# HUNTER-GATHERER: Three Search Techniques Integrated for Natural Language Semantics

**Stephen Beale, Sergei Nirenburg** and **Kavi Mahesh**
Computing Research Laboratory
Box 30001
New Mexico State University
Las Cruces, New Mexico 88003
sb,sergei,mahesh@crl.nmsu.edu

### Abstract

This work[1] integrates three related AI search techniques – constraint satisfaction, branch-and-bound and solution synthesis – and applies the result to semantic processing in natural language (NL). We summarize the approach as "Hunter-Gatherer:"

- branch-and-bound and constraint satisfaction allow us to "hunt down" non-optimal and impossible solutions and prune them from the search space.
- solution synthesis methods then "gather" all optimal solutions avoiding exponential complexity.

Each of the three techniques is briefly described, as well as their extensions and combinations used in our system. We focus on the combination of solution synthesis and branch-and-bound methods which has enabled near-linear-time processing in our applications. Finally, we illustrate how the use of our technique in a large-scale MT project allowed a drastic reduction in search space.

## Introduction

The number of possible semantic analyses in an **average**-sized sentence in the Spanish corpus used in the Mikrokosmos MT project is fifty six million, six hundred eighty seven thousand, and forty. Complex sentences have gone past the trillions. Exhaustive search methods applied to real sentences routinely require several minutes to finish, with larger sentences running more than a day. Clearly, techniques must be developed to diffuse this exponential explosion.

## Hunters and Gatherers in AI

Search is the most common tool for finding solutions in artificial intelligence. The two paths to higher efficiency in search are

1. Reducing the search space. Looking for sub-optimal or impossible solutions. Removing them. Killing them. "Hunting"
2. Efficiently extracting answer(s) from the search space. Collecting satisfactory answer(s). "Gathering"

Much work has been done with regard to the hunters. Finding and using heuristics to guide search has been a major focus. Heuristics are necessary when other techniques cannot reduce the size of the search space to reasonable proportions. Under such circumstances, "guesses" have to be made to guide the search engine to the area of the search space most likely to contain acceptable answers. "Best-first" search (see, among many others, (Charniak et al. 1987)) is an example of how to **use** heuristics.

The "hunting" techniques applied in this research are most closely related to the field of constraint satisfaction problems (CSP). (Beale 1996) overviews this field and (Tsang 1993) covers it in depth. Further references include (Mackworth 1977), (Mackworth & Freuder 1985) and (Mohr & Henderson 1986).

"Gathering" has been studied much less in AI. Most AI problems are content with a single "acceptable" answer. Heuristic search methods generally are sufficient. Certain classes of problems, however, demand **all** correct answers. "Solution synthesis" addresses this need. Solution synthesis techniques (Freuder 1978; see also Tsang & Foster 1990), iteratively combine (gather) partial answers to arrive at a complete list of all correct answers. Often, this list is then rated according to some separate criteria in order to pick the most suitable answer.

In a "blocks" world, CSP techniques and solution synthesis are powerful mechanisms. Many "real-world" problems, however, have a more complex semantics: constraints are not "yes" or "no" but "maybe" and "sometimes." In NL, certain word-sense combinations might make sense in one context but not in another. This is the central problem with previous attempts at using constraint analysis for NL disambiguation (Nagao 1992; Maruyama 1990).[2] We need a method as powerful as CSP for this more complex environment.

[1]Research reported in this paper was supported in part by Contract MDA904-92-C-5189 from the U.S. Department of Defense.

[2]For instance, Nagao eliminates an **ownership** meaning on the basis that a **file-system** is not a **human** agent. As shown in the next section, metonymy and other figurative

| Grupo Roche | a traves de | su | compania | en | espana | adquirir | Dr. Andreu |
|---|---|---|---|---|---|---|---|
| ORGANIZATION | LOCATION INSTRUMENT | OWNER | CORPORATION SOCIAL-EVENT | LOCATION DURING | NATION | ACQUIRE LEARN | HUMAN ORGANIZATION |

Figure 1: Example Sentence

Our proposal is to 1) use constraint dependency information to partition problems into appropriate sub-problems, 2) combine (gather) results from these sub-problems using a new solution synthesis technique, and 3) prune (hunt) these results using branch-and-bound techniques. The rest of this paper addresses each of these issues in turn.

## Constraint Satisfaction Hunters in NL

NL problems can be almost always viewed as bundles of tightly constrained sub-problems, each of which combine at higher, relatively constraint-free levels to produce a complete solution. Beale (1996) argues that syntactic and semantic constraints effectively partition discourse into clusters of locally interacting networks. Here, we summarize those results and report how solution synthesis and branch-and-bound techniques can improve search efficiency.

Figure 1 illustrates the basic lexical ambiguities in a very simple Spanish sentence from the corpus processed by our semantic analyzer. In the figure the Spanish words and phrases are shown with their readings, expressed as corresponding concepts in the underlying ontology. An exhaustive decision tree for this sentence would include 36 possible combinations of word senses, but, when some fairly obvious "literal" semantic constraints are imposed and propagated using arc consistency, all but one of the paths can be eliminated.

Unfortunately, a literal imposition of constraints does not work in NL. For example, *a traves de*, in *a traves de su compania* could very well be **location**, even though a literal constraint would expect *compania* to be a **place**, because corporation names are often used metonymically to stand for "the place of the corporation:"

*I walked to IBM.*
*I walked to where IBM's building is.*

Therefore, the fact that *compania* is not literally a **place** does not rule out the **location** interpretation. In fact, in certain contexts, the **location** interpretation might be preferred. Constraint satisfaction techniques such as arc-consistency, therefore, will be of limited value.

Figure 2 gives a different view of this same NL problem by graphically displaying the constraint dependencies present in Figure 1. These dependencies can be identified simply by iterating through the list of constraints, retrieved from the Mikrokosmos lexicon and ontology (Beale, Nirenburg & Mahesh 1995), and linking any words involved in the same constraint. In Figure 2, three relatively independent sub-parts can be identified. If these sub-parts, or "circles" in our terminology, could be identified, the processing involved in finding a complete solution could be decomposed into three sub-problems. In this paper we assume such a decomposition is possible so that we may concentrate on describing the methods used to combine results from individual circles to form larger and larger solutions, the largest of which will be the solution to the entire problem.

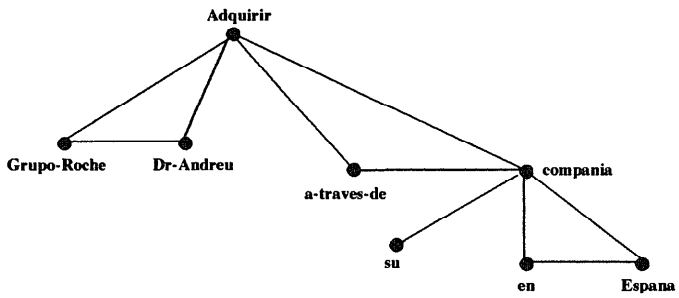


Figure 2: Constraint Dependencies in Sample Sentence

## Solution Synthesis Gatherers in NL

Freuder (1978) introduced Solution Synthesis (SS) as a means to "gather up" all solutions for a CSP without resorting to traditional search methods. Freuder's algorithm (SS-FREUDER) created a set of two-variable nodes that contained combinations of **every** two variables. These two-variable nodes were then combined into three-variable nodes, and so on, until a node containing all the variables, i.e. the solution, was synthesized. At each step, constraints were propagated down and then back up the "tree" of synthesized nodes.

Tsang improved on this scheme with the Essex Algorithms (SS-ESSEX). These algorithms assumed that a list of the variables could be made, after which two-variable nodes were created only between adjacent variables in the list. Higher order nodes were then synthesized as usual, starting from the two-variable nodes. Tsang noted that some orderings of the original list would prove more efficient than others, most notably a "Minimal Bandwidth Ordering" (MBO), which seeks to minimize the distance between constrained variables.

language often overrides such constraints.

The work described here extends and generalizes the concept of MBO. The basic idea of synthesizing solution sets one order higher than their immediate ancestors is discarded. Instead, solution synthesis operates with maximally interacting groupings (circles) of variables of any order and extends to the highest levels of synthesizing. Tsang only creates second order nodes from adjacent variables in a list, with the original list possibly ordered to maximize second order interactions. After that, third and higher order nodes are blindly created from combinations of second order nodes. We extend MBO to the higher levels. The circles of co-constrained variables described in the previous section guide the synthesis process from beginning to end.

The main improvement of this approach comes from a recognition that much of the work in SS-FREUDER and SS-ESSEX was wasted on finding valid combinations of variables which were not related. Even though relatively few words in a sentence are connected through constraints, SS-FREUDER looks for valid combinations of every word pair. Depending on the ordering used, many irrelevant combinations can also be inspected by SS-ESSEX. Furthermore, the ESSEX algorithm tends to carry along unneeded ambiguity. If two related variables are not adjacent in the ESSEX algorithm, their disambiguating power will not be applied until they happen to co-occur in a higher-order synthesis.[3] The current work combines the efficiency of the ESSEX algorithms with the early disambiguation power of the Freuder method.

Our SS-GATHERER algorithm only expends energy on variables directly related by constraints. For instance, for the example in Figure 2, three "base" circles would be formed:

```
1. adquirir, grupo roche, dr andreu
2. adquirir, a traves de, compania
3. compania, en, espana
```

The last two are synthesized into a larger circle:

```
adq., a traves de, compania, en, espana, su
```

This is then synthesized with the first "base" circle above to give the answer to the complete problem.

The bulk of disambiguation occurs in the lower order circles which were chosen to maximize this phenomenon. The correct solution to the example problem was obtained by SS-GATHERER in only five steps. SS-Freuder uses hundreds of extra nodes for this example and SS-ESSEX, 31 extra nodes. Focusing the synthesizer on circles that yield maximum disambiguation power produces huge savings while still guaranteeing the correct solution.

One objection that could be raised to this process is that more work might be needed to create higher-level nodes. For instance, if each variable had three possible values, one needs to test 9 ($3^2$) combinations for each second-order node, but 27 ($3^3$) combinations for third order nodes.[4] If two second-order nodes could be created that would form a third-order node, and each second order node could be completely disambiguated to a single solution, then the third order node could be created without any combinatorics, yielding a total of 18 combinations (9 + 9) that were searched in the case of three values for each variable. Directly creating the third-order node requires the 27 combinations to be searched. However, if the second order nodes do **not** disambiguate, nothing is gained from them. For this reason, base circles can be further sub-divided into groups of second-order nodes, **if** those second-order nodes are connected in the constraint graph.

The algorithm below accepts a list of Circles, ordered from smaller to larger. Each circle has the sub-circles from which it is made identified.

```
1 PROCEDURE SS-GATHERER(Circles)
2    FOR each Circle in Circles
3       PROCESS-CIRCLE(Circle)

4 PROCEDURE PROCESS-CIRCLE(Circle)
     ;;Each Circle in form (Vars-in-Circle Sub-Circles)
5    Output-Plans < -- nil
6    Incoming-Non-Circles < -- REMOVE all
       variables in Sub-Circles from Vars-In-Circle
7    Non-Circle-Combos < --
       Get-Combos(Incoming-Non-Circles)
8    Circle-Combos < --
       Combine-Circles(Sub-Circles)
9    FOR each Non-Circle-Combo in Non-Circle-Combos
10      FOR each Circle-Combo in Circle-Combos
           ;; each incoming circle has consistency
           ;; info stored in arrays:
11         AC-Info < -- access arc constraint
              info from input circles
12         Plan < -- add Non-Circle-Combo
             to Circle-Combo
           ;; Plan is a potential solution for this Circle
           ;; with a value assigned to each variable.
13         IF Arc-Consistent(Plan,AC-Info) THEN
14            Output-Plans < -- Output-Plans + Plan
15            ;; update AC-Info for this circle
16   RETURN Output-Plans
```

The Get-Combos procedure (line 7) simply produces all combinations of value assignments for the input variables. This procedure has complexity $O(a^x)$, where $x$ is the number of variables in the input Incoming-Non-Circles, and $a$ is the maximum number of values for a variable. In the worst case, $x$ will be $n$; this is the case when the initial circle contains all the variables

[3] Freuder's algorithm does not have this disadvantage, because **all** combinations of variables are created, though at great expense.

[4] It should be pointed out that sometimes second order nodes are used in SS-GATHERER, if the dependency structure calls for them. Incidentally, there is nothing special about third-order nodes in SS-GATHERER, although NL constraints seem to produce them the most. It is quite possible that even higher-order nodes could be the starting point.

and no sub-circles. Of course, this is simply an exhaustive search, not Solution Synthesis. In practice, the circles usually contain no more than two variables not involved in input sub-circles, the exceptions almost always pertaining to the base circle, in which case the combine-circles procedure does not add complexity.

The Combine-Circles procedure (line 8) combines all consistent[5] plans already calculated for each input Sub-Circle. In the worst case, where each Sub-Circle contained a single variable, and Sub-Circles contained every variable, then the time complexity would be $O(a^n)$,[6] where $a$ is the maximum size of a variable domain. This is unavoidable, and is the nature of CSPs. However, if the number of circles in Sub-Circles is limited to $c$, and each circle has at most $p$ possible Plans, then the complexity of this step is $O(p^c)$. This step dominates the time complexity of SS-GATHERER. The next section illustrates how this number can be reduced to a "near" constant value.

The FOR loops in lines 9 and 10 simply combine the possible Plan-Combos from the Incoming-Non-Circles with the Circle-Combos calculated for the Sub-Circles. The worst-case time complexity is no worse than the worst-case time complexity for either Combine-Circles or Get-Var-Combos. If Get-Var-Combos produces $a^n$ combinations, then Combine-Circles will produce none, and vice-versa. In practice, Combine-Circles produces $p^c$ combinations while Plan-Combos produces a constant[7] number of combinations. The total complexity of PROCESS-CIRCLE is therefore $O(p^c)$. Again, this number can be reduced to a "near" constant, as shown below. The complexity of SS-GATHERER, then, is $O(p^c)$ times the number of circles, which is proportional to the number of variables, $n$. If $O(p^c)$ can be shown to be a "near" constant, then SS-GATHERER has time complexity that is "near" linear with respect to the number of variables.

For each synthesis, arc consistency may be performed (line 13). As discussed above, however, unmodified CSP techniques such as arc-consistency are not usually helpful in problems with non binary-valued constraints. The next section presents a computational substitute that will produce similar efficiency for these kinds of problems.

## Using Branch-and-Bound in an Uncertain World

The key observation that enables the application of branch-and-bound to solution synthesis problems is that some variables in a synthesis circle are unaffected by variables outside the circle. For example, in the first circle of Figure 2, (Adquirir, Grupo-Roche, Dr-Andreu), neither Grupo-Roche nor Dr-Andreu is connected (through constraints) to any other variables outside the circle. This will allow us to optimize, or reduce, this circle with respect to these two variables. The reduction process uses branch-and-bound techniques.

Implementing this type of branch-and-bound is quite simple using the apparatus of the previous sections. It is a simple matter to determine if, for a given circle, a variable is connected, through constraints, to variables outside the circle. To implement SS-GATHERER with branch-and-bound, we first need to add to the inputs a list of variables that are affected outside the circle.

All that is needed to complete the upgrade of SS-GATHERER is the addition of one procedure and a modification to SET-UP-CONSTRAINTS, the initialization procedure (not shown) so that it sets up the consistency arrays based not on yes-no constraints but rather on values from the 0 to 1 scale. The best approach is to set a THRESHold below which a constraint score should be considered "not satisfied." This will allow the CSP mechanism to eliminate combinations with low-scoring constraint scores. All other combinations will be allowed to go through.

```
1 PROCEDURE PROCESS-CIRCLE(Circle)
...
16a  REDUCE-PLANS(Output-Plans Constrained-Vars)
16   RETURN Output-Plans

17 PROCEDURE REDUCE-PLANS(Plans Constr-Vars)
18    FOR each Plan in Plans
19       Affected-Assignments < -- all value assgnmnts
           from Plan that involve a Constr-Var
20       IF Affected-Assignments is NIL THEN
            ;;This will only happen for the topmost circle
21          Affected-Assignments < -- TOP
22       This-Score < -- Get-Score(Plan)
23       Best-Score < --
           Best-Score[Affected-Assignments]
24       IF (This-Score > Best-Score) THEN
25          Best-Score[Affected-Assignments] < --
              This-Score
26          Best-Plan[Affected-Assignments] < -- Plan
27    RETURN the list of all Best-Plans
```

Why does this work? First of all, each previously processed circle has been reduced, so that the input Circle-Combos will only contain reduced plans. In REDUCE-PLANS, then, we want to keep all possible combinations of variables that are affected outside the circle. Line 19 calculates what these affected combinations are for the input plan. The Best-Score and Best-Plan arrays are then indexed by this (consistently ordered) list of combinations. The goal is that, for each possible combination of assignments of variables affected outside the circle, find the Plan that maximizes that combination. Because all of the other, Unconstrained-Vars, are **not** affected outside the circle, we can find the Plan that maximizes each of the

[5] If one circle has a Plan1 with the assignment $< A, X >$ (value X assigned to variable A) and another Circle has a Plan2 with the assignment $< A, Y >$, then Plan1 and Plan2 are not consistent and cannot be combined.

[6] Combining $n$ variables each with $a$ possible values.

[7] $a^x$, where $x$ is the number of variables in Incoming-Non-Circles, usually 1 or at most 2, except for base circles.

combinations that **are** affected outside the circle.

In the first circle, (Adquirir, Grupo-Roche, Dr-Andreu), only *adquirir* is affected outside the circle. Because there exist other constraints that are not in this circle, we cannot choose a final value for *adquirir*. We will need to retain plans for both possible value assignments: $< adquirir, aquire >$ and $< adquirir, learn >$. On the other hand, *Grupo Roche* and *Dr. Andreu* are not constrained outside the circle. All of the constraints involving them are taken care of within the circle. For this reason, we can find the value assignments of *Grupo Roche* and *Dr. Andreu* that produce the maximum score for the $< adquirir, aquire >$ assignment, and then find the value assignments that produce the maximum score for the $< adquirir, learn >$. All other plans involving non-optimal combinations can be discarded. "Scores" are calculated by comparing constraints, such as a **learn** concept requiring an **animate** agent, with the actual relationships between the value assignments under consideration.

It must be stressed here that discarding the non-optimal plans in no way incurs risk of finding sub-optimal solutions. These are not heuristic decisions being made which might be wrong. Branch-and-bound techniques such as these simply prune off areas of the search space in which optimal solutions can be guaranteed **not** to be found. The only non-certainty present is in the scoring of constraints, which is an inexact science; however, once given a set of scores, these techniques are guaranteed to give the optimal value assignment combinations.

**Branch-and-Bound Results** To illustrate how Branch-and-Bound dramatically reduces the search space, consider the results of applying it to the sample sentence.

| Circle | In-Circles | In-Combos | Reduced-Combos |
|---|---|---|---|
| 1 | none | 2*2*1 = 4 | 2 |
| 2 | none | 2*2*2 = 8 | 4 |
| 3 | none | 2*2*1 = 4 | 2 |
| 4 | 2 and 3 | synth only | 2 |
| 5 | 1 and 4 | synth only | 1 |

The total number of combinations examined is the sum of the input combos; in this case 4+8+4=16. Compare this to an exhaustive search, which would examine (2*1*2*2*2*1*2*1) = 32 combinations. As the input problem size increases, the savings are even more dramatic. This happens because the problem is broken up into manageable sub-parts; the total complexity of the problem is the **sum** of the individual complexities. Without these techniques, the overall complexity is the **product** of the individual complexities. This is the central fact that supports our claim that the number of circles in Sub-Circles is limited to a "near" constant, leading to a "near" linear time complexity for the whole algorithm.

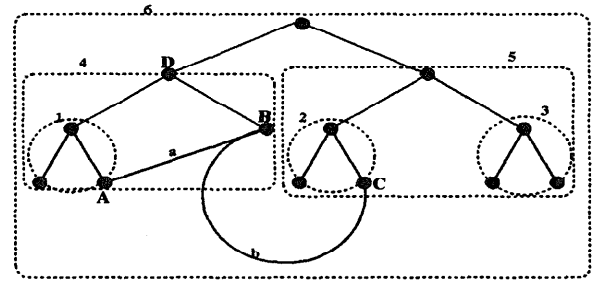


Figure 3: Cross Dependencies

The only way multiplicative growth can occur in SS-GATHERER is when there are constraints across trees, as in Figure 3. In that Figure, several of the circles cannot be fully reduced due to interactions outside the circle. Variable A in Circle 1 cannot be fully reduced[8] because of Arc a. Note, however, that when Circle 4 is synthesized, Variable A **can** be reduced because, at that point, it does not interact outside the larger circle. In Circle 4, Variable B cannot be reduced because it interacts with Variable C. Likewise, Variable C cannot be reduced in Circle 2 because of its interaction with Variable B. In all of these cases, ambiguity must be carried along until no interactions outside the circle exist. For Variables B and C, that does not occur until Circle 6, the entire problem, is processed.

Practically speaking, though, NL problems generally do not allow interactions such as Arc a and Arc b.[9] "Governed" (Haegeman 1991) interactions such as Variable D directly constraining Variable A can occasionally occur, but these only delay reduction to the next higher circle. Thus, some local multiplicative effects can occur, but over the problem as a whole, the complexity is additive.

To illustrate this point, consider what happens as the size of the problem increases. The following table shows actual results of analyses of various size problems. We have tested the SS-GATHERER algorithm extensively on a wide variety of sentences in the context of the Mikrokosmos Machine Translation Project. Over 70 sentences have been analyzed (a relatively large corpus for knowledge-based MT). The claims of near-linear time processing and guaranteed optimal so-

[8] By "fully reduced" we mean all child variables maximized with respect to a single parent, which cannot be reduced because it connects higher up in the tree.

[9] "Long-distance" dependencies do exist, but are relatively rare.

lutions have been verified. These three sentences are representative:

| | Sentence A | Sentence B | Sentence C |
|---|---|---|---|
| # plans | 79 | 95 | 119 |
| exh. combos | 7,864,320 | 56,687,040 | 235 billion |
| SS-GATHERER | 179 | 254 | 327 |

It is interesting to note that a 20% increase in the number of total plans[10] (79 to 95) results in a 626% increase (7.8M to 56M) in the number of exhaustive combinations possible, but only a 42% increase (179 to 254) in the number of combinations considered by SS-GATHERER. As one moves on to even more complex problems, a 25% increase (95 to 119) in the number of plans catapults the exhaustive complexity by 414,600% (56M to 235B) and yet only increases the SS-GATHERER complexity by 29% (254 to 327). As the problem size increases, the minor effects of "local multiplicative" influences diminish with respect to the size of the problem. We expect, therefore, the behavior of this algorithm to move even closer to linear with larger problems (i.e. discourse). And, again, it is important to note that SS-GATHERER is guaranteed to produce the same results as an exhaustive search.

Although time measurements are often misleading, it is important to state the practical outcome of this type of control advancement. Prior to implementing SS-GATHERER, our analyzer failed to complete processing large sentences. The largest sentence above was analyzed for more than a **day** with no results. Using SS-GATHERER, on the other hand, the same sentence was finished in 17 **seconds**. It must be pointed out as well that this is not an artificially selected example. It is a real sentence occurring in natural text, and not an overly large sentence at that.

## Conclusion

We have presented a new control environment for processing Natural Language Semantics. By combining and extending the AI techniques known as constraint satisfaction, solution synthesis and branch-and-bound, we have reduced the search space from billions or more to thousands or less. This paper has concentrated on the combination of branch-and-bound "hunters" with solution synthesis "gatherers."

In the past, the utility of Knowledge-based semantics has been limited, subject to arguments that it only works in "toy" environments. Recent efforts at increasing the size of knowledge bases, however, have created an imbalance with existing control techniques which are unable to handle the explosion of information. We believe that this methodology will enable such work. Furthermore, because this work is a generalization of a control strategy used for simpler binary constraints, we believe that it is applicable to a wide variety of real-life problems. We intend to test this control paradigm on problems outside NLP.

[10]The total number of plans corresponds to the total number of word senses for all the words in the sentence.

## References


Beale, S. 1996. Hunter-Gatherer: Applying Constraint Satisfaction, Branch-and-Bound and Solution Synthesis to Natural Language Semantics, Technical Report, MCCS-96-289, Computing Research Lab, New Mexico State Univ.

Beale, S. and Nirenburg, S. 1995. Dependency-Directed Text Planning. In Proceedings of the 1995 International Joint Conference on Artificial Intelligence, Workshop on Multilingual Text Generation, 13-21. Montreal, Quebec.

Beale, S.; Nirenburg, S. and Mahesh, K. 1995. Semantic Analysis in the Mikrokosmos Machine Translation Project. In Proceedings of the 2nd Symposium on Natural Language Processing, 297-307. Bangkok, Thailand.

Charniak, E; Riesbeck, C.K.; McDermott D.V. and Meehan, J.R. 1987. *Artificial Intelligence Programming.* Hillsdale, NJ: Erlbaum.

Freuder, E.C. 1978. Synthesizing Constraint Expressions. *Communications ACM* 21(11): 958-966.

Haegeman, L. 1991. *An Introduction to Government and Binding Theory.* Oxford, U.K.: Blackwell Publishers.

Lawler, E.W. and Wood, D.E. 1966. Branch-and-Bound Methods: a Survey. *Operations Research* 14: 699-719.

Mackworth, A.K. 1977. Consistency in Networks of Relations. *Artificial Intelligence* 8(1): 99-118.

Mackworth, A.K. and Freuder, E.C. 1985. The Complexity of Some Polynomial Consistency Algorithms for Constraint Satisfaction Problems. *Artificial Intelligence* 25: 65-74.

Maruyama, H. 1990. Structural Disambiguation with Constraint Propagation. In Proceedings 28th Conference of the Association for Computational Linguistics, 31-38. Pittsburgh, Pennsylvania.

Mohr, R. and Henderson, T.C. 1986. Arc and Path Consistency Revisited. *Artificial Intelligence* 28: 225-233.

Nagao, K. 1992. A Preferential Constraint Satisfaction Technique for Natural Language Analysis. In Proceedings 10th European Conference on Artificial Intelligence, 523-527. Vienna.

Tsang, E. 1993. *Foundations of Constraint Satisfaction.* London: Academic Press.

Tsang, E. and Foster, N. 1990. Solution Synthesis in the Constraint Satisfaction Problem, Technical Report, CSM-142, Dept. of Computer Science, Univ. of Essex.