Speech Understanding System using Classification Trees

Kwan Yi
School of Computer Science
McGill University, Montreal

A Thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfilment of the requirements for the degree of M.Sc. in Computer Science.

0-612-37180-8

Abstract

The goal of Speech Understanding Systems(SUS) is to extract meanings from a sequence of hypothetical words generated by a speech recognizer. Recently SUSs tend to rely on robust matchers to perform this task. This thesis describes a new method using classification trees acting as a robust matcher for speech understanding. Classification trees are used as a learning method to learn rules automatically from training data. This thesis investigates uses of classification trees in speech system and some general algorithms applied on classification trees. The linguistic approach requires more human time because of the overhead associated with handling a large number of rules, whereas the proposed approach eliminates the need to handcode and debug the rules. Also, this approach is highly resistant to errors by the speaker or by the speech recognizer by depending on some semantically important words rather than entire word sequence. Furthermore, by re-training classification trees on a new set of training data later, system improvement is done easily and automatically. The thesis discusses a speech understanding system built at McGill University using the DARPA-sponsored Air Travel Information System(ATIS) task as training corpus and testbed.

Résumé

L'objectif d'un système de compréhension de la parole (SCP) est d'extraire le sens d'une séquence de mots hypothétiques générés par un système de reconnaissance de la parole. Les SCP actuels se fient à un système de correspondance robuste pour la réalisation. Cette thèse décrit une méthode nouvelle utilisant les arbres de classification (AC) comme système de correspondance robuste pour le SCP. Les AC sont utilisés comme une méthode d'apprentissage des règles de façon automatique à partir des données d'entraînement. Cette thèse examine l'utilisation des AC dans un SCP ainsi que quelques algorithmes généraux s'appliquant aux ACs. L'approche linguistique coûte plus cher pour supporter un grand nombre de règles tandis que cette approche élimine le besoin d'écrire manuellement et de tracer les règles. Aussi, cette approche est moins sensible aux erreurs causées par le locuteur ou par le système de reconnaissance parce qu'elle dépend plus de certains mots clés qu'à une séquence de mots entiers. De plus, en appliquant AC à un nouveau groupe de données d'entraînement plus tard, le système peut être amélioré facilement et automatiquement. La thèse couvre le SCP construit à l'université McGill utilisant le ≪Air Travel Information System (ATIS)≫ parrainé par l'agence DARPA comme une source d'entraînement et de tests.

Acknowledgements

My present status in life is a result of the efforts and sacrifices of my parents and family.

Renato De Mori, my thesis adviser, taught me a lot about professional attitude towards work in addition to guiding me through my research. As a supervisor, he shows an exemplary concern for the well-being of his students to ensure that they have financial support. Also, he is a man of great courtesy.

My co-supervisor, Roland Kuhn, is another great person with nice personality. He helped me a lot from miscellanous things to core work. He provided me a chance to work in CRIM for the summer 96. I always got more than what I expected and wanted whenever I spoke with him. I can't thank him enough.

I am also thankful to all the members of the speech lab for their help and cooperation. I have to explicitly express my gratitude to Charles Snow and Matteo Contolini for their help. Especially, Charles for reading my thesis and providing miscellaneous invaluable advice.

As a graduate student in school of computer science at McGill University, I have been very happy with our great secretaries in our department.

I am grateful to my friends in CS, Feng Li, Yi-Jen Huang, Bin Cao, Owen, and Winnie for providing many happy distractions and constant encouragement. Emily read my thesis and corrected and provided her advice. Pung Chitra Hay wrote French-version abstract of this thesis for me. Especially, I enjoyed a lot with Imran Ahmed although we don't go for movie yet.

My beloved wife Jeeyoun's great patience, encouragement, love and guidance were essential for my joining and smooth-sailing through my life. My daughter Judy; she gave me a lot of happiness.

To my great parents and mother-in-law, My thanks and my love go to you most of all.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

Introduction

1.1. Backgrounds and Problem Statement

A definition of 'Understanding' is stated in [37].

> To understand something is to transform it from one representation into another, where ...

In the Human Understanding System(HUS), the most important factors might be one's intellectual ability, psychological status, background, etc. However, in the Machine Understanding System(MUS), the factors listed above are unnecessary because the domain in which the MUS works is predetermined. An example in [37] clearly demonstrate this point. Consider a sentence "I need to go to New York as soon as possible". A MUS dealing with the airline information domain would have "understood" if it finds the first available flight to New York. A person knowing that the speaker's family live in New York would have "understood" that there may be a problem in the speaker's family. The purpose of speech/natural language understanding is to extract semantics from written or spoken sentences. By applying the definition of 'Understanding' stated above, it translates to transforming a sequence of words into a semantic representative form. That is, the ability of a MUS to understand is limited to a particular domain, and the key point in a MUS is how to make an understanding representation form.

The speech understanding system designed in this report works under the Air Travel Information System(ATIS) domain. The development of Speech Understanding Systems(SUS) under ATIS is currently driven by two main streams : the corpus based and the linguistic. In the corpus based approach, a system is statistically modeled and the parameters of a statistical model are learned from an annotated corpus. In the linguistic based approach, a system consists of hand-coded rules reflecting linguistic/syntactic/semantic knowledge of its domain. Hybrid approach combines the two approaches. A hybrid system takes a stochastic model and hand-coded rules as understanding tools. In [**22**] the two principles of hybrid approach are discussed. The benefits of the hybrid approach are :

(i) Learn everything that can be learned from available data.

(ii) Rather than attempting to learn complex and rare linguistic cases, use simple ways to incorporate established linguistic knowledge into the system.

There are some hybrid speech understanding systems developed under the DARPA ATIS task. For stochastic tools, semantic classification tree for CHANEL system [**19, 18, 17, 25**] is devised. A Hidden Markov Model is based for the stochastic models in CHRONUS system [**36, 35, 22**] and HUM system [**41, 40**].

The focus of this work is to train classification trees, that is to learn semantic rules from annotated data, and to set up a set of Classification Trees(CTs) as a robust matcher of SUS. In the linguistic approach, due to involvement of many experts extracting their own semantic rules, it requires a lot of time and effort in development of an SUS. On the other hand, this automatic learning of CTs dramatically reduces waste of time and effort. It should be noted that in SUS, the rules learned by classification trees are more robust to errors of a spoken sentence containing ungrammatic and non-linguistic occurrences.

1.2. Application of Classification Trees in Speech Recognition System

This thesis will present how classification trees can be used in a speech understanding system as a way of learning semantic rules, based on annotated corpus.

However, in speech recognition processing, classification trees have already been used successfully. The list of the applications of classification trees in its related domain is presented below :

(i) *Codebook*[1] *Design*

Vector quantization is a technique that maps a stream of high rate digital data into a stream of relatively lower rate digital data [7], which is applied to the input of Hidden Markov Models(HMMs) in speech recognition system. In the design of a codebook, two factors are crucial : time to search codebook and data distortion. Full search on a codebook can reduce data loss whereas it can have increased search-time. Thus, the aim in codebook design is how to have well-designed codebook with the trade-off between two factors. As one way of codebook design, classification trees are used in designing a codebook to speed up the coding of observation vector. Supposed that

[1]Basically this is a set of vectors. In vector quantization, a vector in the set is mapped a data from $\mathcal{K}$-space.

there are M vectors in the codebook, it is obvious that the time finding a vector will require $\Theta(\log_2 M)$ time, if the codebook is designed with trees. Various methods in obtaining codebooks using classification tree are found in [**13, 15, 38, 46**].

(ii) *Language Modeling*

The purpose of speech recognition system is to find w maximizing $P(w)P(y|w)$, where w is a sequence of words and y is an acoustic observation vector. The bigram $f(w_i|w_{i-1})$ or the trigram $f(w_i|w_{i-2}, w_{i-1})$ language model can be used to estimate $P(w)$. In [**20**] researchers at IBM proposed CT-based system with the trigram model, which yields better result than either one of them. In a recent paper [**27**], a language model integrating an acoustics model, using semantic classification trees is described.

(iii) *Acoustic Modeling*

In an acoustic model, due to the unfeasibility of a word modeling for large vocabulary speech recognition, a sub-word modeling is used. To construct a sub-word modeling, several trials based on classification trees are made. In [**8**] the algorithm of triphones clustering, based on classification trees is proposed. Phoneme modeling based on CT was originally proposed in [**24**]. In [**24**], classification trees are employed to represent a phoneme as a function of the phone-context. In [**34**], Kuhn *et al.* extend the work of [**24**] in four ways by (a) taking the Poisson criterion to find the M best questions, (b) applying expansion-pruning instead of growing algorithm, (c) adding a "DON'T KNOW" subtree of each question, and (d) permitting an arbitrary feature schema in question generation.

1.3. Overview of The Thesis

This thesis focuses on the use of classification trees in speech understanding system. We will examine how they can be trained and applied for our purpose of 'understanding a sentence'.

- Chapter 2. *Machine Learning with Classification Trees*

  Presents the theoretical background of classification trees and algorithms for constructing classification trees. Reviews splitting/stopping/pruning methods used in the construction of a classification tree, and compares splitting/pruning methods.

- Chapter 3. *Machine Learning on Semantic Interpretation Rules*

  Describes the SUSs developed under ATIS domain. Most of the chapter is devoted to describe the corpus-based SUSs.

- Chapter 4. *Speech Understanding System at McGill*

  Describes the overall structure of SUS developed at McGill for this thesis, and the structure and function of each module of the system. Descriptions of structures of the trained classification trees and the semantic representation used in this system are given.

- Chapter 5. *Result and Improvements*

  Provides various analysis for the output of the SUS at McGill with ATIS2 test data. Suggests further work to improve capacity of the system.

- Appendix

  Provides list of output of this SUS with ATIS2 test data.

CHAPTER 2

Machine Learning with Classification Trees

2.1. Basic Classification Tree Concepts

A *tree* is commonly defined as a connected, acyclic, undirected graph [2]. For the purposes of the work described in this thesis, a tree is defined recursively as a finite set of one or more nodes such that:

(i) there is a specially designated node called the root;

(ii) the remaining nodes are partitioned into n disjoint sets $T_1, \dots, T_n$ where each T_i is called the subtree of the root $1 \leq i \leq n$

(iii) each subtree can recursively be defined as a root and a partition into disjoint sets

In order to properly introduce classification trees, which will be discussed later, it is important to consider binary trees. A *binary tree* [9] is recursively defined as a finite set of nodes, each of which either :

- is empty or

- consists of a root and two disjoint binary trees called the left subtree and the right subtree.

Trees are frequently used in different applications, in such roles as *set-representations*, *decision-trees*, *game-trees*, etc. As a set-representation, each tree represents a set. Set functions like 'union' or 'find' are simply implemented. For example, the union of two trees is accomplished by making one of the trees a subtree of the other. There are interesting applications of trees in programming games including tic-tac-toe, go, chess, etc. In these examples, the tree contains all of the possible sequences of board configurations and the root node represents the initial board setup. Another application of trees is in decision making. In this application, each node of a tree represents a decision and a tree itself serves as a set of decisions which lead to an answer. *Classification trees* fall into this category of application.

A simple instance of a typical classification tree is shown in figure 2.1. They are binary trees in which all nodes are divided into two types represented as circular ones and square ones. We call the circular nodes "non-terminal" or "decision" ones and the square nodes "terminal" or "class" ones. Each non-terminal node is associated with a question (condition) and each terminal node is associated with a class labeling. In figure 2.1, there are four questions $(x < a, y < c, y < d, x < b)$ inside non-terminal nodes and three classes $(1, 2, 3)$ inside terminal nodes. To classify a data item, the tree is traversed, starting from the root, following the left branch if the condition on the current node is true and following the right branch if not. By performing this simple procedure recursively, a terminal node

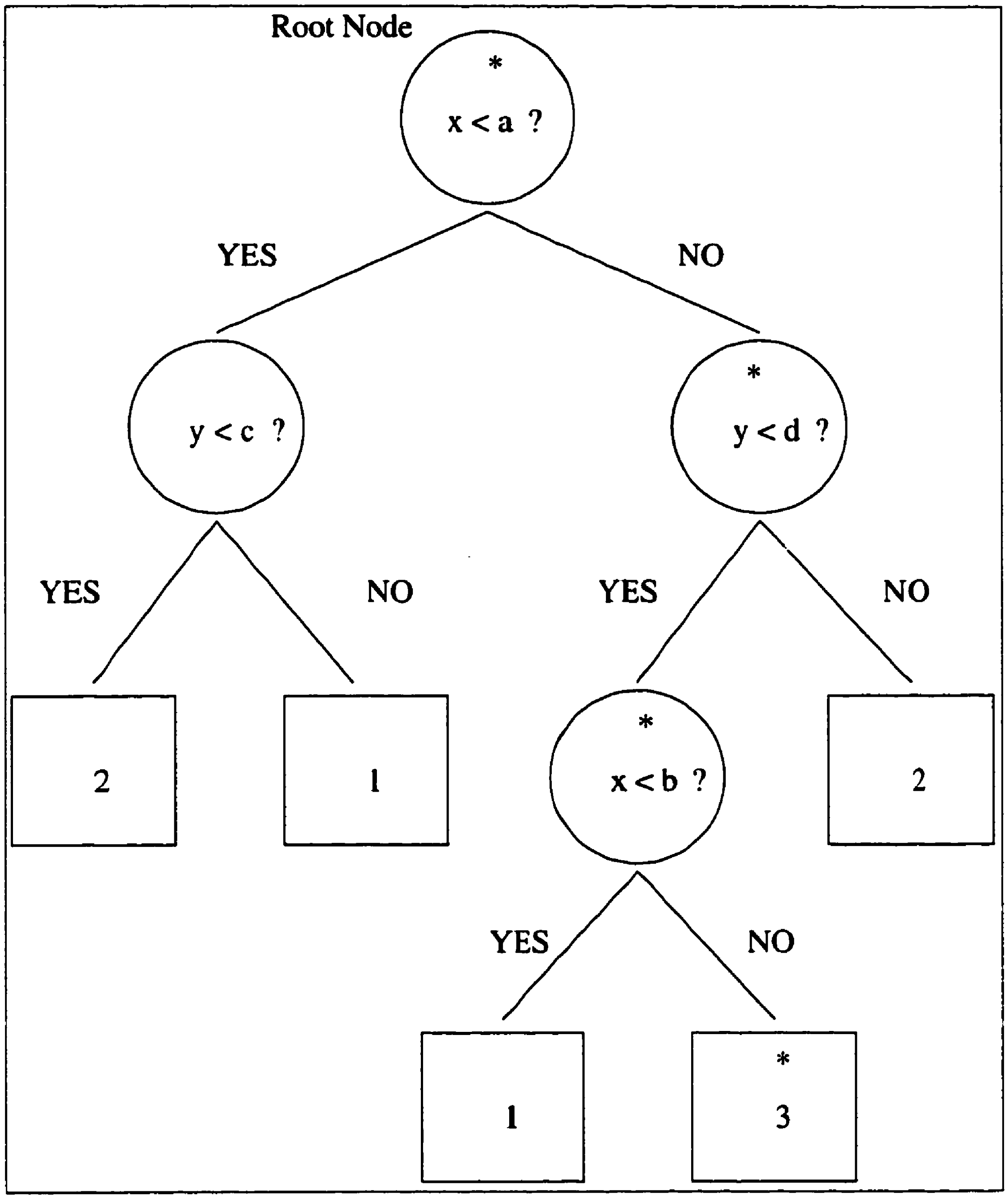


FIGURE 2.1. Example of Classification Tree.

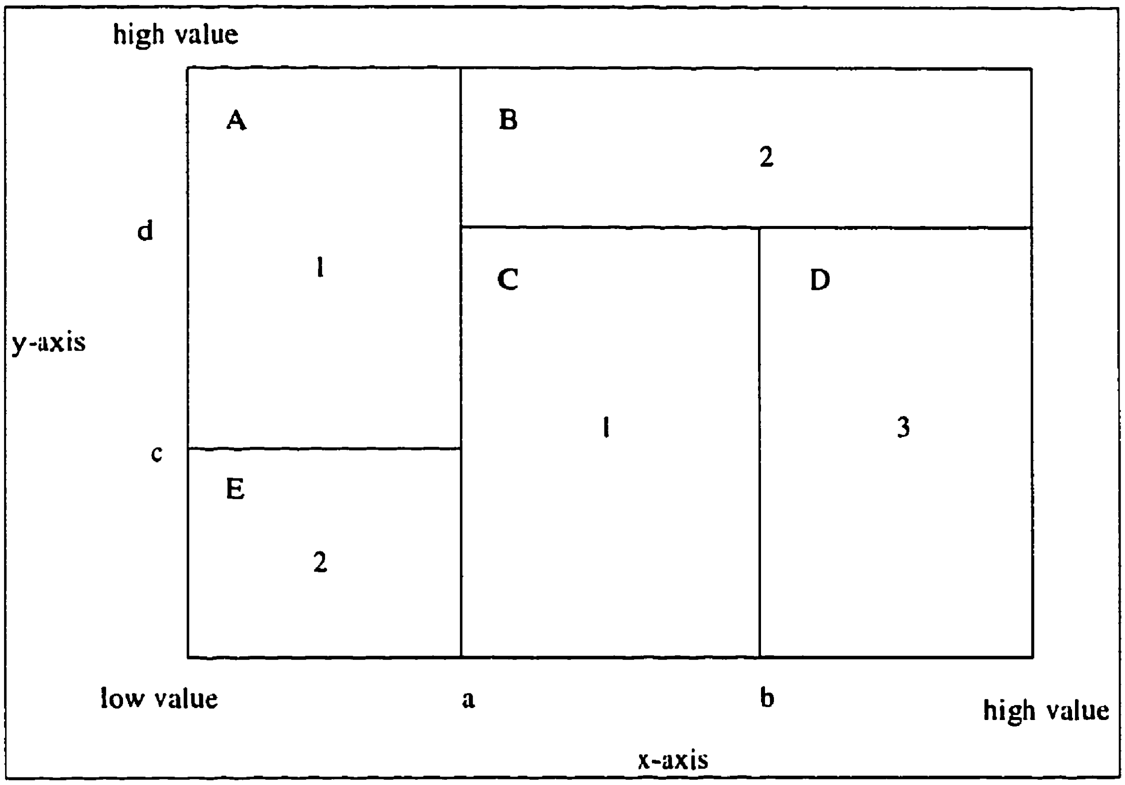


FIGURE 2.2. Geometric View of Classification Tree

is eventually reached, at which point the item based on which the questions were answered is ascribed to the class whose label is displayed in the terminal node.

Suppose we have a feature vector[1] (x, y), where $x \geq b > a$ and $y < d$ in figure 2.2, which has to be classified. Following the classification steps of the tree, we find that the feature vector is classified as class 3. The path we have followed is marked by $*$s.

Classification trees may also be considered from a geometric viewpoint. In figure 2.2, a 2-dimensional picture corresponding to the classification tree of figure 2.1 is shown. Within the conditions displayed in the non-terminal nodes in figure 2.1, x and y are variables and

[1]A feature vector indicates a measurement vector; the terminology "feature" is commonly used in pattern recognition application.

a, b, c, and d are constants. Since there are two variables, we can consider the measurement space to be 2-dimensional with axes x and y two axes, values of constants a, b, c, d are real-valued points in the Cartesian plane that is partitioned into area A, B, C, D, E as indicated in figure 2.1. From this viewpoint, each question given by a non-terminal node in the tree serves as a constraint-function on the measurement space. So rather than choosing left or right branches based on the conditions of tree-nodes, we examine the space in the graph depending on the outcome of the function. For example, consider a measurement vector $X = (x, y) = (e, f)$, where x and y are variables and e and f are constants satisfying $e < a$ and $f < c$. The only area satisfying the two conditions is E, so we classify the vector as class 2. Thus generally speaking, a function representing a question in a node recursively separates the measurement space considered into two parts until we get to a segment of space where we can't split any more.

In order to arrive to a precise formulation, a notation used in this thesis is now introduced. Let measurement space χ be defined as the set of all measurement vectors in the space of dimension q. C is the set of classes such that $C = \{1, \ldots, J\}$. A function $d(x)$ on any vector x in the measurement space is defined such that any value of a function $d(x)$ is in the class set C. A_k is a subset of the measurement space satisfying $d(x) = k$ for any vector x in the measurement space. In addition, for any two different sets, A_i and A_j, the intersection of A_i and A_j is null; that is, $A_i \cap A_j = \emptyset$.

The role of a classification tree is to classify an input vector x of a measurement space by leading it to A_k, if $d(x) = k$, and assigning class k to the vector. The classification tree represents a partition of a measurement space with a class labeling for each partition.

Class labeling is performed as described previously, by following tree branches depending on node conditions.

The classification technique using trees has been widely and successfully applied. Later, we will examine in detail some specific applications. The tree representation has several advantages: The tree structure is easy to understand because of the simplicity of its concept. It is also compact enough to be stored on a file. In terms of speed, classification is relatively swift having a computational time complexity proportional to the height of the tree.

2.2. Research on Classification Trees

The construction of a classification tree from training data sets has been the object of research in pattern recognition, decision theory, and statistics.

A taxonomy of machine learning involving classification trees has been proposed in [**14**] along the following lines.

- Classification on the basis of the underlying learning strategies used
- Classification on the basis of the representation of knowledge or skill acquired by the learner
- Classification in terms of the application domain of the performance system for which knowledge is acquired.

With the advent of knowledge-based expert systems, the methods mentioned above have been also used for knowledge representation. In expert systems, knowledge is collected by an interaction between a knowledge engineer and a domain specialist. There may exist limitations to this method of knowledge-collection in terms of knowledge completeness. In other words, is it possible for a domain specialist to express all that he/she knows about the area? For example, when a computer scientist is asked a question such as "Could you describe everything that you know about Data Structures?", how completely could he/she cover the subject? Could the scientist even be certain of covering the whole of his/her knowledge? This is clearly a problem.

Concerning models and methods for knowledge representation and for knowledge acquisition has been a major issue. Many different approaches have been suggested in the form of *structures* [**10**], *discrimination nets* [**6**], *production rules* [**4**], *semantic network* [**3**] etc.

For this purpose, Quinlan [**31**] proposed a new approach to knowledge representation and acquisition consisting in using decision trees, trained from examples, for knowledge representation and acquisition. The decision tree technique has the property of completeness in that it classifies every element of a domain. Nevertheless, it might have an inherent error rate in classifying this whereas other techniques may be more robust, but limited coverage due to incomplete knowledge collected. Quinlan's method of deriving a tree from data to represent domain knowledge and dividing the domain into several classes was founded on Hunt's previous work [**10**]. Breiman *et al.* [**21**] also provide an excellent theoretical foundation for classification trees.

Classification trees have been used in diverse applications including speech recognition, which we'll explore in detail in the next chapter, pattern recognition, medical diagnosis and prognosis, and remote sensing. In [**28**], application areas are discussed in more detail. Also, various methods of designing classification trees to adjust to various problems have been developed. These method will be reviewed in later sections of this chapter.

2.3. Methods for Classification Tree Construction

In [**42**], heuristic methods of constructing classification tree are categorized :

(i) Bottom-Up approaches[2]

(ii) Top-Down approaches[3]

(iii) Hybrid approaches [**16**]

(iv) Growing-Pruning

The Growing-Pruning approach is based on Top-Down approaches and it is simply an extension of Top-Down approaches in that a pruning step is introduced to prevent over-trained[4] or under-trained[5] trees. Our description will concentrate on Top-Down approaches and the Growing-Pruning approach, with more focus on the latter.

The power of classification trees trained on data sets lies in the possibility to classify data which are not used in the training process. The maximum predictive power can be obtained when a tree is obtained having the right size. Predictive power decreases with over-trained or under-trained trees.

[2]In these approaches a classification tree is constructed from bottom to top. For example, using some distance measure, the two nodes with smaller distance are merged to form a root node of the two nodes.

[3]These approaches keep expanding down the tree from the root using a splitting rule, and then the expansion stops when a stopping rule is reached.

[4]An over-trained tree is a tree which is pruned late and/or is too big due to fitting the data on which it was trained too well.

[5]A under-trained tree is a tree which is pruned earlier and/or is too small.

In an effort to obtain right-sized trees, top-down approaches expand down the tree starting from the root node and keep splitting until some stopping rules are satisfied. In the growing-pruning approach, a tree is trained while a combination of splitting, stopping and pruning rules are applied.

We will discuss some of the methods devised and tested in constructing classification trees to determine the best method for obtaining right-sized trees.

2.3.1. Preliminaries

Since it has been proved in [**11**] that the construction of optimal classification trees is an NP-complete problem, the focus in construction of classification trees has naturally shifted to the problem of how to obtain near-optimal solutions. Various heuristic methods for the construction of classification trees have been proposed at various levels. Some effective heuristic strategies are discussed in the following subsections.

2.3.1.1. *Definitions*

Some terminology for describing classification tree construction (see [**21**] for more details) is now introduced.

- Let (X, Y) be jointly distributed random variables with X in R^q and Y in the set $S = \{1, \ldots, J\}$. X is a measurement, a feature or a pattern vector and Y is associated with a class label in the set S.
- A measurement space χ is defined as containing all possible measurements, feature or pattern vectors.

- A learning sample is denoted by ζ :

$$\zeta = \{(X_1, j_1), \dots, (X_n, j_n)\}$$

  where $X_n \in \chi$ and $j_n \in \{1, \dots, J\}, n = 1, \dots, N$

- An attribute is an element of a feature vector.

- A classifier, a classification rule or a decision rule is a function $d(X)$ mapping χ in $\mathcal{R}^{\text{u}}$ into one of the numbers $1, \dots, J$. Also, a classifier is a partition of χ into J disjoint subsets , $A_1, \dots, A_J$, $\chi = \bigcup_j A_j$ such that for every χ in A_j the predicted class is j.

- The true misclassification rate, $R^*(d)$, is the proportion of vectors misclassified by classification rules, d.

$$R^*(d) = P(d(X) \neq Y)$$

  .

- Let $R(d)$ denote the estimated misclassification rate.

$$R(d) = M/N$$

  where M is the number of samples in L such that $d(X_n) \neq j_n$ (see definition 3), and N is the total number of samples in L.

- When the training sample is used to estimate $R^*(d)$, $R(d)$ is called the re-substitution estimate of $R^*(d)$ and when the test sample is used to estimate $R^*(d)$, $R(d)$ is called a test sample estimate of $R^*(d)$.

- An impurity function Φ is a function defined on the set of all J-tuples of numbers $(p_1, \dots, p_J)$ satisfying $p_j \geq 0$ $(j = 1, \dots, J)$ and $\sum_j p_j = 1$ with the following properties :
  - Φ is a maximum only at the point $(\frac{1}{J}, \dots, \frac{1}{J})$
  - Φ achieves its minimum only at the points $(1, 0, \dots, 0)$, $(0, 1, 0, \dots, 0)$, $\dots$, $(0, 0, \dots, 0, 1)$
  - Φ is a symmetric function of $(p_1, \dots, p_J)$
- Given an impurity function Φ, we define the impurity measure $i(t)$ of any node t as $i(t) = \Phi(p(1/t), p(2/t), \dots, p(J/t))$.
- A branch T_t of tree T with root node t in T consists of node t and all the descendants of t in T.
- Pruning a branch T_t from a tree T entails deleting from T all descendants of t. We denote the pruned tree as $T - T_t$.
- A branch T_t of T with root node t consists of the node t and all descendants of t in T.

2.3.1.2. *Elements of Classification Tree Construction*

In [**21**], Breiman *et al.* pointed out that three elements are required for constructing a classification tree based on growing-pruning algorithm:

(i) A set Q of questions

(ii) A rule for selecting the best split at any node

(iii) A criterion for choosing the right-sized tree

ATTRIBUTE				CLASS
Outlook	Temperature	Humidity	Windy	
sunny	hot	high	false	N
sunny	hot	high	true	N
overcast	hot	high	false	P
rain	mild	normal	false	P
rain	cool	normal	false	P

TABLE 2.3.2.1. A small example of correct Data Set

ATTRIBUTE				CLASS
Outlook	Temperature	Humidity	Windy	
overcast	hot	high	false	N
sunny	hot	high	true	N
overcast	hot	high	false	P
rain	mild	normal	false	P
rain	mild	normal	false	P

TABLE 2.3.2.2. A small example of an erroneous Data Set

The set Q of questions depends entirely on the specific application. The second and third elements will be examined in sections 2.4.3 and 2.4.4 respectively.

2.3.1.3. *Limitations of Data Sets*

In the construction of a classification tree, it is assumed that the training set is representative of the measurement vectors and the corresponding class labels are available.

In reality, however, it might not be possible to obtain a complete training data set that covers the entire measurement space well. Thus, we may be unable to avoid encountering the situation in which our classification tree has been trained on contaminated data. Based on this reality, then, the process of preparing the data might be as much as the choice of technique because, even if we have adopted a state-of-the-art technique, it will still be impossible to obtain a good tree due to the bias of the data set.

We give a simple example in tables 2.3.2.1 & 2.3.2.2 showing how easily data can be corrupted and what serious damage could be caused by a small amount of corruption. Comparing the first and third rows in our example tables, we notice a few facts. The value of the outlook attribute has been corrupted to 'overcast' from 'sunny'. As a result of changing one value, the first and third rows have the same data but with different class labels, which leads to a contradiction. This simple example shows that even one corrupted value of a feature vector can render the vector itself useless, as well as any other vector which uses the same values but have different class labels. This corruption might very well lead to a different and more complex training tree to fit the corrupted data. Thus, to reduce the unwanted effects of biased data in constructing trees, we may need to use techniques of data collection with verification/correction.

Data collection is a process for obtaining near-optimal data sets (optimal in the sense that it tries to cover the entire measure space uniformly.) *Data verification/correction* is a process that attempts to correct for situations in which contamination of data is unavoidable.

To cope with erroneous training sets the tree design algorithm must have the following two properties [**31**] :

(i) The algorithm must be able to work with partial data sets

(ii) The algorithm must be able to decide that testing further attributes will not improve the predictive accuracy of the classification tree

To satisfy the first requirement, Quinlan suggests having more classes instead of only one class on the terminal node or a probabilistic class which is ratio class rather than fixed class. This topic will be treated in detail in section 2.4.4. To satisfy the second property, the use of the chi-square test for stochastic independence has been proposed. Such a method has been found to be effective in preventing the generation of overly complex trees that attempt to incorporate erroneous data, without affecting performance for error-free data.

2.3.2. Splitting Rules

When expanding a tree, we must assign a question to each node, which acts as a classifier. Among the set of candidate questions, the most suitable question is selected depending on the splitting rule adopted. Ben-Bassat [**1**] divides feature-evaluation rules into the following three categories:

(i) Rules based on Information or Entropy

(ii) Rules based on Distance Measures

(iii) Rules based on Dependence Measures

2.3.2.1. *Rules based on Information or Entropy*

A. Information Gain

The method of Information Gain is based on Shannon's entropy, defined as $H = -\sum_i p_i \log p_i$ where p_i is an a priori probability of class i. The basic idea of the information heuristic implemented in the ACLS program [**44**, **45**] and the ID3 program [**30**, **29**] is to select an attribute whose information gain is maximal for a node.

The algorithm is as follows. Suppose a training data set has n objects of class N and y objects of class Y. The information derivable from relative frequencies of class membership, $I(y, n)$, is expressed as:

$$I(y,n) = (-\frac{y}{y+n})(\log \frac{y}{y+n})(-\frac{n}{y+n})(\log \frac{n}{y+n}) \qquad (2.3.2.1)$$

Suppose an attribute A has v possible values $(A_1, \dots, A_v)$ and y_i and n_i are the numbers of objects of class Y and N respectively, having the ith value A_i of A. The expected information requirement after testing attribute A and weighting each node by relative frequencies of class membership based on the attribute $E(A)$ can be written as :

$$E(A) = (\sum_{i=1}^{v} \frac{y_i + n_i}{y+n}) I(y_i, n_i) \qquad (2.3.2.2)$$

Finally, the information gained by a node on attribute A is :

$$gain(A) = I(y,n) - E(A) \qquad (2.3.2.3)$$

and the attribute with the highest information gain will be selected for the node under consideration [**33**].

B. Information Gain Ratio

The experimental work of Kononenko *et al.* [**12**] shows that information gain favors attributes with many values. To compensate for this fact, Quinlan [**31**] suggests the information gain ratio method for selecting an attribute. The information of an attribute value can be expressed as :

$$G(A) = -\sum_{i=1}^{v} \frac{p_i + n_i}{p + n} \log \frac{p_i + n_i}{p + n} \qquad (2.3.2.4)$$

For the choice of an attribute, the ratio gain(Attribute)/G(Attribute) is used and the attribute whose ratio is the largest is selected for the node under consideration.

2.3.2.2. *Rules based on distance measures*

The criteria in this category measure separability, divergence or discrimination between classes [**28**]. The most popular rule of this type is the Gini Index rule [**21**]. Also, Breiman *et al.* suggested another method called the Twoing Criterion, which is useful for cases in which there is a relatively large number of classes. In the Twoing Rule, however, the amount of computational time is proportional to the number of classes, which is disadvantageous. As in other distance-based measures, Bhattacharya [**23**], and Kolmogorov-Smirnoff distances [**39**] are used.

A. Gini Criterion

The basic idea of the Gini Criterion is that, when considering splitting criteria at an internal node, the node whose offspring nodes are the most "pure" is selected.

The Gini impurity function is as follows:

$$i(t) = \sum_{i \neq j} p(i/t)p(j/t) \tag{2.3.2.5}$$

where $p(i/t)$ is the probability that the sample training set belongs to the class i, given a node t.

The "goodness" of the split S is defined as

$$\Delta i(S,t) = i(t) - i(t_L)P_L - i(t_R)P_R \tag{2.3.2.6}$$

where P_L and P_R are the proportion in which data falls on the left-child and right-child of node i, respectively.

Finally, choose a split s^* which gives the largest decrease in impurity :

$$\Delta i(s^*,t) = \max_{s \in S}(\Delta i(s,t)) \tag{2.3.2.7}$$

B. Twoing Criterion

The Twoing Criterion method was proposed as a better way to deal with the multi-class problem [**21**]. The Two Criterion function $\Theta(s,t)$ is defined as :

$$\Theta(s,t) = \frac{P_L P_R}{4} \sum_j \mid p(\frac{j}{t_L}) - p(\frac{j}{t_R}) \mid$$

The best split for any node t and split s of t into t_R and t_L is a split which maximizes the Twoing function $\Delta(s,t)$.

2.3.2.3. *Rules based on dependence measures*

The measures in this category refer to the statistical dependence between two random variables [**28**].

2.3.3. Stopping Rules

The stopping rule determines when to stop splitting nodes. Stopping rules are based on two tree design methods: top-down tree design, in which a tree grows without pruning and growing-pruning tree design, in which the growing and pruning processes alternate until the final tree is obtained.

A threshold is set to limit the splitting rule. For example, suppose our splitting rule is the Gini Criterion. The threshold $0 < \beta < 1$ can be defined as :

$$\max_{s \in S} \Delta i(s,t) < \beta \tag{2.3.2.8}$$

β acts as a degree of maximum impurity difference.

Also, as another simple way of stopping rule, a tree is split until each terminal node has fewer than N items(N is close to 1).

The threshold method does not, however, lead to a right-sized tree. Since the threshold is applied equally to all nodes, some nodes of the tree obtained using this method are over-trained and some nodes are under-trained. That is, since a terminal node could have a child node which can get higher value of Δi, converting such a node to terminal node prevents prospective child nodes from being considered. Thus, the tree fails to have high predictive power on a data set which is not used for the training process even though it can fit to the training set well. Some experimental results using various thresholds are reported in [**21**].

2.3.4. Pruning Rules

As an alternative to stopping rules for top-down tree design, pruning rules were added as a step in the tree training process. The stopping rule for the growing-pruning process uses either the threshold method mentioned above with $\beta = 0$ or the stopping condition of reaching a node that has fewer than N items where N is close to 1. A pruning rule will be applied to the over-trained tree to obtain an intermediate right-sized tree.

2.3.4.1. *Cost Complexity Pruning*

In [**21**], Breiman *et al.* implemented a pruning method in the CART (Classification And Regression Tree) program. The proposed pruning method involves finding a node, which is called the "weakest link" in the tree grown, and turning it into a terminal node. Here is how to find the weakest link in a tree T.

For each node n of tree T, the re-substitution estimate of node n, $R(n)$ is computed (see definitions 5 and 6). Specifically, $R(n)$ is the ratio of the number of misclassified data items on a node n to the total number of sample data items. $R(T)$, the re-substitution estimate of tree T is defined as the sum of the $R(n)$s at all terminal nodes of tree T as follows:

$$R(T) = \sum_{n \in \tilde{T}} R(n) \tag{2.3.2.9}$$

where $\tilde{T}$ is the set of terminal nodes of tree T.

The re-substitution estimate of subtree T_n rooted at node n, $R(T_n)$, is defined similarly to R(T):

$$R(T_n) = \sum_{n \in \tilde{T}_n} R(n) \tag{2.3.2.10}$$

Finally, we define a function $g_l(n)$, where n is an internal node of tree T, by :

$$g_l(n) = \frac{R(n) - R(T_n)}{\mid \tilde{T}_n \mid -1} \tag{2.3.2.11}$$

Then a node n having the minimum value of the function over all internal nodes of a tree T is called the weakest link in tree T. That is, the weakest link can be represented by :

$$g_l(\tilde{n}) = \min_{n \in T} g_l(n) \tag{2.3.2.12}$$

The rationale of weakest link is follows. The complexity cost, α, is the cost of one extra leaf in the tree. The total cost of a subtree rooted at node n is :

$$R(T_n) + \alpha \mid \tilde{T}_n \mid \tag{2.3.2.13}$$

The total cost of node n is, if the sub-tree rooted at node n is pruned :

$$R(n) + \alpha \tag{2.3.2.14}$$

The two previous equations are equal when :

$$R(T_n) + \alpha \mid \tilde{T}_n \mid = R(n) + \alpha \tag{2.3.2.15}$$

This leads to :

$$\alpha = \frac{R(n) - R(T_n)}{\mid \tilde{T}_n - 1 \mid} \tag{2.3.2.16}$$

Suppose we have a list of complexity costs $\alpha_1 \leq \cdots \leq \alpha_s$ for a tree, where each complexity cost α_i of node i is such that it makes the cost of node i equivalent to the cost of its subtree. By the definition of weakest link, the node having the smallest complexity cost α_1 is chosen as weakest link. Suppose only node i has α_1 as complexity cost. Since α_1 is the smallest, in the internal nodes except node i,

$$R(T_n) + \alpha \mid \tilde{T}_n \mid < R(n) + \alpha \tag{2.3.2.17}$$

where $n \neq i$ and n is internal node of tree. In conclusion, if we take a smallest complexity cost and apply it to all internal nodes to compare the cost of node and cost of its subtree, then when the smallest complexity cost is unique, there exist only one internal node of tree which satisfying the cost of node is less than the cost of its subtree.

2.3.4.2. *Reduced Error Pruning*

This method is rather straightforward and considers complexity cost and misclassification rate as two crucial factors in deciding pruning. The reduced error pruning method uses a test set rather than a training set to obtain the misclassification rate during the pruning process.

For every non-terminal (non-leaf) subtree S of T, the misclassified (error) rate over the test set is examined using the same method as was used in the cost complexity method. If a subtree rooted at node n is replaced by a node n, and the new tree either keeps the same misclassification number or improves its misclassification rate, then the new tree is taken for improvement of the misclassified rate or for smaller size of tree with the same misclassification number. This procedure continues until no further improvement in the size of the tree or in the misclassification rate has been achieved. As with cost complexity pruning, the method generates a sequence of trees. In addition, the final tree is the smallest subtree with the most accuracy of the original tree over the test set.

2.3.4.3. *Pessimistic Pruning*

As long as the training set is used to measure the misclassification ratio of a tree which was trained with it, since the generated tree has been tailored to the training set, the error ratio does not provide a reliable estimate when unseen cases are classified.

Consider a subtree T_n rooted at node n and define K as the number of data items falling into node n and J as the number of data items misclassified on the terminal nodes under T_n. For example, the ratio of misclassification for the subtree T_n is considered as $\frac{J+\frac{1}{2}}{K}$ instead of $\frac{J}{K}$. In [**32**], Quinlan suggests a more pessimistic view of error rates of misclassification. Suggesting that the misclassified error rate for unseen cases is $\frac{J+\frac{|\tilde{T}_n|}{2}}{K}$. Suppose E is the number of data items misclassified from the training set. The pessimistic pruning method prunes subtree T_n, rooted at node n, if $E + \frac{1}{2}$ is within the range of standard error of

$J + \frac{|\tilde{T}_n|}{2}$. When we compare the two misclassified ratios of a node n and a subtree rooted at node n, and take into consideration the standard error, we find that there are more chances to replace a subtree by a node. Since Quinlan's method examines all non-leaf subtrees once and doesn't consider pruned subtrees, the pruning process is relatively fast.

Gelfand *et al.* [**43**] proposed a simple, cheap pruning method. As the method proposed by Breiman *et al.* , first of all, $R(n)$ is calculated for each node n of tree T. However, the values obtained for $R(n)$ on the same tree with two pruning methods are totally different. The reason is that in CART's method, $R(n)$ is calculated with the same data set as used when tree-growing is done whereas in Gelfand's pruning method, the different data set is used when $R(n)$ is calculated. The Gelfand *et al.* pruning algorithm is described as follows :

```
1. For any node $n$ of the tree, set $R(n)$
2. For each terminal node $n$ of tree, set $S(n) = R(n)$
3. For each internal node $n$ of tree,
      a. set $S(n) = S(left(n)) + S(right(n))$
      b. if $R(n) \leq S(n)$ then
            prune the subtree consisting of descendants of node $n$
            set $S(n) = R(n)$.
```

The value of $S(n)$ on a node n is to represent total number of data items misclassified by a subtree rooted at a node n. Thus, while scanning whole nodes of tree from bottom to root, prune any subtree rooted at any node n when the ratio of misclassification based on a subtree rooted at node n is equal to or greater than the ratio of misclassification of node n.

2.3.5. Class-Selection Rules

Assigning a class to each terminal node seems generally to be quite easy. In most cases, a majority rule is used to decide a class from several candidates. In other words, to select a class for a terminal node, select the class having the most cases. In case of tie with two cases, we can apply tie-breaking rule. The rule is to choose the class of the parent if it's one of the classes with the most items. Otherwise, break the tie arbitrarily.

2.3.6. Comparison of Methods

Because there are so many splitting/pruning rules and so many different applications of them, there have been a large number of empirically comparative studies for determining which method is the most effective in constructing 'good' trees. We review here some of these studies.

2.3.6.1. *Comparison of Splitting Rules*

So far, most studies have shown that there is little difference between the splitting rules. In [**21**], Breiman *et al.* conjectured that obtaining a good tree is not dependent on the choice of splitting rule; rather, that the stopping/pruning rules are more crucial in tree construction. In [**26**], Mingers also concluded that the selection of a splitting rule is not important to the quality of the tree; however, some studies have indicated that a specific rule is superior to the others in certain applications. Fayyad [**5**] indicated that C-SEP criterion does better than Gini criterion and information gain for some specific application.

Kononenko *et al.* [**12**] pointed out that information gain tends to favor attributes with a large number of possible values and tried to solve the problem by requiring that all tests have only two outcomes. Quinlan [**31**] suggested information gain ratio as a remedy. The information gain criterion selects the attribute that maximizes the ratio from among attributes with an average or better information gain.

For more details about such comparisons, see [**28**] and [**42**].

2.3.6.2. *Comparison of Pruning Rules*

In [**26**] and [**32**], there are some comparisons of each pruning method. J. Mingers shows some experimental results with 5 pruning methods (Error-Complexity Pruning; Critical Value Pruning; Minimum-Error Pruning; Reduced-Error Pruning; Pessimistic Error Pruning) which were applied to six distinct domains. These tests showed that Error-Complexity Pruning and Reduced-Error Pruning were the most accurate, while Minimum-Error Pruning and Pessimistic Pruning were the least accurate.

In [**32**], however, Quinlan arrives to a completely different conclusion. Experimenting with four different pruning rules (cost-complexity pruning, reduced-error pruning, pessimistic pruning, production rule form) that were applied to six distinct domains which were not the same as those in [**26**], results show that the performance of pessimistic pruning is marginally better than cost-complexity pruning averaged over all domains.

As some empirical comparisons show, there is no single method that is superior to the other methods over all domains.

2.3.6.3. *Correlation of Splitting and Pruning Rules*

In [**26**], Mingers attempts to find a relationship between splitting and pruning rules. To accomplish this, he selected 4 different split methods: G-statistic, G-statistic with Marshall's correction, χ-square distribution, and gain-ratio; and five pruning rules: critical, minimum-error, error-complexity, pessimistic pruning, and reduced-error. He obtained empirical results on the rate of misclassification by applying trees trained by using 4 different split methods associated with each pruning method to each of six domains. Finally, he asserted that there is no evidence that the splitting method has relation to pruning method.

CHAPTER 3

Machine Learning of Semantic Interpretation rules

3.1. Introduction

There are currently three main approaches for building the interpretation module for speech understanding systems. They are characterized by the way in which interpretation knowledge is obtained. In a machine learning approach, knowledge is acquired automatically by computers and used for building statistical models. Parameters of statistical models are learned from an annotated corpus. In the linguistically-based approach, knowledge is made of hand-coded rules. A hybrid approach is a combination that overcomes some weak points of each approach, for example, adding some rules to the statistical model in order to handle events which seldom occur in a corpus.

Some of the most important Speech Understanding Systems(SUS) [**17, 18, 22, 41, 25, 36, 35**] developed and evaluated on the ATIS(Air Travel Information System) projects are :

- The CHANEL System of CRIM(Centre de Recherche Informatique de Montréal), which is a hybrid system with a charted bottom-up parser and a SCTs(Semantic Classification Trees)-based robust matcher.
- The CHRONUS System of AT&T, which is another hybrid system consisting of a stochastical conceptual model whose parameters are automatically trained, and a component with manually written rules.
- The DELPHI System of BBN, which is made up of a chart-based unification parser and a fallback module with extended grammatic/pragmatic rules.
- The HUM System of BBN, which is a hybrid system with a conceptual Hidden Markov Models(HMM) and hand coded rules.
- The SRI(Stanford Research Institute) GEMINI System, which is a unification-based parser of syntactic and semantic rules. The SRI 'Template Matcher' is a template-based system with slots filled by a pattern-matching mechanism.
- The PHOENIX System of CMU(Carnegie Mellon University), which is a template-based system like SRI's Template Matcher but with a slightly different scoring mechanism: to fill a slot, the grammar related to the slot must be satisfied.
- The TINA System of MIT (Massachusetts Institute of Technology), which consists of a global syntactic parser and a robust matcher. TINA's hand-coded rules are part of a probabilistic context-free grammar.

CHANEL, CHRONUS, and HUM are corpus-based systems based on machine learning techniques, whereas the other systems are linguistically-based and entirely dependent

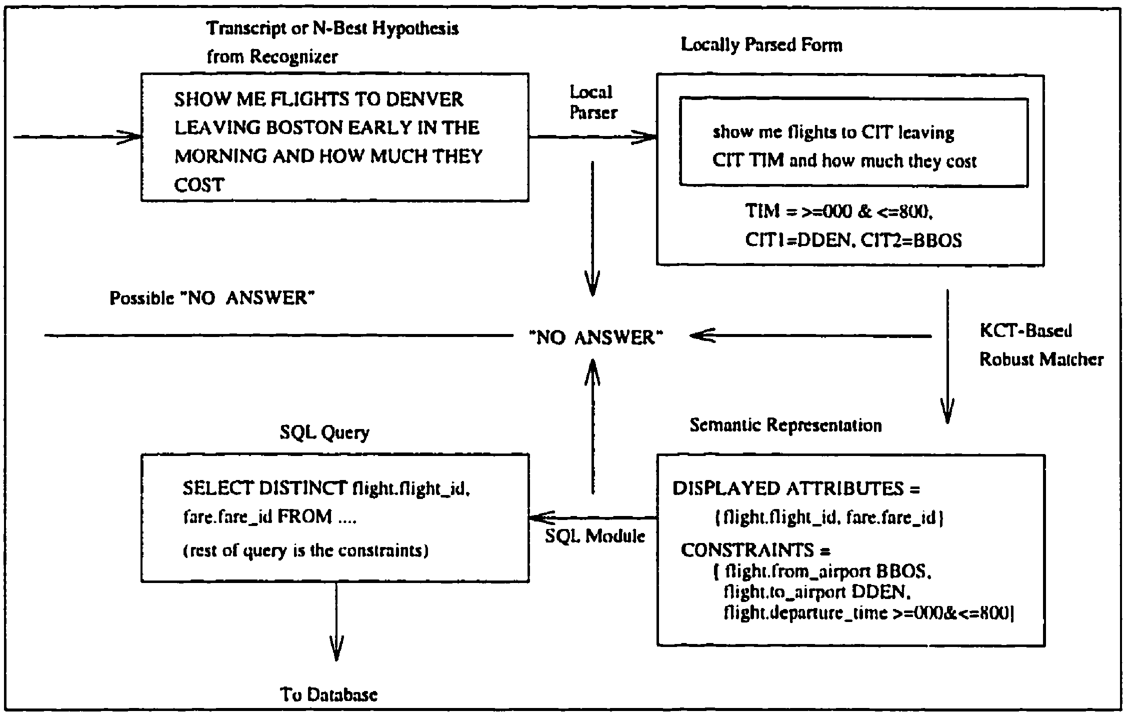

Transcript or N-Best Hypothesis from Recognizer
SHOW ME FLIGHTS TO DENVER LEAVING BOSTON EARLY IN THE MORNING AND HOW MUCH THEY COST
Local Parser
Locally Parsed Form
show me flights to CIT leaving CIT TIM and how much they cost
TIM = >=000 & <=800, CIT1=DDEN, CIT2=BBOS
Possible "NO ANSWER"
"NO ANSWER"
KCT-Based Robust Matcher
Semantic Representation
SQL Query
SELECT DISTINCT flight.flight_id, fare.fare_id FROM ....
(rest of query is the constraints)
SQL Module
DISPLAYED ATTRIBUTES = {flight.flight_id, fare.fare_id}
CONSTRAINTS = {flight.from_airport BBOS, flight.to_airport DDEN, flight.departure_time >=000&<=800}
To Database

FIGURE 3.1. Overall structure of the CHANEL System [17]

on manually written rules [19]. Systems incorporating machine learning techniques are discussed in detail below.

3.2. The CHANEL System

3.2.1. System Overview

CHANEL(CRIM Hybrid Analyzer for Natural Language) is a hybrid linguistic analyzer for the ATIS domain. As shown in Fig. 3.1, the system is composed of two main modules: the local parser and the SCT(Semantic Classification Tree)-based robust matcher [1] for the recognition of the semantic content conveyed by a sentence.

[1]Dr.R. Kuhn used 'KCT(Keyword Classification Tree)' in his Ph.D thesis [17]; however, the 'SCT(Semantic Classification Tree)' is used here to follow his example in a recent paper [18].

Much like a conventional approach to understanding speech, the local bottom-up parser preprocesses a string of words and labels semantic phrase components. The SCT-based robust matcher, which consists of a forest of SCTs, is the core of CHANEL. Each SCT in the robust matching module is trained with methods described in Chapter 2. For more detail, see also [18]. The advantages of using automatically constructed trees over hand-coded rules are to enhance robustness in presence of non-linguistic phenomena, recognizer errors or spontaneous speech phenomena like false starts, abrupt changes of subject, ungrammaticalities, etc.

An example of sentence processing through each module is shown inside of each box in Fig. 3.1. In the locally-parsed form (the output of the Local Parser), city names(DENVER, BOSTON) and a phrase(EARLY IN THE MORNING) representing a time are replaced by word-categories such as CIT and TIM. Although some phrases of a sentence have been replaced by word-categories, the true value of each category is stored and passed to the next process. The semantic representation emerging from SCT-Based Robust-Matcher is very similar to its SQL Query. Each element listed in DISPLAYED ATTRIBUTES section represents a table name and a column name of a database. The list of column names and the corresponding values listed in the CONSTRAINTS section describes constraints applied to corresponding columns in DISPLAYED ATTIRIBUTES.

3.2.2. Description of Components

3.2.2.1. *The Local Chart Parser*

The parser component is based on the lexical grammar formalism and some DCG rules for defining categories having a linear structure.

A lexical grammar has two components: [25]

(i) A lexicon(word or word category) is linked to its linguistic knowledge represented by a structure describing its syntactic and semantic characteristics.

(ii) A set of combining rules describes the general mechanism for combining structures.

The parsing algorithm is applied in the following three steps:

(i) For each word or sequence of words in a sentence, all possible interpretations are considered by the predefined lexical grammars for a lexicon. For instance, the phrase 'before two in the afternoon' is interpreted both as 'before(A) number(2) time(afternoon)' and 'before(A) number(2) pm(A)'. Since the predefined rules of two lexical items, *time* and *pm*, are satisfied with 'in the afternoon' at the same time, two possible translations are considered, usually, time(afternoon) and pm(A).

(ii) The next step is a bottom-up combination process using the pre-described combining rules. Obviously, 'before(A) number(2) pm(A)' is selected instead of 'before(A) number(2) time(afternoon)' to have 'before(1400)' because the numbers following 'before' should be interpreted as a time. Thus, the interpretation of 'number pm' is correct semantically.

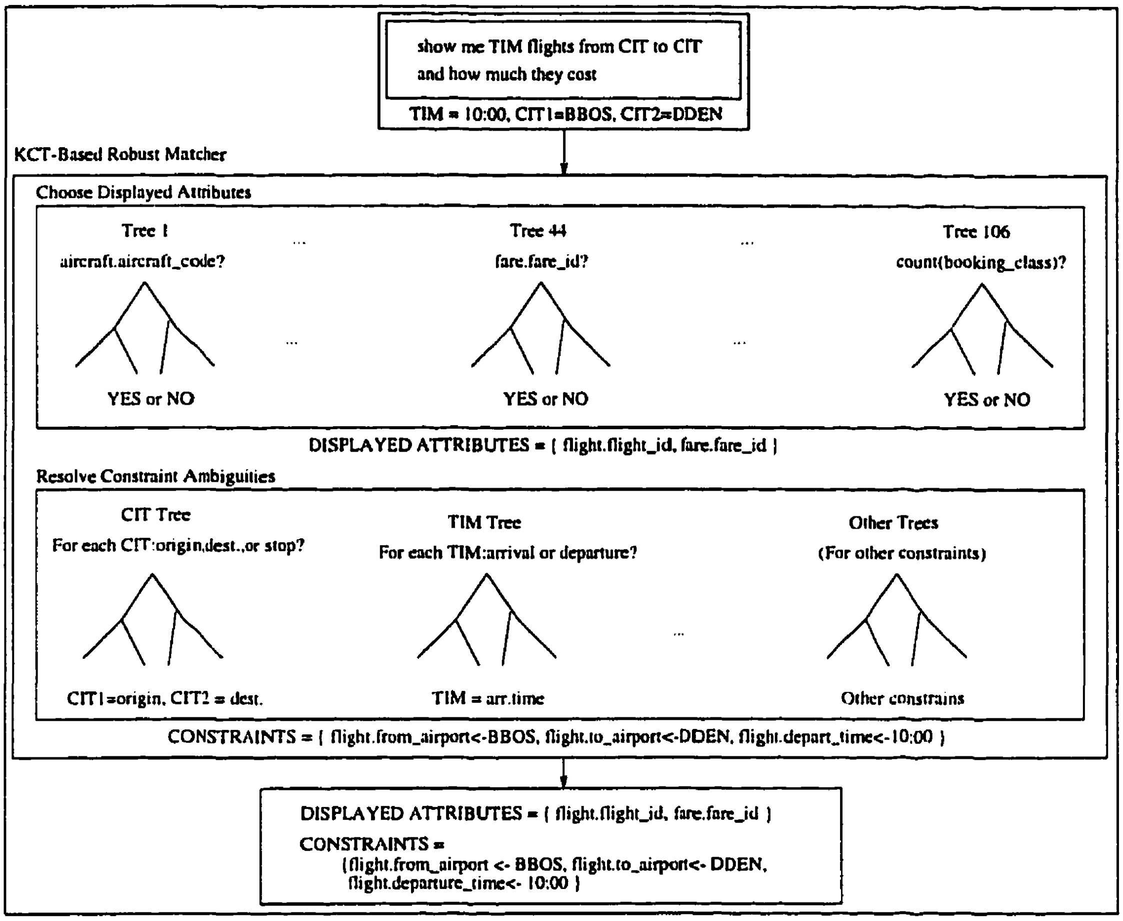

show me TIM flights from CIT to CIT
and how much they cost
TIM = 10:00, CIT1=BBOS, CIT2=DDEN
KCT-Based Robust Matcher
Choose Displayed Attributes
Tree 1
aircraft.aircraft_code?
Tree 44
fare.fare_id?
Tree 106
count(booking_class)?
YES or NO
YES or NO
YES or NO
DISPLAYED ATTRIBUTES = { flight.flight_id, fare.fare_id }
Resolve Constraint Ambiguities
CIT Tree
For each CIT:origin,dest.,or stop?
TIM Tree
For each TIM:arrival or departure?
Other Trees
(For other constraints)
CIT1=origin, CIT2 = dest.
TIM = arr.time
Other constrains
CONSTRAINTS = { flight.from_airport<-BBOS, flight.to_airport<-DDEN, flight.depart_time<-10:00 }
DISPLAYED ATTRIBUTES = { flight.flight_id, fare.fare_id }
CONSTRAINTS =
{flight.from_airport <- BBOS, flight.to_airport<- DDEN,
flight.departure_time<- 10:00 }

FIGURE 3.2. The SCT-Based Robust Matcher [17]

(iii) The last step is to find necessary semantic data in an entire sentence. For 'flight_airl(421, UA)' as a result of second step, the semantic data('AIR=UA' and 'FNB=421') are found.

3.2.2.2. *The SCT-Based Robust Matcher*

Figure 3.2 shows the structure of the robust matcher in the CHANEL System. It is composed of a large number of trees, each of which carries an appropriate semantic representation.

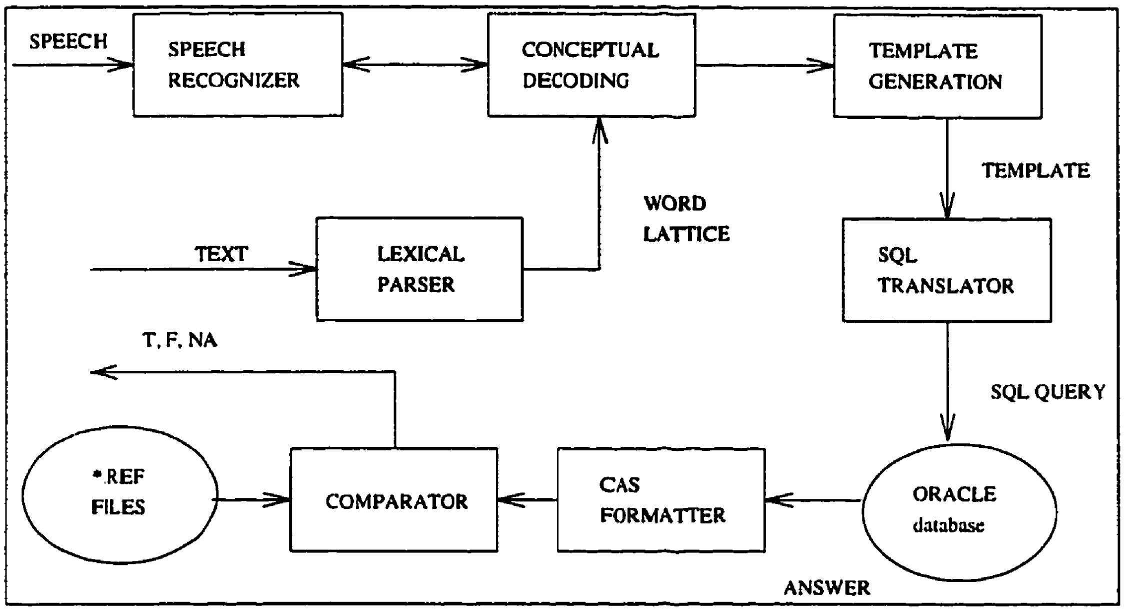


FIGURE 3.3. Overall structure of the CHRONUS System [36]

The SCTs are divided into two groups : 'displayed attributes' and 'local constraints'. The functions of the two types of trees are slightly different. A tree in the 'displayed attributes' group conveys semantic rules for a certain column of a database table. For instance, if the result of a sentence on the 'aircraft.aircraft_code' tree turns out to be 'YES', the sentence is considered to have an expression of 'aircraft code'.

A tree in the 'local constraints' group uses rules for a table column and its value. For example, if the result of the CIT tree turns out to be 'origin', then the 'city name' phrase is counted as the city name for 'departure'.

3.3. The CHRONUS System

3.3.1. System Overview

0	1	I	14	15	$PREFER(S)$
1	2	$WOULD$	15	16	TO
2	3	$LIKE(S)$	16	17	$FL(Y\|IES)$
3	4	TO	17	18	ON
4	5	$GO(ES)$	18	20	$[A](< aircraft\_make > BOEING)$
5	6	$FROM$	18	23	$[A](< aircraft > 74M)$
6	7	NEW	19	20	$(< aircraft\_make > BOEING)$
6	8	$(< city > NNYC)$	19	23	$(< aircraft > 74M)$
6	8	$(< state > NY)$	20	21	$(< numbers > 7)$
8	9	TO	20	22	$(< numbers > 740)$
9	10	SAN	20	23	$(< numbers > 747)$
9	11	$(< city > SSFO)$	21	22	$(< numbers > 40)$
11	12	$(< day\_name > SATURDAY)$	21	23	$(< numbers > 47)$
12	13	$MORNING(S)$	22	23	$(< numbers > 7)$
13	14	I			

TABLE 3.3.3.1. Example of a Lattice Structure [**22**]

Fig. 3.3 shows the structure of the CHRONUS system. The *Speech Recognizer* produces a string of word hypotheses based on acoustic signals. The top hypothetic word string produced by the *speech recognizer* is not directly connected to the *Conceptual Decoding*. The *Lexical Parser* preprocesses numbers, acronyms, and compound words from the string and generates a lattice structure containing interpretations of the string. For example, the substring "B SEVEN FOUR SEVEN" could be interpreted as "B 747" or "B7 47" or "B74

wish:	I WOULD LIKE TO GO
origin:	FROM NEW YORK
destin:	TO SAN FRANCISCO
day:	SATURDAY
time:	MORNING
aircraft:	I PREFER TO FLY ON A BOEING SEVEN FORTY SEVEN

TABLE 3.3.3.2. Example of a Conceptual Segmentation [22]

AIRLINE:	UA
ORIGIN_CITY:	NNYC
DESTINATION_CITY:	SSFO
WEEKDAY:	SATURDAY
ORIGIN_TIME:	0≺1200
AIRCRAFT:	74M
SUBJECT:	FLIGHT

TABLE 3.3.3.3. Example of a Template [22]

7", etc by the *Lexical Parser* [36]. Table 3.3.3.1 shows the lattice structure for the following sentence:

I WOULD LIKE TO GO FROM NEW YORK TO SAN FRANCISCO SATURDAY MORNING PREFER TO FLY ON A BOEING SEVEN FORTY SEVEN

The *Conceptual Decoding* produces a semantic representation, called a *conceptual segmentation*, of the input strings using the Viterbi algorithm. Table 3.3.3.2 shows an example of conceptual segmentation for the lattice in table 3.3.3.1.

The *Template Generation* transforms a conceptual segmentation form into an SQL-like form. Table 3.3.3.3 is the template for the conceptual segmentation of table 3.3.3.2.

The *SQL Translator* generates an SQL query and extracts relevant information from Oracle database. Also, the *comparator* module compares an result of the system with a reference answer provided by a training set.

3.3.2. Description of Components

3.3.2.1. *Conceptual Decoding*

This module plays the main role in extracting semantic units from a sequence of words. The task is accomplished by giving a *concept label* to each phrase of a sentence using statistical techniques, which we will describe next.

The ideal goal of this module is to find the sequence of words W and the concepts C which will maximize the conditional probability of W and C given the acoustic signal A:

$$\max_{W\times C} P(W,C|A) \tag{3.3.3.1}$$

The right-hand side of Equation 3.3.3.1 can be rewritten using the Bayes rule as:

$$\max_{W\times C} P(W,C|A) = \max_{W\times C} \frac{P(A|W,C)P(W|C)P(C)}{P(A)} \tag{3.3.3.2}$$

Based on the reasonable assumption that the acoustic model of a word is independent of the concept of a word, Equation 3.3.3.2 is equivalent to :

$$\max_{W \times C} P(W, C|A) = \max_{W \times C} \frac{P(A|W)P(W|C)P(C)}{P(A)} \tag{3.3.3.3}$$

Obviously, due to the fixed value of $P(A)$, Equation 3.3.3.3 can be rewritten as :

$$\max_{W \times C} P(W, C|A) = \max_{W \times C} P(A|W)P(W|C)P(C) \tag{3.3.3.4}$$

In the CHRONUS System, $P(A|W)$ is implemented with HMMs(Hidden Markov Model) of phonetic sub-word units. For the remaining terms,

$$P(W|C)P(C) = \prod_{i=2}^{M} P(w_i|w_{i-1}...w_1, C)P(w_1|C) \prod_{i=2}^{M} P(c_i|c_{i-1}...c_1)P(c_1) \tag{3.3.3.5}$$

For approximation of Equation 3.3.3.5, AT&T assumes that :

$$P(w_i|w_{i-1}...w_1, C) = P(w_i|w_{i-1}...w_{1-n}, c_i) \tag{3.3.3.6}$$

and

$$P(c_i|c_{i-1}...c_1) = P(c_i|c_{i-1}...c_{1-m}). \tag{3.3.3.7}$$

In addition, a bigram language model is used to approximate Equations 3.3.3.6 and 3.3.3.7. Finally, Equation 3.3.3.4 can be rewritten as :

$$\max_{W \times C} P(W, C|A) = \max_{W \times C} \prod_{i=2}^{M} P(w_i|w_{i-1}, ci) \cdot \prod_{i=2}^{M} P(c_i|c_{i-1}). \qquad (3.3.3.8)$$

The implementation is made with HMMs whose states represent *concept relations* and whose observation probabilities lie in the bigram form of words. For the training HMM, a corpus of training sentences are hand-coded. The parameters of HMM are learned from the corpus.

3.3.2.2. *Lexical Parser*

The *Lexical Parser* preprocesses the hypotheses produced by the *Speech Recognizer*, and transforms them into a lattice structure. This module applies some morphological/syntactic/semantic rules on the top hypotheses and reorganizes the lattice into a list of *word classes*, each of which is a base form for possible morphological/syntactic/semantic variants of a word. There are rules to transform each word as a form of *word class* [**35**]:

(i) Articles are generally associated with the word that follows (e.g. 'THE FLIGHT' is transformed into '[THE]FLIGHT, etc.).

(ii) Words with morphological variants are grouped together (e.g. 'GO', 'GOES', 'GOING' are represented by a super-word 'GO(ES)(ING)', etc.).

(iii) Some compound phrases are converted into hyphenated phrases (e.g. ONE WAY becomes ONE-WAY, etc.).

(iv) Acronyms and numbers are dealt with by regular grammars (e.g. TWA, USAIR, etc.).

(v) semantically meaningful words are grouped together, like city names. (e.g. SAN FRANCISCO, DALLAS FORT WORTH, etc.).

(vi) For a given concept, words can be grouped together according to their uses in the phrases. For example, for the concept ORIGIN the words DEPART, LEAVE, ARRIVE can be considered to have the same concept in the circumstances of the following sentences:

(a) THE FLIGHT THAT DEPART(S) FROM DALLAS

(b) THE FLIGHT THAT LEAVE(S) FROM DALLAS

(c) THE FLIGHT THAT ARRIVE(S) FROM DALLAS

As shown in Table 3.3.3.1, the *lexical parser* organizes multiple hypotheses into a lattice structure. It defers the decision of choosing the most possible hypotheses until the next step, *conceptual decoding*.

3.3.2.3. *Template Generator and SQL Translator*

The *Template Generator* consists of a set of hand-coded rules. It simply translates conceptual segmentations into pairs of key-words and values according to the rules. A pair made up of a key-word and a value correspond to a pair made up of an attribute and an entity in the database.

The *SQL Translator* is composed of a set of hand-coded rules like the Template Generator. It produces an SQL database-query to extract appropriate data reflecting template information.

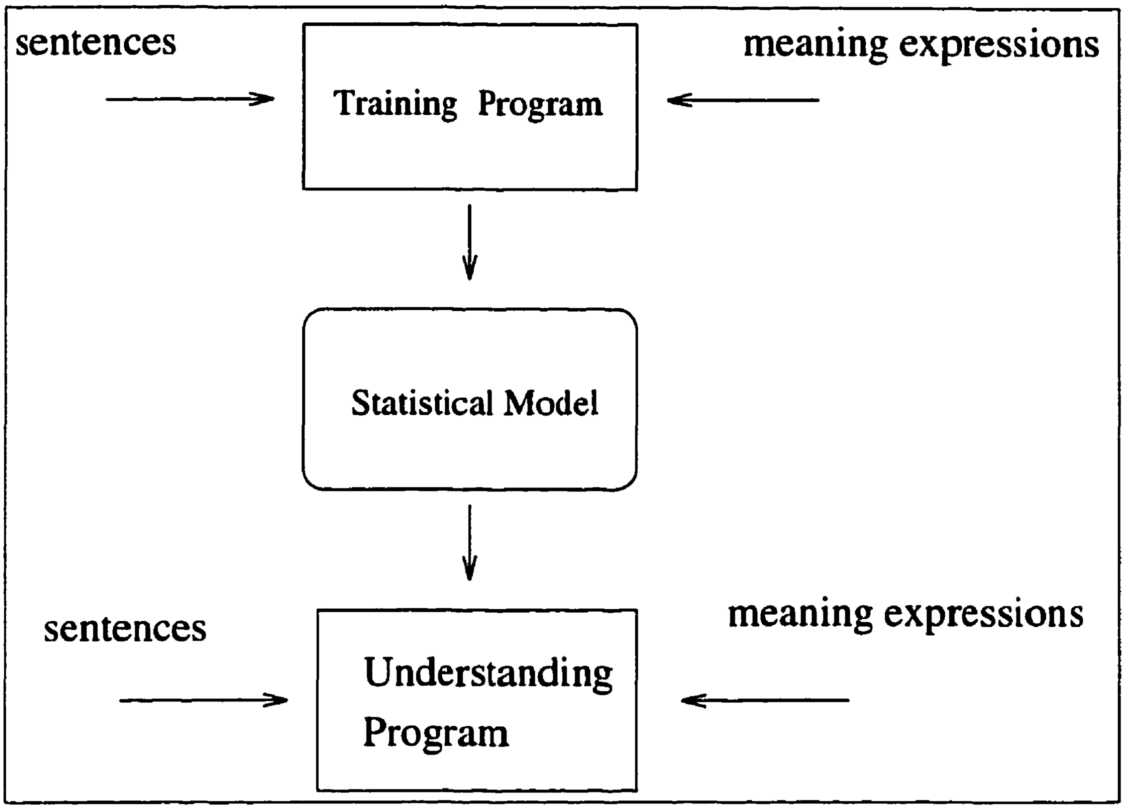


FIGURE 3.4. Overall structure of HUM System [41]

3.4. The HUM System

3.4.1. System Overview

The overall structure of the HUM System is shown in Figure 3.4. A 'Training Program' is used to estimate the parameters of the statistical model of this system. The statistical model trained by the 'Training Program' is used for extracting the meaning of expressions in a sentence. The 'Understanding Program' finds the most likely meanings supposed by a word sequence by using the statistical model. The Hidden Understanding Model, the statistical model of this system, was motivated, as the name implies, from HMM(Hidden Markov Models) [**41**].

A representative tool, ***tree-structured meaning representations***, was used to express meaning. This tool is based on a tree structure in which each internal node is either an *individual concept* or a *component concept.* A *component concept* is a sub-concept of its parent node, which is an *individual concept.* Each terminal node is directly connected to a word or a string of words.

3.4.2. Description of Components

The HUM System recognizes the semantics hidden in a sentence W as a meaning M such that $P(M|W)$ is maximized. By Bayes Rule,

$$P(M|W) = \frac{P(W|M)P(M)}{P(W)} \tag{3.4.3.1}$$

Since $P(W)$ is fixed, $P(M|W)$ can be achieved by maximizing the product of $P(W|M)$ and $P(M)$.

$$P(M|W) = P(W|M)P(M) \tag{3.4.3.2}$$

The statistical model of the HUM System, shown in Figure 3.5, is made up of a *semantic language model* and a *lexical realization model.* $P(M)$ in Equation 3.4.3.2 is implemented with the *semantic language model,* and $P(W|M)$ by means of the *lexical realization model.* The combination of the two models is equivalent to finding the most likely meaning M given a word sequence W.

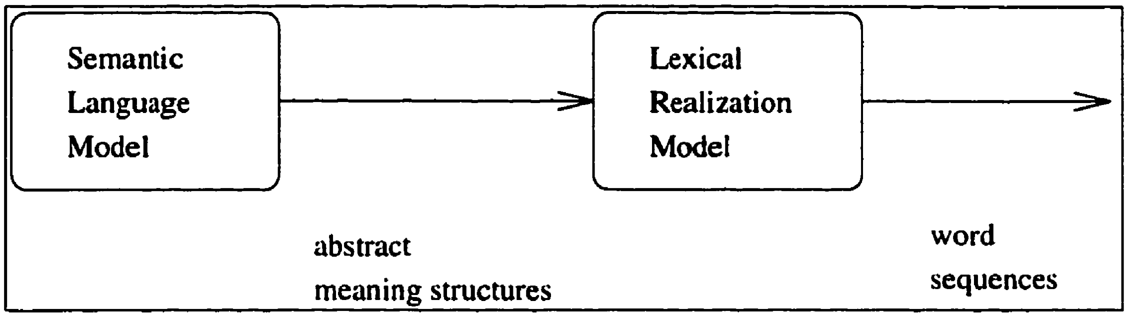


FIGURE 3.5. Statistical Model of HUM System [41]

3.4.2.1. *Semantic Language Model*

In the semantic language model, *a probabilistic state transition network* is constructed to implement an abstract concept. All sub-concepts of an *abstract concept* consist of states of a network representing the abstract concept. Two more states, 'enter' and 'exit', are added to indicate the entry and exit points. A network is a complete directed graph except the two added states cause the addition of directed paths from the 'enter' state to other sub-concept states and from each sub-concept state to the 'exit' state. Each arc in a network is associated with a probability. For example, the arc from *airline* to *date* on a *flight* network has a probability $P(date|airline, fight)$, which represents the probability of going into the *date* state from the *airline* state on the *flight* network.

The semantic language model is a network of combination of all *probabilistic state transition networks*, or a *probabilistic recursive transition network.*

3.4.2.2. *Lexical Model*

From the point of view of structure, the *Lexical Realization Model* is the same as the *Semantic Language Model* except that the terminal node of the *tree structure meaning* *representation* is used as a state in the lexical model. There are two virtual states to indicate the first word and the last, '*begin*' and '*end*'. The value on an arc is estimated with a probability for a transition from one word to another given a particular context. Thus, $P(please|*begin*, show\_indicator)$ is the probability that *please* is the first word of a *show_indicator* phrase, and $P(*end*|me, show\_indicator)$ is the probability that *me* is the last word of a *show_indicator* phrase.

3.4.2.3. *Training and Understanding Components*

The main concern in the training procedure is how to estimate the transition probabilities of the semantic language model and the lexical realization model. In the HUM System [**41**], the estimates of the probabilities are given by the following for the semantic language model:

$$\hat{\mathrm{P}}(state_n|state_m, context) = \frac{C(state_n|state_m, context)}{C(state_m, context)}, \qquad (3.4.3.3)$$

and the estimate for the lexical realization model is given by the following:

$$\hat{\mathrm{P}}(word_n|word_m, context) = \frac{C(word_n|word_m, context)}{C(word_m, context)} \qquad (3.4.3.4)$$

For more robust estimates, Equations 3.4.3.3 and 3.4.3.4 are smoothed with $\hat{\mathrm{P}}(state_n|context)$ and $\hat{\mathrm{P}}(word_n|context)$ [**41**].

The issue in understanding components is how to find the meaning of a given string of words such that $P(W|M)P(M)$ is maximized. As mentioned earlier, $P(W|M)P(M)$ is the probability of a path through the combined network of two probabilistic networks. If the whole network is searched to find a maximized path, the algorithm would take exponential time to length of sentence. The search time is reduced by applying dynamic programming and the Viterbi algorithm [**41**].

CHAPTER 4

Speech Understanding System (SUS) at McGill

4.1. System Overview

The speech understanding system at McGill university is developed on Air Travel Information System (ATIS) domain sponsored by DARPA. The main task of the speech understanding system is to extract semantics from the output sentence of a speech recognizer. As other hybrid systems like CHANEL, HUM, and CHRONUS, stated in the previous chapter, this system is also a hybrid system incorporating linguistic-based approach at a local level and corpus-based approach at a global level.

The structure of the Speech Understanding System is shown in Fig. 4.1. From the structure point of view, it consists of two modules :

- *RTN(Recursive Transition Network)-based parser* for a local-level lexical/ syntactical/ semantical/ analysis (local in the sense that adjacent consecutive words are examined).
- *CT(Classification Tree)-based Robust Matcher* for a global-level semantic analysis (global in the sense that whole sentence is examined).

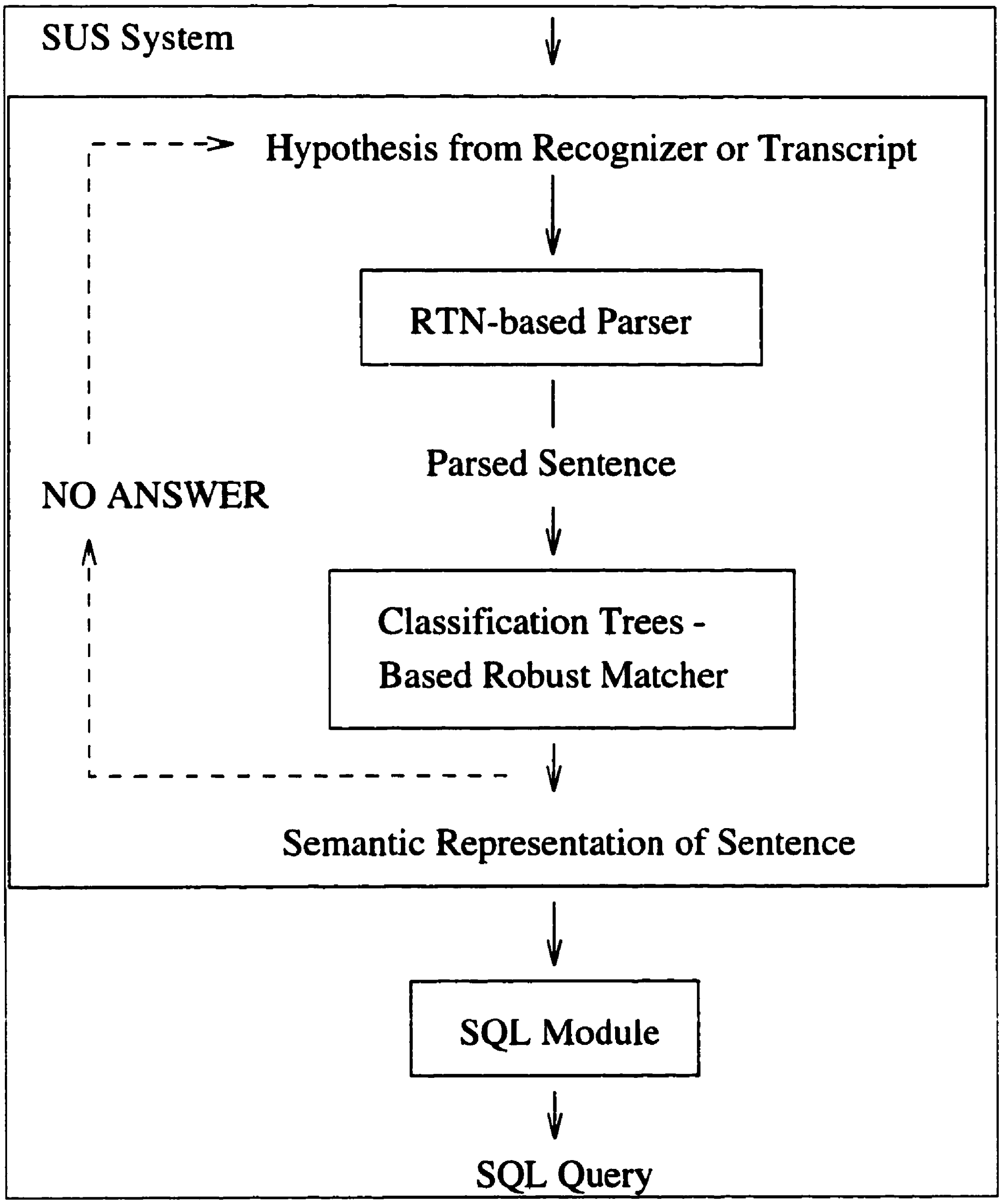

SUS System
Hypothesis from Recognizer or Transcript
RTN-based Parser
Parsed Sentence
NO ANSWER
Classification Trees - Based Robust Matcher
Semantic Representation of Sentence
SQL Module
SQL Query

FIGURE 4.1. Overall structure of Speech Understanding System in McGill

In local-level analysis, the RTN-based top-down parser converts a string of words into a hierarchical structure containing word categories that correspond to semantic constituents. In global-level analysis, a trained classification tree functions as a semantic extractor. A classification tree is a data structure which learns semantic rules from annotated training data. A trained classification tree represents semantic rules of a concept as a form of question in each node. This system generates output similar to SQL query, which is called *intermediate codes*. The intermediate codes are translated into a SQL query by SQL module at the next step. Instead of intermediate codes, "NO ANSWER" output could be generated if there is no semantics to be detected, due to either a failure of system or a sentence without the meanings the system is looking for.

4.2. The Local Parser Module

The local parser is based on RTNs (Recursive Transition Network), each of which codes a semantic structure such as FLIGHT-NUMBER, TIME, etc. Rules representing a semantic concept are coded in a RTN. The parsing process proceeds in a way that if the final state of a RTN is reached by a sentence, the part of sentence read through the RTN is replaced with a symbol indicating the RTN.

A triple of data enclosed in parenthesis is used to represent the output of a local parser. A triple data contains the word, a symbol representing its category and its value. All tripled data are stored in a queue and the local parser proceeds with the queue. Examples of this representation are shown on tables 4.2.4.1 and 4.2.4.2. Structure of local parsers is shown on figure 4.2.

(please	UNDEF	???	)
(list	SHOW_INDICATOR	1	)
(the	UNDEF	???	)
(eastern	AIRLINE-NAME	EA	)
(flight	UNDEF	???	)
(two	NUM	2	)
(ten	NUM	10	)
(from	UNDEF	???	)
(d-f-w	AIRPORT-NAME	DFW	)
(to	UNDEF	???	)
(san-francisco	CITY-NAME	SFO	)
(on	UNDEF	???	)
(the	UNDEF	???	)
(june	MONTH	6	)
(twenty	NUM	20	)
(second	ORD	2	)
(between	UNDEF	???	)
(three	NUM	3	)
(and	UNDEF	???	)
(seven	NUM	7	)
(p-m	TIME_PERIOD	19	)
(NULL	NULL	NULL	)

TABLE 4.2.4.1. Example of Intermediate Form

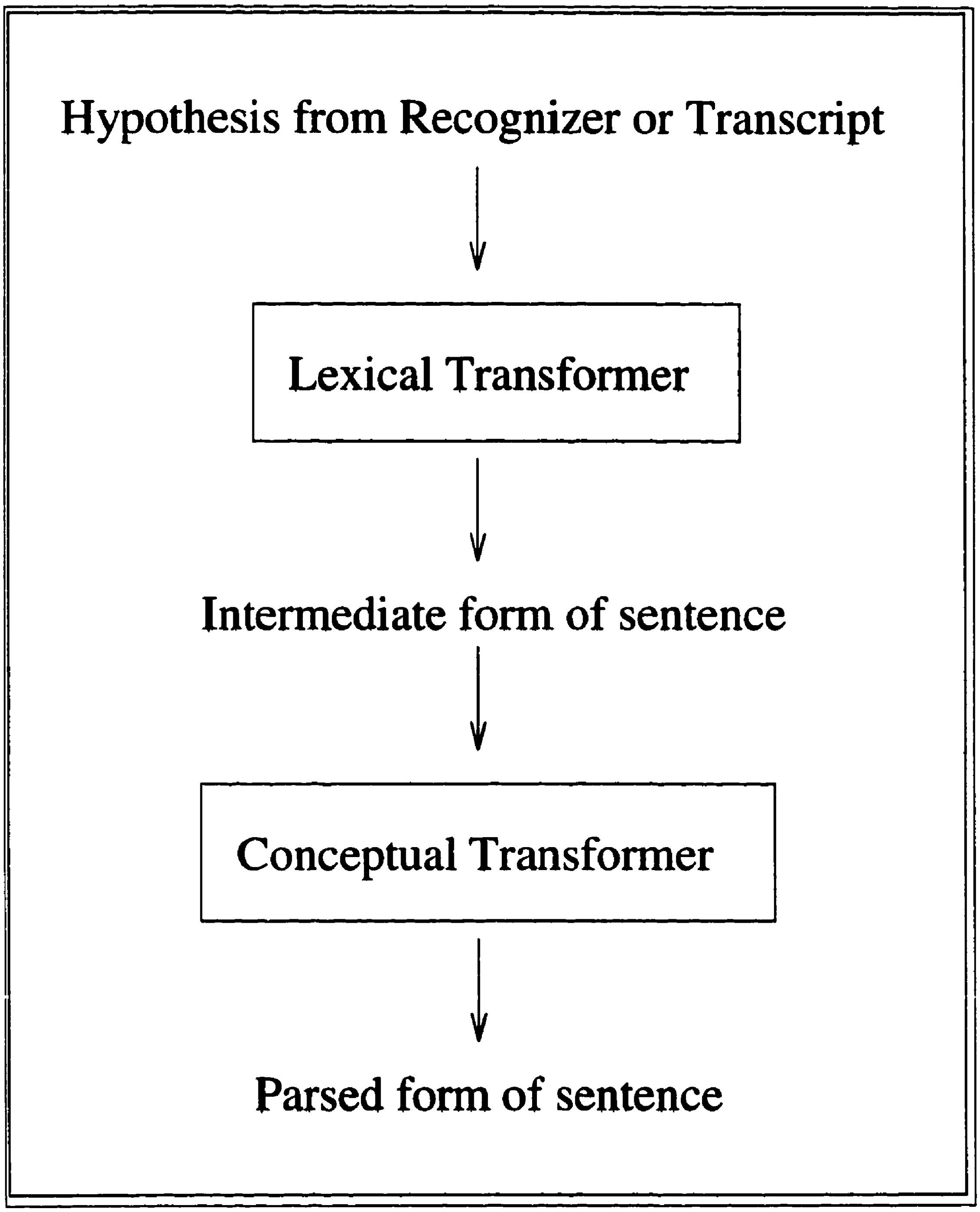

Hypothesis from Recognizer or Transcript
Lexical Transformer
Intermediate form of sentence
Conceptual Transformer
Parsed form of sentence

FIGURE 4.2. Overall structure of Local Parser

(please	UNDEF	???	)
(list	SHOW_INDICATOR	1	)
(the	UNDEF	???	)
(AIRLINE-NAME	EA	eastern	)
(FLIGHT-NUMBER	210	*	)
(from	UNDEF	???	)
(AIRPORT-NAME	DFW	d-f-w	)
(to	UNDEF	???	)
(CITY-NAME	SFO	san-francisco	)
(on	UNDEF	???	)
(the	UNDEF	???	)
(DATE-MONTH	6	june	)
(DATE-DAY	22	*	)
(TIME	*	$\geq$15:0&&$\leq$19:0	)
(NULL	NULL	NULL	)

TABLE 4.2.4.2. Example of Parsed Form

Local parser module has two components : Lexical Transformer and Conceptual Transformer. The Lexical Transformer module concatenates pre-defined words to treat them as one word and looks up individual words from a sentence in the dictionary, and then extracts its category and value from the dictionary. The Conceptual Transformer module precedes syntactic and semantic parsings with a large RTN combining all RTNs. A RTN represents semantic rules for a semantic concept like FLIGHT-NUMBER, TIME. A sentence is read through each RTN to see if the final node of the RTN can be reached by the sentence, which means that the sentence satisfies a RTN so that it has the semantics coded in the RTN. If a sentence turns out to contain the semantics which a RTN supports, the conceptual transformer module replaces the phrase of input string comprising the semantics with the symbol representing the semantics.

For instance, consider a sentence "please list the eastern flight two ten from d f w to san francisco on the june twenty second between three and seven p m". The *intermediate form* and the *parsed form* of the sentence are displayed on table 4.2.4.1 and table 4.2.4.2, respectively. Following the Lexical Transformer step, it would be "please SHOW_INDICATOR the AIRLINE-NAME flight NUM NUM from AIRPORT-NAME to CITY-NAME on the MONTH NUM ORD between NUM and NUM TIME_PERIOD". It is labelled as follows: "please SHOW_INDICATOR(list) the AIRLINE-NAME(eastern) flight NUM(two) NUM(ten) from AIRPORT-NAME(d-f-w) to CITY-NAME(san-francisco) on the MONTH(june) NUM(twenty) ORD(second) between NUM(three) and NUM(seven) TIME_PERIOD(p-m)". According to the Conceptual Transformer step, it would be "please SHOW_INDICATOR the AIRLINE-NAME FLIGHT-NUMBER from AIRPORT-NAME to CITY-NAME on the DATE-MONTH DATE-DAY TIME". Thus, it is labelled as follows: "please SHOW_INDICATOR (1:list) the AIRLINE-NAME (EA:eastern) FLIGHT-NUMBER (*:210) from AIRPORT-NAME (DFW:d-f-w) to CITY-NAME (SFO:san-francisco) on the DATE-MONTH (6:june) DATE-DAY (22:*) TIME (>=15:0&&<=19:0:*)". The string before ':' inside parenthesis is the value of a concept and the one after ':' is the original string of a concept. However, if the original string consists of more than two words, it is replaced with '*'.

In addition, the adoption of the symbol SHOW_INDICATOR is not directly related to the issue of semantics but related to the issue of handling training data. For instance, after parsing procedure, two sentences "please show the eastern flight two ten from d f w to san francisco" and "please list the eastern flight two ten from d f w to san francisco" have the same parsed form, since each of two different words, 'show' and 'list', is changed to the same symbol, SHOW_INDICATOR. If the two sentences of the exactly same semantic and syntactic structures have the same parsed forms, it could help the training mechanism learn such a structure quickly. This point could be important in a situation where there is a lack of training data.

4.3. Building Classification Trees

Recall that three elements in construction of classification tree, stated in Chapter 2, are:

- A set Q of questions
- A rule for selecting the best split at any node
- A criterion for choosing the right-sized tree

Getting the best split at a node is equivalent to choosing the best question from a set of questions for a node. The Gini criterion described in Chapter 2 to select the best question for a node is used for this thesis. Also, the iterative expansion-pruning algorithm described in Chapter 2 to get a right-sized tree is adopted.

A set of questions now is closely related to the application of a speech understanding system. The node structure for a question was originally devised at CRIM. Two structures are applied on each node : *known structure* and *decision structure.* Two structures are of the same form. Each structure is a regular expression consisting of *symbol* and *gaps.* The four regular expressions for each gap + in a *known structure* are considered with a symbol w :

- $< w >$: single symbol w.
- $< +w >$: sequence of length of at least two ending with symbol w.
- $< w+ >$: sequence of length of at least two beginning with symbol w.
- $< +w+ >$: sequence of length of at least three containing symbol w that is neither the first nor the last.

The *known structure* for the root of a classification tree is defined as $< + >$. The operation rules on a *known structure* are to apply each of four regular expressions to each gap + in the *known structure* with a given symbol. If there are L symbols in the lexicon, there are (4 x L) operations applicable to each gap + in *known structure.*

When a regular expression at a node is selected according to the Gini criterion, the *decision structure* of the node is determined by applying selected regular expression to the *known structure* of the node. For instance, for a node with *known structure,* $< +w >$, four possibilities for the *decision structure* given a symbol, s, are $< sw >$, $< s+w >$, $< +sw >$, and $< +s+w >$. This is obtained by replacing the only gap + in $< +w >$ with four gap operations, $< s >$, $< s+ >$, $< +s >$, and $< +s+ >$. The *known structure* of "YES child" node is the *decision structure* of its parent node, while the *known structure* of "NO child" is identical to the *known structure* of its parent.

4.4. The CT-based Robust Matcher Module

Figure 4.3 shows the structure of CT-based robust matcher. The parsed sentence preparsed by *local parser* is submitted into the *robust matcher*. The *robust matcher* is composed of two parts (*Displayed Attributes Module* and *Constraints Module*), which correspond to two parts of *semantic representation* (*Displayed Attributes* and *Constraints*). The ultimate purpose of this system is to find out *something* satisfying *some conditions*. Attribute corresponds to *something* and constraint corresponds to *something with conditions*. The *Displayed Attributes Module* generates 'list of attributes', while the *Constraints Module* produces 'list of constraints'. An attribute from 'list of attributes' is related to a name concatenating table(relation) and column(attribute) in ATIS database, while a constraints from 'list of constraints' is composed of an attribute and its value. List of constraints has two types of constraints : *Local Constraints* and *Global Constraints*.

4.4.1. Displayed Attributes Module

The SQL translation of class A[1] sentence of the ATIS2 training data has the structure of "SELECT DISTINCT *list of attrubutes* FROM ...". There are 74 different attributes in SQL code of ATIS2. The *flight.flight_id* is the most common attribute appearing in 2308

[1]class A sentence is semantically independent, while class D sentence is semantically dependent on previous sentence.

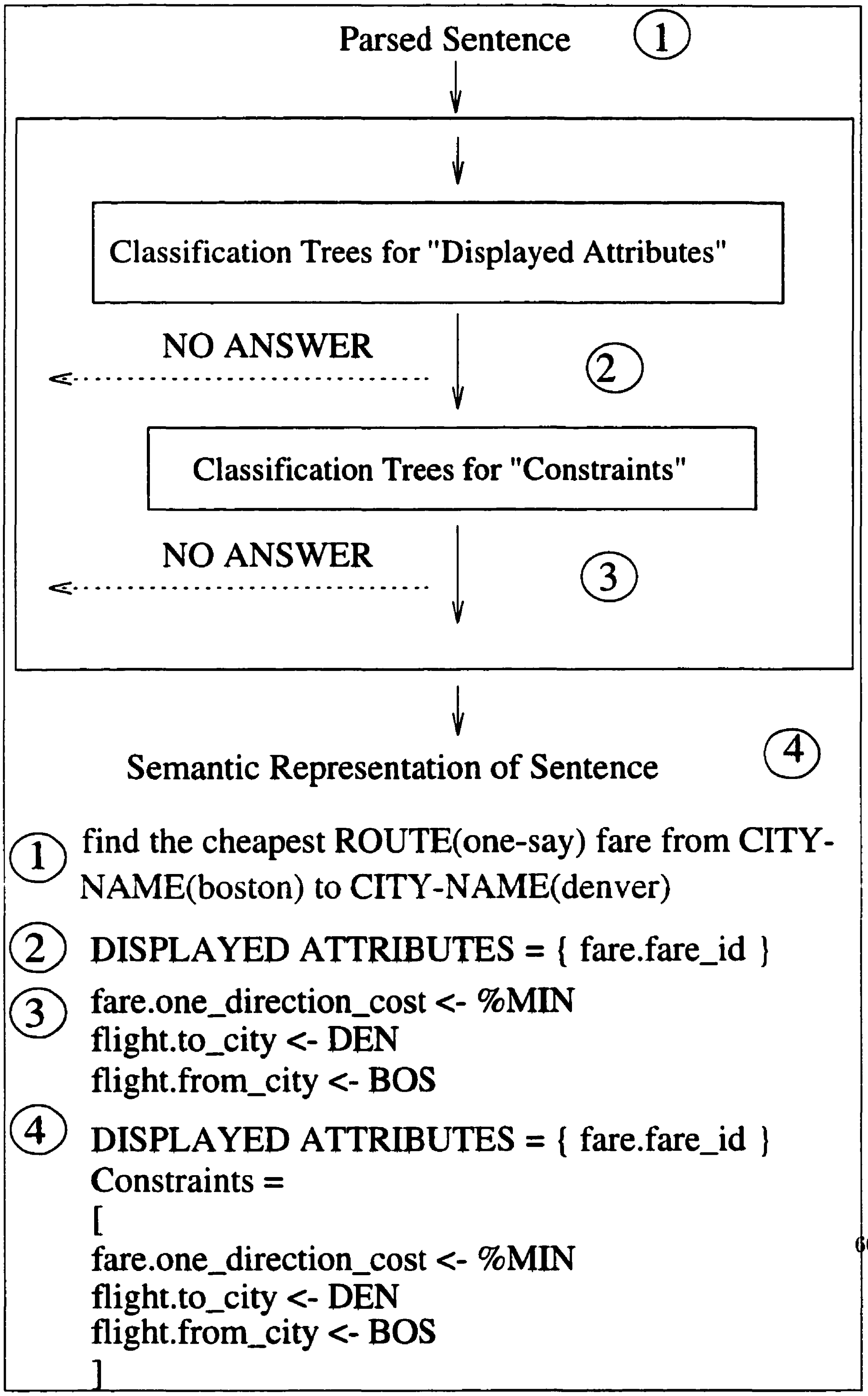

Parsed Sentence
1
Classification Trees for "Displayed Attributes"
NO ANSWER
2
Classification Trees for "Constraints"
NO ANSWER
3
Semantic Representation of Sentence
4
1 find the cheapest ROUTE(one-say) fare from CITY-NAME(boston) to CITY-NAME(denver)
2 DISPLAYED ATTRIBUTES = { fare.fare_id }
3 fare.one_direction_cost <- %MIN
flight.to_city <- DEN
flight.from_city <- BOS
4 DISPLAYED ATTRIBUTES = { fare.fare_id }
Constraints =
[
fare.one_direction_cost <- %MIN
flight.to_city <- DEN
flight.from_city <- BOS
]

FIGURE 4.3. Overall structure of Robust Matcher

please show all flights from CITY-NAME to AIRPORT-NAME DAY ⇒ 0
please show flights arriving in CITY-NAME from CITY-NAME ⇒ 0
please show me again the CLASS-NAME fares from CITY-NAME to CITY-NAME ⇒ 0
please tell me the times of the flights between CITY-NAME and CITY-NAME ⇒ 1
show me the costs and times for flights from CITY-NAME to CITY-NAME ⇒ 1

TABLE 4.4.4.1. Parsed Form of Some Training Data for *flight.departure_time* Attribute

out of 3102 sentences of class A. Only 34 out of 74 attributes appear in at least 10 sentences and 16 other attributes appear only once. A function call, MIN(flight.departure_time), is found in one sentence.

Some different attributes are highly correlated and some always appear together in SQL queries. In the latter case, the different attributes have identical classification tree. For example, *airport.airport_location, airport.country_name, airport.minimum_connecct_time, airport.state_code,* and *airport.time_zone_code* attributes are displayed in only two sentences.

Some training data for *flight.departure_time* attribute are shown in table 4.4.4.1.

The number '0' or '1' shown at the end of a sentence indicates whether or not the SQL query of the sentence contains the *flight.departure_time* attribute between 'SELECT DISTINCT' and 'FROM'. Thus, the sentence labelled '1' has *flight.departure_time* attribute in its SQL query and the one labelled '0' does not. All the labelled training data are involved in a training of a classification tree. Thus, the training data for the attribute *flight.flight_id* has 2308 sentences labelled '1' and 794 sentences labelled '0'. The iterative expansion-pruning algorithm we adopt requires to maintain two training data sets. Each data set is kept to have approximately equal numbers of labelled sentences without duplication of data for each class. In the training data set for attributes having only one sentence labelled '1', however, the same sentence labelled '1' appears in each data set.

4.4.2. Constraints Module

4.4.2.1. *Global Constraints*

Recall that a constraint in the semantic representation is composed of an attribute and its value, or a range of possible values. Properties of a global constraint are spread out over a sentence instead of being localized. Unlike local constraint, it does not require to be pre-processed by a local parser.

In the current version of system, 13 global constraint classification trees were trained :

- *MAX and MIN for an one-way fare and a round-trip fare*

  A classification tree is trained for a global constraint. All sentences containing 'MIN (fare.one_direction_cost)' in their SQL queries are collected for training data. The 'cheapest', 'least expensive', 'lowest cost', 'lowest price', or 'lowest fare' phrase appears on sentences satisfying the condition.

- *MAX and MIN for the capacity of aircraft*

  Only 2 sentences contains the phrase of maximum capacity of aircraft. 'largest seating capacity' and 'greatest seating capacity' are the corresponding parts of the sentences. 'smallest seating capacity', 'smallest plane', or 'smallest number of passengers' appears on the sentences for the minimum of aircraft capacity.

- *MAX and MIN for departure time of flight*

For the maximum of departure time, the proper sentences contains 'last flight', 'latest flight', etc. For the minimum, 'first flight', 'earliest flight', etc are the examples.

- *MIN for arrival time of flight*

  Only 1 sentence, 'what flight from boston to atlanta arrives earliest in atlanta', satisfies this criterion.

- *nonstop of flight*

  'flight.stops = 0' condition appears on the SQL queries of the training data for this constraint. 'please list all flights between boston and san francisco nonstop' and 'what nonstop flights between boston and washington arrive after five o'clock p m' are the exemples.

- *flight with meal*

  The training data for this constraint include two types of sentences. In the first type of sentence, a meal name is not specified, but the property of this constraint is defined explicitly. That is, phrases like 'with a meal' or 'serving a meal' are found in the sentences. In the second type of sentence, a meal name is concretely specified, such as '**breakfast** served' or 'serve **dinner**'.

- *'direct' and 'connect' of flight connection*

  In SQL translations of the training data, 'flight.connections = 0' is found for *direct flight* constraint, while 'flight.connections > 0' is found for *indirect flight* constraint.

- *airline code and flight number of flight*

  A classification tree for this constraint does not exist, since the attribute and value for this constraint are already found in local parser module.

what flights are available from %CITY-NAME to CITY-NAME	⇒ 1
what flights are available from CITY-NAME to %CITY-NAME	⇒ 0

TABLE 4.4.4.2. Example of Training Data for 'city:origin' constraint

what flights are available from %CITY-NAME to CITY-NAME	⇒ 0
what flights are available from CITY-NAME to %CITY-NAME	⇒ 1

TABLE 4.4.4.3. Example of Training Data for 'city:destination' constraint

4.4.2.2. *Local Constraints*

After a sentence is parsed by a local parser, semantics of some parsed substrings are left unresolved. Meanings of such an ambiguous substring are identified by classification trees trained for local constraints. Consider a sentence 'what flights are available from denver to baltimore'. The parsed form of the sentence is 'what flights are available from CITY-NAME to CITY-NAME' where the value of the first CITY-NAME is 'denver' and the value of the second is 'baltimore'. As we can see, CITY-NAME after 'from' is the origin city and CITY-NAME after 'to' is the destination city. The semantics of unidentified substrings are revealed in local constraints module.

In the training data for a local constraint, data duplication is unavoidable. As shown in table 4.4.4.2, the two almost same sentences appear with different labelling. In table 4.4.4.3, the two sentences appear again with the reversed labelling according to dealing with the reversed role of constraint. We distinguish the same symbols being considered, by adding '%' on the symbol. Thus, labelling of a sentence is decided by the role of the symbol preceded by '%'.

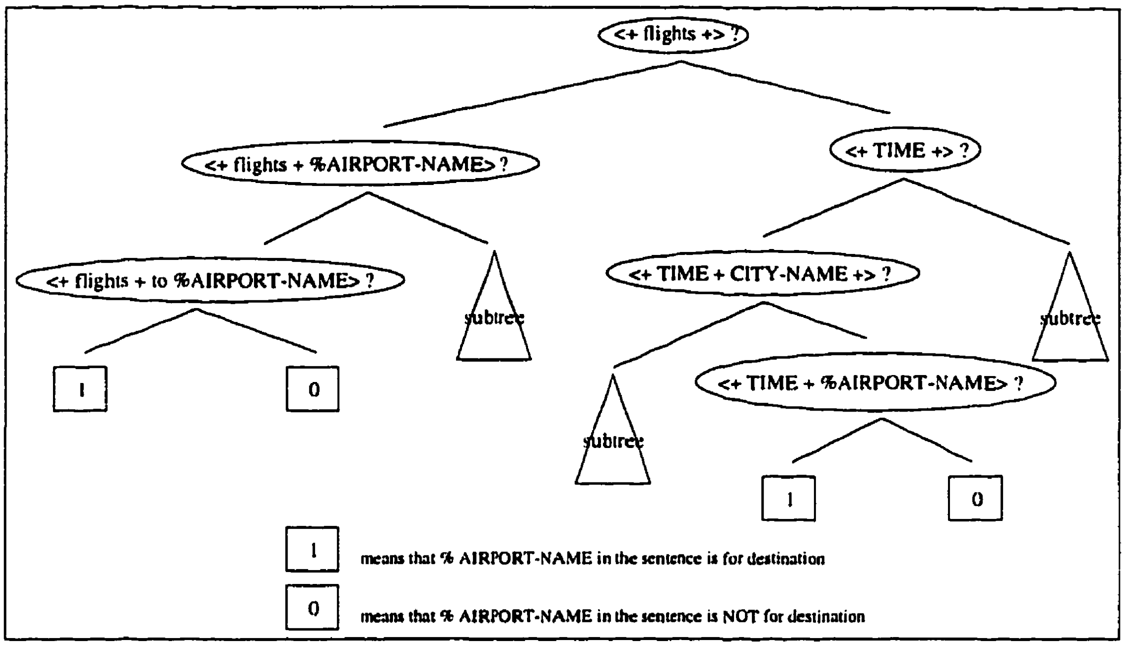

<+ flights +> ?
<+ flights + %AIRPORT-NAME> ?
<+ TIME +> ?
<+ flights + to %AIRPORT-NAME> ?
subtree
<+ TIME + CITY-NAME +> ?
subtree
1
0
<+ TIME + %AIRPORT-NAME> ?
subtree
1
0
1 means that % AIRPORT-NAME in the sentence is for destination
0 means that % AIRPORT-NAME in the sentence is NOT for destination

FIGURE 4.4. CT for the local constraint *destination of airport name*

Here are the list of 12 local constraint CTs implemented in the current version of system.

- *for AIRPORT NAME*

  An airport name can be an origin, a destination, a stopover, a site served by an airline, or a location for ground transportation.

- *for CITY NAME*

  A city name can be an origin, a destination, a stopover, a site served by an airline, or a location for ground transportation.

- *for TIME*

  A time can be an arrival time or a departure time.

Figure 4.4 shows a CT that decides if an airport name is for destination.

```
Recognized Sentence ⇒ are there any flights from san francisco to boston leaving
in the afternoon and arriving between three and seven p m
DISPLAYED ATTRIBUTES = { flight.flight_id }
Constraints =
[
flight.to_city ← BOSS
flight.from_city ← SFOO
flight.arrival_time ← >=15:0&&<=19:0
]
```

TABLE 4.5.4.1. Example 1 of A Semantic Representation

4.5. The Semantic Representation

The semantic representation is the intermediate code that represents the information carried by a sentence, which will be converted to a corresponding SQL query. The semantic representation we adopt here is based on the one devised at CRIM and used for CHANEL system, and is modified for our purpose.

The semantic representation has two parts, *attributes* and *constraints*. *Attribute* is of the "relation.attribute" form and *constraint* is of the "relation.attribute ← value" form. '&&' and '||' are logical operators indicating the AND and OR, respectively. '>=', '<=', '<', and '>' are relation operators. '%ZERO', '%MIN', and '%MAX' are special symbols. For example, '%MIN' indicates a minimum value and '%MAX' is a maximum value.

```
Recognized Sentence ⇒ i'm looking for a one way flight from boston to baltimore
washington what is the cheapest flight
DISPLAYED ATTRIBUTES = { flight.flight_id }
Constraints =
[
fare.one_direction_cost ← %MIN
flight.to_city ← WSH
flight.from_city ← BOS
]
```

TABLE 4.5.4.2. Example 2 of A Semantic Representation

Some examples of semantic representation are shown in tables 4.5.4.1 and 4.5.4.2. For the interpretation of 'baltimore washington' in table 4.5.4.2, we follow the principles of interpretation provided by ATIS2 database.

4.6. Speech Understanding Systems at McGill and CRIM

SUS at McGill and CHANEL are hybrid systems consisting of two components, a parser and a robust matcher, as the other SUSs developed under ATIS domain. Two SUSs have many similarities in that the parsers are linguistic-based and the robust matchers are corpus-based. Furthermore, both SUSs adopt Classification Trees to perform semantic extraction. However, two systems take different parsers and similar robust matchers in performing similar work. Differences of two SUSs are specified as follows.

- The parser at McGill is based on Recursive Transition Networks (RTN) formalism which analyzes a sentence locally and finds its semantic values from important words, whereas in CHANEL, the parser relies on lexical grammar formalism with DCG rules to detect semantic structures.
- Two systems have similar semantic representation language. McGill system adopts the one used in CHANEL which has been modified for our purpose.
- The robust matcher of SUS at McGill have 13 classification trees for global constraints. Some global constraints do not need to be implemented with classification trees because parser module collects enough information to determine such constraints. Determination of having some global constraints or not, is done during parsing process.
- In McGill's system, a classification tree represents only one semantic concept and determines if a sentence has a semantic concept represented by the tree or not. That is, one semantic concept is exactly corresponding to a classification tree. Therefore each classification tree is independently trained/re-trained. In CHANEL, a classification tree is trained for one or more semantics. It can save time in training and reduce the number of trees.
- McGill's SUS is implemented with C++ for the parser and robust matcher and PERL for the interconnection of two modules, whereas CHANEL is written in C for robust matcher and Lisp for parser.

CHAPTER 5

Results and Discussion

5.1. Experiments and Analysis

The single-symbol CTs for the SUS developed at McGill were trained on 3248 class A ATIS2 NL sentences. The test results, which will be shown later, were from 399 class A Feb92 ATIS2 NL sentences and on 441 class A Nov92 ATIS2 NL sentences.

The SU system developed at McGill consists of three components : a parser, displayed Attributes (DA), and constraints. The parser part was not tested here. The remaining parts were tested and the results are analyzed in this chapter. Also, since the SQL module generating SQL codes from the semantic representation (the output of this SU system) is not prepared yet, the test for the SQL module is not dealt with here. The test results were obtained by comparing the semantic representations produced by this system and the SQL codes provided by ATIS2.

	Feb92	*Nov92*	*Total*
Number of successes	317	367	684(81%)
Number of fails	65	54	119(14%)
Number of no answers	17	20	37(4%)
Total	399	441	840(100%)

TABLE 5.1.5.1. ATIS2 Class A NL Test Result 1 on CTs of Displayed Attributes

5.1.1. Benchmark Results and Analysis

5.1.1.1. *CTs for Displayed Attributes*

The February and November 1992 ATIS2 benchmark results are shown in tables 5.1.5.1 - 5.1.5.5.

Table 5.1.5.1 shows the results for CTs of Displayed Attributes on ATIS2 Class A NL sentences. On the test data, a 'Success' occurs when the set of the displayed attributes of this system includes the answer prescribed by ATIS2. The other cases are considered to be 'Fails'.

For the test sentence 'which airlines depart from boston', the SQL codes of ATIS2 reads 'select distinct airline.airline_code from airline where ...'. The result of the system on the sentence is the following :

DISPLAYED ATTRIBUTES = { airline.airline_code }

Constraints =

{

flight.from_city ← BBOS

}

With the sentence 'are there any united flights from boston to san francisco stopping in denver', the SQL codes are 'select distinct flight.flight_id from flight where ...'. This system generates the output as follows :

```
DISPLAYED ATTRIBUTES = { flight.flight_id }
DISPLAYED ATTRIBUTES = { flight.airline_code }
Constraints =
{
flight.airline_code ← UA
flight.stopover_city ← DDEN
flight.to_city ← SSFO
flight.from_city ← BBOS
}
```

In the first example, two answers are matched exactly, whereas in the second case, the system output includes the ATIS2 answer.

The 'Success' cases in table 5.1.5.1 are divided into two case : *exact match* and *inclusive exact match*[1], shown in table 5.1.5.2.

[1]This is the case that an answer of the system includes ATIS2 answer.

	Feb92	*Nov92*	*Total*
Number of successes	193	241	434(52%)
Number of inclusive successes	124	126	250(30%)
Number of fails	65	54	119(14%)
Number of no answers	17	20	37(4%)
Total	399	441	840(100%)

TABLE 5.1.5.2. ATIS2 Class A NL Test Result 2 on CTs of Displayed Attributes

5.1.1.2. *CTs for Constraints*

As stated earlier, at the time of the development of this system, an SQL module for this system had not been prepared, so this system does not produce SQL code. Thus, the results for constraint CTs was obtained by comparing the output of constraint CTs with the corresponding SQL provided by ATIS, for each test sentence.

Tables 5.1.5.3 and 5.1.5.4 show two different results on CTs for constraints. Table 5.1.5.3 shows experimental results for entire test sentences, whereas the table 5.1.5.4 indicates results of the test sentences for which constraints are implemented by means of CTs. In particular, the test sentences containing constraints of which the corresponding CTs are not trained, have been excluded from the results shown in table 5.1.5.4. Some examples that are not considered for the test shown in table 5.1.5.4 follow : (The sentence components in italics indicate those which the constraint CTs of this system does not support.)

- list the first nonstop flight from boston to washington on *june twenty fifth.*
- how many t w a flights have *first class.*

	Feb92	*Nov92*	*Total*
Number of successes	144	166	310(37%)
Number of fails	238	255	493(59%)
Number of no answers	17	20	37(4%)
Total	399	441	840(100%)

TABLE 5.1.5.3. ATIS2 Class A NL Test Result 1 on Constraint CTs

	Feb92	*Nov92*	*Total*
Number of successes	144	166	310(72%)
Number of fails	71	15	86(20%)
Number of no answers	17	20	37(9%)
Total	232	201	433(100%)

TABLE 5.1.5.4. ATIS2 Class A NL Test Result 2 on Constraint CTs

- the cost of all flights from pittsburgh to boston on *wednesday of next week.*
- please give me flight information from denver *to pittsburgh to atlanta* and return to denver.

From the point of view of SU system performance, the results shown in table 5.1.5.3 are more interesting; however, in the viewpoint of how well the CTs-based approach works for an SU system in terms of CTs performance, the results shown in table 5.1.5.4 are more significant. The improvements in the previous two approaches will be discussed later

An analysis of the errors in semantic extraction using constraint CTs is shown in the table 5.1.5.5. Note that in table 5.1.5.3, the number of *Fails* is 493, whereas the total

	Feb92	*Nov92*	*Total*
Number of errors in global constraint CTs	47	27	74(11%)
Number of errors in local constraint CTs	65	108	173(26%)
Number of unimplemented errors	167	240	407(62%)
Total	279	375	654(100%)

TABLE 5.1.5.5. Error Analysis for the Constraint CTs

number of errors in table 5.1.5.5 is 654 because more than one error may occur in one sentence. 62% among total errors were occurred due to non-implementation of such CTs.

5.2. Improvements

The improvements will be discussed from two perspectives : system performance and CTs performance. For the sake of completeness, we will review general improvements to CT-based SU from [**17**].

(i) *System Performance Point of View*

Features which may improve system performance are listed below.

- Multiple Frames

  The handling of multiple frames is not implemented in this system. Most cases of multiple frame occurs in multiple departure cities(airports), multiple arrival cities(airports), or multiple stopover cities(airports). In CRIM's system, this feature was implemented but the results were unsatisfactory.

- Incompatibility between outputs of CTs

A large number of errors in the semantic representations generated occurs due to the incompatibility between DA and constraints since in the current version of the system, the output of each CT is entirely independent of the output of all other CTs. This, together with the fact that only single frames are handled, means that two different cities could be classified with the same semantics. Further research on the relationship between outputs of CTs is necessary.

- Unimplemented CTs in Global and Local Constraints (Class)

  The results with Feb. and Nov. 1992 test sentences reveal that this system needs more CTs to resolve certain constraints. These constraints and some examples of the corresponding phrases are given below :

  - *DATE*

    'august fifth', 'september fifth nineteen ninety one', 'week from wednesday', 'tomorrow', 'today', 'on wednesday of next week', 'next friday'.

  - *DAY*

    'on a wednesday'.

  - *CLASS*

    'first class', 'business class'.

  - *MEAL*

    'serve breakfast'.

  - *FARE*

    'coach fare', 'economy fare', 'coach fare'.

- *RENT*

  'rent a car'.

- *COST*

  'less than eight hundred and sixty dollars'.

In the current version of the system, the semantic CT for meal service was designed to indicate the existence of meal service, not to refer to a specific meal service.

(ii) *CTs Performance Point of View*

The largest number of errors occurred in the interpretation of TIME phrases associated with other semantic components : when a TIME phrase appears in an independent phrase, the TIME CTs captures the semantic correctly. However, when a TIME phrase is integrated with other constituents, such as 'monday afternoon'or 'wednesday night', the TIME CTs do not work properly.

General Improvements to CT-Based SU

- Uses a dialogue session for collecting training data
- Sets up meta-rules for determining that there is something wrong with a semantic representation
- Uses hierarchies of CTs
- Uses a CT-driven lexical search

5.3. Conclusion

This thesis describes a method for learning classification rules from training data using trees. The learning mechanism is based on the probabilistic approach. All patterns for a node of a CT are tried and one of them is selected for the pattern of the node. All selected patterns on a tree consist of classification rules. We applied this method to semantic extraction in speech understanding systems. Similar methods had been designed and applied to many application fields including speech recognition system, but this work is unique in its treatment of question patterns which are used for question of each node.

Finally, I would like to stress some points for using this method.

- *Training Data*

  Since this method is based on a probabilistic approach to training data, obtaining training data covering all possible cases is essential for successful results. Some DAs in training DA CTs have only a few sentence satisfying those DAs in ATIS. In this case, the method using hand-coded rules works better.

- *Training Process*

  During training, the process of selecting proper sentences from a large amount of training data takes a large portion of time. To speed up the development of the system, further research should be done to determine how to reduce the time taken by the training process (including the selection of the sentences that are needed).

- *Process of Obtaining Training Data*

After a system has been developed, there should be a process to collect training data. Once some training data have been acquired, the system may be re-set up with the training data collected so far.

REFERENCES

[1] M. Ben-Bassat. Use of distance measures, information measures and error bounds on feature evaluation. In P. R. Krishnaiah and L. N. Kanal, editors, *Classification, Pattern Recognition and Reduction of Dimensionality, volume 2 of Handbook of Statistics*. Springer-Verlag, Berlin, 1979.

[2] J.A. Bondy and U.S.R. Murty. *Graph Theory with Applications*. MacMillan Press, 1976.

[3] R.J. Brachman. On the epistemological status of semantic networks. In N.V. Findler, editor, *Associative Networks*. Academic Press, 1979.

[4] B.G. Buchanan and T.M. Mitchell. Model-directed learning of production rules. In *Pattern Directed Inference Systems*. Academic Press, 1978.

[5] U.M. Fayyad and K.B. Irani. Multi-interval discretization of continuous valued attributes for classification learning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1022–1027. IJCAI, Morgan Kaufmann Inc., 1993.

[6] E.A. Feigenbaum and H.A. Simon. Performance of a reading task by an elementary receiving and memorizing program. In *Behavioral Science*, volume 8. University of Michigan Press, 1963.

[7] R. Gray. Vector quantization. In A. Waibel and K.-F. Lee, editors, *Readings in Speech Recognition*. Morgan Kaufmann Inc., 1990.

[8] H.-W. Hon and K.-F. Lee. Recent progress in robust vocabulary-independent speech recognition. In *Proceedings of the DARPA Speech and Natural Language Workshop*, pages 258–263. DARPA, Morgan Kaufmann Inc., February 1991.

[9] E. Horowitz and S. Sahni. *Fundamentals of Data Structures in Pascal.* Computer Science Press, 1987.

[10] E.B. Hunt. *Concept Learning:An Information Processing Problem.* Wiley Co., 1962.

[11] L. Hyafil and R.L. Rivest. Constructing optimal binary decision tree is NP-complete. *Information Processing Letters*, 5(1):15–17, 1976.

[12] I. Kononenko, I. Bratko, and E. Roskar. Experiments in automatic learning of medical diagnostic rules. Technical report, Jozef Stefan Institute, Ljubljana, Yugoslavia, 1984.

[13] J. Makhoul, S. Roucos, and H. Gish. Vector quantization in speech coding. *Proc. IEEE*, 73:1551–1588, 1985.

[14] J.G. Carbonell, R.S. Michalski, and T.M. Mitchell. An overview of machine learning. In *Machine Learning:An Artificial Intelligence Approach.* Tiago Publishing Co., 1983.

[15] K. Oehler, E. Riskin, and R. Gray. Unbalanced tree-growing algorithms for practical image compression. In *Proceedings of the International Conference on Acoustics, Speech and Signal Proceesing*, pages 2293–2296. IEEE, 1991.

[16] B. Kim and D.A. Landgrebe. Hierarchical decision tree classifiers in high-dimensional and large class data. Technical Report TR-EE-90-47, Purdue University, 1990.

[17] R. Kuhn. *Keyword Classification Trees for Speech Understanding Systems.* PhD thesis, McGill University, 1993.

[18] R. Kuhn and R. De Mori. The application of semantic classification trees to natural language understanding. *IEEE Transactions on pattern analysis and machine intelligence*, 17(4), 1995.

[19] R. Kuhn and R. De Mori. Sentence interpretation. In *Spoken Dialogues with Computers.* Academic Press, 1997.

[20] L. Bahl, P. Brown, *et al.* A tree-based statistical language model for natural language speech recognition. In A. Waibel and K.-F. Lee, editors, *Readings in Speech Recognition.* Morgan Kaufmann Inc., 1990.

[21] L.B. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. *Classification and Regression Trees.* Wadsworth and Brooks, 1984.

[22] E. Levin and R. Pieraccini. Concept-based spontaneous speech understanding system. *Proceedings EUROSPEECH*, pages 555–558, 1995.

[23] Y.K. Lin and K.-S. Fu. Automatic classification of cervical cells using a binary tree classifier. *Pattern Recognition*, 16(1):69–80, 1983.

[24] L.R. Bahl, P.V. De Souza, *et al.* Decision trees for phonological rules in continuous speech. In *Proceedings of the International Conference on Acoustics, Speech and Signal Proceesing*, pages 185–188. IEEE, 1991.

[25] E. Millien and R. Kuhn. A robust analyzer for spoken language understanding. *Proceedings EUROSPEECH*, pages 1331–1334, 1993.

[26] J. Mingers. An empirical comparison of pruning methods for decision tree induction. *Machine Learning*, 4:227–243, 1989.

[27] R. De Mori, J. Fischer, *et al.* An integrated model of acoustics and language using semantic classification trees. In *Proceedings of the International Conference on Acoustics, Speech and Signal Proceesing*, volume 1, pages 419–422. IEEE, 1996.

[28] K.V.S. Murthy. *On Growing Better Decision Trees from Data.* PhD thesis, Johns Hopkins University, 1996.

[29] J.R. Quinlan. Learning efficient classification procedures. In *Machine Learning:An Artificial Intelligence Approach*, volume 3. Tiago Publishing Co., 1983.

[30] J.R. Quinlan. Learning from noisy data. In *Proceedings of the Second International Workshop on Machine Learning Workshop.* Morgan Kaufmann Inc., 1983.

[31] J.R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[32] J.R. Quinlan. Simplifying decision trees. *International Journal of Man-Machine Studies*, 27:221–234, 1987.

[33] J.R. Quinlan. Decision trees and multi-valued attributes. In J.E. Hayes, D. Michie, and J. Richards, editors, *Machine Intelligence*, 11. Oxford University Press, 1988.

[34] R. Kuhn, A. Lazarides, *et al.* Improved decision trees for phonetic modeling. In *Proceedings of the International Conference on Acoustics, Speech and Signal Proceesing*, pages 552–555. IEEE, 1995.

[35] R. Pieraccini, E. Tzoukermann, *et al.* A speech understanding system based on statistical representation of semantics. In *Proceedings of the International Conference on Acoustics, Speech and Signal Proceesing*, pages 193–196. IEEE, 1992.

[36] R. Pieraccini, E. Tzoukermann, *et al.* Progress report on the CHRONUS system: ATIS benchmark results. In *Proceedings of the DARPA Speech and Natural Language Workshop*, pages 67–71. DARPA, February 1992.

[37] E. Rich and K. Knight. *Artificial Intelligence*. McGraw-Hill, Inc, 1993.

[38] E. Riskin and R. Gray. Lookahead in growing tree-structured vector quantizers. In *Proceedings of the International Conference on Acoustics, Speech and Signal Proceesing*, pages 2289–2292. IEEE, 1991.

[39] E. Rounds. A combined non-parametric approach to feature selection and binary decision tree design. *Pattern Recognition*, 12:313–317, 1980.

[40] S. Miller, M. Bates, *et al.* Recent progress in hidden understanding models. *Proceedings of the Spoken Language Technology Workshop*, pages 22–25, 1995.

[41] S. Miller, R. Bobrow, R. Schwartz, and R. Ingria. Statistical language processing using hidden understanding models. *Proceedings of the Spoken Language Technology Workshop*, pages 48–52, 1994.

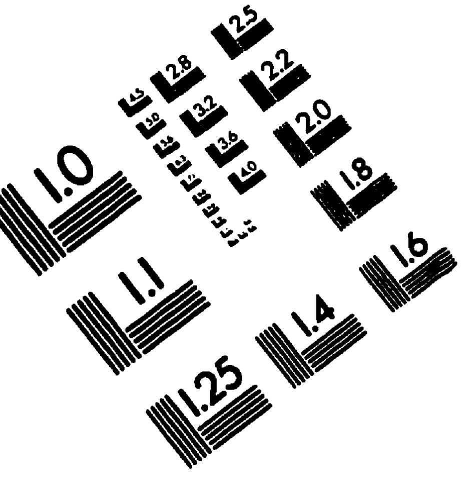
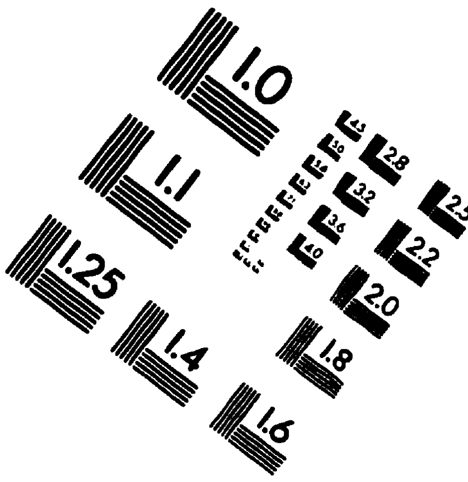
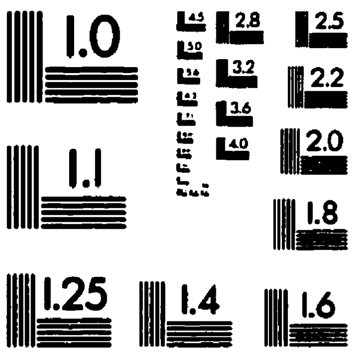
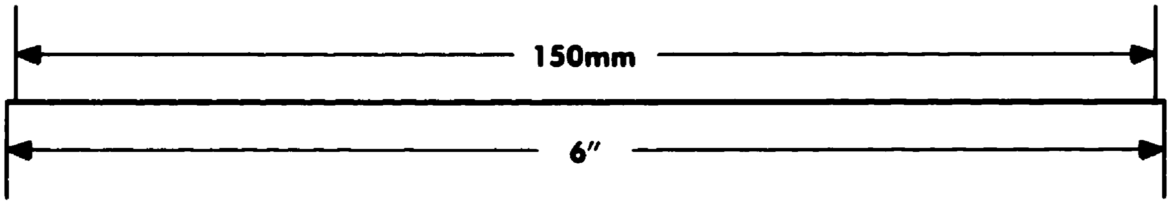
[42] S.R. Safavian and D. Landgrebe. A survey of decision tree classifier methodology. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(3), 1991.

[43] S.B. Gelfand, C.S. Ravishankar, and E.J. Delp. An iterative growing and pruning algorithm for classification tree design. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(2), 1991.

[44] A. Shapiro. *The role of inductive learning in expert systems.* PhD thesis, University of Edinburgh, 1983.

[45] B.A. Shepherd. An appraisal of a decision tree approach to image classification. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 8. IJCAI, 1983.

[46] S.Z. Kiang, G. Sullivan, *et al.* Recursive optimal pruning of tree-structured vector quantizers. In *Proceedings of the International Conference on Acoustics, Speech and Signal Proceesing*, pages 2285–2288. IEEE, 1991.





150mm
6"