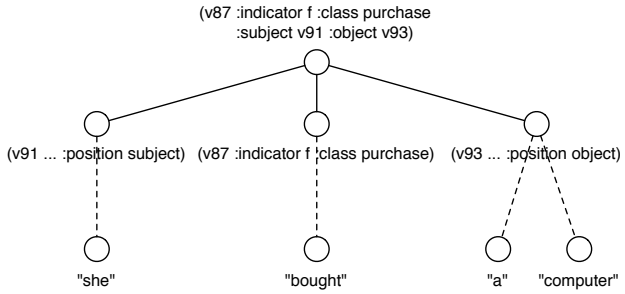

Figure 4: A word forest created from the Acorn grammar.

RHS expansions create (1) ordering constraints, (2) new branches, and (3) feature modifications to the current term. Options (1) and (2) are typically done with ordering rules such as the following RHS:

```
(-> (?s :position subject)
    (?rest) (?o :position object))
```

The variables are either bound from the LHS conditions, or are unbound (conditions that follow the *&optional* indicator in the LHS) and ignored during RHS expansion. The *?rest* variable is a special case variable which refers to the current term and its features that do not appear in the LHS (by default, features in the LHS that are matched are removed from the term, unless they follow a *&keep* indicator). In the above example, there will be a new conjunction branch with three child nodes in the word forest, as shown in figure 4.

When this rule is matched, the ?s node will bind its variable that must point to one of the terms in the input utterance's *:terms* list. Processing will now begin with that term, attaching any features in the RHS to it (in this example, :position subject), at the top of the grammar. Once completed, processing will continue with the current term (?rest) until the grammar is exhausted. Finally, the third term (?o ...) will begin at the top of the grammar. As discussed in section 3.2, any trickle-down features in the current term are appended to the three terms when processing begins/continues on each of them.

A term attempts to match each rule in the grammar until a RHS creates a leaf node. This is accomplished by a RHS expansion into an initial atom that is a string. Finally, inline functions are allowed to be used in the grammar. The following example calls the function *stringify* and its returned value is bound to the *?str* variable. These calls are typically used to access the lexicon.

```
(grule stringify
       (:lex ?lex)
       ;; convert lexical item to string
       ((stringify ?lex) ?str)
  &optional
       (:cat ?cat)
  >>
  (-> (?str :cat ?cat)))
```

PathFinder

The PathFinder module of Acorn is the second stage, responsible for determining the most likely path through the forest. In this stage, the hypotheses from the grammar are analyzed and the top word ordering is chosen based on n-gram stochastic models derived from corpora.

The algorithm we implemented in PathFinder is largely the same as the one described in [Langkilde, 2000]. It is a dynamic programming algorithm that stores the top m phrases at each decision point based on the leading and trailing words in the phrase. When dealing with n-grams, we only need to keep track of the first $n-1$ and the last $n-1$ words in each phrase. Our approach not only tracks these *features* as Langkilde calls them, but PathFinder also sorts the top m phrases and prunes any duplicates. Pruning duplicates offers an advantage in runtime when the phrases are merged with neighboring phrases. The complexity analysis is still $O(m*m) = O(m^2)$, but in practice, pruning phrases reduces the number of phrases to some number less than m.

The largest change to the algorithm is that we added dynamic interpolation of language models. PathFinder can load any number of models and interpolate them together during n-gram analysis using an input set of weights. PathFinder also has the capability to use feature-based models and word history models.

Feature models, such as part of speech n-grams, model the features[1] of forest leaves instead of the lexical items. The Forest Creation stage is able to output features in addition to lexical items, as seen in the RHS of this forest leaf:

```
N6 :POS NN :BASE COMPUTER -> "COMPUTERS"
```

There are two 'features' on this leaf, *pos* and *base*. Parameters can be passed to PathFinder that command it to use the features instead of the RHS string when applying a language model to the forest. This option is not evaluated in this paper, but is a promising option for future work.

Word history models keep track of the current discourse and monitor word usage, providing a history of word choice and calculating a unigram probability for each word. The PathFinder is updated on each utterance in the dialogue and applies a decaying word history approach, similar to the work in [Clarkson and Robinson, 1997]. This model is not evaluated in this paper, but is useful in portraying the breadth of coverage that a stochastic phase can provide to dialogue.

[1]Here we refer to features in the grammar phase, as in feature-values. These are not to be confused with the features of Langkilde in the forest search phase.

:action	:co-theme	:property
:addressee	:cognizer	:purpose
:affected	:compared-to	:rank
:agent	:cost	:result
:along-with	:effect	:sit-val
:associated	:entity	:state
:attribute	:event-relative	:theme
:beneficiary	:experiencer	:time-duration-rel
:cause	:of	:value
:center	:patient	

Figure 5: The main semantic features in Acorn's grammar.

4 Evaluation

The three factors that are most important in evaluating dialogue generation is portability, coverage, and speed. Other factors include naturalness, flexibility, and many more, but the above three are evaluated in this paper to address concerns of domain independent generation and real-time dialogue. During one's efforts to address the latter concern by constraining the size of the word forest, it is very easy to lose the former.

4.1 The Grammar

Acorn's grammar contains 189 rules and is heavily semantic based, although the semantic features and concepts are transformed into syntactic features before word ordering is decided. It is possible to input a syntactic utterance, but this evaluation is only concerned with semantic input. The grammar was created within the context of a computer purchasing domain in which the dialogue system is a collaborative assistant that helps the user define and purchase a computer. We had a corpus of 216 utterances from developers of the system who created their own mock dialogues. The grammar was constructed mainly based on these parsed utterances. Other domains such as an underwater robotic mine search and a database query interface were used to represent as many semantic roles as possible. The list of the main semantic features in Acorn's grammar is provided in figure 5.

4.2 Evaluation Methodology

Each utterance that was able to be parsed in our target dialogues was automatically transformed into the input syntax of Acorn. These inputs were pushed through Acorn, resulting in a single, top ranked utterance. This utterance was compared to the target utterance using the Generation String Accuracy metric. This metric compares a target string to the generated string and counts the number of word movements (M), substitutions (S), deletions (D), and insertions (I) (not counting deletions and insertions implicitly included in movements). The metric is given below (L is the number of tokens in the target string):

$$1 - \frac{M + I + D + S}{L} \quad (1)$$

Before comparison, all contractions were split into single lexical items to prevent the metric from penalizing semantically similar phrases (e.g. *aren't* to *are not*). The Simple

Utterance Lengths				
number of words	1-2	3-5	6-9	10-
Number of utterances	661	177	109	39

Figure 6: Number of utterances of each word length. The majority are grounding/acknowledgements (661 utterances out of 986). We only evaluated those of length 3 or more, 325 utterances.

String Accuracy metric was also applied to provide comparison against studies that may not use the Generation Metric; however, the Generation Metric intuitively *repairs* some of the former's failings, namely double penalization for word movement. More on these and other metrics can be found in [Bangalore *et al.*, 2000].

4.3 Domain Independent Evaluation

Acorn was evaluated using the Monroe Corpus [Stent, 2000], a collection of 20 dialogues. Each dialogue is a conversation between two English speakers who were given a map of Monroe County, NY and a description of a task that needed to be solved. There were eight different disaster scenarios ranging from a bomb attack to a broken leg, and the participants were to act as emergency dispatchers. It is a significantly different domain from computer purchasing and was chosen because it offers a corpus that has been parsed by our parser and thus has readily available logical forms for input to Acorn. The length of utterances are shown in figure 6.

The four dialogues that had most recently been updated to our logical form definitions were chosen for the evaluation. The remaining sixteen are used by PathFinder as a bigram language model of the domain's dialogue. Two series of tests were run. The first includes the lexical items as input to Acorn and the second only includes the ontology concepts. Generation String Accuracy is used to judge the output of the system against the original utterances in the Monroe dialogues. While there have been other generation metrics that have been proposed, such as the Bleu Metric [Papineni *et al.*, 2001], the Generation String Accuracy metric still provides a measure of system improvement and a comparison against other systems. Bleu requires more than one correct output option to be of worthwhile (*'quantity leads to quality'*), so is not as applicable with only one target utterance.

4.4 Domain Specific Evaluation

In order to compare the domain independent evaluation with a domain specific evaluation, the same evaluation described in 4.2 was used on the computer purchasing corpus that includes the logical forms on which Acorn's grammar is based. As described in 4.1, the domain is an assistant that collaboratively purchases computers online for the user. There are 132 utterances of length three or more in this corpus. The n-gram models were automatically generated using a hand formed word grammar of sample sentences. Both Simple and Generation String Accuracy were used to compare the output of Acorn to the target utterances in the corpus.

Domain Independent: Monroe Rescue

Simple String Accuracy			
	Baseline	Random Path	Final
Lexical Items	0.28	0.55	0.67
Semantic Concepts	N/A	0.38	0.59

Generation String Accuracy			
	Baseline	Random Path	Final
Lexical Items	0.28	0.59	0.70
Semantic Concepts	N/A	0.40	0.62

Figure 7: The Simple and Generation String Accuracy results of Acorn in the Monroe domain. The two baseline metrics and the final Acorn scores are given.

4.5 Baselines

Two baselines were included in the evaluation as comparative measures. The first is named simply, *baseline*, and is a random ordering of the lexical inputs to Acorn. Instead of using a grammar to choose the ordering of the input lexical items, the *baseline* is a simple procedure which traverses the input terms, outputting each lexical item as it comes across them. When there are multiple modifiers on a term, the order of which to follow first is randomly chosen. This baseline is only run when lexical items are provided in the input.

The second baseline is called *Random Path* and serves as a baseline before the second phase of Acorn. A random path through the resulting word forest of the first phase of Acorn is extracted and compared against the target utterance. This allows us to evaluate the usefulness of the second stochastic phase. Both these baselines are included in the following results.

4.6 Results

Two different tests were performed. The first included lexical choice in the input utterances and the second included only the ontology concepts. The accuracy scores for the Monroe domain are shown in figure 7. A semantic input with all lexical items specified scored an average of 0.70 (or 70%) on 325 input utterances. A purely semantic input with just the ontology classes scored 0.62 (or 62%).

The results from Acorn in the Computer Purchasing Domain are shown in figure 8. Both the semantic and lexical evaluations were run, resulting in an average score of 0.85 (85%) and 0.69 (69%) respectively.

In order to judge usefulness for a real-time dialogue system, the runtime for both phases of Acorn was recorded for each utterance. We also ran HALogen for comparison. Since its grammar is significantly different from Acorn's, the output from HALogen is not relevant since little time was spent in conforming its grammar to our logical form; however, the runtimes are useful for comparison. The times for both Acorn and HALogen are shown in figure 9. With a purely semantic input, Acorn took 0.16 seconds to build a forest and 0.21 seconds to rank it for a total time of 0.37 seconds. HALogen took a total time of 19.29 seconds. HALogen runs quicker when lexical choice is performed ahead of time, finishing in

Domain Specific: Computer Purchasing

Simple String Accuracy			
	Baseline	Random Path	Final
Lexical Items	0.28	0.66	0.82
Semantic Concepts	N/A	0.47	0.67

Generation String Accuracy			
	Baseline	Random Path	Final
Lexical Items	0.28	0.69	0.85
Semantic Concepts	N/A	0.49	0.69

Figure 8: The Simple and Generation String Accuracy results of Acorn in the Monroe domain. The two baseline metrics and the final Acorn scores are given.

System Runtime			
	Build	Rank	Total Runtime
Acorn Lexical	0.06s	0.00s	0.06s
Acorn Semantic	0.16s	0.21s	0.37s
HALogen Lexical	2.26s	0.47s	2.73s
HALogen Semantic	11.51s	7.78s	19.29s

Figure 9: A comparison of runtimes (in seconds) between Acorn and HALogen. Both the lexical item and the semantic concept input are shown.

2.73 seconds. The reason is mainly due to its over-generation of noun plurals, verb person and number, and morphology.

Finally, the runtime improvement of using the grammar rule indexing algorithm was analyzed. All utterances of word length five or more with correct parses were chosen from the dialogues to create forests of sufficient size, resulting in 192 tests. Figure 10 shows the average forest building time with the indexing algorithm versus the old approach of checking each grammar rule individually. A 30% improvement was achieved.

5 Discussion

While it is difficult to quantify, the implementation of trickle-down features and Empty-Creation and Filling rules accommodate well the construction of a grammar that can capture head/foot features. The forest creation algorithm of HALogen and others is much too cumbersome to implement within, and representing lexical movement is impossible without it. The above result of 62% coverage in a new domain is comparable, and arguably better than those given in Langkilde [Langkilde-Geary, 2002]. This paper uses a semantic utterance input which is most similar to the *Min spec* test of Langkilde. The *Min spec* actually included both the lexical choice and the surface syntactic roles (such as logical-subject, instead of theme or agent), resulting in a Simple String Accuracy of 55.3%. Acorn's input is even more abstract by only including the semantic roles. Its lexical input, most similar to the *Min spec*, but still more abstract with thematic roles, received 70%. This comparison should only be taken at face value since dialogue utterances are shorter than the WSJ, but it provides assurance that a constrained grammar can produce good output even with a more abstract input. It must also be noted that the String Accuracy approaches do not take into account synonyms and paraphrases that are semantically equivalent.

	Build Forest Runtime
Normal Grammar	0.30s
Indexed Grammar	0.21s
% Improvement	30%

Figure 10: Runtimes of a sequential grammar rule search for matching rules versus the rule indexing approach described in this paper. The average runtime for 192 word forests is shown.

These results also evaluate the amount of effect the stochastic phase of this approach has on the overall results. Figure 7 shows that the average random path through the word forest (the result of the first grammar-based phase) was only 0.40 (40%). After PathFinder chooses the most probable path, the average is 0.62 (62%). We can conclude that the grammar is still over-generating possible realizations and that this approach does require the second stochastic phase to choose a realization based on previously seen corpora.

The difference between the results in the known domain (computer purchasing) and the new domain (monroe rescue) is 85% to 70% (69% to 62% without lexical items). While the difference is too great to claim domain independence on a semantic input, one of the main advantages of the over-generation grammar is that it requires less work to construct a new grammar when domains are switched. Here we see 70% achieved for zero invested time. A study that analyzes the time it takes a programmer to reach 85% has yet to be done.

The runtime improvement of our approach is more drastic than originally thought possible. An average runtime of 0.37 seconds is decidedly within the time constraints of an effective dialogue system. While the 30% improvement in grammar indexing is also significant, the larger gains appear to be results of finer morphology and person/number agreement between verbs and their subjects. Compared with 19.29 seconds of the previous implementation, it shows that a middle ground between over-generation and statistical determination is a viable solution.

Finally, more work is needed to produce better output. The majority of errors in this approach are modifier placement choices. Without a formal grammar, the final placement decisions are ultimately decided by an n-gram language model, resulting in short-sighted decisions. Even though 85% from a semantic input is a good result, modifiers tend to be the one area that falls behind. Several examples of this can be seen in Appendix B where some poor generations are shown.

6 Related Work

Stochastic work on the FERGUS system [Chen *et al.*, 2002] uses a TAG grammar to produce a word lattice of possible realizations. The lattice is traversed to find the most likely path. The work in [Chen *et al.*, 2002] generated sentences in

0.28 seconds for an Air-Travel Domain. This paper differs in that the input to FERGUS is a shallow syntactic tree, containing all lexemes and function words. In addition, surface syntax trees were mapped one-to-one with each template in the Air-Travel domain. There was little, if any, flexibility in the semantic input. This paper presents a result of 0.37 seconds that includes both the sentence planner, surface realizer, and a grammar that generates multiple realizations based on both syntax *and* semantics.

Work was done on the Oxygen system [Habash, 2000] to improve the speed of the two-phased Nitrogen generator, a predecessor to HALogen. The work pre-compiled a declarative grammar into a functional program, thus removing the need to match rules during forest creation. This paper differs in that similar performance was achieved without the need for pre-compiling nor a more complex grammar syntax. This paper also described lexical movement and trickle down features not supported in Oxygen.

Chambers [Chambers and Allen, 2004] used HALogen in a dialogue system and performed a human evaluation of the mixed syntax/semantic input. Their input converted their domain independent logical form into the HALogen input. This work differs in that we obviously did not use the HALogen system, but implemented a more efficient two-phased approach. The work by Chambers and Allen did not analyze runtime, perform sentence planning (not a full semantic input), nor provide results from the common String Accuracy metrics for comparison to other approaches.

7 Conclusion

Stochastic approaches to natural language processing are often criticized for being too slow, particularly in recent attempts in language generation. This paper describes Acorn, a system that generates dialogue utterances in an average of 0.37 seconds. The approach and its additional advances in word forest creation were described, such as a technique called trickle-down features that allow a grammar to pass head/foot features through a generation input, enabling language phenomena such as wh-movement to be represented. The grammar syntax and an evaluation of the coverage in an unknown domain were presented. The coverage is comparable and the runtime drastically out-performs previous approaches.

A Example Semantic and Lexical Input

Below is an example utterance from the Monroe corpus and its purely semantic and lexical input to Acorn. In this example, only the words *have*, *helicopter*, and *Strong Memorial* are absent in the semantic input. The resulting generation output from Acorn is also shown.

Original utterance:
'and i also have a helicopter at strong memorial'

Semantic Input to Acorn:

```
((utt :speechact sa_tell :mods v05 :saterm v88 :terms
  ((v88 :indicator speechact :class sa_tell :content v27
        :mods v05)
   (v05 :indicator f :class conjunct :lex and :of v88)
   (v27 :indicator f :class have :co-theme v63 :theme v09
        :mods v64 :mods v23 :tense present)
   (v09 :indicator pro :class person :context-rel i)
   (v23 :indicator f :class additive :lex also :of v27)
   (v63 :indicator a :class air-vehicle)
   (v64 :indicator f :class spatial-loc :lex at :of v27 :val v75)
   (v75 :indicator the :class facility
        :name-of (strong memorial)))))
```

Lexical Input to Acorn:

```
((utt :speechact sa_tell :mods v05 :saterm v88 :terms
  ((v88 :indicator speechact :class sa_tell :content v27
   :mods v05)
   (v05 :indicator f :class conjunct :lex and :of v88)
   (v27 :indicator f :class have :lex have :co-theme v63
        :theme v09 :mods v64 :mods v23 :tense present)
   (v09 :indicator pro :class person :lex i :context-rel i)
   (v23 :indicator f :class additive :lex also :of v27)
   (v63 :indicator a :class air-vehicle :lex helicopter)
   (v64 :indicator f :class spatial-loc :lex at :of v27 :val v75)
   (v75 :indicator the :class facility :lex strong-memorial
        :name-of (strong memorial)))))
```

Acorn Generation:
'and i have a helicopter also at strong memorial'

B Example Poor Output

Below are some target and generated utterances from Acorn, illustrating several common errors, and are not examples of success. The first utterance is the real target one, and the second is the Acorn generated utterance.

1. "i think i have a disability with maps"
   "i think i have disability with maps"
2. "they should have stayed in front of the tv"
   "in a front of the tv should stay they"
3. "and i also have a helicopter at strong memorial"
   "and i have a helicopter also at strong memorial"
4. "i can't see it on the map"
   "i can not on the map see it"
5. "probably all of them are hospitals"
   "probably hospitals are all them"
6. "are you talking to me"
   "are you talking me"
7. "and there are three people on a stretcher at the airport "
   "and three people on a stretcher are at the airport"
8. "then there's one stretcher patient at the mall"
   "then stretcher one patient is at the mall"
9. "so that guy should just walk to the hospital"
   "so that guy should walk to the hospital just"
10. "i think that's a very good plan"
    "i think that is very good plan"

C Example Good Output

Below are a list of target utterances that Acorn matched exactly, word for word. It is obviously not a complete list.

1. "i'm not doing this on purpose"
2. "we can bring it to strong memorial"
3. "it's on elmwood and mount hope "
4. "so the heart attack person can't go there"
5. "and bring them to saint mary's"
6. "do you have any suggestions?"
7. "we can put him in one ambulance"
8. "because we have only six wounded"
9. "i think that's a good idea"
10. "and the other one is at the airport"
11. "what can i say?"

References

[Bangalore *et al.*, 2000] Srinivas Bangalore, Owen Rambow, and Steve Whittaker. Evaluation metrics for generation. In *INLG*, Saarbrucken, Germany, August 2000.

[Callaway, 2003] Charles Callaway. Evaluating coverage for large symbolic nlg grammars. In *IJCAI*, Acapulco, Mexico, August 2003.

[Chambers and Allen, 2004] Nathanael Chambers and James Allen. Stochastic language generation in a dialogue system: Toward a domain independent generator. In *Proceedings of the 5th SIGdial Workshop on Discourse and Dialogue*, Boston, USA, May 2004.

[Chen *et al.*, 2002] John Chen, Srinivas Bangalore, Owen Rambow, and Marilyn A. Walker. Towards automatic generation of nautral language generation systems. In *COLING*, Taipei, Taiwan, 2002.

[Clarkson and Robinson, 1997] P.R. Clarkson and A.J. Robinson. Language model adaptation using mixtures and an exponentially decaying cache. In *Proceedings of ICASSP-97*, pages II:799–802, 1997.

[Dzikovska *et al.*, 2003] M. Dzikovska, M. Swift, and J. Allen. Constructing custom semantic representations from a generic lexicon. In *5th International Workshop on Computational Semantics*, 2003.

[Elhadad and Robin, 1996] M. Elhadad and J. Robin. An overview of surge: A reusable comprehensive syntactic realization component. Tech Report 96-03, Ben Gurion University, Beer Sheva, Israel, 1996.

[Habash, 2000] Nizar Habash. Oxygen: A language independent linearization engine. In *AMTA-2000*, Cuernavaca, Mexico, October 2000.

[Kipper *et al.*, 2000] Karin Kipper, Hoa Trang Dang, and Martha Palmer. Class-based construction of a verb lexicon. In *Proceedings of the 17th National Conference on Artificial Intelligence*, Austin, TX, 2000.

[Langkilde-Geary, 2002] Irene Langkilde-Geary. An empirical verification of coverage and correctness for a general-purpose sentence generator. In *INLG*, New York, 2002.

[Langkilde, 2000] Irene Langkilde. Forest-based statistical sentence generation. In *NAACL*, 2000.

[Papineni *et al.*, 2001] K. Papineni, S. Roukos, T. Ward, and W. Zhu. Bleu: a method for automatic evaluation of machine translation. Research Report RC22176, IBM, September 2001.

[Stent, 2000] A. Stent. The monroe corpus. Research Report 728, Computer Science Dept., University of Rochester, March 2000. 99-2.