Carolyn Penstein Rosé

This chapter discusses the application of genetic programming (GP) to the problem of robust language understanding in the context of a large scale multi-lingual speech-to-speech translation system. Efficiently and effectively processing sentences outside of the coverage of a system's linguistic knowledge sources is still an open problem in computational linguistics, a problem that must be faced if natural language interfaces will ever be practical. In this chapter, the GP based ROSE approach to robust language understanding is demonstrated to yield a significantly better time/quality trade-off than previous non-GP approaches. GP is used to search for the optimal way to assemble fragments of a meaning representation. The ROSE approach is the first application of a program induction technique to a problem of this type.

4.1 Introduction

This chapter introduces a new application of genetic programming (GP) to the problem of robust language interpretation and argues that GP is well suited to solve this problem effi ciently and effectively. Specifi cally, GP is used for the purpose of allowing a language understanding system to recover in cases where a parser fails. An empirical evaluation demonstrates that the GP approach yields a signifi cantly better time/quality trade-off than previous non-GP approaches. The ROSE system, RObustness with Structural Evolution, which is described in this chapter, serves as one example of how GP can be used to solve this problem, opening a new area of application for the GP community.

The theory of language understanding underlying the work described in this chapter has its roots in Conceptual Dependency Theory [Schank, 1975]. The basic tenets of this theory state that the purpose of language understanding is for the listener to construct a representation of the meaning of the input sentence. The representation of the meaning is independent of the actual words used to communicate that meaning. Meaning representations are built from a predefi ned set of primitives where meaning is encoded in the primitives themselves as well as in the relationships represented between those primitives in the structure resulting from the understanding process. However, unlike in Conceptual Dependency Theory, the work described here does not advocate a specifi c set of primitives. Instead, it can be used with whatever set of primitives are deemed useful for a particular domain.

A language understanding process constructs a meaning for a sentence by matching a set of grammar rules against the sentence that describe possible relationships between words and how these relationships are encoded in meaning representation structures. The impossibility of exhaustively enumerating the patterns of encoded relationships that are found in spontaneous spoken language make the problem of robust language interpretation particularly challenging. Sentences whose structure cannot be completely described by the set of rules in the system's parsing grammar are a common occurrence in naturally occurring language. Since parsing grammars in real systems generally only cover a subset of English or whatever language it is built for, its parser may fail even on sentences that are techni-

cally grammatical. These sentences are called extra-grammatical sentences since they are technically not ungrammatical but are nevertheless outside of the coverage of the system's grammar rules. Although the rules that are used to analyze sentences do not cover an entire extra-grammatical expression, often they are at least sufficient for covering its important sub-expressions. It is therefore possible to recover the majority of the meaning of the whole expression if the relationships between the meanings of the analyzed sub-expressions can be determined. Thus, the problem of robust interpretation can be thought of as the process of extracting the meaning of the sub-expressions within an extra-grammatical expression and then determining the relationships between those sub-expressions.

The ROSE approach to the problem of extra-grammaticality is to use a robust parser [Lavie, 1995] to extract the meaning of sub-expressions inside of an extra-grammatical expression. It then uses genetic programming to search the space of possible relationships between the meaning representation structures of those grammatical sub-expressions in order to build a representation of the whole sentence [Rosé, 1997]. Thus, genetic programming is used for the purpose of repair. The ROSE system was developed and evaluated in the context of the large-scale JANUS multi-lingual machine translation project [Lavie et al., 1996; Woszcyna et al., 1993; Woszcyna et al., 1994]. The JANUS project deals with the scheduling domain where two speakers attempt to schedule a meeting together over the phone. It is evaluated on spoken input that has been transcribed by a human. Thus, it contains all of the ungrammaticalities and disfluencies of spoken language without the additional complication of speech recognition errors.

It is ROSE's application of GP that makes it possible for it to operate efficiently. First, because the genetic search can do the work of assembling the meaning representation structures for the analyzed sub-expressions, the ROSE system avoids the overwhelming overhead of maximally flexible parsing approaches such as the Minimum Distance Parsing approach [Lehman, 1989; Hipp, 1992] that attempt to perform the whole task of sentence level interpretation at parse time. ROSE's two stage approach allows it to use a more restrictive, and thus more efficient, partial parser, and to use extra resources (i.e., the GP based repair stage) only in cases where repair is both necessary and possible. Furthermore, since the GP based combination algorithm constructs a ranked set of near-optimal or optimal hypotheses about the meaning of the sentences by searching the space of possible combinations of sub-expressions, it avoids the need for hand-coded repair rules that are featured in otherwise similar approaches [van Noord, 1996; Danieli and Gerbino, 1995].

4.2 Abstraction on the Problem of Parse Repair

The goal of ROSE's genetic search is to construct a meaning representation for an expression from the meaning representations of its sub-expressions. This approach assumes that meanings of sentences can largely be represented compositionally. In other words, the meaning of an expression can be represented in terms of the relationships between the

meanings of its immediate sub-expressions. For example, consider the sentence "The cat chases the dog." This sentence describes an action with two participants, one of whom is doing the action, and one to whom the action is being done. The meaning of "the cat" is a representation of the animal that the expression refers to. Likewise, "the dog" has as its meaning the animal that it refers to. "Chases" refers to the action of an actor running after the actee. It is not enough to know the meanings of these sub-expressions, however. In order to understand this sentence, it is necessary to recognize the relationships between these sub-expressions. In other words, it is necessary to realize that it is the cat that is doing the action of chasing and the dog to whom the action of chasing is being done. Although the assumption of compositionality [Gamut, 1991] has some notable exceptions [van den Berg et al., 1994], it has been demonstrated to be a useful simplifying assumption for the purpose of robust semantic interpretation in other recent work [Bod and Kaplan, 1998; Bod, 1998].

This section describes an abstraction of the problem in order to illustrate its scope and lay the foundation for the application of genetic programming to solve it.

4.2.1 The Basic Problem

Each primitive representing a unit of meaning can be thought of as an object with a hook at the top and some number of holes at the bottom. These primitives can then be linked by inserting the hook from the top of one corresponding object in one of the holes at the bottom of another. Each hole represents a relationship between two units of meaning, i.e., the one corresponding to the object with the hole and the one corresponding to the object with the hook. The act of linking these objects together by inserting hooks into holes is analogous to constructing the meaning of an expression from the meanings of its sub-expressions.

Figure 4.1 illustrates how different ways of linking the same three objects results in different meanings. Notice that each hole at the bottom of each primitive is labeled with a role. This role indicates the relationship that is denoted by inserting the hook from one object into the corresponding hole. For example, if the object corresponding to the CAT primitive is inserted into the *actor* hole in the object corresponding to the CHASE primitive, it denotes that the cat is the one who is doing the chasing. Likewise, if the CAT primitive is inserted into the *actee* hole in the CHASE primitive, it denotes that the cat is the one being chased. Furthermore, if the CHASE primitive is inserted into the *behavior* hole in the CAT primitive, it denotes that the chasing is something that the cat in question does. Thus, searching the space of meanings composed of the same primitives is analogous to searching the space of configurations of corresponding objects.

The number of possible configurations grows quickly as the number of objects increases. It is impossible to precisely compute the number of possible configurations for a general set of n objects since the actual number of holes in an object varies from corresponding primitive to corresponding primitive. But it can be computed if one makes the simplifying assumption that holes do not carry meaning so that the effective number of holes per object

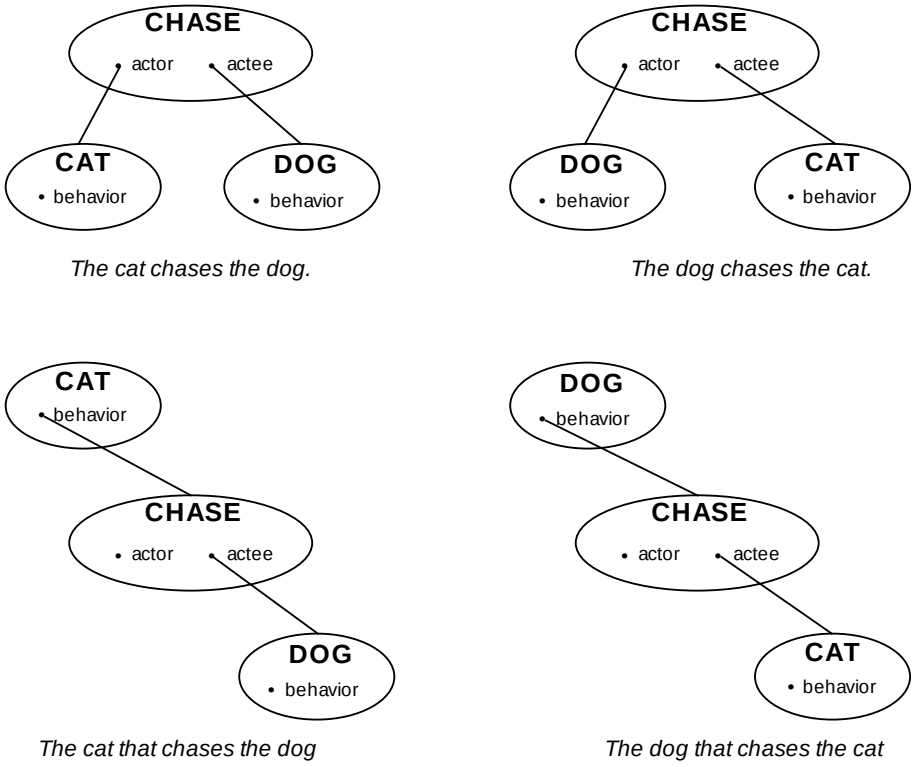


Figure 4.1
Here we see how a number of different meanings can be constructed from the same set of semantic primitives. The act of linking primitive units of meaning together is analogous to constructing the meaning of an expression from the meanings of its subexpressions.

Table 4.1
The number of possible configurations grows quickly as the number of objects increases. This table reports a lower bound on the number of possible configurations per number of objects.

Number of Objects	Search Space Size
1	1
2	4
3	18
4	116
5	1120
6	13782
7	212800
8	3801800

becomes one. Table 4.1 illustrates how the number of possible configurations grows quickly as the number of objects increases even with this simplifying assumption. The average number of objects produced by the parser for each example in the evaluation presented in Section 4.5 was 5.66. Since, as demonstrated in Figure 4.1, one hole is not equivalent to another hole in meaning, the actual search space size grows much faster than indicated in Table 4.1. Nevertheless, $S(n)$ provides a lower bound. It was computed using Equation 4.1.

$$S(n) = \sum_{i=1}^{n} \left(\begin{array}{c} n \\ i \end{array} \right) \left(\begin{array}{c} i \\ 1 \end{array} \right) \sum_{j=1}^{i-1} T(i-1, j) \tag{4.1}$$

The target configuration may be composed of any subset of the original set of n objects. Thus, the lower bound search space size with n objects is the sum of the number of ways to construct a configuration from every subset of the full set of n objects. The number of configurations with exactly i objects is the number of ways to select one of those objects as the root times the number of ways to divide the remaining objects into subsets and then construct a configuration from each subset. $T(i-1, j)$ is the number of ways to divide $i-1$ objects into j subsets and then construct a configuration from each subset. To compute the actual search space size, instead of the lower bound, T(i-1,j) would be multiplied by the number of ways to insert the resulting j objects into the number of holes in the selected root object.

4.2.2 Program Induction as a Solution

The problem of searching for the correct configuration of objects is easily cast as a program induction problem because of its recursive nature. Composite objects are assembled by inserting hooks from sub-objects into a root sub-object. These sub-objects may themselves be composed of other sub-objects, and so on. If one assumes the existence of a function called `COMBINE` that can insert the hook from the object that is its second argument into a hole in another object that is its first argument, one can write a program to construct any single configuration of objects. Thus, the target configuration can always be constructed by a program consisting only of instances of the subset of objects needed to construct the target configuration and instances of the `COMBINE` function.

Figure 4.2.2 shows how instances of `COMBINE` can be composed in order to construct a configuration. Notice how `OBJECT3` is first inserted into `OBJECT2`. Next, `OBJECT4` is inserted into `OBJECT2`. The composite object with `OBJECT2` as its root is then inserted into `OBJECT1`. Finally, `OBJECT5` is inserted into `OBJECT1`.

Although every target configuration can be constructed using instances of `COMBINE` and instances of the subset of objects needed for that configuration, not every program consisting only of instances of those objects and instances of `COMBINE` produces a legal configuration. In particular, more than one instance of the same object may not legally ap-

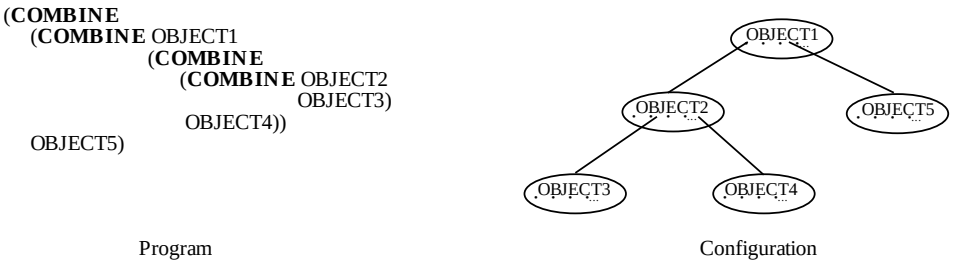


Figure 4.2
Here we see the result of evaluating a program consisting of the `COMBINE` function and primitive semantic objects. OBJECT3 is first inserted into OBJECT2. Next OBJECT4 is inserted into OBJECT2. Then the composite object with OBJECT2 as its root is inserted into OBJECT1. Finally, OBJECT5 is inserted into OBJECT1. The resulting configuration is displayed on the right.

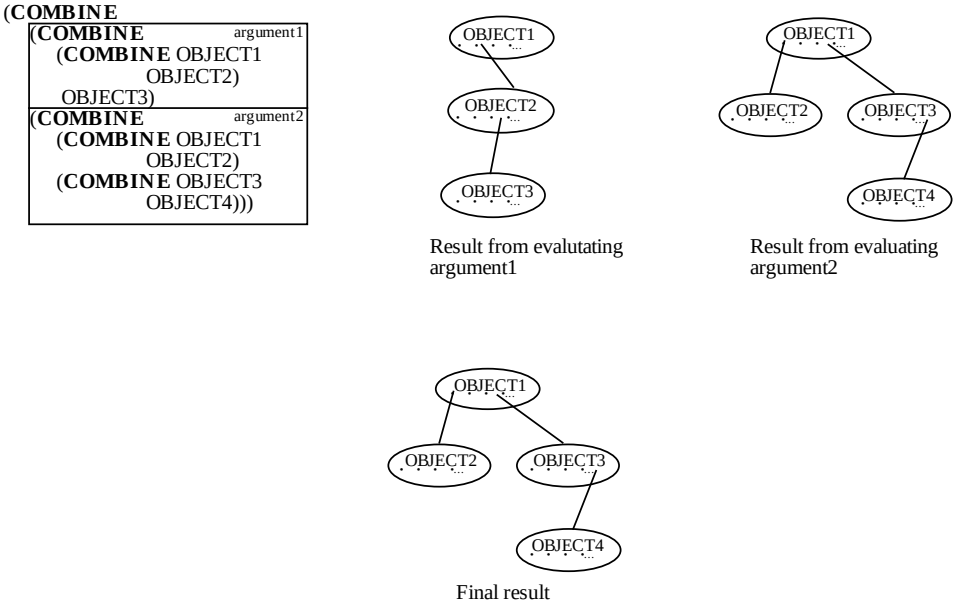


Figure 4.3
Here we see an example of how `COMBINE` avoids inserting multiple instances of the same object within a configuration. If the two objects passed in as arguments overlap with one another, the largest of the two is returned as the result.

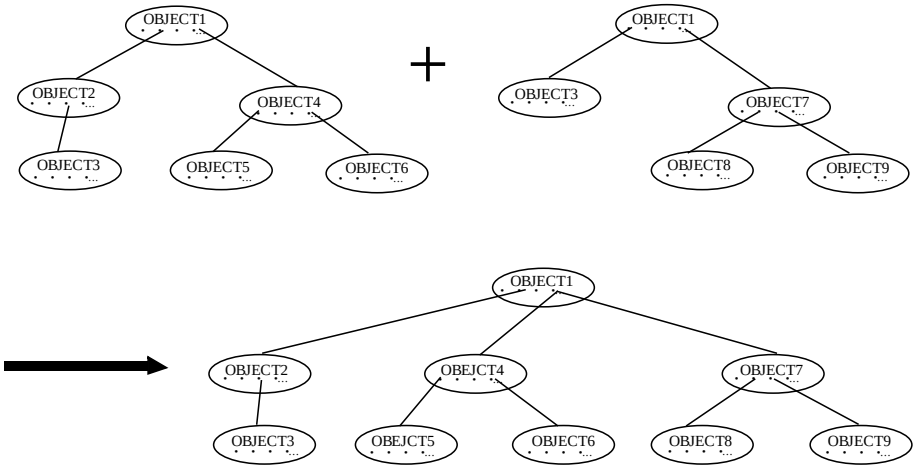


Figure 4.4
If the objects passed into `COMBINE` can not be combined by inserting the second one into the first one and they have the same root primitive object, they can be combined by merging. Here we see how merging produces a more complete resulting configuration.

pear within the same configuration. So with the most straightforward version of `COMBINE`, every program that contains more than one instance of the same object produces a result that is not a legal configuration. In order to ensure that every possible program produces a legal configuration, `COMBINE` must be altered in such a way as to prevent more than one instance of the same object from appearing within the same result. Thus, it inspects its two arguments to test whether they are composed of instances of any of the same objects. If they are, rather than inserting the second object into the first object, it returns the one composed of the largest number of primitive objects. See Figure 4.2.2 for an example of this. The instance of `COMBINE` at the top level, once its arguments are evaluated, is presented with two composite objects as arguments. Its first argument is composed of objects one through three. Its second argument is composed of objects one through four. Since there is overlap between the two arguments, they cannot be combined into a single legal configuration by inserting one into the other. Instead, the one composed of the largest number of primitive objects is returned. Thus, in this case, the one composed of objects one through four is returned as the result.

As discussed in Section 4.2, each hole on an object corresponds to a relationship that may hold between the associated object and another object. To precisely specify how to construct a configuration, a decision must be made within the `COMBINE` function of which hole to insert the second object into on the first object. Each hole has restrictions on it about what types of objects may be inserted into it. For example, it does not make sense

for an inanimate object to be the actor of an action. A rock cannot chase a mouse. These restrictions are specified in a meaning representation specification that lists the full set of possible primitives, which relationships are associated with each primitive, and what types of objects can be inserted into holes corresponding to those relationships. The `COMBINE` function makes reference to this specification so that it can ensure that every object it inserts into a hole in another object is appropriate for the specified relationship. In some cases, there will not be an appropriate hole in one object to insert the other object into. In that case, as in the case above, the largest of the two objects will be returned as the result.

In some cases, as in Figure 4.2.2, it is not possible to insert the second object into the first object, but the two objects can be combined by merging instead. Objects that have the same root can be merged into a single, possibly more complete, object. Merging takes place by first pruning the programs that generated the two arguments that were input to `COMBINE`. This pruning is done such that the only instances of `COMBINE` in the program that remain are those that when evaluated made insertions that contributed to the construction of those arguments. The remaining instances of `COMBINE` from the two programs are then combined into a single set. The largest subset of the cumulative set of instances of `COMBINE` is then extracted such that when they are composed into a single program, every `COMBINE` will be able to make an insertion when it is evaluated. Whenever it is possible to merge, it is preferable over simply returning the largest argument since it has the potential for returning a result that is more complete. Thus, whenever it is not possible to insert, but it is possible to merge, `COMBINE` will merge rather than return one or the other argument. In Figure 4.2.2 we see configurations resulting from two programs that can be merged as well as the configuration resulting from the merged program.

It would be possible to have a separate `MERGE` function in the function set, but it makes sense to keep both actions within the same function. The cases in which it is possible to insert and when it is possible to merge are almost always mutually exclusive in practice, and where they are not, inserting is almost always preferable to merging. Thus, it is possible to get by with a single function that can accommodate each of the three possible cases. This is preferable to having two separate functions since it avoids the case where one or the other function is assigned inappropriately. Note that this is also a reason why GP is more appropriate for this application than GA with configurations as its data structure [Michalewicz, 1994]. Although it would be possible to do almost the equivalent of ROSE's application of GP by using GA with a configuration data structure, typed crossover, and a merge operator, the algorithm would lose control over which specific cases the merge operator is applied to.

Figure 4.2.2 contains the definition of the `COMBINE` function as it is used in ROSE. It takes two possible composite objects as arguments. If it can insert the second object into a hole in the first object, it selects a hole and then does the insertion. If it is not possible to do an insertion, it tests whether it is possible to merge the two chunks. If this is not possible either, it returns the object composed of the largest number of primitive objects. With this `COMBINE` function it is true both that every target configuration can be built by a

```
COMBINE takes as input two objects
    If the two objects do no overlap with one another
    and there is a hole in the first object in which to
    insert the second object
        Select a hole
        Insert the second object into the selected hole in the first object
    Otherwise if both objects have the same primitive root object
        Merge the two objects
    Otherwise return the most comprehensive object
```

Figure 4.5
Working definition of `COMBINE`

program with only `COMBINE` in its function set and that every possible program consisting of instances of primitive objects and instances of `COMBINE` produces a legal configuration.

4.3 ROSE's Application of GP

ROSE operates in two stages: partial parsing and combination, each of which are described in this section. The focus of this chapter is the GP based combination stage. However, since the output of the partial parsing stage provides the input to the combination stage, it is necessary to describe both.

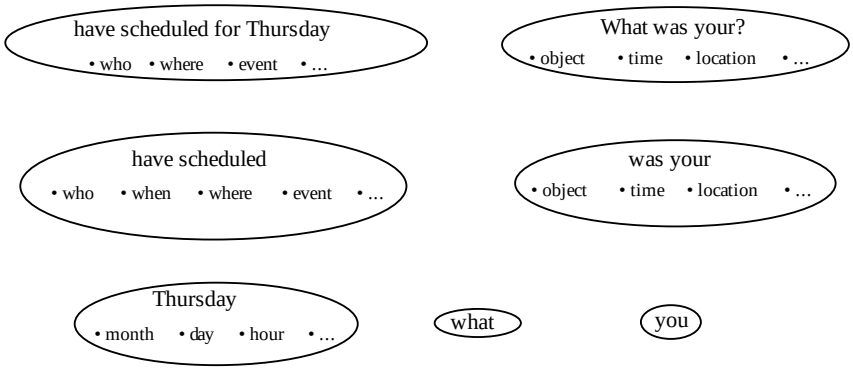
4.3.1 The Partial Parsing Stage

The robust parser used in ROSE is Lavie's GLR* parser [Lavie, 1995] modified to produce analyses for contiguous portions of a sentence. The partial parsing stage is described in depth in [Rosé, 1997]. In this chapter it is described only briefly.

The goal of the partial parsing stage is to obtain an analysis for islands of the speaker's sentence if it is not possible to obtain an analysis for the whole sentence. In Figure 4.6 we see an abstract representation of the parser output for "What did you say about what was your schedule for Thursday?" The parser produced seven partial analyses covering different parts of the sentence. The details of the underlying representation are not shown, but note that it is such that partial analyses can be assembled into configurations in the same way that primitive objects were in Section 4.2. The one difference between the objects constructed by the parser and those described in Section 4.2 is that some of these objects could be said to overlap with one another because they cover overlapping parts of the same sentence. For example, the object corresponding to "have scheduled for Thursday" and the object corresponding to "have scheduled" both cover the word "schedule" from the original sentence. Therefore, these two objects can not both appear together in a legal configuration. Likewise, the "you" object overlaps with the "what was your" and "was

Sentence: What did you say about what was your schedule for Thursday?

Partial Parse:



Ideal Configuration:

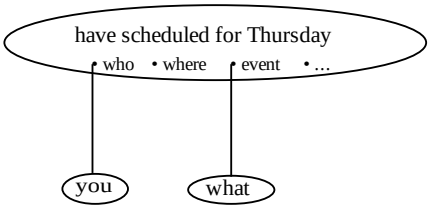


Figure 4.6
Here we see a representation of the objects produced by the partial parser for the sentence "What did you say about what was your schedule for Thursday." The ideal configuration is displayed at the bottom.

Table 4.2
This tableau describes how GP is set up for the Combination problem.

Objective:	Find the program that generates the target configuration
Terminal Set:	Objects produced by the partial parser
Function Set:	`COMBINE`
Raw Fitness:	Evaluated by an indirect measure. See section 4.3.2.2.
Standardized Fitness:	Same as raw fitness
Hits:	None. Since the target configuration is not known, it is impossible to have absolute certainty that you have a hit
Wrapper:	None.
Parameters:	Population Size = 32, Generations = 4
Success Predicate:	None.

your" objects. Thus, the test for overlap in `COMBINE` must ensure not only that the same parser object appear at most once in the resulting configuration, but also that the same part of the sentence only be represented once in a resulting configuration.

Because in the parser's output "schedule" is misanalyzed as an action rather than as an object, there is no way to assemble a configuration to represent the exact meaning of this sentence with the available objects. The configuration with the closest meaning to the original sentence that can be constructed from the available objects represents the meaning "What do you have scheduled for Thursday?"

Notice that the ideal configuration does not include both of the objects covering the largest portions of the sentence. Instead, it includes one of them along with the two smallest objects. Thus, the task of selecting the ideal subset of objects produced by the parser to include in the output configuration is not a trivial task. It cannot be predicted with a straightforward metric such as picking the subset that includes the largest non-overlapping objects.

4.3.2 The Combination Stage

The combination stage takes as input the objects returned by the parser. The goal is to evolve a program that builds the ideal configuration out of these objects.

4.3.2.1 Applying Genetic Programming

The GP based combination algorithm used in ROSE is Koza's lisp kernel [Koza, 1992]. The goal of this combination algorithm is to evolve a program that when executed constructs the ideal configuration from the objects returned by the parser. Notice that in Figure 4.6, the ideal configuration is composed of a subset of the full set of objects returned by the parser. Thus, two things are accomplished in parallel by this application of GP. The correct subset of objects is selected, and at the same time, the correct way to assemble them is determined.

The terminal set for the combination problem is the set of objects returned by the parser. The function set contains only the `COMBINE` function. The initial random population of

programs is generated with the ramped half and half method with a maximum depth of 5. Rather than selecting terminal symbols with a uniform distribution, each object is selected with a frequency proportionate to the percentage of the sentence it covers. These large objects are not guaranteed to be the more appropriate objects in the set for building the correct configuration, but in practice they tend to be better than the other smaller objects. By selecting objects as terminal symbols in this way, the most comprehensive objects appear more frequently in the population. Thus, it is more likely for a larger object to appear in a configuration produced by an individual, but it is not impossible for a smaller object to appear.

Once a population of individuals is generated, the fitness of each individual program is computed. The fitness of each individual is calculated using the function described in Section 4.3.2.2 that combines multiple goodness indicators. Each instance of `COMBINE` has a static local variable that keeps track of which hole in the parent object was selected for inserting the child object. In this way, it can be ensured that each time the program is evaluated, the same result will be produced. The fitness function is trained in such a way that the fitness score assigned to each individual program is indicative of the goodness of the resulting configuration.

Once the fitness for each individual is computed, subsequent generations are computed using fitness proportionate reproduction. The fitness proportionate reproduction fraction was set to 10%, mutation 10%, and crossover 80%, with 75% of crossovers constrained to occur at function points. Since every program composed of instances of the objects produced by the parser and instances of `COMBINE` is guaranteed to produce a legal configuration, straightforward versions of crossover and mutation can be used. Crossover in ROSE's combination algorithm simply selects a sub-program from each parent program and swaps them. The maximum depth for individuals after crossover is set to 15. Generally, a depth of 15 is far larger than necessary for including the steps necessary for constructing the target configuration. However, since it is common for portions of the evolved program to have no effect on the resulting configuration as discussed in Section 4.2.2, it is necessary to allow larger programs to be evolved. With a larger population and larger number of generations, programs with a smaller depth and equivalent performance could be evolved. However, in order to force the combination algorithm to perform as efficiently as possible, it is allowed here to produce sloppier programs.

When the resulting programs are evaluated, wherever configurations resulting from the execution of sub-programs are no longer appropriate for inserting into the hole where they were previously inserted, a new hole is selected. Similarly, mutation takes place by constructing a random subprogram in the same way that the initial population was generated and inserting it in place of a randomly selected sub-program in the parent program. The maximum depth for randomly generated sub-programs is 4.

4.3.2.2 Fitness Evaluation for the Combination Problem

The programs created by the GP based combination algorithm can be thought of as repair hypotheses. Once a population of hypotheses is generated, each individual in the population is evaluated for its fitness. Evaluating the fitness of repair hypotheses is the most difficult part of applying genetic programming to repair. The ROSE approach to fitness evaluation of repair hypotheses is one of the aspects that makes this application of genetic programming unique. An ideal fitness function for repair would rank hypotheses that generate structures closer to the target structure better than those that are more different. In comparing relative goodness of alternative repair hypotheses, the repair module must consider not only which subset of objects returned by the parser to include in the final result, but also how to put them together. However, it does not know what the ideal configuration is. Since the repair module does not have access to this information, it must rely upon indirect evidence for determining which hypotheses are better than others. A fitness function is trained to use this indirect evidence for the purpose of ranking repair hypotheses in a useful manner.

Four pieces of indirect evidence about the relative goodness of repair hypotheses can be computed for each repair hypothesis: the number of primitive objects in the resulting structure (NUM_CONCEPTS), the number of insertions involved (NUM_STEPS), a statistical score reflecting the goodness of insertions (STAT_SCORE), and the percentage of the sentence that is covered by the resulting configuration (PERCENT_COV). These parameters are generally useful in ranking hypotheses. Both NUM_CONCEPTS and PERCENT_COV provide an estimate of the completeness of solutions. NUM_STEPS provides an estimate of the simplicity of the program. And as much as the statistical goodness score gives a reliable indication of goodness of fit between objects and relationships and quality of parser produced objects, repair hypotheses with better than average statistical score are more likely to be better hypotheses. The statistical score of a hypothesis is calculated by averaging the statistical scores for each repair action, where the statistical score of each included object is the statistical score assigned by the parser to the analysis of that object, and the statistical score of inserting an object into a hole in another object is defined by the information gain between the hole and the type of the inserted object. The information gain between the hole and the type of the inserted object is a measure of how strongly the hole predicts which type of object will fill it.

Intuitively, one would prefer more complete hypotheses over less complete ones. And following the principle of Occam's razor, other things being equal, one would prefer simpler solutions over more complex ones. Since simpler solutions may be less complete, and more complete hypotheses might be more complex, the trained fitness function must learn how to balance these two competing qualities. A fitness function trained with GP was used to combine these four pieces of information in order to rank alternative repair hypotheses.

The fitness function was trained over a corpus of 48 randomly selected sentences from a separate corpus from that used in the evaluation discussed in Section 4.5. Each of these 48 sentences were such that repair was required for constructing a reasonable interpretation. In this corpus each sentence was coupled with its corresponding ideal meaning representa-

tion structure. To generate training data for training the repair fitness function, the repair module was run using an ideal fitness function that evaluated the goodness of hypotheses by comparing the meaning representation structure produced by the hypothesis with the ideal structure. It assigned a fitness score to the hypothesis equal to the number of primitive objects in the largest substructure shared by the produced configuration and the ideal configuration. The four scores that serve as input to the trained fitness function were extracted from each of the hypotheses constructed in each generation after the programs were ranked by the ideal fitness function. The resulting ranked lists of sets of scores were used to train a fitness function that can order the sets of scores the same way that the ideal fitness function ranked the associated hypotheses. Thus, although there were only 48 sentences in the training set, there were four times that many training cases. While this is still a relatively small number of training cases, no over fitting effects were observed.

The genetic programming algorithm used to train the repair fitness function took as terminals the four scores plus a random real number function. The function set included addition, subtraction, multiplication, and division. The fitness of alternative proposed fitness functions generated during the genetic search was computed by first ordering the set of scores in each training example using the hypothesized function. The length of the greatest common subsequence between the ideal ordering and the generated ordering was then computed. The greatest common subsequence was computed using Dijkstra's well known algorithm [Cormen et al., 1989]. The fitness of each hypothesized repair fitness function was the average greatest common subsequence score over the entire training corpus of 48 sentences. A single run with a population size of 1000 was used, and the training process continued for approximately 2000 generations, until subsequent generations didn't produce a function with performance better than the previous generation.

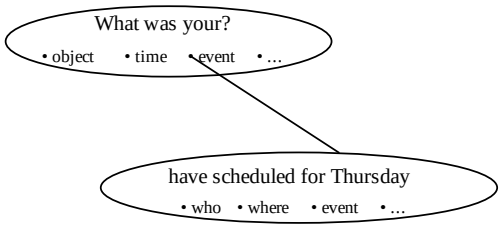
4.4 Why GP?

Repairing extra-grammatical sentences is an unusual application for GP in that it requires a relatively small population size and number of generations. In practice, a population size of 32 and 4 generations has been determined to be adequate for repairing extra-grammatical sentences in the scheduling domain.

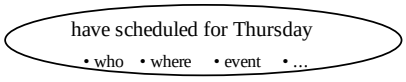
Although it may appear on the surface that a simpler control structure such as a priority queue would suffice, such an approach would require the system to decide which set of repairs to start with and then which alternative hypotheses logically follow in an ordered manner. However, since the goodness of hypotheses is determined by factors that make competing predictions (completeness versus simplicity), no such prioritization can effectively be determined a priori. Likewise, what distinguishes hypotheses from one another is both which subset of objects is included in the hypotheses and how the objects are composed. It is not clear what principle should be used in generating successive hypotheses to test - whether it is better to change the subset of objects included in the hypothesis or

Sentence: What did you say about what was your schedule for Thursday?

Locally Optimal:



Less Complete but More Correct:



Ideal Configuration:

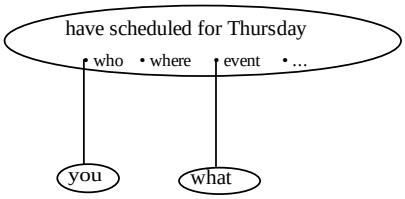


Figure 4.7
Here we see an example of a locally optimal solution for the example in Figure 4.6. It contains the two largest objects returned by the parser. No single operation on this configuration can improve the quality of the solution since removing the suboptimal object results in a solution that is more correct, but less complete.

simply to change the way they are assembled. Finally, it is not clear what stopping criteria one would use for an application of the priority queue method, or how one would avoid locally optimal solutions.

The problem of locally optimal solutions is a big one in this application. A locally optimal solution from the example in Figure 4.6 is found in Figure 4.4. The locally optimal solution is composed of the two largest objects produced by the parser. It can not be made to cover any more of the sentence by inserting any additional objects since every other object in the set returned by the parser overlaps with one of the objects already included in the confi guration. However, the object covering "what" and "you" included in that solution corresponds to a suboptimal analysis of that portion of the sentence. Before the best confi guration can be built, the object corresponding to "What was your?" must be removed, resulting in a confi guration that covers less of the sentence than the suboptimal solution, but not containing the suboptimal analysis. The optimal confi guration can then be constructed by inserting the objects corresponding to "what" and "you" separately into the object corresponding to "have scheduled for Thursday". Thus, the result must temporarily be made less complete in order to be made both correct and complete.

Thus, GP's opportunistic search method is very sensible in this case, although the appropriateness of GP-like approaches such as SIHC [O'Reilly and Oppacher, 1995] have yet to be tested as alternatives for this application. GP fi rst samples its search space widely and shallowly and then narrows in on the regions surrounding the most promising looking points. It has the ability to search a large space effi ciently. As described in Section 4.2.1, the number of alternative confi gurations grows quickly as the number of objects produced by the parser increases. The number of alternative individuals evaluated by the genetic programming algorithm can be fi xed ahead of time by setting the population size and the maximum number of generations. By limiting the population size to 32 and the number of generations to 4, the repair module is constrained to search only 128 alternatives. Thus, if the number of objects produced by the parser is any more than four, the GP approach searches only 11% or less of the lower bound estimate on the number of alternative hypotheses. If the number of objects produced by the parser is exactly four, the number of alternatives explored is similar. The average number of objects produced by the parser in the evaluation presented in Section 4.5 was 5.66. In 55.7% of the cases, the parser produced more than four objects. Thus, the GP approach yields a signifi cant savings in time in more than half of the cases where repair is used. The great success with examining such a small portion of the search space is perhaps due to the statistical bias within the `COMBINE` function.

4.5 Evaluation

ROSE's performance was evaluated in terms of effi ciency and effectiveness in comparison with the two main competing approaches to robust interpretation, namely the maximally

Table 4.3
This table reports the percentage of sentences for the alternative interpretation strategies that either produced a nil result or were assigned a grade of Bad, Partial, Okay, or Perfect by an impartial human judge.

	NIL	Bad	Partial	Okay	Perfect	Total Acceptable
MDP 1	21.4%	3.4%	3.4%	18.4%	53.4%	71.8%
MDP 3	16.2%	4.2%	5.0%	19.6%	55.0%	74.6%
MDP 5	8.4%	8.2%	6.0%	21.0%	56.4%	77.4%
GLR with Restarts	9.2%	6.4%	12.8%	19.4%	52.2%	71.6%
GLR with Restarts + Repair	0.4%	9.6%	11.8%	23.4%	54.8%	78.2%
GLR*	2.2%	8.8%	11.6%	21.4%	56.0%	77.4%
GLR* + Repair	0.6%	8.8%	10.6%	23.6%	56.4%	80.0%

flexible parsing approach [Lehman, 1989; Hipp, 1992] and the restrictive partial parsing approach [Lavie, 1995; Abney, 1996; Ehrlich and Hanrieder, 1996; Srinivas et al., 1996; Federici et al., 1996; Jensen and Heidorn, 1993; Hayes and Mouradain, 1981; Kwasny and Sondheimer, 1981; Lang, 1989]. The purpose of this evaluation was to demonstrate the appropriateness of employing GP for the purpose of repairing extra-grammatical sentences. ROSE's two stage approach, employing a GP based combination algorithm, is demonstrated here to achieve a better effectiveness/efficiency trade-off than either of the above mentioned single stage approaches.

In order to compare the alternative approaches keeping as many factors constant as possible, the same parser was used in each case, parameterized to control the flexibility of the algorithm. The GLR* parser [Lavie, 1995; Lavie and Tomita, 1993] is used in five different parameter settings described in depth in [Rosé, 1997], Chapter 10, Section 3. The most restrictive version, `GLR w/restarts`, constructs analyses for contiguous portions of the input text. The less restrictive `GLR*` setting allows the parser to skip over words in order to construct analyses for non-contiguous portions of the input. The more flexible `MDP 1`, `MDP 3`, and `MDP 5` settings allow the parser to either insert or delete up to 1, 3, or 5 words respectively in order to search for an analysis for each extra-grammatical sentence. `GLR w/restarts` and `GLR*` serve as representatives of the restrictive partial parsing approach. `MDP 1`, `MDP 3`, and `MDP 5` serve as representatives of the maximally flexible parser approach. ROSE is evaluated using each of the restrictive partial parsers, referred to as `GLR w/restarts + repair` and `GLR* + repair` respectively. Thus, seven different specific approaches are evaluated in the experiments described here. Each approach was evaluated using the same semantic grammar with approximately 1000 rules, with the same lexicon of approximately 3000 lexical items, on the same previously unseen test corpus of 500 sentences.

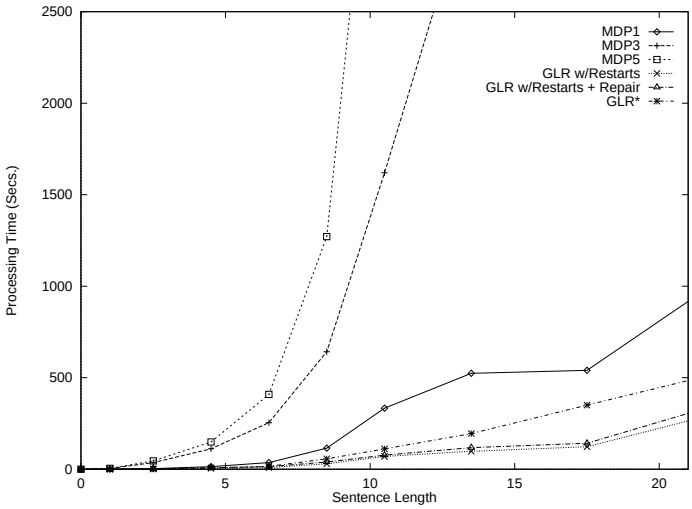


Figure 4.8
This diagram displays mean run times for six alternative interpretation strategies as it varies for different sentence lengths. Notice that the three **MDP** approaches are far slower than the other approaches with or without the GP based repair stage.

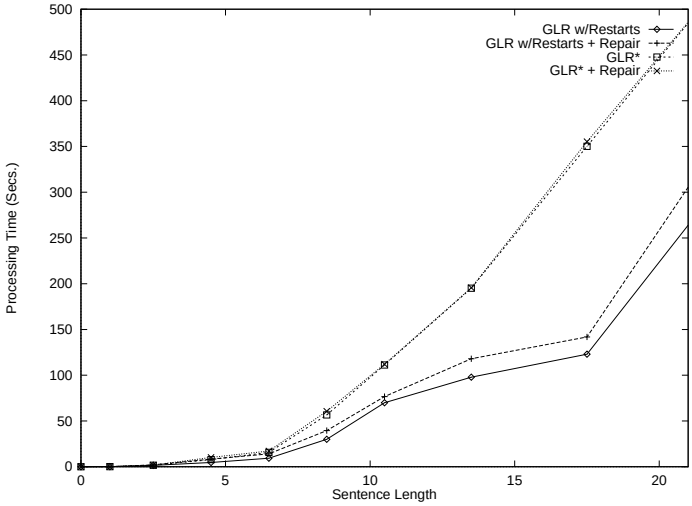


Figure 4.9
Here we see mean processing time for two alternative partial parsers, namely **GLR w/restarts** and **GLR***, with and without repair. Notice that the GP based repair stage does not dramatically increase the practical run time of the parsers.

For each sentence, the meaning representation structure returned by the alternative interpretation processes was passed to a generation component that generates a sentence in English. This text was then graded by a human judge as Bad, Partial, Okay, or Perfect in terms of interpretation quality. The human judge was a staff person on the JANUS project experienced in grading interpretation quality but not having been involved in the development of any portion of the system being evaluated. The judge was not aware of which approach produced each result. A grade of "Partial" indicates that the result communicated part of the content of the original sentence while not containing any incorrect information. "Okay" indicates that the generated sentence communicated all of the relevant information in the original sentence but not in a perfectly fluent way. "Perfect" indicates both that the result communicated the relevant information and that it did so in a smooth, high quality manner.

Run times for all seven approaches are found in figures 4.8 and 4.9. Notice that all of the MDP approaches are significantly slower than the other approaches. Also notice that the partial parsing approaches with repair are not significantly slower than their corresponding partial parsing approaches without repair. In particular, in Figure 4.9 we see that run times for `GLR*` and `GLR* + repair` are barely distinguishable. This small difference in run times between the versions with repair and without are accounted for by the fact that the repair stage is not time consuming (taking 30 seconds on average) and is only used when both necessary and possible.

By comparing the run times for the alternative conditions in Figures 4.8 and 4.9 with the interpretation quality scores found in Table 4.3, it becomes evident that the two-stage ROSE approach achieves a better effectiveness/efficiency trade-off than either the maximally flexible parsing approach or the restrictive partial parsing approach. Predictably, `MDP 5` shows an improvement over `MDP 1`, with an associated significant cost in run time. Also, not surprisingly, the very restrictive `GLR w/restarts`, while it is fastest, has a correspondingly lower associated interpretation quality. However, `GLR w/restarts + repair` outperforms all of the single stage approaches, second only to `GLR* + repair`, which is slightly slower although still faster than `MDP 1`. Though these results demonstrate certain trends in the performance of the alternative approaches, the differences in interpretation quality overall are very small. Nevertheless, the very significant difference in runtime performance demonstrates that the two-stage ROSE approach is a clear winner.

The contribution made by ROSE's GP based Combination stage is more evident when considering the maximum potential repair that is possible using chunks produced by the parser. For example, with `GLR w/restarts + repair`, the percentage of sentences in the test corpus where it was true both that repair was necessary and that the parser produced sufficient chunks for actually constructing an acceptable hypothesis was only 8.6%. Therefore, the 6.6% of additional acceptable hypotheses produced by ROSE constitutes 76.7% of the maximum potential improvement. Though this still leaves room for further work, it demonstrates that a significant percentage of the maximum improvement that is possible to achieve with repair was indeed realized by ROSE's application of GP.

4.6 Challenges

This chapter describes ROSE, RObustness with Structural Evolution, an application of GP to the problem of robust interpretation of extra-grammatical sentences. The genetic programming algorithm searches for the near-optimal or optimal ways to assemble analyses for fragments of an extra-grammatical sentence into a single meaning representation structure. ROSE is demonstrated here to achieve a better effectiveness/efficiency trade-off than either the restrictive partial parsing approach or the maximally flexible parsing approach.

These promising results point the way towards a number of avenues for future exploration of GP applied to the problem of robust interpretation. For example, since each individual generated by ROSE is a program that constructs a single meaning representation structure, it works best only in cases where an utterance contains a single sentence. For processing multi-sentence utterances, the GP algorithm would have to be able to construct individuals composed of multiple programs each assembling a subset of the chunks produced by the parser. Not only would the genetic search be responsible for determining which chunks to include in the final analysis, but it would also have to decide how to partition this set into subsets each representing a single sentence, and then how to compose the chunks for each sentence into a single meaning representation structure.

Secondly, since ROSE works by assembling the chunks returned by the parser, the resulting meaning representation structure can only represent the portions of the sentence that the parser is able to construct a partial analysis for. One can imagine that something similar to the Random Constant used in many numerical GP applications could be used to make guesses about the missing portions (perhaps statistically). The genetic search would then be responsible for determining when it was the case that an essential part of the analysis was missing and what was likely to be missing.

Finally, just as ROSE is used for determining how sub-sentence units of meaning relate to one another, one can imagine GP being used to determine how the meanings for individual sentences fit together into a larger discourse structure. [Mason and Rosé, 1998] reports on some preliminary work in this area, specifically in evolving constraint functions for operators to function within a plan-based discourse processor [Rosé et al., 1995].

Acknowledgements

The work described in this chapter is part of the author's dissertation research. Lori Levin, Barbara Di Eugenio, Jaime Carbonell, Alon Lavie, Johanna Moore, and Sandra Carberry served on the committee that advised this research and all deserve recognition for the valuable contribution they made in countless direct and indirect ways. In particular, Lori Levin, who served as the author's thesis advisor, tirelessly read and commented on numerous drafts of this work and kept the author focused and on track throughout her journey towards the completion of her PhD. Alon Lavie more than anyone else acted as a partner in

this research, collaborating on its evaluation, but even more importantly providing critical feedback on many of its core underlying ideas. Most of all, Eric Rosé deserves the author's deepest gratitude for originally introducing her to genetic programming, but most of all for providing constant support, encouragement, and companionship without which this work would have been impossible. This chapter is dedicated to the author's precious baby daughter Rachel, the apple of her mother's eye.

This continuing research is partly supported by the Office of Naval Research, Cognitive and Neural Sciences Division (Grant #N00014-93-I-0812).

Bibliography

Abney, S. (1996), "Partial parsing via finite-state cascades," in *Proceedings of the Eighth European Summer School In Logic, Language and Information, Prague, Czech Republic*.

Bod, R. (1998), "Spoken dialogue interpretation with the dop model," in *Proceedings of COLING-ACL '98*.

Bod, R. and Kaplan, R. (1998), "A probabilistic corpus-driven model for lexical-functional analysis," in *Proceedings of COLING-ACL '98*.

Cormen, T. H., Leiserson, C. E., and Rivest, R. L. (1989), *Introduction to Algorithms (Ch 25)*, The MIT Press.

Danieli, M. and Gerbino, E. (1995), "Metrics for evaluating dialogue strategies in a spoken language system," in *Working Notes of the AAAI Spring Symposium on Empirical Methods in Discourse Interpretation and Generation*.

Ehrlich, U. and Hanrieder, G. (1996), "Robust speech parsing," in *Proceedings of the Eighth European Summer School In Logic, Language and Information, Prague, Czech Republic*.

Federici, S., Montemagni, S., and Pirrelli, V. (1996), "Shallow parsing and text chunking: a view on underspecification in syntax," in *Proceedings of the Eighth European Summer School In Logic, Language and Information, Prague, Czech Republic*.

Gamut, L. T. F. (1991), *Logic, Language, and Meaning Volume 2: Intensional Logic and Logical Grammar*, University of Chicago Press.

Hayes, D. and Mouradain, G. V. (1981), "Flexible parsing," *Computational Linguistics*, 7(4).

Hipp, D. R. (1992), *Design and Development of Spoken Natural-Language Dialog Parsing Systems*, PhD thesis, Dept. of Computer Science, Duke University.

Jensen, K. and Heidorn, G. E. (1993), *Parse Fitting and Prose Fixing (Ch 5), In Natural Language Processing: the PLNLP Approach*, Kluwer Academic Publishers.

Koza, J. (1992), *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press.

Kwasny, S. and Sondheimer, N. K. (1981), "Relaxation techniques for parsing grammatically ill-formed input in natural language understanding systems," *American Journal of Computational Linguistics*, 7(2).

Lang, B. (1989), "Parsing incomplete sentences," in *Proceedings of the 12th International Conference on Computational Linguistics (COLING 89)*.

Lavie, A. (1995), *A Grammar Based Robust Parser For Spontaneous Speech*, PhD thesis, School of Computer Science, Carnegie Mellon University.

Lavie, A., Gates, D., Gavalda, M., Mayfield, L., and Levin, A. W. L. (1996), "Multi-lingual translation of spontaneously spoken language in a limited domain," in *Proceedings of COLING 96, Kopenhagen*.

Lavie, A. and Tomita, M. (1993), "GLR* - an efficient noise-skipping parsing algorithm for context free grammars.," in *Proceedings of the Third International Workshop on Parsing Technologies*.

Lehman, J. F. (1989), *Adaptive Parsing: Self-Extending Natural Language Interfaces*, PhD thesis, School of Computer Science, Carnegie Mellon University.

Mason, M. and Ros´e, C. P. (1998), "Automatically learning constraints for plan-based discourse processors," in *Working Notes for the AAAI Spring Symposium on Machine Learning and Discourse Processing*.

Michalewicz, Z. (1994), *Genetic Algorithms + Data Structures = Evolution Programs*, New York: Springer-Verlag.

O'Reilly, U. and Oppacher, F. (1995), "A comparative analysis of genetic programming," in *Advances in Genetic Programming II*, P. J. Angeline and J. K. Kinnear (Eds.), The MIT Press.

Ros´e, C. P. (1997), *Robust Interactive Dialogue Interpretation*, PhD thesis, School of Computer Science, Carnegie Mellon University.

Ros´e, C. P., Eugenio, B. D., Levin, L. S., and Ess-Dykema, C. V. (1995), "Discourse processing of dialogues with multiple threads," in *Proceedings of the ACL*.

Schank, R. (1975), *Conceptual information processing*, North Holland, Amsterdam.

Srinivas, B., Doran, C., Hockey, B., and Joshi, A. (1996), "An approach to robust partial parsing and evaluation metrics," in *Proceedings of the Eighth European Summer School In Logic, Language and Information, Prague, Czech Republic*.

van den Berg, M., Bod, R., and Scha, R. (1994), "A corpus based approach to semantic interpretation," in *Proceedings of the Nineth Amsterdam Colloquium*.

van Noord, G. (1996), "Robust parsing with the head-corner parser," in *Proceedings of the Eighth European Summer School In Logic, Language and Information, Prague, Czech Republic*.

Woszcyna, M., Aoki-Waibel, N., Buo, F. D., Coccaro, N., Horiguchi, K., Kemp, T., Lavie, A., McNair, A., Polzin, T., Rogina, I., Ros´e, C. P., Schultz, T., Suhm, B., Tomita, M., and Waibel, A. (1994), "JANUS 93: Towards spontaneous speech translation," in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*.

Woszcyna, M., Coccaro, N., Eisele, A., Lavie, A., McNair, A., Polzin, T., Rogina, I., Ros´e, C. P., Sloboda, T., Tomita, M., Tsutsumi, J., Waibel, N., Waibel, A., and Ward, W. (1993), "Recent advances in JANUS: a speech translation system," in *Proceedings of the ARPA Human Languages Technology Workshop*.