PHRASAL DOCUMENT ANALYSIS FOR MODELING

By
Ritesh D. Sojitra
sojitra@vt.edu

Thesis submitted to the faculty of Virginia Polytechnic Institute and State University in the partial fulfillment of the requirements for the degree of

Master of Science
In
Electrical Engineering

Dr. W. R. Cyre, Chairman
Dr. J. R. Armstrong
Dr. F. G. Gray

September 11, 1998
Blacksburg, Virginia

Keywords:
Chunks, Information Extraction, Modeling, ModelMaker, Noun Phrase, Parser

PHRASAL DOCUMENT ANALYSIS FOR MODELING

Ritesh D. Sojitra

Abstract

Specifications of digital hardware systems are typically written in a natural language. The objective of this research is automatic information extraction from specifications to aid model generation for system level design automation. This is done by automatic extraction of the noun phrases and the verbs from the natural language specification statements. First, the natural language sentences are parsed using a chart parser. Then, a noun phrase and verb extractor scans these charts to obtain the noun phrases with their frequencies of occurrence. The noun phrases are then classified by semantic types. Also the verbs are automatically assigned their respective roots and classified. Finally, each sentence is summarized as a sequence of "chunks": noun phrases, verbs and prepositions. Vectors are generated from these chunks and imported into MS Excel for plotting occurrence graphs of noun phrases and verbs with respect to the sentences in which they occur. Finally, inter-term dependencies between noun phrases, and between noun phrases and verbs were studied. The frequencies of occurrence, the classification of chunks, the occurrence graphs and the inter-term dependencies together give useful information about the subject, the hardware components and the behavior of a system described by a natural language specification document.

ACKNOWLEDGEMENT

I thank Dr. Walling R. Cyre, my academic and research advisor, for the guidance and support he provided throughout the duration of this project. I am especially grateful for his numerous ideas and suggestions that helped me get through some of the problems, and often made me realize the aspects I had overlooked. I also thank Dr. James R. Armstrong and Dr. F. Gail Gray for serving on my committee and for all their assistance and support for this project.

I thank Andreas I. Gunawan, graduate research assistant under Dr. Walling R. Cyre and presently a Hardware Engineer at Texas Instruments, for the help he provided me through the course of this thesis. Finally, I thank my parents, all my family members and friends for their continuous support and encouragement.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

1.1 Motivation

The complexity of design processes has increased due to the explosive growth in complexity of digital system designs. Design of digital systems usually begins with the development of a set of specifications outlining the requirements for the desired system. Usually, the structural information is expressed in formal diagrams, whereas the behavioral description is expressed in the accompanying natural language text. The behavioral models of a system required for simulation, analysis, test and automatic synthesis must be constructed from the natural language description. Building behavioral models is an intensive and repetitive task that requires considerable skill. To be competitive, these models must be generated quickly and accurately. By avoiding errors, and by providing better tools for the design process, costs and delays can be contained. This thesis reports on the use of automatic information extraction to scan natural language specifications and extract useful information to help build behavioral models. Information extraction can reduce design cycle times and improve modeling efficiency by eliminating much of the manual search and interpretation required in the modeling process.

The objective of the work proposed here is to make a significant contribution to automating the extraction of modeling information from source descriptions such as product specifications, product proposals and U.S. patents. The approach used in this work is phrasal analysis of natural language. This approach has been used previously for other purposes, as discussed in Chapter 2 of this thesis.

1.2 Research Approach

The approach of this work is based on noun phrase and verb extraction for providing useful information to a modeler. A program called NPer (Noun-Phraser-er) was written

in C++ for this purpose. The domain of the specification documents selected for the study was Direct Memory Access controllers. One commercial product data sheet [16] and five US Patents [3, 4, 6, 9, 17] were used as input specification documents. The analysis dictionary was based on three of the documents [3, 4, 16]. Also, the grammar was updated to identify more noun phrases and thus increase recall and precision of the program.

A parser applies [8] grammar rules to input sentences to form parse charts representing the syntactic structure of the English sentences. The parse charts serve as an input to the chunk extractor, which extracts noun phrases, verbs and prepositions. These chunks serve as a basic database of modeling information. Chunk occurrence lists and chunk occurrence vectors were developed for tracking the occurrence and co-occurrence of chunks in a specification document. These lists and vectors were used to plot chunk occurrence graphs and chunk co-occurrence graphs. NPer is written in C++ and runs on a Windows 95/NT platform. Validation tests for recall and precision were run on the other three US Patents [6, 9, 17].

1.3 Research Contributions

The research contributions made by the author of this thesis are listed below.

- Dictionary

  The dictionary used for NPer was updated from 2500 words, which was used for the ModelMaker [14] project, to 3000 words. Two documents [3, 4] describing Direct Memory Access controllers were used to update this dictionary. Words in the dictionary were classified into semantic categories. These categories are used to classify noun phrases and verbs.

- Grammar

  Seven new grammar rules were added to recognize noun phrases that were not recognized by the parser using the previous grammar.

- Root Forms of Verbs

  A database listing root forms of about 500 verbs was created. These roots are used to classify the verb chunks. The format of the file containing root forms of verbs is similar to the format of the dictionary.

- Vocaber

  Vocaber is a program used to sort and count frequencies of occurrence of words in a text. This program, previously written in Pascal, was converted to C++, and was used to calculate frequencies of occurrence of noun phrases and verbs.

- Chunk Extracting Algorithm

  A new function was developed to perform the noun phrase, verb and preposition extraction from a chart built by the parser. This function scans through all the constituents of an existing chart and extracts chunks out of it.

- Storing the Chunks

  A new C++ class was developed to store the extracted chunks. This class stores the chunk name, its identification number, the sentence number in which the chunk occurs, the syntactic category of the chunk, the semantic category of chunk and the chunk head word. An object of this class was used to access the chunks for generating occurrence lists and occurrence vectors for plotting occurrence and co-occurrence graphs.

- Plotting Chunk Occurrence Graphs

  Chunk occurrence lists were imported to MS Excel to plot occurrence graphs. Occurrence graphs are useful for observing where components, signals or behaviors are discussed in a specification.

- Plotting Chunk Co-occurrence Graphs

  A Matlab program was written which takes chunk occurrence vectors as its input to calculate co-occurrence frequencies and plot the co-occurrence graph. Co-

occurrence graphs were used to study the degree of relationships between different chunks.

- Frequency of Occurrence

  A study of frequency of occurrence of chunks was done to show that chunks that occur often in a text are better indicators of what the text is about.

- Predicting Different Sections of Text

  A study of occurrence graphs was done to predict different sections of a text, such as abstract, background, summary, detailed description and claims. It was shown that the detailed description section of a text is of more importance to a modeler.

- Frequency of co-occurrence

  A study of frequency of co-occurrence between two noun phrases was done to reveal important information about the connection between two devices. Similarly, a study of frequency of co-occurrence between noun phrases and verbs was done to indicate the functions of devices.

- Degree of Relationship

A study of co-occurrence graphs was done to reveal the degree of relationship between two chunks. Co-occurrence graphs gave information about device interfacing and device functioning of the system described in a text.

CHAPTER 2: RELEVANT WORK

Considerable research has been carried out, and is being carried out, in the field of knowledge acquisition and machine translation. This chapter discusses some of the research done in the area of information extraction and inter-term dependencies. This discussion includes a brief description of, Abbott's technique for programming from English specifications [1], Grefenstette's Sextant system that studies similarities between words [12], Vadera's and Meziane's approach for generating formal English specifications from informal English specifications [19], and Dong's and Agogino's work on information extraction for constructing design representations [11].

2.1 Program Design by Informal English Descriptions

Abbott presents a technique for developing programs from informal but precise English descriptions [1]. His paper describes how to make a transformation from noun phrases into data types and objects, verbs and attributes into operators, direct references into variables, and other English equivalents into control structures, for Ada programming. First, an "informal strategy" [1] for the problem is developed. This is done by stating the problem in the same words and concepts as the problem itself. Then this casual plan of action is analyzed for data types, objects, operators, and control structures. These are then partitioned into a package and a subprogram. Thus in the final program, the package contains the data types, the variables and the operators, whereas the subprogram contains the steps of operations or functions in terms of the data types, the variables, the operators and the control structures that are defined in the package.

The modeling technique in this thesis is similar to the programming methodology proposed by Abbott, but begins with existing documents. Abbott observes the parallelism between noun phrases and data types, and between verbs and operators. This thesis also deals with noun phrases that can be classified as entities, variables or components, and verbs that can be directly related to states or processes. Abbott does not

define a means for extracting these noun phrases and verbs automatically, whereas this thesis deals specifically with automatic information extraction. Abbott's end product is a program in Ada, whereas the end product in this thesis depends entirely on the modeler. It can be any modeling language, such as, a hardware description language or state charts.

2.2 Sextant: Extracting Semantics from Raw Text

Sextant, by Grefenstette, is a complete system that uses finer-grained syntactic contexts to discover similarities between words [12]. It is based on the argument that words that are used in a similar way throughout a corpus are semantically similar. Sextant takes raw text as input and divides it into words, which are then looked up in a simple dictionary. The dictionary supplies possible grammatical categories for each word. Using the local context around the word, a decision on the likely grammatical category for each word is made. Then, each sentence is bracketed into noun phrases and verb phrases. The extracted noun phrases are the longest possible, complex noun phrases, which include prepositions and conjunctions. Once this is done, a five-pass method is used to extract syntactic relations between words within these phrases, and across these phrases. Thus the modifiers are connected to head nouns within phrases, and verbs are connected to subjects and objects. Finally, a similarity measure is calculated and Sextant produces a list of most similar words as an output.

This thesis also deals with noun phrases and verbs, and relationships between them. But, in this thesis, after the words are given their grammar categories from the dictionary, syntactic analysis is performed to construct parse trees. A parse tree may represent a complete sentence, or one or the constituents of the sentence. The noun phrases and verbs are then extracted from these parse trees. Also, the noun phrases extracted are longest possible simple noun phrases. This thesis does not extract similar words, but looks at the frequencies of co-occurrence of phrases. This thesis is based on the proposition that phrases that have high frequency of co-occurrence are related to each other.

2.3 From English to Formal Specifications

This paper, by Vadera and Meziane, proposes an interactive approach to transform an informal English specification to a formal specification [19]. The approach helps with the process of identifying data types from English specifications, and helps identify any ambiguities and incompleteness in English specifications. An English text document that contains an informal specification is used as input for the system. As the system does not handle conjunctions and pronouns, sentences that contain conjunctions are broken into separate sentences, and pronoun references are identified separately. The nouns that represent the entities, and the verbs that represent the relationships between the entities, are identified. Also quantifiers are identified, which consist of the universal quantifier indicated by words such as "every" and "all", the unique existential quantifier indicated by definite article "the", and the existential quantifier indicated by indefinite article "a". The degree of relationship is determined from these quantifiers associated with each entity. Thus, for each sentence, the system produces one or more logical forms as shown below [19].

Example: Sentence → *The complex aircraft uses a radar.*
Logical form → *all(aircraft(X)&complex(X),*
ex(radar(Y),use(X,Y)))

The system involves active participation by the user to select the logical form that corresponds to the intended meaning of the English sentence. Also the user is allowed to filter any irrelevant entity or relationship from the list. Finally, an entity-relationship model [19] is produced as a result of the above process.

The thesis presented here is similar to this work in that both look at the usefulness of nouns and verbs to provide specific information to the user. However, this thesis does not produce an entity-relationship model, but concentrates on automatically extracting all the useful information for modeling. The system described in this thesis takes care of pronouns and prepositions, which is not done in Vadera's and Meziane's work.

Moreover this thesis does not require the user to help the system parse the English specification document and generate the final output. The parsing of the English sentences is done automatically with the help of a dictionary and a grammar. The role of the modeler is to apply the final output of the system for model generation.

2.4 Text Analysis for Constructing Design Representations

Dong and Agogino present a technique to determine a design representation from a corpus of design documents [11]. Based upon contextual clues and syntactic patterns in the design documents, the relation between the design and the representation for the design is determined. A computable learning method is developed to extract the content of the design model to help information sharing among designers. First, a set of content-carrying terms is developed based on a word score metric. The score for a word depends upon the frequency of occurrence of the word in individual documents in the corpus and an inverted weight distribution measurement for the number of documents containing the term. Contextual similarity of these content-carrying terms is determined by measuring the frequency of occurrence of any two of these terms in the documents. Once the prescribed vocabulary is developed, the contextual similarity between these content-carrying terms is represented by building a Bayesian belief network. Finally, the design document learning system is integrated with an agent-based collaborative design system, known as "smart drawing" [10], for fetching design information. The agent environment consists of the database of design documents, including the drawings, design specifications, design notes and memos and e-mails written between designers. This agent builds a model of the design by periodically generating the list of content carrying words and using the document database for additional data. In response to the user's request, the agent can retrieve relevant design information.

This thesis has many similarities and differences with Dong's and Agogino's work. This thesis describes a single user tool to automatically extract useful chunks of information, which would help the user model the system being described in that particular natural language specification document. Dong and Agogino mainly talk about

information retrieval from a database, which would help sharing of information among various function workgroups. Both research areas are similar in the sense that both exploit the content carrying power of the noun phrases and verbs, and their frequency of occurrence. Dong and Agogino use Bayesian belief networks to discover inter-term dependencies, whereas this thesis uses the frequencies of co-occurrence among these terms.

CHAPTER 3: BACKGROUND

This chapter describes the theory behind the research carried out in this thesis. First, English language grammar and terminology, related to this research are briefly discussed. Then, some technical terms used in this thesis are defined. While discussing the terminology, this chapter briefly talks about the importance of nouns, noun phrases, verbs, and prepositions for information extraction from natural language specifications.

3.1 English Language Grammar

Since digital system descriptions do not use the full features of the English language, the subject of this work is a technical sublanguage [7]. Grammar is the part of a language that deals with how the words of a language are arranged into phrases and sentences. The sentence is a basic element of language and communication [15], and is the highest unit in the English grammar, here.

3.1.1. Context-Free Grammar

A context-free grammar provides an especially simple way of describing the structures of a language that can be used in a systematic way to generate the sentences of a language. Though context-free grammar is not adequate for English, it performs well enough in the present application. A context-free grammar consists of a set of rules. A rule has a result and a constituent list. Whenever the list of constituents can be found contiguously in a sentence, they can be replaced by a constituent of type result. The rules define a sentence of a language, and large and small constituents of a sentence. The result is followed by an arrow, followed by a sequence of constituents forming the rule such as

$$np \rightarrow det\ noun$$

where, “np” denotes a noun phrase

“det” denotes a determiner

“noun” denotes a noun

There can be one or more rules defining a constituent type. A grammar consists of a set “Σ” of terminals (the alphabet), a set of non-terminals, a special non-terminal “S”, and a set of grammar rules. In this study, the alphabet is a vocabulary of English words. The non-terminals are lexical categories and constituent type. “S” denotes a sentence. In a context-free grammar, all rules are of the form

$$A \rightarrow X$$

where, A is a non-terminal

X is a non-empty string of terminals and non-terminals

s	→	np	vp
np	→	det	np
np	→	adj	noun
vp	→	verb	
det	→	“the”	
adj	→	“host”	
noun	→	“processor”	
verb	→	“reads”	

Figure 3.1 Simple Context-Free Grammar

A simple context-free grammar is shown in Figure 3.1. As shown in Figure 3.1, a noun phrase followed by a verb phrase can form a sentence. However, a noun phrase can be formed by two different sets of constituents. A determiner followed by a noun phrase can form a noun phrase, or an adjective followed by a noun can form a noun phrase. A verb can form a verb phrase. The words “the”, “host”, “processor” and “reads” are terminals of this grammar.

3.1.2 The Noun Phrase Grammar

This thesis used the noun phrase grammar listed in Figure 3.2. Table 3.1 lists the non-terminals used in the grammar. Most of these rules were developed in the ASPIN [8] system. Table 3.2 shows some new rules to identify noun phrases, added during the course of this work.

np	→	pdet	det	adjs	head	
np	→	pdet	det	head		
np	→	pdet	adjs	head		
np	→	pdet	head			
np	→	det	ord	#	adjs	head
np	→	det	ord	#	head	
np	→	det	ord	adjs	head	
np	→	det	ord	head		
np	→	det	#	adjs	head	
np	→	det	#	head		
np	→	det	adjs	head		
np	→	det	head			
np	→	ord	#	adjs	head	
np	→	ord	#	head		
np	→	ord	adjs	head		
np	→	ord	head			
np	→	#	adjs	head		
np	→	#	head			
np	→	adjs	head			
np	→	head				
np	→	head	range			
np	→	pron				
np	→	np	#			
np	→	np	(	np	)	
head	→	noun				
head	→	id				
head	→	noun	head			
head	→	id	head			
adjs	→	adj				
adjs	→	adj	adjs			
adjs	→	adj	,	adjs		

Figure 3.2 Noun Phrase Grammar Rules

Table 3.1 Listing of Non-terminals

Non-terminal	Description
#	cardinal number
adj	adjective
adjs	adjective sequence
adv	simple adverb
det	determiner
head	head noun
id	identifier
np	noun phrase
ord	ordinal such as first, last
pdet	pre-determiner such as all, none
pron	pronoun
ven	past participle form of verb
ving	present participle form of verb

Table 3.2 Newly Added Grammar Rules

Rules	Examples
np → (np)	*(brief description)*
np → [np]	*[detailed description]*
np → det ving head	*the data processing apparatus*
np → np ving head	*data processing apparatus*
np → det ven head	*the added circuit*
np → det adv ven head	*the newly added circuit*
adj → (adj)	*the dma (host) processor*

The first two rules were added to recognize noun phrases enclosed by the parentheses or the square brackets. The next two rules were added to recognize noun phrases containing the present participle form of verbs as pre-modifiers. The next two rules were added to recognize noun phrases containing adverbs along with simple past form of verbs used as pre-modifiers. The last rule was added to identify an adjective enclosed in the parenthesis. A head can be a noun, an identifier, a noun followed by a head, or an identifier followed by a head.

3.2 English Language Terminology

Words and punctuation are the terminals of the language. This includes nouns, pronouns, verbs, adjectives, adverbs, prepositions, and conjunctions. Sequences of these

terminals make up phrases such as noun phrases, verb phrases and prepositional phrases. This thesis is concerned with automatic detection of these phrasal constituents.

3.2.1 Lexical Categories

Words in sentences are assigned lexical categories (a subset of non-terminals) that describe their role in the structure of a sentence, such as noun, verb, adjective and preposition. The lexical categories of words are listed in a dictionary.

Examples: "*fastest*": *adj*,
"*microprocessor*": *noun*,
"*that*": *prep*,
"*the*": *det*,
"*transmits*": *verb*

3.2.2 Nouns

Nouns are words that name or refer to a thing, quality, action or idea. Their arrangement with verbs helps to form the sentence core, which is essential to every complete sentence. They can function as the subject of a verb or as the direct object of a verb, as well as the objects of prepositional phrases.

- Noun as the subject of a verb.
  Example: *The memory stores the data.*

- Noun as the direct object of a verb.
  Example: *The memory stores the data.*

- Types of Nouns
  Following are the types of nouns found in the English language.

  - Proper Nouns

A proper noun begins with a capital letter and usually is a name of a thing or a place.

Examples: *Intel-586, Motorola-68000*

- Common Nouns

  All nouns, except proper nouns, are common nouns.

- Derivational form of Nouns

  These are nouns derived from other parts of speech by adding a suffix. They are as follows.

  - Nouns from Verbs

    Nouns can be formed form verbs by adding suffixes such as “tion”, “ment”, “al”, “age”, “ance”, “ence”, “(e)ry”, “sion”, “ure”, “er”, “ant” and “or”.

    Examples: *transmission, interruption, arrangement, arrival, package, performance, correspondence, delivery, division, failure, analyzer, applicant, behavior*

    In technical English, the infinitive form is also used as a noun.

    Examples: *read, write, interrupt*

  - Nouns from Adjectives

    Nouns can be formed from adjectives by adding suffixes such as “ity”, “ness” and “th”.

    Examples: *activity, effectiveness, width*

In digital systems descriptions, a noun can represent a device or value, such as processor, instruction, and signal. Nouns derived from verbs represent actions, whereas nouns derived from adjectives are attributes. Nouns may also refer to documentation elements such as figures, sections and tables.

3.2.3 Adjectives

An adjective modifies a noun or a pronoun. It generally precedes the noun it refers to, but can sometimes follow a noun. Adjectives can be used alone, or with one or more modifiers. They usually attribute a quality or characteristic to the noun they modify.

Examples: *a fast microprocessor, the signal is low, it is slow*

3.2.4 Pronouns

A pronoun is a word that is substituted in place of a noun that has occurred earlier. Pronouns have a weak semantic content because their meaning is derived from the context in which they appear.

Example: *It is a very slow process.*

This is a data processing apparatus.

3.2.5 Noun Phrases

A noun phrase is a sequence of words that function as a noun, and is made up of a head noun and all its modifiers. A noun phrase can appear as the subject, object or complement of a verb, an object of a preposition, or an appositive.

- Structure of a Noun Phrase

  In general, a noun phrase consists of pre-modifiers, the head and post-modifiers.

  np → *pre-modifiers head-noun post-modifiers*

  - Head

    The head of a noun phrase can be a noun or a pronoun.

    Example: *the quick **reception** of the control signal*

- Pre-modifiers

  Pre-modifiers comprise of all the describing constituents of a noun phrase before the head. A noun phrase can have one or more pre-modifiers. Pre-modifiers can be adjectives, ordinals, nouns and verbs.

  Examples: *the quick* ***reception*** *of the control* ***signal***,
  the 16-bit synchronous stored-program ***computer***

  Determiners also form pre-modifiers.

  Examples: *the* ***memory***, *a* ***microprocessor***, *an* ***event***, *this* ***device***, *another* ***signal***, *their* ***function***

  Pre-determiners also form pre-modifiers.

  Examples: *few* ***states***, *all* ***processors***, *some of the* ***registers***

- Post-modifiers

  Post-modifiers comprise of all the describing constituents after the head. They are typically prepositional phrases or clauses. A noun phrase can have one or more post-modifiers.

  Examples: *the quick* ***reception*** *of the control signal*,
  the ***data*** *which was stored earlier*

  Some appositives form post-modifiers and are peculiar to patents.

  Example: *the data processing* ***apparatus*** *101*

- Types of Noun Phrases

  Usually noun phrases are classified into two categories.

  - Simple Noun Phrases

    A simple noun phrase is a noun phrase without prepositional phrases, clauses and conjunctions. It can have pre-modifiers and certain post-modifiers.

Examples: ***circuit,***

*the **circuit,***

*the control signal generator **circuit,***

*the **MPU** 1*

- Complex Noun Phrases

  Noun phrases other than simple noun phrases are complex noun phrases.

  Example: *the 16-bit **outputs** of the flag register and the controller,*

  *the **data** stored in the memory*

The semantic category of a noun phrase can be discovered, by identifying the semantic category of the head. Therefore, if we detect noun phrases of texts, we can understand the texts to some extent, as shown in Section 5.2 of this thesis.

3.2.6 Verbs

A verb is a word that expresses an event, action, or a state of being. There are five main forms of verbs depending on the suffixes they use.

Table 3.3 Forms of Verbs

Forms of Verbs	Suffix Used	Examples
Simple Present	Root Form - No suffix	*generate, process*
Simple Present	-s	*generates, processes*
Simple Past	-ed	*generated, processed*
Present Participle	-ing	*generating, processing*
Past Participle	-en	*written, taken, shown*

These forms are used in strings of words called verb sequences, as shown in Figure 3.4. The verb usually expresses the central topic of the sentence. The structure of a sentence is determined by the verb and the noun phrases that modify or elaborate it.

Table 3.4 Sequences of Verbs

Tenses	Active Voice	Passive Voice
Present	*generate, generates*	*is/are generated*
Past	*generated*	*was/were generated*
Future	*shall/will generate*	*shall/will be generated*
Present Perfect	*has/have generated*	*has/have been generated*
Past Perfect	*had generated*	*had been generated*
Future Perfect	*shall/will have generated*	*shall/will have been generated*
Present Continuous	*is/are generating*	*is/are being generated*
Past Continuous	*was/were generating*	*was/were being generated*
Future Continuous	*shall/will be generating*	*shall/will be being generated*
Present Perfect Continuous	*has/have been generating*	*has/have been being generated*
Past Perfect Continuous	*had been generating*	*had been being generated*
Future Perfect Continuous	*shall/will have been generating*	*shall/will have been being generated*

3.2.7 Prepositions

A preposition relates or positions a noun phrase to another element of the sentence. Prepositions are used as connectives between two noun phrases, or a noun phrase and a verb. The object of a preposition, which can be a noun or a noun phrase, must follow the preposition.

- Simple Prepositions

  These consist of only one word.

  Examples: *on, at, in, by, for, during, within, before, after, to, from, about*

- Compound Prepositions

  These consist of more than one word.

  Examples: *according to, depending on, regardless of, in spite of*

Prepositional phrases answer questions such as, whose, which one, number, what kind, how, when, where and why. Thus prepositions provide important semantic

information by indicating the type of role or relationship between the object of a prepositional phrase and the noun phrase or the verb it modifies. In the sub-language of this thesis, "of" is never used to modify a verb and "in" rarely modifies a verb.

Example: *The data is stored in the memory.*

3.3 Definition of Terminology

The results in this thesis are based upon technical terms such as tokens, chunks, their semantic classification and frequencies of occurrence. This section briefly discusses these terms.

3.3.1 Tokens

A token can be a word, number or punctuation (tokens are terminals of the grammar). The input text is read, character by character, and a token is formed from a set of characters. Following rules describe tokens [14]. These rules are used in the Lexical scanner to find tokens in text.

- A token can be
  - a string of letters
  - a number
  - a punctuation mark
  - a special punctuation

- A string of letters can be
  - a letter followed by letters or
  - a letter followed by a number or
  - a period followed by letters or
  - a hyphen, "-", or
  - a slash, "/", or
  - an underscore, "_" or
  - any letter between A to Z or
  - any letter between a to z

- A number can be
  - a digit followed by a number or

- a number followed by a letter or
- a special punctuation followed by a number or
- any single digit number between 0 to 9

- A punctuation mark can be
  - a colon, ":", or
  - an exclamation mark, "!", or
  - a left curly bracket, "{", or
  - a left parenthesis, "(",or
  - a left square bracket, "[", or
  - a question mark, "?", or
  - a right curly bracket, "]", or
  - a right parenthesis, ")", or
  - a right square bracket, "]", or
  - a semi-colon, ";"

- A special punctuation mark can be
  - a period, ".", or
  - a comma, ","

- Any other character string is classified as a space.

3.3.2 Chunks

Chunks [18] are one of the final outputs of the software described in this thesis (chunks are based on a subset of the non-terminals of the grammar). Each sentence in a natural language specification document is summarized as a sequence of chunks. Each chunk has the following parts.

- Identification Number

  This is a unique serial number given to each chunk.

- Sentence Number

  This is the number of the sentence to which the phrase or the word belongs.

- Syntactic Type

  This field specifies the grammatical type (non-terminal) of the phrase or the word in a chunk. It is “np” for a noun phrase, “v” for a verb and “p” for a preposition,

- Headword

  In a noun phrase, headword is the head noun or the head pronoun.

  Example: Noun Phrase → *the data processing apparatus*

  Headword → *apparatus*

  In case of a verb, the headword is the root form of that verb.

  Example: Verbs → *generates, generated, generating, generate*

  Headword → *generate*

  Headword in case of prepositions is the preposition itself.

- Semantic Type

  This field denotes the conceptual type of the headword. For prepositions this field is marked “nil”, as they do not carry any conceptual type. Different conceptual types, with examples, are discussed in the next section.

- Determiner

  The determiner in case of noun phrases is shown in this field. In case of verbs and prepositions this field is marked “nil”.

- Term

  A term can be a noun phrase without its determiner, a verb in the same form as found in the sentence, or a preposition.

For example, consider the sentence,

The DMA processor reads the data from the memory.

Assuming that it is the first sentence in the document, it will be decomposed into the following chunks.

ID	Sentence Number	Syntactic Type	Headword	Semantic Type	Determiner	Term
1	*1*	*np*	*processor*	*device*	*the*	*dma processor*
2	*1*	*v*	*read*	*action*	*nil*	*reads*
3	*1*	*np*	*data*	*value*	*the*	*data*
4	*1*	*p*	*from*	*nil*	*nil*	*from*
5	*1*	*np*	*memory*	*device*	*the*	*memory*

Figure 3.3 Examples of Chunks

Thus, chunks are feature structures in which the meaning of constituents of sentences is encoded. This database can be sorted in MS Access or MS Excel, depending upon various criteria, for analyzing the text. Different graphs can be plotted after extracting useful information from the database, and conclusions can be made as discussed in Chapter 5 of this thesis.

3.3.3. Semantic Classification

Each headword in a chunk is classified by concept type using the conceptual hierarchy shown in Figure 3.4 and described in Table 3.5.

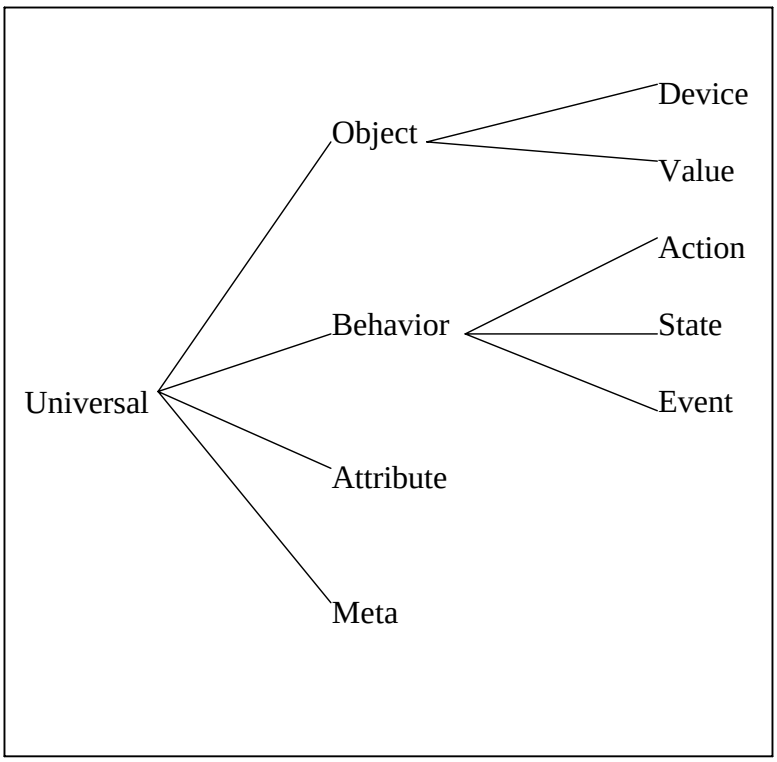

Figure 3.4 Conceptual Type Hierarchy

Table 3.5 Conceptual Types

Conceptual Type	Represents	Examples
Object	A device or a value.	*Input, output, block, board*
Device	A hardware unit.	*Memory, register, circuit*
Value	A software unit or information.	*Program, instruction, bit, data, signal*
Behavior	A behavior of a hardware or a software unit.	*Remain, correspond, require, support*
Action	A function of an object or a device.	*Read, write, transfer, operate, receive*
State	A state in which an object or a device functions.	*Mode, condition, case*
Event	A beginning or an end of an action.	*Begin, cause, initiate, address, signal*
Attribute	A characteristic of an object or behavior.	*Architecture, group, logic, type, plurality, synchronous*
Meta	All words related to documentation.	*Figure, claim, background, description, embodiment*

Classifying the headwords into their conceptual types gives information about the semantic content of the chunks.

3.3.4 Frequency of Occurrence

Frequency of occurrence is defined as the number of times a particular chunk occurs in a document. There is a high probability that chunks, which are related to the subject of a document, will be repeated, as shown in Section 5.3 of this thesis. The idea is that the frequency measurement correlates with the text semantics. Chunks that occur often in a text are better indicators of what the text is about, as shown in Section 5.3 of this thesis.

3.3.5 Chunk Occurrence Lists and Occurrence Graphs

Occurrence graphs are two-dimensional plots between chunks and the sentences in which they occur. Occurrence lists are a list of integers used to plot occurrence graphs. Each chunk is represented by a list, which contains the sentence numbers in which the chunk occurs.

Example: *1 2 12 34 56 57 78 90 113 123 456*

The example list denotes that the chunk occurs in sentence number 1, 2, 12, 34, 56, 57, 78, 90, 113, 123 and 456. The occurrence graphs give information about word patterns in the input text. One can easily predict different sections of a text and their subjects by studying these occurrence graphs, as shown in Section 5.4 of this thesis.

3.3.6 Frequency of Co-occurrence

Two terms co-occur if they both occur in a sentence of the document. Frequency of co-occurrence between two terms is defined as

$$\sum_{i=1}^{N}(FW1 \times FW2) \tag{3.1}$$

where FW1 = number of times the first chunk occurs in a sentence

FW2 = number of times the second chunk occurs in the same sentence

N = total number of sentences in the document

Frequency of co-occurrence between two terms can convey the degree of relationship between these two terms. The higher the frequency of co-occurrence, the higher is the probability that the terms are related. Study of frequency of co-occurrence between two noun phrases can reveal important information about the connection between two devices. Also, study of frequency of co-occurrence between a noun phrase and a verb can indicate the function of a device. This is shown in Section 5.5 of this thesis.

3.3.7 Chunk Occurrence Vectors and Co-occurrence Graphs

Co-occurrence graphs are three-dimensional plots between different chunks and their frequencies of co-occurrence. Chunk occurrence vectors are used to calculate the frequencies of co-occurrence between chunks for plotting co-occurrence graphs. The lengths of these vectors are equal to the number of sentences in the input text. These vectors contain integers that represent the frequency of the chunk in each sentence of the document.

Example: *1 0 0 0 2 3 0 0 0 0*

The example vector denotes that the chunk occurs once in the first sentence, twice in the fifth sentence, thrice in the sixth sentence and zero times in all other sentences. Co-occurrence graphs give information about the degree of relationship between two chunks. As shown in Section 5.5 of this thesis, study of these graphs reveals a good amount of information about device interfacing and device functioning of the system described in a text.

3.3.8 Recall and Precision

Recall [13] measures the extent to which correct noun phrases are extracted. Similarly, precision [13] measures the degree to which noun phrases are extracted correctly.

$$Recall = NPcorrect / NPtotal \tag{3.2}$$

$$Precision = NPcorrect / (NPcorrect + NPincorrect) \tag{3.3}$$

where NPcorrect = number of correct noun phrases extracted
NPincorrect = number of incorrect noun phrases extracted
NPtotal = total number of noun phrases in a document

The incorrect noun phrases arise because the grammar imperfectly represents the language. For example, according to the grammar “output” is a noun. Note that the number of extracted noun phrases (NPcorrect + NPincorrect) is usually less then the total number of noun phrases in the document (NPtotal).

CHAPTER 4: IMPLEMENTATION

This chapter discusses the functioning of the software developed for this thesis. The program is called NPer, and has two components: the parser and the chunk extractor. First, the functioning of the parser is discussed, then the functioning of NPer as a whole is described. This chapter begins with some theory on chart parsing.

4.1 Theory of a Chart Parser

Parsing is a process in which a natural language sentence is analyzed into a hierarchical structure that corresponds to the structure of a sentence. Parsing identifies constituents of a sentence, which can be used for future analysis. In order to parse sentences, a parser must identify words in a sentence, and match these sequences of words with grammar rules. A context-free grammar is used for this purpose. There are two methods to match these rules against a sequence of constituents: the top-down method and the bottom-up method.

In the top-down method, a parser selects a top-level rule, for a sentence, and searches for the smaller constituents specified by the rule. Then it searches for further rules for decomposing these constituents into smaller constituents. The process continues until individual words are reached. If the sequence of rule matching leads to a combination of constituents that can be represented by the words in the sentence, then the parse is successful, otherwise the parser backtracks to a higher-level constituent and tries an alternative rule.

The bottom-up method is quite opposite to the top-down method. In this procedure, a parser begins with the words in the sentence and finds out their lexical categories. Then it tries to satisfy a lower-level grammar rule, such as a rule that defines a noun phrase. If a rule is satisfied, higher-level rules are tested with these new constituents formed by the lower-level rules. The process continues until all the words in

the input sentence satisfy a top-level rule defining a sentence. The parser used in the work reported in this thesis is a bottom-up parser.

In a chart parser, keeping track of constituents is done by a chart. A chart is a graph of nodes representing points between words in a sentence, linked by edges representing words and constituents. The edges are given a label that identifies the grammatical category of the constituent represented by the edge. Figure 4.2 shows a partial chart for the sentence "The host processor executes a program", using the simple grammar shown in Figure 4.1.

s	→	n	pred
s	→	np	vp
n	→	np	
np	→	det	np
np	→	adj	noun
np	→	noun	
vp	→	verb	

Figure 4.1 Grammar Rules for Partial Chart

Initially, the chart only contains the edges for individual words. Next edges for their lexical categories are added when the words are looked up in dictionary. The parser uses the chart to look for constituents to satisfy a rule. Then, whenever a rule succeeds, the parser adds the constituent to the chart. Thus, the parser builds larger constituents from smaller ones until all possibilities have been considered. For example, the rule

$$np \rightarrow det\ noun$$

combines the determiner and the noun constituents representing "a" and "program", respectively, to form a new noun phrase constituent representing "a program".

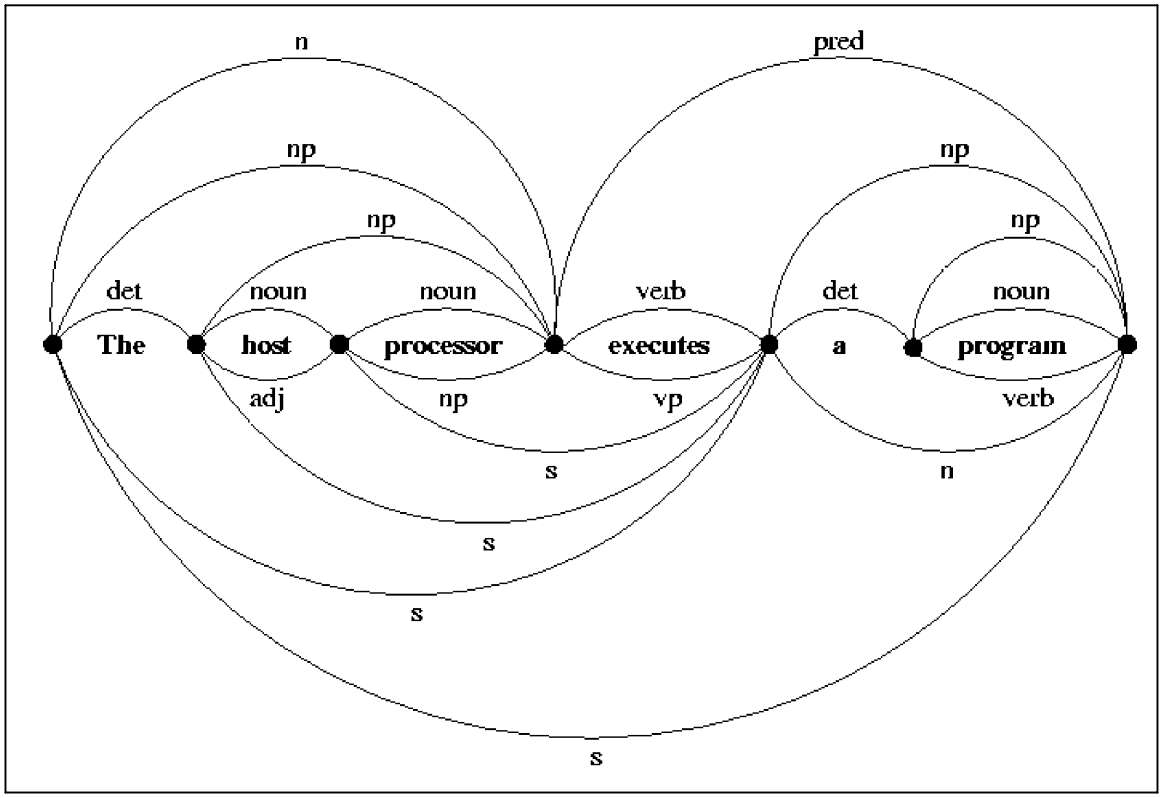

Figure 4.2 Partial Chart for a Sentence

The parser described in this thesis scans words from left to right of a sentence and builds the constituents. Once a word is processed, all possible constituents ending in that word have been built from the lexical categories of the word, and from constituents ending in previous words of the sentence.

4.2 The Parser Implementation

The parser discussed here is a part of a project called the Automated Specifications Interpreter (ASPIN) system [8]. It was written in C and implemented the bottom-up parsing strategy. Later, this parser was converted to C++ in the ModelMaker [14] project. The parser takes an English document as its input. The input file must be an ASCII text file. No pre-formatting of the text document is required. The functioning of the parser can be divided into two parts: initialization and building the charts. An

object-oriented description of the parser can be found in Gunawan's work on the ModelMaker [14].

4.2.1 Initialization

The first step in initializing the parser is to load a dictionary. The parser uses two types of dictionaries: a main dictionary and a multiword dictionary. The main dictionary defines about 3000 words with their possible lexical and semantic categories. The dictionary was manually constructed from words accumulated throughout the ASPIN [8] project, the ModelMaker [14] project and this thesis. The multiword dictionary defines groups of two or more words forming a lexical category. A sample of the main dictionary is shown in Figure 4.3 and that of the multiword dictionary is shown in Figure 4.4. The ASCII input file, "rdict.txt", carries the main dictionary and the ASCII input file, "mdict.txt" carries the multiword dictionary.

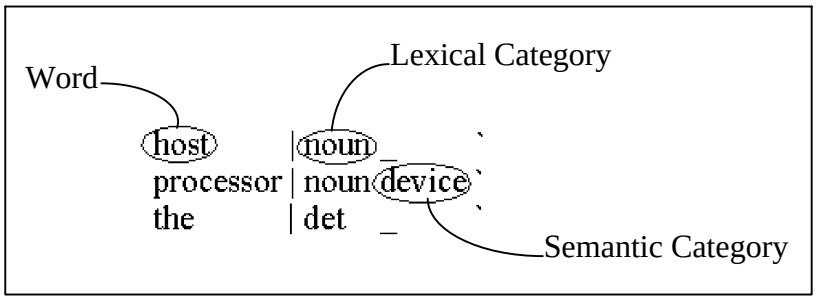

Figure 4.3 Main Dictionary

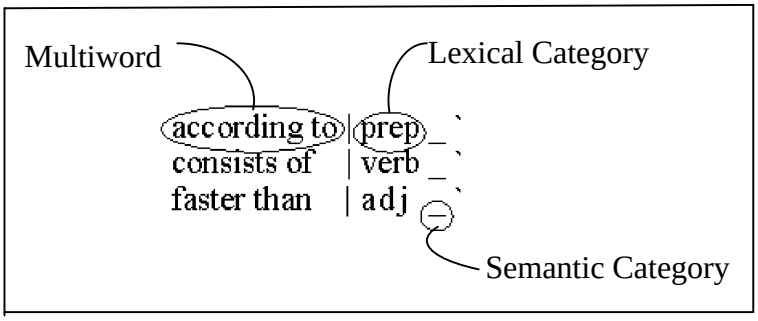

Figure 4.4 Multiword Dictionary

The second initialization step is to load the grammar rules. The grammar used by the Parser was developed manually in a study of English used to describe microprocessors [2]. A few new grammar rules to recognize noun phrases were added during the course of this work. Some sample grammar rules are shown in Figure 4.5. The ASCII input file, "gram.txt", contains the grammar rules. The main constituent, represented by the main constituent number in a rule, determines the semantic category of the result.

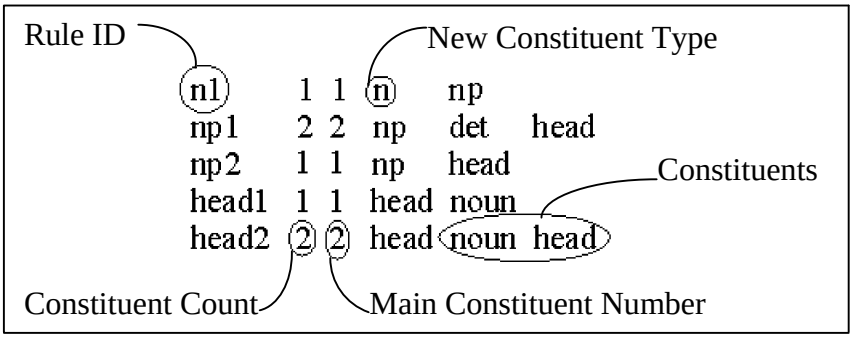

Figure 4.5 Grammar Rules

After the dictionary and the grammar rules are loaded, an empty chart is initialized.

4.2.2 Building the Chart

The input text is read and decomposed into tokens by the Lexical Scanner. The rules for identifying tokens were described earlier in Section 3.3.1. As soon as a token is formed, it is added to the chart for parsing. If the token is not found in the dictionary, it is defined as an "identifier" and added to the chart. This is done to take care of acronyms and identifiers such as device names and device types. One disadvantage of this is that all the unknown words in a text will be defined as identifiers and used as nouns. To avoid this misclassification, all the words of a text should be defined in the dictionary before using the parser. If the token is found in the dictionary, its lexical category is added to the chart as an edge.

Constituents, including lexical categories, form the edges of a chart. Some of the constituents can represent a sentence, whereas others represent lesser constituents such as noun phrases, verbs or prepositions. The constituents for an edge are listed with the node following that edge. Each node in the chart is given an identification number, called the node number. Also, each constituent attached to a node has an identification number, called the constituent number, and the starting node number and the finishing node number of the constituent. Each constituent has the constituent numbers of the constituents forming it, and the rule number of the grammar rule defining the constituent. This allows the parser to keep track of combinations of constituents already used to satisfy a rule. The algorithm for the parser is shown in Figure 4.6.

1. Load the main and the multiword dictionary.
2. Load the grammar rules.
3. Initialize the token as empty.
4. Get a token from the input file.
   If the token is a period, goto 9.
   If end of the document, goto 10.
5. Create a new node for the token and insert it at the end of the nodes of the chart.
6. Get the lexical and the semantic categories of the token from the dictionary.
7. For each lexical category of the token, create a new constituent for the present node.
8. Try satisfying all the grammar rules with different combinations of the existing constituents.
   If a rule succeeds, create a new constituent for the present node, goto 8.
   If all the rules and combinations of constituents have been tried, goto 4.
9. Output the chart, goto 3.
10. Stop.

Figure 4.6 Algorithm for the Parser

Figure 4.7 shows the partial chart generated for the example sentence in Figure 4.2. The dictionary used is shown in Figure 4.3 and the grammar used is shown in Figure 4.5.

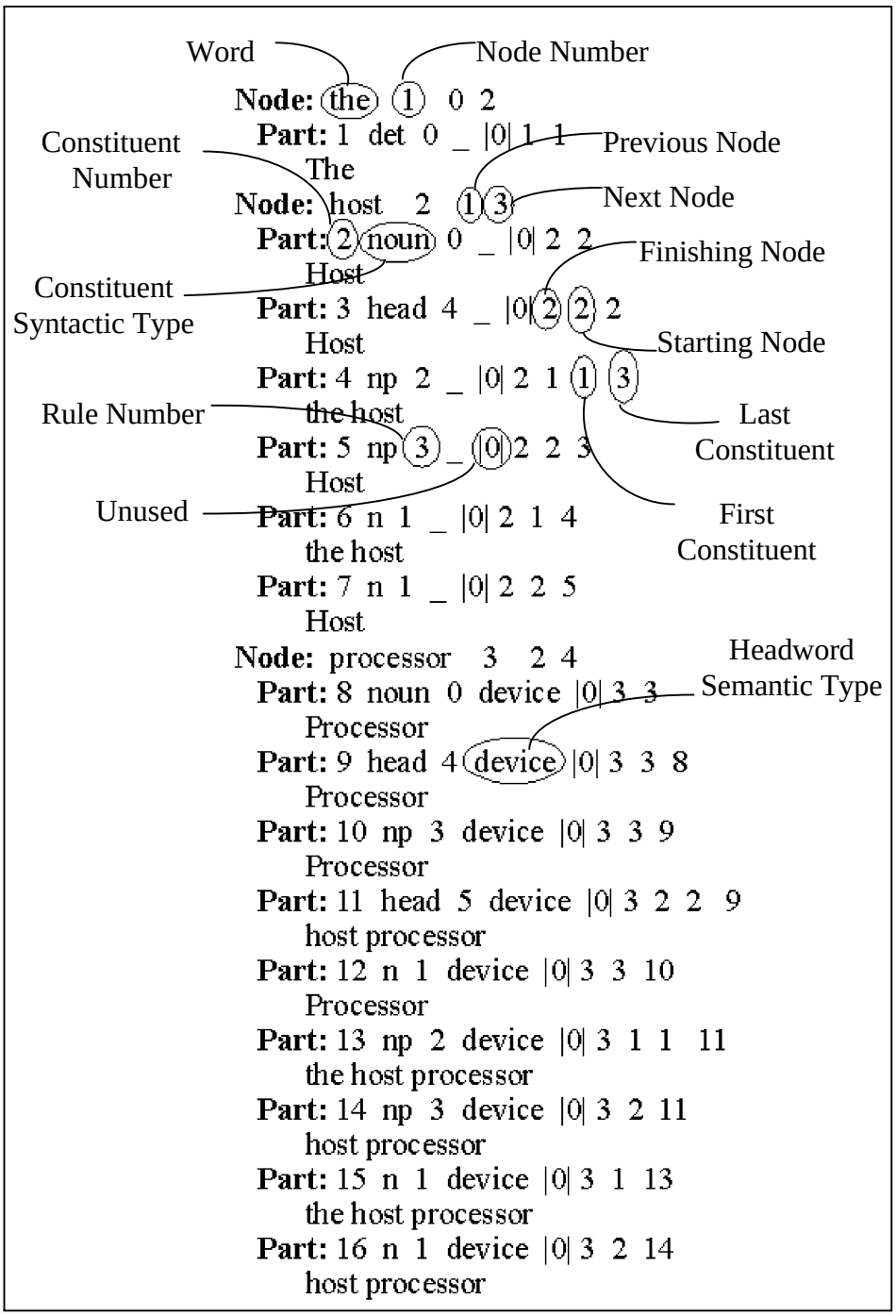

Figure 4.7 Partial Chart for the Example Sentence

As seen in the chart, different rules and different combinations of constituents are tried to from new constituents. Constituent 13 shows that the result was a noun phrase, and is formed using the grammar rule "np1" and by combining constituent 1 of node 1 and constituent 11 of node 2. Constituent 1 was formed directly from the dictionary,

whereas constituent 11 was formed using the rule “head2”, and constituent 2 of node 2 and constituent 9 of node 3. Again, constituent 2 was formed directly from the dictionary, but constituent 9 was formed using rule “head1” and constituent 8 of the same node. Constituent 8 was formed directly from the dictionary.

Similarly, constituent 16 is a nominal, which is formed using rule “n1” and constituent 14 of node 3. Constituent 14 was formed using rule “np2” and constituent 11 of node 3. Formation of constituent 11 is the same as discussed above. The data in the chart can be used, as an input to various algorithms to extract required information.

4.3 NPer

A chunk extractor method was written in C++ and was integrated with the above parser. This method operates on the final parse chart of a sentence generated by the parser and extracts the required chunks. The functioning of the chunk extractor consists of loading the root forms of verbs, extracting the chunks out of the chart, calculating frequencies of occurrence of noun phrases and verbs, and generating occurrence lists and occurrence vectors for plotting required graphs. The algorithm summarizing NPer is shown in Figure 4.8.

1. Load the root forms of verbs.
2. Get a parse chart for a sentence from the parser.
   If end of the document, goto 10.
3. Traverse the chart to extract all constituents forming noun phrases, verbs and prepositions.
4. Sort this list of constituents for extracting the longest simple noun phrases.
5. Traverse the chart again to output the required constituents in form of chunks.
   While traversing the chart, calculate the frequency of occurrence for each noun phrase and verb.
   Goto 2.
6. Output the list of noun phrases with their frequencies of occurrence.
7. Prompt the user for inputting the cut-off frequency.
8. Calculate the occurrence lists and occurrence vectors.
9. Output the occurrence lists, occurrence vectors and the list of chunks used to generate these lists and vectors.
10. Stop.

Figure 4.8 Algorithm for NPer

Table 4.1 Output Files

Output File	Content
Chunk.txt	Contains generated chunks in the same order as the sentences of the input text.
NounPhrase.txt	Lists all noun phrases in alphabetical order with frequencies of occurrence.
Occurrence.txt	Contains occurrence lists for plotting occurrence graphs.
InterTerm.txt	Contains occurrence vectors for plotting co-occurrence graphs.
PhraseList.txt	Lists chunks that are used to generate occurrence lists and occurrence vectors.

4.3.1 Loading Root Forms of Verbs

Root forms of verbs are loaded after the main dictionary, the multiword dictionary and the grammar rules are loaded. The input file, “root.txt”, contains these root verbs. The entry-field structure of this file is similar to the main dictionary. This is done so that the method that is used to load and access the main dictionary can also be used to load and access these root verbs. A sample of this file is shown in Figure 4.9.

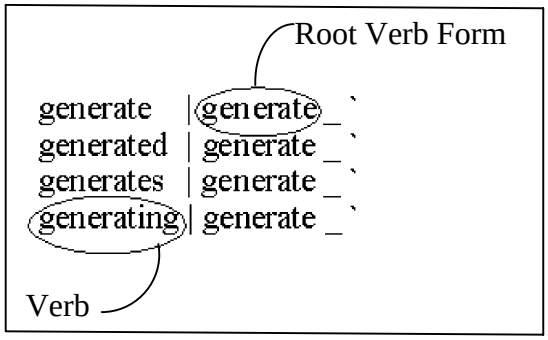

Figure 4.9 Root Forms of Verbs

4.3.2 Extracting the Chunks

As discussed earlier, a constituent in a chart has information stored about the starting node and the finishing node of the constituent. The length of a constituent can easily be calculated from this information. The chunk-extracting algorithm utilizes this node information to extract noun phrases, verbs and prepositions from the chart.

The chunk extractor starts at the first node and traverses through the chart, looking for the constituents of types: noun phrase, verb and preposition. This data is stored in a temporary array, in terms of the starting node and the ending node of the constituent. Since this traversing does not look for the longest possible constituent, many constituents extracted can be fragments of other constituents. For example, while traversing the chart shown in Figure 4.7, all the following constituents will be extracted, as noun phrases: “the host”, “host”, “processor”, “the host processor” and “host processor”. But only the longest simple noun phrase, “the host processor”, is desired. So the data stored in the temporary array, is searched for the longest constituent. This sorting is done depending on the starting and the finishing nodes of the constituents. The rules for comparing any two constituents to find the longest constituent are listed below.

Let S1 = the starting node of the first constituent.
F1 = the finishing node of the first constituent.
S2 = the starting node of the second constituent.
F2 = the finishing node of the second constituent.

- If S1=S2 and F1=F2, then both the constituents are same. Delete the second constituent and keep the first constituent. This happens because the grammar rules are redundant, i.e., the same constituent can be formed by satisfying more than one sub-set of grammar rules. This case is illustrated in Figure 4.10.

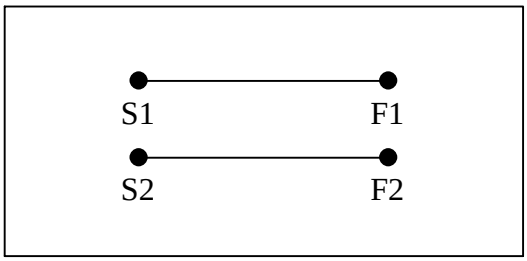

Figure 4.10 Case 1: S1=S2 and F1=F2

- If F1=S2=F2, then the second constituent is just one word and is a fragment of the first constituent. Delete the second constituent and keep the first constituent. See Figure 4.11.

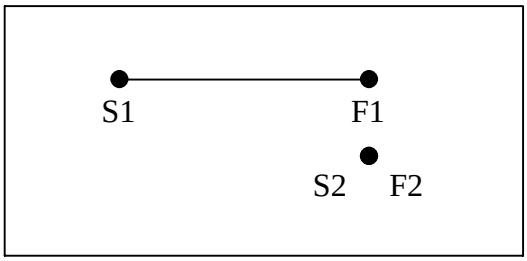

Figure 4.11 Case 2: F1=S2=F2

- If S1<S2 and F1=F2, then the second constituent is a fragment of the first constituent. Delete the second constituent and keep the first constituent. See Figure 4.12.

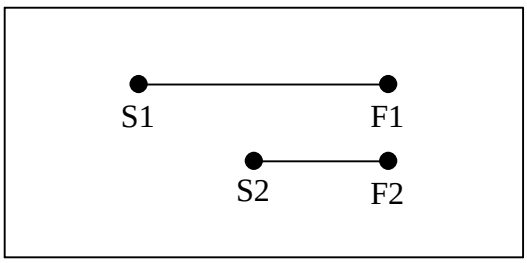

Figure 4.12 Case 3: S1<S2 and F1=F2

- If S1>S2 and F1=F2, then the first constituent is a fragment of the second constituent. Delete the first constituent and keep the second constituent. See Figure 4.13.

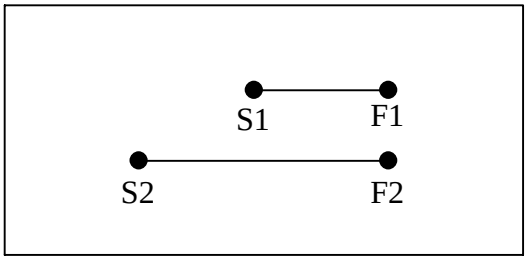

Figure 4.13 Case 4: S1>S2 and F1=F2

- If S1=S2 and F1<F2, then the first constituent is a fragment of the second constituent. Delete the first constituent and keep the second constituent. See Figure 4.14.

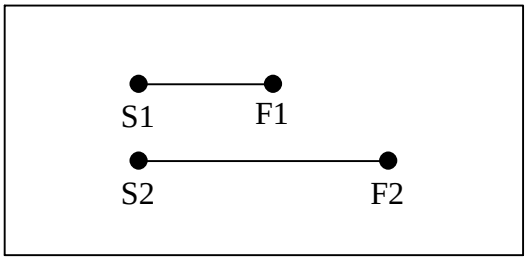

Figure 4.14 Case 5: S1=S2 and F1<F2

- If S1>S2 and F1<F2, then the first constituent is a fragment of the second constituent. Delete the first constituent and keep the second constituent. See Figure 4.15.

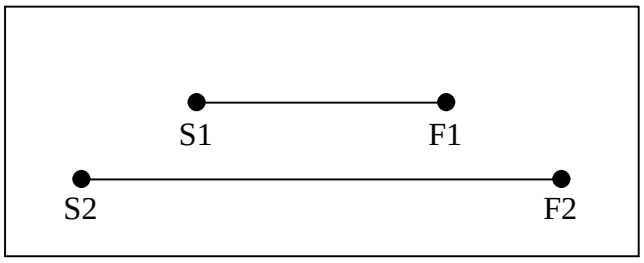

Figure 4.15 Case 6: S1>S2 and F1 < F2

- Other possibilities are discarded, as they can never exist because of the procedure of creating the chart.

After finding the longest simple noun phrases, the chart is traversed again with the available node information. The required constituents are identified from the chart and are output to the output file "Chunks.txt". The headword and its semantic category for each constituent are also output to the output file. The main constituent number (Figure 4.5) of the grammar rule forming a particular constituent identifies the headword of that constituent. If the headword happens to be a verb, its root form is output to the output file. The format of the chunks dumped into the output file is discussed in detail in Section 3.3.2 of this thesis. Looking at our previous example, all other parts are fragments of the noun phrase, "the host processor". So all the parts except "the host processor" will be deleted and the chunk will be generated for this particular noun phrase. Thus, NPer extracts the longest simple noun phrases. Figure 4.16 shows the chunks extracted for the example sentence in Figure 4.2.

1	1	np	processor	device	the	host processor
2	1	v	execute	action	nil	executes
3	1	np	program	value	a	program

Figure 4.16 Chunks for the Example Sentence

4.3.3 Calculating Frequency of Occurrence

For generating a list of noun phrases with their respective frequencies of occurrence, a previously developed program was used. This program, known as Vocaber, was written in Pascal and was used to find the frequencies of occurrence of words in a text. The program was converted into C++ and integrated with NPer to calculate the frequencies of occurrence of chunks. This program can be found as "NPFreq.h" and "NPFreq.cpp", among the NPer source files.

The program uses a binary tree to calculate and store the frequencies. Each node of the binary tree has two parts: the node value and the node frequency. The node value is the literal noun phrase or verb, and the node frequency is the frequency of occurrence of the chunk. Each chunk found by the chunk extractor is passed to this Vocaber. This class uses the algorithm shown in Figure 4.17 to accumulate frequencies of occurrence of noun phrases and verbs. After a document is fully analyzed, this class dumps an alphabetically ordered list of noun phrases, with their respective frequencies of occurrence, into the output file, "NounPhrase.txt". The frequencies of occurrence of noun phrases and verbs are utilized to calculate occurrence lists and occurrence vectors to draw the required graphs.

```
1. Get a chunk (noun phrase or verb) from the chunk extractor.
        If end of the document, goto 8.
2. If no binary tree exists,
        Root node name = chunk;
        Node frequency = 1;
        Goto 1.
   Else, goto 5.
3. Node name = chunk;
   Node frequency++;
   Goto 1.
4. Node frequency++;
   Goto 1.
5. Present node = root node;
6. If chunk < present node, goto the left of present node.
        If left node is not empty and left node == chunk, goto 4.
        If left node is not empty and left node != chunk,
                Present node = left node;
                Goto 6.
        If left node is empty, goto 3.
7. If chunk > present node, goto the right of present node.
        If right node is not empty and right node == chunk, goto 4.
        If right node is not empty and right node != chunk,
                Present node = right node;
                Goto 6.
        If right node is empty, goto 3.
8. Output the alphabetically ordered list of noun phrases with their frequencies of
   occurrence.
9. Stop.
```

Figure 4.17 Algorithm for Calculating Frequency of Occurrence

4.3.4 Generating Occurrence Lists and Occurrence Vectors

After the chunks are extracted and the frequencies of noun phrases and verbs are calculated, NPer generates occurrence lists and occurrence vectors to plot occurrence graphs and co-occurrence graphs, respectively. NPer prompts the user to give a cut-off frequency for generating lists and vectors. The chunks chosen for vector generation will have their frequencies of occurrence above this cut-off frequency. NPer searches for the high-frequency chunks and outputs the occurrence lists in "Occurrence.txt" output file, and occurrence vectors in "InterTerm" output file. NPer also generates the output file, "PhraseList.txt", with a list of these chunks. The occurrence lists can be imported into MS Excel to plot the occurrence graphs, whereas the occurrence vectors can be input to the Matlab program shown in Appendix B to plot the co-occurrence graphs.

4.4 Object Oriented Description of NPer

The object-oriented description of the parser has been described in detail in the ModelMaker [14] project. It is recommended that the reader refer the ModelMaker [14] project to understand the object-oriented description of the parser, before reading this section.

4.4.1 C++ Header and Source Files

NPer header and source files can be separated into two parts as shown in Table 4.2. The code was written in C++ for the Microsoft Windows 95/NT platform. Microsoft Visual C++ Developer Studio was used to compile the code. This program can be compiled for a different platform without changing much of its code.

Table 4.2 C++ Header and Source Files

The Parser		The Chunk Extractor	
Header Files	**Source Files**	**Header Files**	**Source Files**
Chart.h	Chart.cpp	Chunker.h	Chunker.cpp
Dictionary.h	Dictionary.cpp	NPFreq.h	NPFreq.cpp
Grammar.h	Grammar.cpp		
Lexer.h	Lexer.cpp		
MultiWord.h	MultiWord.cpp		
Parser.h	Parser.cpp		

- Parser.h and Parser.cpp

  These files contain the CParser class. The "Parser.cpp" also contains the C++ main function. The constructor of this class is responsible for the functioning of NPer.

- Chunker.h and Chunker.cpp

  These files form the core of the chunk extractor. They are needed to extract the chunks out of the parse chart. The CChunker and CChunk are the main classes of these files. CChunker class has member functions that extract the chart and generate the occurrence lists and occurrence vectors. Objects of CChunk class store the chunks in a linked list format.

- NPFreq.h and NPFreq.cpp

  These files contain Vocaber and are responsible for calculating frequencies of occurrence of noun phrases and verbs. The CNPFreq class is the main class of this file. A member function of this class generates a binary tree for storing noun phrases and verbs, and their respective frequencies of occurrence.

4.4.2 Main Class Descriptions

The primary class of NPer is of the CParser class. The CParser class uses the other main classes: CChart, CChunker, CDictionary, CGrammar, CLexer and CMWDictionary. The classes, CChart, CDictionary, CGrammar, CLexer and

CMWDictionary, are explained in detail in the ModelMaker [14] project. Booch class diagrams [5] are used to illustrate the class interactions. In these class diagrams, a dotted irregular outline denotes a class. A line from one class to another, with a dot on the first, indicates that the first class uses objects of the second class. An integer on the line shows the number of objects used.

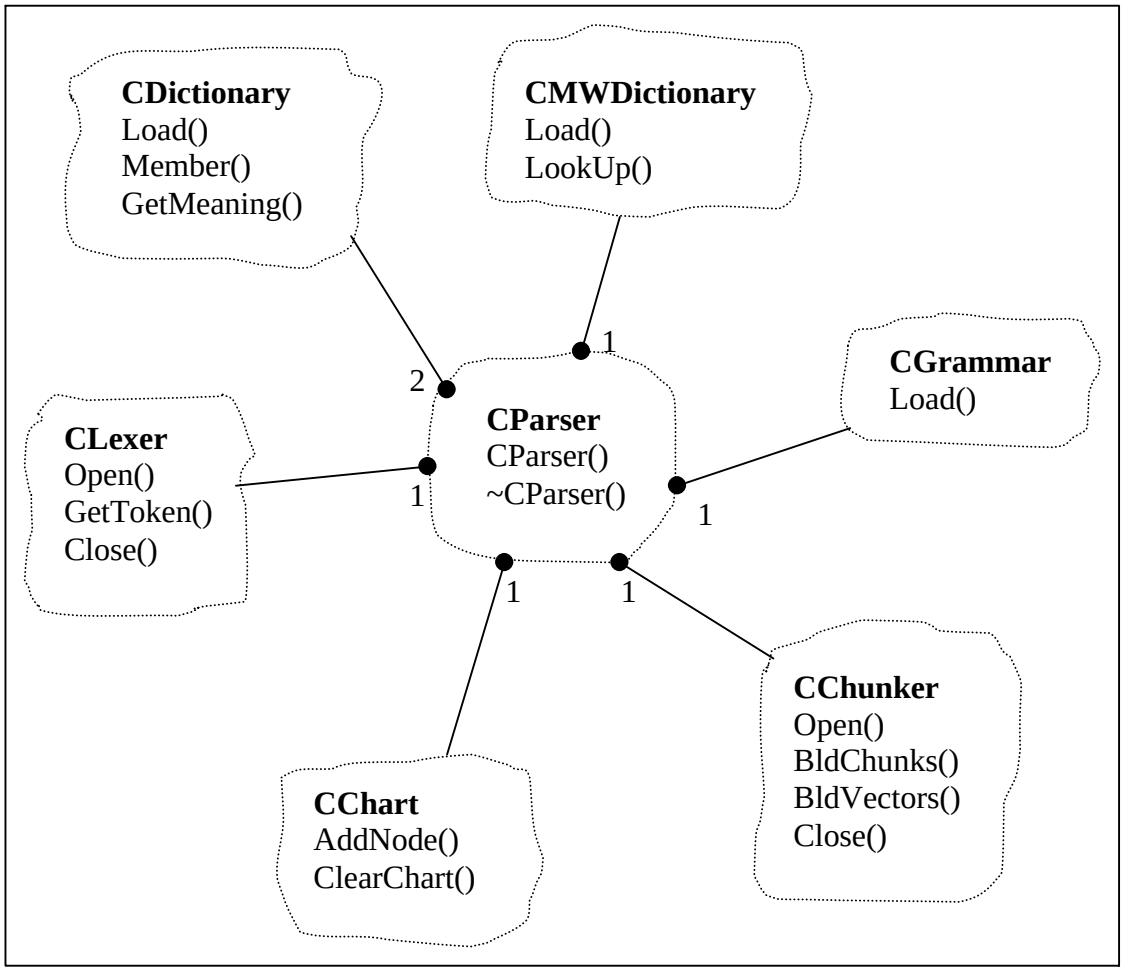

Figure 4.18 CParser Class Diagram

- CParser

  An object of this class is instantiated by the C++ main function. On its instantiation, the constructor of this class creates the CDictionary objects, the CMWDictionary object, the CGrammar object, the CLexer object, the CChart object and the CChunker object. For loading and storing the main dictionary and the root

forms of verbs, two CDictionary objects are used. The CMWDictionary object and the CGrammar object load and store the multiword dictionary and the grammar rules, respectively. The CParser uses the CLexer object to read the input file and form a token. The token is passed to the CChart object to be added to the chart. The CChunker object uses the chart from the CChart object, to extract the required chunks and vectors. The class diagram of CParser is shown in Figure 4.18.

- CChunker

As discussed earlier, the constructor of the CParser class creates the CChunker object. The CChunker object uses the BldChunks() member function to extract the chunks from the parse chart, according to the algorithm shown in Figure 4.8. The CChunker object creates the CChunk object to store these chunks. While extracting the chunks, the CChunker object creates two CNPFreq objects to calculate frequencies of occurrence of noun phrases and verbs. After the frequencies of occurrence are calculated, the BldVector() member function of this class generates occurrence lists and vectors required to plot the occurrence and the co-occurrence graphs. This class also has the Open() and the Close() member functions that open and close output files. The class diagram of CChunker is shown in Figure 4.19.

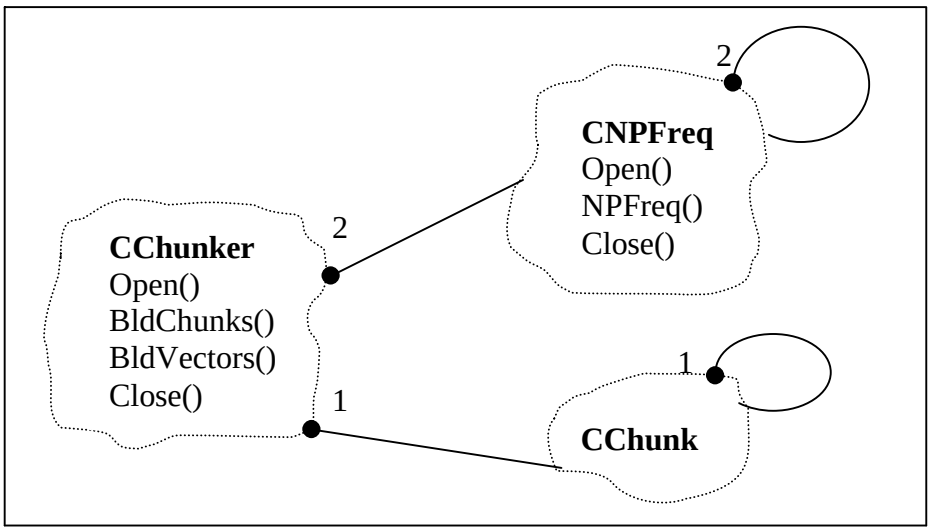

Figure 4.19 CChunker Class Diagram

- CChunk

  This class is used to store chunks in a linked list format. An object of this class will have member variables to store the chunk identification number, the sentence number in which the chunk occurs, the syntactic category of the chunk, the headword of the chunk, the semantic category of the headword, the determiner if any, the phrase or the word, and an object of the same class to point to the next chunk. The class diagram of this class is shown in Figure 4.20.

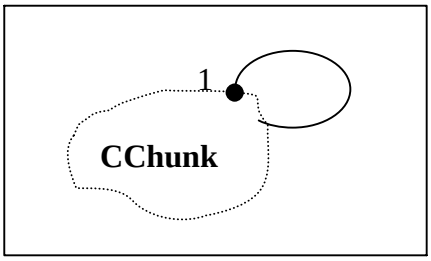

Figure 4.20 CChunk Class Diagram

- CNPFreq

  As shown in Figure 4.21, this class only creates two objects of the same class to form the binary tree. The only function of this class is to calculate frequencies of occurrence of noun phrases and verbs. This is done by the NPTree() member function according to the algorithm shown in Figure 4.17. This class also has the Open() and the Close() member functions that open and close the output file.

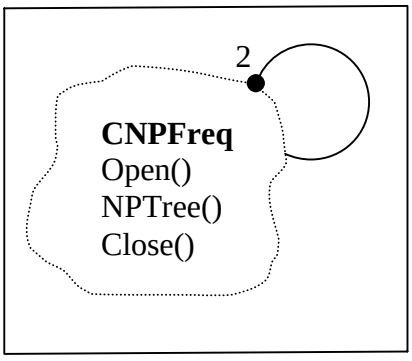

Figure 4.21 CNPFreq Class Diagram

CHAPTER 5: RESULTS

Two sets of documents were used to generate the results of this project. The training set consisted of three documents [3, 4, 16] for which the dictionary of NPer was defined. The test set consisted of another three documents [6, 9, 17]. The dictionary and the grammar were not updated for the test set. All the documents were descriptions of Direct Memory Access (DMA) controllers. This chapter discusses the results obtained by the implementation of NPer, and also defends the claims made in the previous chapters.

5.1 Recall and Precision

Two important information extraction metrics, recall and precision, were used to measure the performance of NPer. Table 5.1 shows the values of recall and precision for extraction of noun phrases from the test documents. Recall is measured using Equation 3.2 and precision is measured using Equation 3.3, from Section 3.3.8 of this thesis. The error in extracting noun phrases occurs because words in the dictionary sometimes have multiple lexical categories, such as “output”. “Output” can act as a verb, as in “The processors output the data”, and can also act as a noun, as in “The output of the processor”. In the first case, NPer recognizes “output” as a noun and as a verb, whereas in the second case, NPer recognizes “output” as a noun. Moreover, the parser used for NPer cannot handle some sentences, if their chart becomes too large. This is because some word patterns may satisfy multiple sets of rules. At present, NPer discards large sentences for which the number of constituents increases beyond 10000.

Table 5.1 Recall and Precision

Document	Number of Words	Total Sentences	Sample Sentences	% Recall	% Precision
Training Set					
US Patent 4729090	3386	136	136	90.94	86.14
DMA 8237A	5025	290	145	87.19	90.28
US Patent 5381538	11241	428	214	88.22	86.83
Test Set					
US Patent 4847750	12240	474	237	77.93	76.54
US Patent 5283883	13247	372	136	80.73	78.71
US Patent 5465340	11005	356	128	78.13	75.26

All the sentences in US Patent 4729090 [4] were considered for calculating recall and precision, as the number of sentences in the document was small. For other documents, every other sentence was considered. The values of recall and precision did not change significantly after about 75% of the sampled sentences were considered. Table 5.1 shows that the training set has greater recall and precision than the test set. This is because the dictionary was defined for the training set of documents. However the values for the test set are not that far from that of the first set. This suggests that the dictionary covers most of the words used to describe the domain. The dictionary coverage for test documents is shown in Table 5.2. The values can further be improved by defining the dictionary for a few more documents on DMA.

Table 5.2. Coverage of Dictionary

Document	Unknown Words	Total Distinct Words	% Coverage
US Patent 4847750	394	1199	67.14
US Patent 5283883	270	786	65.65
US Patent 5465340	258	893	71.11

The grammar rules also affect the recall and precision values. With the same dictionary, NPer has higher recall and precision than that of the noun phrase extractor

used for the ModelMaker [14] project, as shown in Table 5.3. This is because a new algorithm, which is discussed in Section 4.3.2, was developed which extracted all possible simple noun phrases and also new grammar rules were added to recognize noun phrases. Two documents from the training set were used for the comparison with ModelMaker [14]. The sampling was done as described above.

Table 5.3 Comparison

	ModelMaker's Noun Phrase Extractor		**NPer**	
Document	**% Recall**	**% Precision**	**% Recall**	**% Precision**
DMA 8237A	85.44	83.41	87.19	90.28
US Patent 4729090	80.37	67.88	90.94	86.14

5.2 Noun Phrases

As claimed in Section 3.2.5, extraction of noun phrases plays an important role in understanding a text as they define entities and concepts. A list of noun phrases in a design text represents all the components used to design the system being described. Figure 5.1 shows the block diagram of the DMA controller described in US Patent 4729090 [4]. Figure 5.2 shows a part of US Patent 4729090 [4] noun phrase list extracted by NPer.

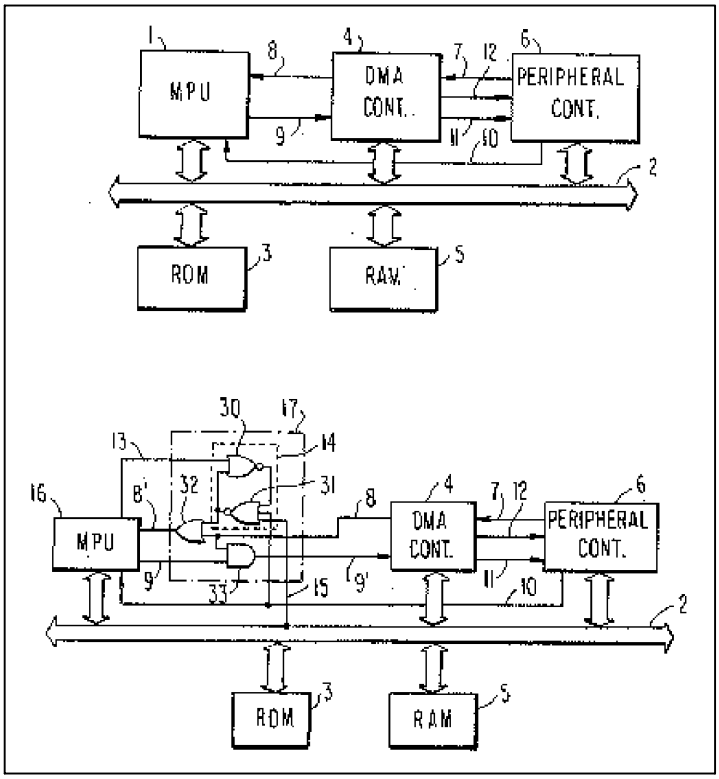

Figure 5.1 US Patent 4729090 Block Diagram [4]

mpu 1
bus 2
common bus 2
common data bus 2
rom 3
dma control circuit 4
dma control unit
dma control unit 4
dma controlling unit 4
memory (ram) 5
ram 5
control circuit 6
controlling circuit 6
peripheral control circuit 6
peripheral control unit 6
data transmission request signal 17
bus request signal 8
signal (acknowledgment signal) 9
signal 9
termination signal 10
transmission termination signal 10
transmission completion signal 11
start signal 12
dma request flag setting signal 13
dma start signal 13
flag setting signal 13
setting signal 13
signal 13
dma request flag 14
dma request flag register 14
flag register 14
initializing signal 15
microprocessor 16
microprocessor 16 (host processor)
microprocessor unit 16
mpu 16
mpu (microprocessor unit) 16
added circuit 17
newly added circuit 17
gates 30
gate 31
gate 32
gate 33

Figure 5.2 Noun Phrases Representing Components

As seen, the components in Figure 5.1 appear as distinct noun phrases in Figure 5.2. Also the carriers between the components appear as signals in the noun phrase list. The numbers at the end of the noun phrases in Figure 5.2 are the component and the carrier numbers in Figure 5.1. This convention is used in US Patents. Also, more than one noun phrase may represent the same component or carrier. For example, the noun phrases “dma control circuit 4”, “dma control unit”, “dma control unit 4” and “dma controlling unit 4”, all represent the same component. Thus a noun phrase list can help a modeler identify components and signals for modeling. Table 5.4 shows the coverage of devices and signals represented by noun phrases. The figures used were high-level block diagrams of DMA controllers described in the documents.

Table 5.4 Coverage of Devices and Signals Represented by Noun Phrases

Document	Total Devices in Figures	Devices not Listed
US Patent 4729090	21	0
DMA 8237A	36	1
US Patent 5381538	37	1
US Patent 4847750	61	2
US Patent 5283883	10	0
US Patent 5465340	23	1

Noun phrases also list many actions performed by these components. These actions can be directly translated into processes for modeling. Many such actions for US Patent 4729090 [4] are shown in Figure 5.3.

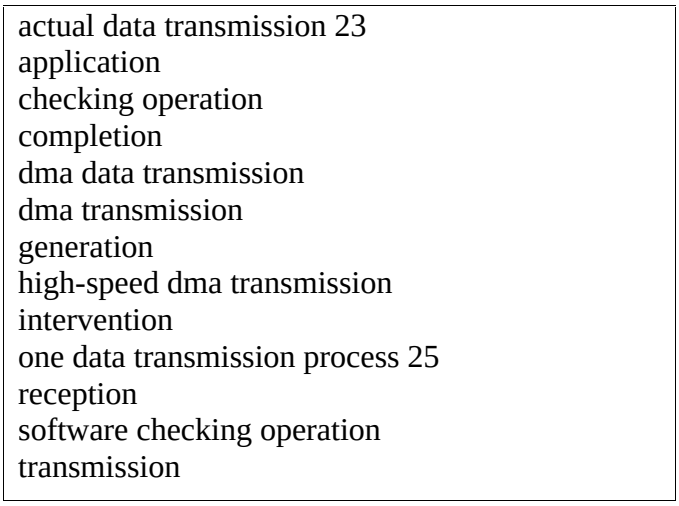

Figure 5.3 Noun Phrases Representing Actions

5.3 Frequency of Occurrence

As claimed in Section 3.3.5, the topic of a text can be predicted by noun phrases that have high frequencies of occurrence. Since all the training and the test documents deal with DMA controllers, we hypothesize that noun phrases with high frequencies of occurrence describe this domain. Table 5.5 shows examples of noun phrases with high frequencies of occurrence for the training set. They clearly represent the domain of the documents.

Table 5.5 Noun Phrases and Frequencies of Occurrence

Document	Total Noun Phrases	Frequency of occurrence	Noun Phrases
US Patent 4729090	825	37	host processor
		23	dma control unit 4
		19	bus usage
		17	peripheral controller
		16	bus
		15	common bus
		14	bus request signal 8
		14	dma controller
		13	bus 2
		13	data processing apparatus
		12	peripheral control unit 6
		11	mpu 16
		11	response
		10	data transmission
		10	dma transmission
DMA 8237A	1245	57	8237a
		23	channel
		15	dma service
		14	microprocessor
		13	dreq
		11	cpu
		11	data
		11	transfer
US Patent 5381538	2525	68	data information
		68	fifo register circuit 104
		63	information
		59	byte
		55	dma controller 52
		38	control information
		37	valid signal
		33	control signal generator circuit 102
		32	data
		32	fifo
		28	bus size
		22	memory
		21	circuit 103
		21	memories 16
		20	backup memory 108
		19	backup circuit 110
		19	dma control
		19	transfer

Noun phrases such as “data”, “input”, “output” and “signal”, also have high frequencies of occurrence. Such noun phrases indicate that a text is a digital design text, but do not contribute towards representing the subject of the text.

5.4 Chunk Occurrence Graphs

A design document is mostly divided into sections such as an abstract, background, detailed description and claims. What interests a modeler most is the detailed description as this part of a document typically has the greatest behavioral content. As claimed in Section 3.3.5, patterns in occurrence graphs can differentiate between different sections of a text. Occurrence graphs for the test documents were plotted for noun phrases and verbs having frequencies of occurrence above a cut-off frequency. The occurrence graph for US Patent 4729090 [4] is shown in Figure 5.4. The cut-off frequency and the list of chunks used to plot the occurrence graph are shown in Table 5.6. As indicated in the figure, sections can clearly be differentiated by breaks in groups of points on an occurrence graph; the largest section is the detailed description.

Table 5.6 Chunk List for Occurrence Graph for US Patent 4729090

Chunk Number	Chunk
1	host processor
2	dma controller
3	memory
4	data processing apparatus
5	provide
6	signal
7	bus request signal
8	set
9	apply
10	bus usage
11	bus
12	use
13	send
14	grant
15	receive
16	dma transmission
17	perform
18	peripheral controller
19	conventional data processing apparatus
20	couple
21	common
22	response
23	execute
24	data transmission
25	bus controller
26	flag register
27	generate
28	dma control unit 4
29	peripheral control unit 6
30	mpu 1
31	bus 2
32	bus request signal 8
33	signal 9
34	mpu 16

Cut-off Frequency of Occurrence: 7

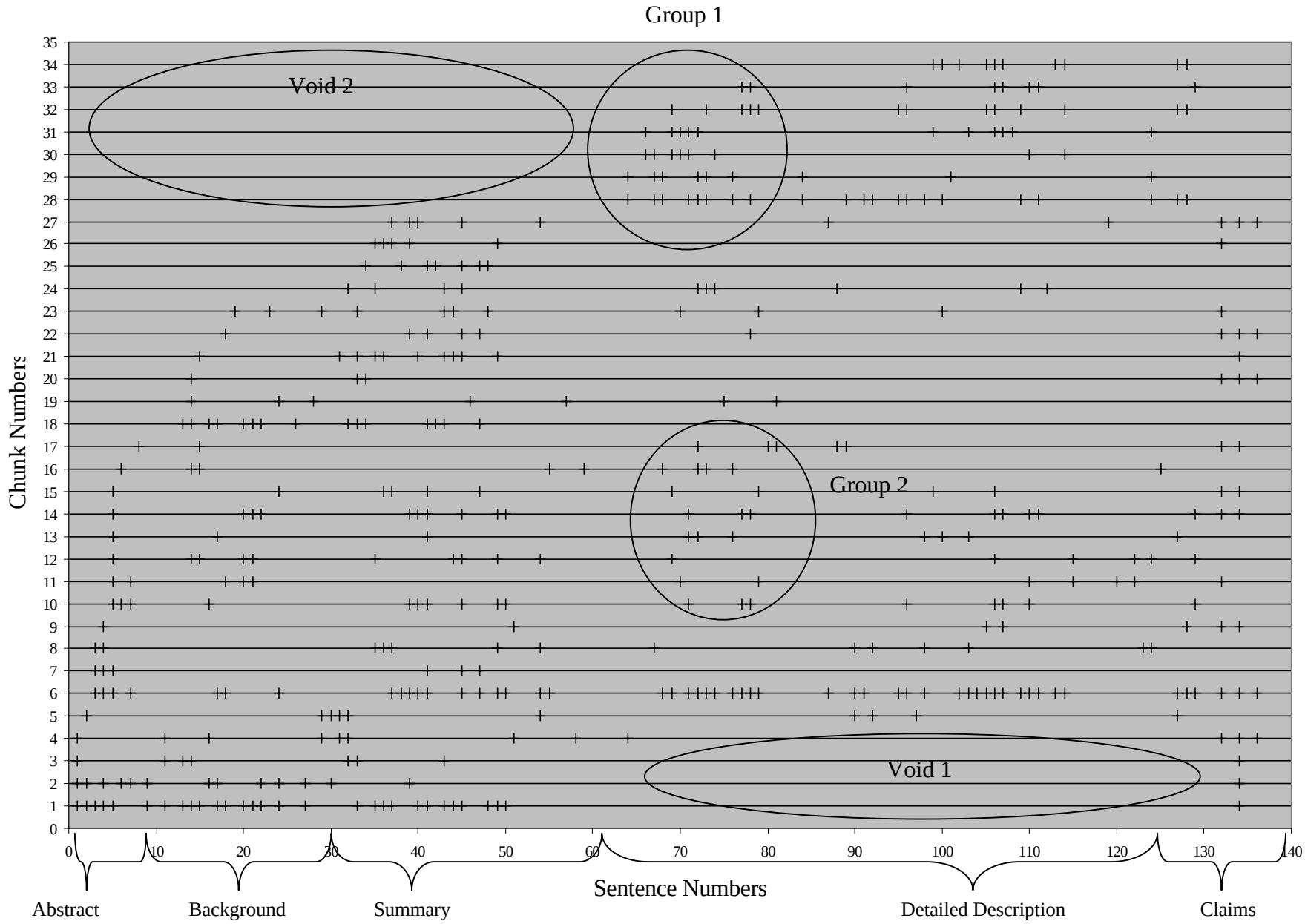

Figure 5.4 Occurrence Graph for US Patent 4729090

One can clearly observe in Figure 5.4 that chunk numbers 1 to 4 occur only in the abstract, the background and the summary sections, whereas chunk numbers 28 to 34 appear only in the detailed description. This observation shows that chunk numbers 1 to 4 represent components used as general terminology in the beginning of the US Patent. Chunk numbers 28 to 30 are specific components used for designing the DMA controller being discussed. It can be observed that chunk numbers 6 to 17 appear in the whole document. These chunks represent components, values and actions used commonly in a digital design document. One can also observe voids in the occurrence graphs. These voids indicate that the chunks do not appear in those particular sentences. For example, void 1 indicates that chunk numbers 1 to 4 are only being discussed in the abstract, the background, the summary and the claims. This void appears in the detailed discussion. Thus one can conclude that the components represented by chunks 1 to 4 are not that important from the modeler's view point, as they are only used for introduction and not for actual designing. Void 2 indicates that chunks 28 to 35 are specific components used for design, and so they do not appear in the beginning of the document. These components are important in defining the entities for modeling.

Groups of points indicate that chunks in that group appear very close to each other in the text. For example group 1 in Figure 5.4 suggest that chunk numbers 28 to 33 are related. This can clearly be shown from the sentences 65 to 79 of US Patent 4729090 [4] which are listed in Figure 5.5. Emphasis is added to show chunks in the sentences.

- The data processing apparatus comprises a microprocessor unit (MPU) 1, a program memory (ROM) 3, a *DMA control unit 4*, a read/write memory (RAM) 5 and a *peripheral control unit 6*.
- These units and memories are all connected to a common *bus 2*.
- The *MPU 1* fetches instructions according to a program from a program ROM 3 via the *bus 2*.
- Set commands from the *DMA control unit 4* and the *peripheral control unit 6* allow the *MPU 1* data transmission between the *peripheral control unit 6* and the RAM 5.
- Thereafter when the *peripheral control unit 6* requires a *DMA transmission*, a request is made to the *DMA control unit 4* using a data transmission request signal 7.
- Immediately after *receiving* this signal, the DMA controlling unit 4 requests the *MPU 1* for the *use* of the *bus 2*, by means of a *bus request signal 8*.
- At this time, if the *MPU 1* is executing a program and is using the *bus 2*, the *MPU 1* cannot release the *bus* immediately.
- Consequently, upon terminating the processing, the *MPU 1* releases the *bus 2* and *sends* a *bus usage grant* signal (acknowledgment signal) 9 to the *DMA control unit 4*.
- The *DMA control unit 4 sends* a start signal 12 to the *peripheral control unit 6* for designating a *DMA transmission*, whereby the *DMA control unit 4 performs* data transmission between the *peripheral control unit 6* and the RAM 5 through the *bus 2*.
- Upon completion of data transmission, the *DMA control unit 4* issues a transmission completion signal 11 to the *peripheral control unit 6* to inform the completion of *DMA transmission* and then disenables the *bus request signal 8*.
- The DMA control unit 6 informs the *MPU 1* of the completion of data transmission by means of a termination signal 10.
- As described above, the conventional data processing apparatus in FIG. 1 requires a long period of time until the *DMA control unit 4 sends* the DMA start signal 13 to the *peripheral control unit 6* after the *peripheral control unit 6* requests the *DMA transmission*.
- Particularly, a very long period of time is necessary from requesting a *bus usage* by means of the *bus request signal 8* till reception of the *bus usage grant signal 9*.
- This is because the microprocessor 1 produces the *bus usage grant signal 9* in response to the *bus request signal 8* from the *DMA control unit 4*.
- Further the microprocessor 1 must execute checking operations to check whether the *bus request signal 8* is *received* or not and to check whether the *bus* should be released or not after the check of the *bus request signal 8*.

Figure 5.5 US Patent 4729090 Sentence Numbers 65 to 79

Similarly, the chunks in two groups appearing for the span of same sentence numbers are also related. For example chunks in group 1 and group 2 in Figure 5.4 are related as they both span sentence numbers 65 to 79. This can also be seen from Figure 5.5. Specific relationships between these chunks can be studied by drawing a co-

occurrence graph for these chunks. The occurrence graphs for other test documents can be found in Appendix C of this thesis.

5.5 Chunk Co-occurrence Graphs

Two chunks co-occur if they occur in the same sentence. The frequency of co-occurrence was calculated using Equation 3.1 in Section 3.3.6 of this thesis. Chunks can have a frequency of co-occurrence greater than one per sentence. For example, in the following sentence "host processor" occurs twice and "common bus" occurs once. So their frequency of co-occurrence for that sentence is two.

The apparatus further includes a host processor for executing a program according to instructions, and a common bus for coupling together the memory, the peripheral controller and the host processor.

As claimed in Section 3.3.7, co-occurrence graphs can represent the degree of relationship between two chunks. A co-occurrence graph of US Patent 4729090 [4] is shown in Figure 5.6. Table 5.7 lists the chunks used for plotting the co-occurrence graph and Table 5.8 lists the chunk pairs appearing in Figure 5.6. As seen in Figure 5.6, chunk number 7, "host processor", and chunk number 15, "common bus", have a co-occurrence frequency of 19. This suggests that both devices are connected and use each other for their functioning. This can be seen in the following set of sentences extracted from US Patent 4729090 [4]. Emphasis is added to show chunks in the sentences.

Table 5.7 Chunk List for Co-occurrence Graph for US Patent 4729090

Chunk Number	Chunk
1	bus
2	bus 2
3	bus controller
4	bus request signal
5	bus request signal
6	bus usage
7	common bus
8	conventional data processing apparatus
9	data processing apparatus
10	data transmission
11	dma control unit 4
12	dma controller
13	dma transmission
14	flag register
15	host processor
16	memory
17	mpu 1
18	mpu 16
19	peripheral control unit 6
20	peripheral controller
21	response
22	signal 9
23	apply
24	couple
25	execute
26	generate
27	grant
28	perform
29	provide
30	receive
31	send
32	set
33	use

Cut-off Frequency of Co-occurrence: 10

Table 5.8 Chunk Pairs for US Patent 4729090

Co-ordinate Pair	Chunk	Chunk
1-15	bus	host processor
1-27	bus	grant
1-33	bus	use
6-27	bus usage	grant
7-15	common bus	host processor
9-24	data processing apparatus	couple
11-19	dma control unit 4	peripheral control unit 6
15-20	host processor	peripheral controller
15-27	host processor	grant
15-33	host processor	use
21-24	response	couple
21-26	response	generate
21-27	response	grant
23-24	apply	couple
23-27	apply	grant
24-26	couple	generate
24-27	couple	grant
24-30	couple	receive
26-27	generate	grant
26-30	generate	receive

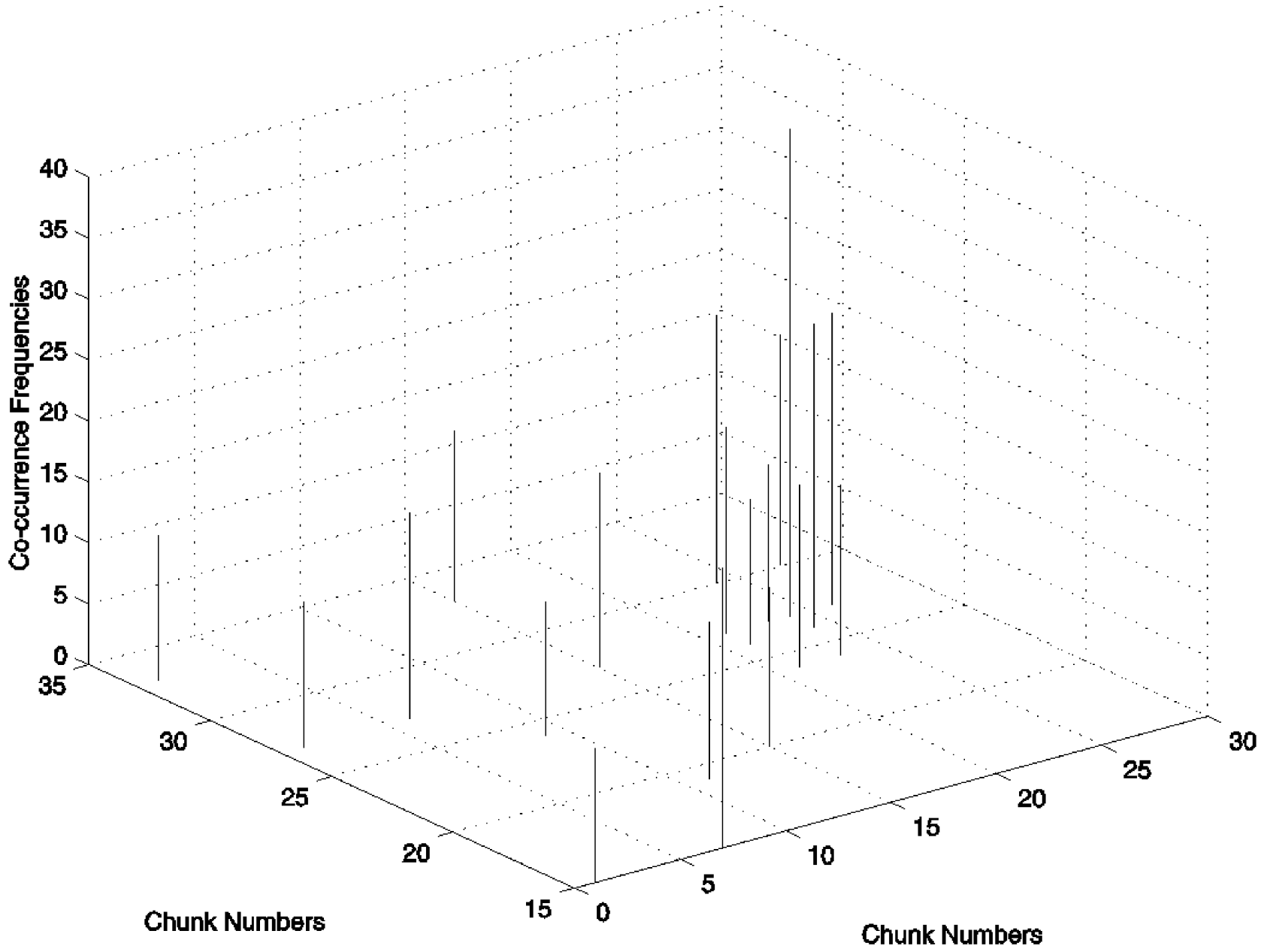

Figure 5.6 Co-occurrence Graph for US Patent 4729090

- The DMA transmission must be performed in a period when the *common bus* is in an idle state, that is, when the *host processor* does not use the *common bus*.
- The apparatus further includes a *host processor* for executing a program according to instructions, and a *common bus* for coupling together the memory, the peripheral controller and the *host processor.*
- The "host processor" sets a flag register when the *host processor* does not need to use the *common bus* or after the *host processor* has terminated a data transmission using the *common bus*.
- The *host processor* receives an output of the flag register and electrically cuts itself off from the *common bus* when the flag register is being set.
- The bus usage grant signal may be generated when the *host processor* electrically cuts itself off from the *common bus* or after the *common bus* has been electrically separated from the *host processor.*
- As the result the peripheral controller executes a data transmission to or from the memory by using the *common bus* without intervention of the *host processor.*
- At this time, the *host processor* may also execute a program which does not need to use the *common bus*.
- According to the present invention, the *host processor* does not generate the bus usage grant signal in response to a bus request signal from the bus controller, but generates it by itself when the *host processor* does not need to use the *common bus* or after a data transmission processing using the *common bus* has been terminated.

Figure 5.7 Set 1: US Patent 4729090 Sentences

Similarly, chunk number 15, "host processor", and chunk number 27, "grant", suggest that either the "host processor" is granting something to another device or the "host processor" is being granted something by another device. In this case, the "host processor" grants another device, the permission to use a bus, which is well illustrated by the following sets of sentences.

- As the result, if the *host processor* does not need to use the bus, the bus usage right is *granted* at this time to the peripheral controller.
- On the other hand, if the *host processor* is using the bus or needs to use the bus at this time, the bus usage right is not *granted* to the peripheral controller until its *host processor* has finished the use of the bus.
- The DMA controller *grants* the bus usage right to the peripheral controller according to the result of the aforementioned checking operation by the *host processor.*

Figure 5.8 Set 2: US Patent 4729090 Sentences

Also, chunk number 15, “host processor, chunk number 33, “use”, co-occur in the following sentences. Note that all forms of a given verb are recognized.

- The *host processor* receives the bus request signal and sends a bus usage grant signal to the added circuit when the *host processor* does not need to *use* a bus or after a processing *using* the bus has been completed, and electrically cuts off itself from the bus.
- The DMA transmission must be performed in a period when the common bus is in an idle state, that is, when the *host processor* does not *use* the common bus.
- As the result, if the *host processor* does not need to *use* the bus, the bus usage right is granted at this time to the peripheral controller.
- On the other hand, if the *host processor* is *using* the bus or needs to *use* the bus at this time, the bus usage right is not granted to the peripheral controller until its *host processor* has finished the *use* of the bus.
- The *host processor* sets a flag register when the *host processor* does not need to *use* the common bus or after the host processor has terminated a data transmission *using* the common bus.

Figure 5.9 Set 3: US Patent 4729090 Sentences

A peculiarity of US Patents that effect co-occurrence needs to be mentioned. The last section of a US Patent is a list of claims. Each claim in a patent is ended by a semi-colon instead of a period. So NPer does not consider an end of a claim as an end of a sentence. This results in a high frequency of co-occurrence for the chunks co-occurring in the claims section of the document. Such chunks with high frequencies of co-occurrence are omitted from the co-occurrence graphs shown. This does not affect the results since modeling information is generally found in the detailed description section of a document. Thus, a high co-occurrence frequency between two noun phrases suggests that both the devices are directly connected or interact, and a high co-occurrence frequency between a noun phrase and a verb suggest the behavior of the device. The higher the co-occurrence frequency, the higher the probability of the chunks being related. The co-occurrence graphs for other test documents can be found in Appendix C of this thesis.

CHAPTER 6: CONCLUSIONS

The results of this project indicate that the noun phrases, the chunk frequencies of occurrence, the occurrence graphs and the co-occurrence graphs give useful information about the subject, the hardware components and the behavior of a system described by a natural language specification document. This chapter summarizes the capabilities and the limitations of the NPer system and suggests directions for future work.

6.1 System Capabilities

NPer is able to analyze a large number of system description sentences. The dictionary contains about 3000 words and the grammar contains 128 rules. The values of recall and precision show that NPer is very efficient in extracting noun phrases from documents describing DMA controllers, even when the dictionary is not updated for a specific document.

In digital systems descriptions, a noun phrase can represent a device or value, such as a processor, instruction, and signal. Noun phrases having head nouns derived from verbs represent actions. This can be seen in results generated using NPer. A modeler can use this information in building entities and processes for a model of a system. Also a modeler can use NPer for sorting out documents on a particular domain by studying frequencies of occurrence of noun phrases.

A modeler is mostly interested in the detailed description section of a document as it typically has the greatest behavioral content. Occurrence graphs generated using NPer give a clear indication of different sections in a text such as abstract, background, summary, detailed description and claims. Also, a modeler can use NPer for differentiating between specific and general components, to help a modeler efficiently build entities from specific components. Groups and voids in an occurrence graph represent whether the components in those groups are general terminology or specific

components, depending in what section of the document they occur. Also, information about the connection and the working of a particular device is useful to a modeler to define entities and processes more accurately, and also to define connections between different entities. This information is seen in co-occurrence graphs generated using NPer.

At present, NPer works on a Windows 95/NT platform, but can be easily compiled on a different platform without changing much of its code. Since the code is object-oriented, the functions and classes can be integrated into different tools to extract information for future research.

6.2 System Limitations

Due to ambiguity in the English language and a limited set of grammar rules, NPer is not able to recognize all noun phrases. The error in extracting noun phrases occurs because words in the dictionary sometimes have multiple lexical categories. Moreover, the parser used for NPer cannot handle some sentences, if their charts become too large. This is because some word patterns may satisfy multiple sets of rules. Also, the occurrence graphs and the co-occurrence graphs were drawn using MS Excel and a Matlab program respectively. NPer is not capable of drawing these graphs from chunk occurrence lists and vectors.

6.3 Directions for Future Work

New words in the dictionary and new grammar rules can be added to increase the vocabulary handling capacity of NPer. Also, words for other domains can be added. As shown in Table 5.3, NPer has better recall and precision than that of ModelMaker's noun phrase extractor [14]. NPer can be integrated into the ModelMaker [14] tool to increase the recall and the precision of noun phrase extraction by ModelMaker [14]. Moreover, searches on verbs and semantic categories can also be implemented into ModelMaker [14], as NPer can supply chunks for such searches. Chunk occurrence lists and vectors can be used for searches with respect to frequencies of occurrence and frequencies of co-

occurrence. This would help modeler look for related chunks that are helpful in obtaining information about connections and functions of devices. NPer can be made capable of drawing the occurrence graphs and the co-occurrence graphs. Also a class can be written that takes occurrence graphs as input and outputs different groupings of chunks to aid a modeler understand relationships between chunks. Such a class can also be used to filter out different sections of a document, and provide a modeler with only the detailed description. This would increase accessibility to the information extracted for modeling.

BIBLIOGRAPHY

[1] Abbott, R. J., "Program Design by Informal English Descriptions", Communications of the ACM, Volume 26 (11), Association for Computing Machinery, New York, 1983.

[2] Aho, A. V. and Ullman, J. D., The Theory of Parsing, Translation and Compiling, Prentice Hall, Englewood Cliffs, New Jersey, 1973.

[3] Amini, N., Boury, B. F. and Lohman, T. J., "DMA Controller Including a FIFO Register and a Residual Register for Data Buffering and having Different Operating Modes", United States Patents, US Patent Number 5381538, 1995.

[4] Baba, E., "DMA System Employing Plural Bus Request and Grant Signals for Improving Bus Data Transfer Speed", United States Patents, US Patent Number 4729090, 1988.

[5] Booch, G., Object Oriented Analysis and Design with Applications (Second Edition), Benjamin/Cummings Publishing Company, Redwood City, California, 1994.

[6] Creedon, T., O'Neill, E. G. and O'Connell, A., "Direct Memory Access Controller Handling Exceptions During Transferring Multiple Bytes in Parallel", United States Patents, US Patent Number 5465340, 1995.

[7] Cyre, W. R., "Design of Restricted Sublanguage", Proceedings of Theoretical Approaches to Natural Languages Understanding Workshop, Halifax, Novia Scotia, 1985.

[8] Cyre, W. R., "Extracting Design Models from Natural Language Descriptions", Tenth International Conference on Industrial Engineering Applications of Artificial Intelligence Expert Systems, Atlanta, Georgia, 1997.

[9] Daniel, R. A., "Peripheral DMA Controller for Data Acquisition System", United States Patents, US Patent Number 4847750, 1989.

[10] Dong, A., Agogino, A. M., Moore F. and Woods, C., "Preprints Advances in Formal Design Methods for CAD", Managing Design Knowledge in Enterprise-

wide CAD, Key Center of Design Computing, University of Sydney, Sydney, Australia, 1995.

[11] Dong, A. and Alice, A. M., "Text Analysis for Constructing Design Representations", Artificial Intelligence in Design '96, Kluwer Academic Publishers, Dordrecht, Netherlands, 1996.

[12] Grefenstette, G., "Sextant: Extracting Semantics from Raw Text", Integrated Computer-Aided Engineering, Volume 1 (6), John Wiley and Sons, New York, 1994.

[13] Grishman, R. and Sundhein, B., "Design of the MUC-6 Evaluation", Proceedings of the Sixth Message Understanding Conference, Morgan Kaufmann Publishers, San Mateo, California, 1995.

[14] Gunawan, A. I., ModelMaker: A Tool for Rapid Modeling from Device Descriptions, Master's Thesis, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, 1998.

[15] Harris, M. D., Introduction to Natural Language Processing, Reston Publishing Company, Reston, Virginia, 1985.

[16] Intel, Peripherals, Intel Corporation, Mt. Prospect, Illinois, 1990.

[17] Mishler, C., "Method and Direct Memory Access Controller for Asynchronously Reading/Writing Data from/to a Memory with Improved Throughput", United States Patents, US Patent Number 5283883, 1994.

[18] Rose, C. P. and Lavie, A., "An Efficient Distribution of Labor in a Two Stage Robust Interpretation Process", Second Conference on Empirical Methods in Natural Language Processing, Providence, Rode Island, 1997.

[19] Vadera, S. and Meziane, F., "From English to Formal Specifications", The Computer Journal, Volume 37 (9), Oxford University Press, Newark, New Jersey, 1994.

APPENDIX A: NPer USER MANUAL

Introduction

This User Manual is designed to help device modelers in using NPer for information extraction. The manual discusses system requirements for NPer, listing all input files needed to run NPer, and the output files generated by NPer. For more detailed information about any particular feature of NPer, refer to the body of this thesis.

Application Files

NPer runs on a Microsoft Windows 95/NT platform. It can be easily complied for a different platform without changing much of its code. All the files essential for proper functioning of NPer are listed below. It is necessary to place all these files in the same directory.

- NPer.exe

  This is the main executable file for NPer. Invoking NPer with this file is discussed later.

- Rdict.txt

  This file contains the main vocabulary dictionary used by NPer. All the vocabulary words are listed in a different line in an alphabetical order. Each entry will have a word and one or more pair of lexical and semantic categories for the word. Each pair of lexical and semantic categories is followed by the pipe, "|", separator. A grave accent, "`", identifies the end of an entry. A sample format of this file is shown in Figure A.1. An asterisk, "*", instead of the lexical category means that the word also occurs as the last word of a phrase defined in the multiword dictionary, "mdict.txt".

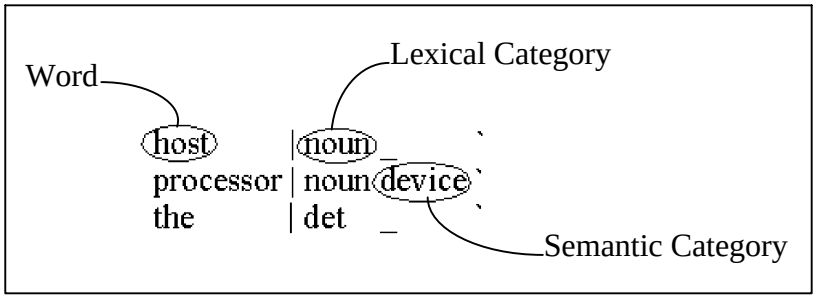

Figure A.1 Main Dictionary

- Mdict.txt

  This file contains the multiword dictionary for the use of NPer. The structure of the entry field of this file is the same as that of the main dictionary. However, for every entry in this file there must be a corresponding asterisk on the last word of the entry in the main dictionary file. A sample format of this file is shown in Figure A.2.

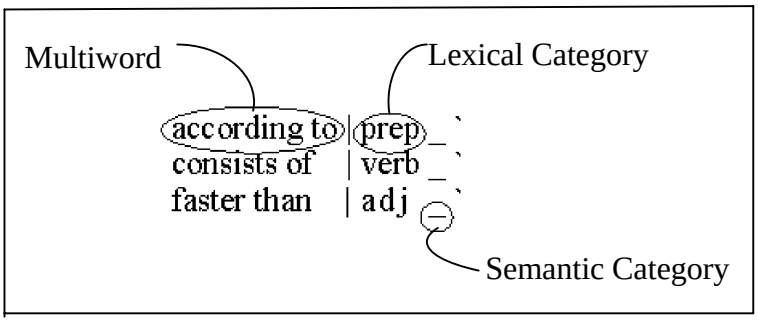

Figure A.2 Multiword Dictionary

- Gram.txt

  This file contains the grammar rules used by NPer to parse English sentences. Each line represents one grammar rule, and includes the rule identifier, the number of constituents forming the rule, the number of the main constituent, the syntactic category defined by the rule, and the list of constituents forming the rule. For example, rule "np1" corresponds to the grammar rule

  np → det head

The constituents are listed in the order required to form the new constituent. The main constituent number selects the word whose semantic category determines the semantic category of the result constituent. A sample format of this file is shown in Figure A.3.

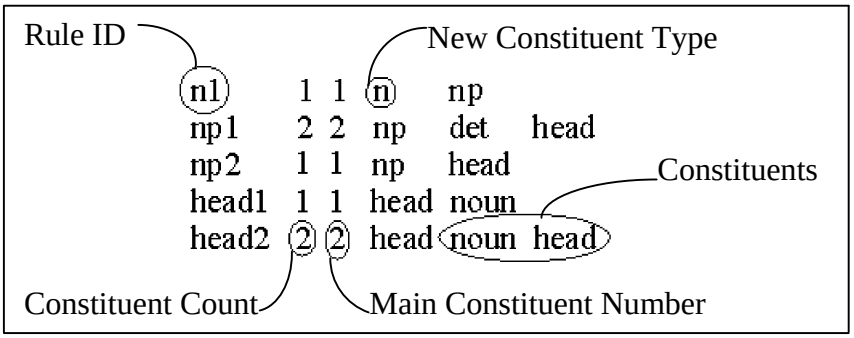

Figure A.3 Grammar Rules

- Root.txt

  This file contains the root forms of verbs used by NPer. Each entry has the verb with its root form listed. The verb and its root form are separated by the pipe, "|", separator, and the end of the entry is marked by an underscore, "_", sign and a quotation mark. A sample format of this file is shown in Figure A.4.

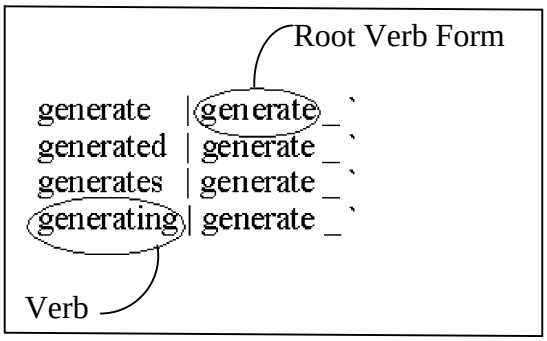

Figure A.4 Root Forms of Verbs

NPer Input Files

NPer takes an English specification document, in the standard ASCII text format, as its input. No special formatting of the input file is necessary. However, graphics and

tables, should be deleted before using it as an input to NPer. Figure A.5 shows a sample input file.

A data processing apparatus employing a DMA system has a newly added circuit. The added circuit is provided between the host processor and the DMA controller. The host processor produces a bus request signal into the register of the added circuit. The added circuit applies the bus request signal to the host processor. The host processor receives the bus request signal and sends a signal to the added circuit. The DMA controller sends a requests to the added circuit when a DMA transmission is required.

Figure A.5 Sample Input File

Invoking NPer

To invoke NPer, open a DOS command prompt window and go to the appropriate directory that stores, "NPer.exe", "rdict.txt", "mdict.txt", "gram.txt" and "root.txt", application files for NPer. Then type

NPer <input_file_name>

where "<input_file_name>" is the name of the natural language specification document that describes the system being modeled. If the input file is not in the same directory as the application files for NPer, then the entire path name for the input file is required. For example, if the input file name is "4729090.txt", enter

NPer 4729090.txt

on the DOS command prompt. After NPer generates the chunks, it prompts the user for the cut-off frequency. Only the terms having frequencies above the cut-off frequency will be used to generate vectors for plotting the occurrence and calculating the frequencies of co-occurrence. The occurrence and the co-occurrence graphs are plotted by importing these vectors into MS Excel.

NPer Output Files

There are five output files created by NPer. All these files are in ASCII text format.

- Chunk.txt

  "Chunks.txt" stores all the generated chunks in the same order as the sentences of the input text. Figure A.6 shows "Chunks.txt" for input file in Figure A.5.

1 | 1 | np | apparatus | device | a | data processing apparatus
2 | 1 | v | employ | action | nil | employing
3 | 1 | np | system | device | a | dma system
4 | 1 | v | have | behavior | nil | has
5 | 1 | np | circuit | device | a | newly added circuit
6 | 2 | np | circuit | device | the | added circuit
7 | 2 | v | provide | action | nil | provided
8 | 2 | prep | between | nil | nil | between
9 | 2 | np | processor | device | the | host processor
10 | 2 | np | controller | device | the | dma controller
11 | 3 | np | processor | device | the | host processor
12 | 3 | v | produce | action | nil | produces
13 | 3 | np | signal | value | a | bus request signal
14 | 3 | prep | into | nil | nil | into
15 | 3 | np | register | device | the | register
16 | 3 | prep | of | nil | nil | of
17 | 3 | np | circuit | device | the | added circuit
18 | 4 | np | circuit | device | the | added circuit
19 | 4 | v | apply | action | nil | applies
20 | 4 | np | signal | value | the | bus request signal
21 | 4 | prep | to | nil | nil | to
22 | 4 | np | processor | device | the | host processor
23 | 5 | np | processor | device | the | host processor
24 | 5 | v | receive | action | nil | receives
25 | 5 | np | signal | value | the | bus request signal
26 | 5 | v | send | action | nil | sends
27 | 5 | np | signal | value | a | signal
28 | 5 | prep | to | nil | nil | to
29 | 5 | np | circuit | device | the | added circuit
30 | 6 | np | controller | device | the | dma controller
31 | 6 | v | send | action | nil | sends
32 | 6 | v | request | action | nil | requests
33 | 6 | prep | to | nil | nil | to
34 | 6 | np | circuit | device | the | added circuit
35 | 6 | np | transmission | action | a | dma transmission
36 | 6 | v | require | behavior | nil | required

Figure A.6 “Chunk.txt” Output File

- NounPhrase.txt

  “NounPhrase.txt” lists all the noun phrases in alphabetical order, with their respective frequencies of occurrence. Figure A.7 shows “NounPhrase.txt” for the input file in Figure A.5.

5	added circuit
3	bus request signal
1	data processing apparatus
2	dma controller
1	dma system
1	dma transmission
4	host processor
1	newly added circuit
1	register
1	signal

Figure A.7 "NounPhrase.txt" Output File

- PhraseList.txt

"PhraseList.txt", lists the phrases and words that are used to generate the occurrence lists and occurrence vectors. Note that this file is different from "NounPhrase.txt" because along with noun phrases used to generate lists and vectors, this file will also contain verbs used to generate lists and vectors. Figure A.8 shows "PhraseList.txt" for input file in Figure A.5.

5	added circuit
3	bus request signal
1	data processing apparatus
2	dma controller
1	dma system
1	dma transmission
4	host processor
1	newly added circuit
1	register
1	signal
1	apply
1	employ
1	have
1	produce
1	provide
1	receive
1	request
1	require
2	send

Figure A.8 "PhraseList.txt" Output File

- Occurrence.txt

  "Occurrence.txt" contains occurrence lists for plotting the occurrence graphs between the chunks and the sentences in which they occur. Figure A.9 shows "Occurrence.txt" for the input file in Figure A.5. The order of these occurrence lists is the same as Figure A.8. For example, "host processor" has "2 3 4 5" occurrence list.

```
2 3 4 5 6
3 4 5
1
2 6
1
6
2 3 4 5
1
3
5
4
1
1
3
2
5
6
6
5 6
```

Figure A.9 "Occurrence.txt" Output File

- InterTerm.txt

  "InterTerm.txt" contains occurrence vectors for calculating frequencies of co-occurrence. Co-occurrence frequencies can then be used to plot co-occurrence graphs. Figure A.10 shows "InterTerm.txt" for the input file in Figure A.5. The order of these occurrence vectors is the same as Figure A.8. For example, "host processor" has "0 1 1 1 1 0" occurrence vector.

```
0 1 1 1 1 1
0 0 1 1 1 0
1 0 0 0 0 0
0 1 0 0 0 1
1 0 0 0 0 0
0 0 0 0 0 1
0 1 1 1 1 0
1 0 0 0 0 0
0 0 1 0 0 0
0 0 0 0 1 0
0 0 0 1 0 0
1 0 0 0 0 0
1 0 0 0 0 0
0 0 1 0 0 0
0 1 0 0 0 0
0 0 0 0 1 0
0 0 0 0 0 1
0 0 0 0 0 1
0 0 0 0 1 1
```

Figure A.10 “InterTerm.txt” Output File

APPENDIX B: MATLAB CODE

The following Matlab code calculates frequencies of co-occurrence of terms and plots the inter-term dependencies graph. The input file for this code will be "InterTerm.txt", generated by NPer. This code needs the Matlab software loaded, and is platform independent. To execute this program just type "InterTerm" on the Matlab prompt. Be sure to include the input vector file in the same directory that is being used to run Matlab.

```
% Matlab File "InterTerm.m"

% Clear all the previous values and graphs.
Clear all;
Close all;

% Input the total number of phrases for which the vectors are to be read.
TPN=input('The Total # of Phrases: ');

% Input the total number of sentences in the text.
TSN=input('The Total # of Sentences: ');

% Input the cut-off frequency to select the phrases to be plotted.
LIM=input('The Cut-off Co-occurrence Frequency: ');

% Open the "InterTerm.txt" input file.
fid=fopen('InterTerm.txt', 'r');

% Read the vectors from the input file.
a=fscanf(fid, '%d', [TSN,TPN]);
```

```
% Close the input file.
st=fclose(fid);
% Calculate the frequencies of co-occurrence.
% Plot the inter-term dependencies graph.
b=0;
PN1=1;
while (PN1<=TPN)
        PN2=PN1+1;
        while (PN2<=TPN)
                b=0;
                for i=1:TSN
                        b=b+(a(i,PN1)*a(i,PN2));
                end;
                if (b>LIM)
                        c(1)=b;
                        c(2:1:TSN)=0;
                        d(1:1:TSN)=PN1;
                        e(1:1:TSN)=PN2;
                        plot3(d,e,c);
                        grid on;
                        hold on;
                end;
                PN2=PN2+1;
        end;
        PN1=PN1+1;
end;

% Give labels to the graph.
xlabel('Chunk Numbers');
ylabel('Chunk Numbers');
zlabel('Co-occurrence Frequencies');
```

APPENDIX C: TEST RESULTS

This appendix shows occurrence graphs and the co-occurrence graphs plotted for the test documents. The occurrence graphs are discussed in Section 3.3.5 and Section 5.4, and the co-occurrence graphs are discussed in Section 3.3.7 and Section 5.5 of this thesis. The list of graphs attached in this appendix is shown in Table C.1. The cut-off frequency and the chunks used to generate the occurrence graphs and the co-occurrence graphs are listed in a table preceding the respective graphs. Also, the chunk pairs appearing in the co-occurrence graphs are list in a table preceding the co-occurrence graphs. The chunk numbers on the graphs correspond to the chunk numbers on the list.

Table C.1 List of Graphs

Table C.2 Chunk List for Occurrence Graph for DMA 8237A

Chunk Number	Chunk
1	8237a
2	use
3	request
4	dreq
5	transfer
6	decrement
7	dma service
8	channel
9	program
10	autoinitialize
11	address
12	idle cycle
13	data bus
14	clear
15	reset
16	set
17	read
18	acknowledge
19	cpu
20	hold
21	generate
22	output
23	enable
24	address
25	data
26	memory
27	write
28	/eop
29	tc
30	register
31	hrq
32	microprocessor
33	transfer
34	perform

Cut-off Frequency of Occurrence: 8

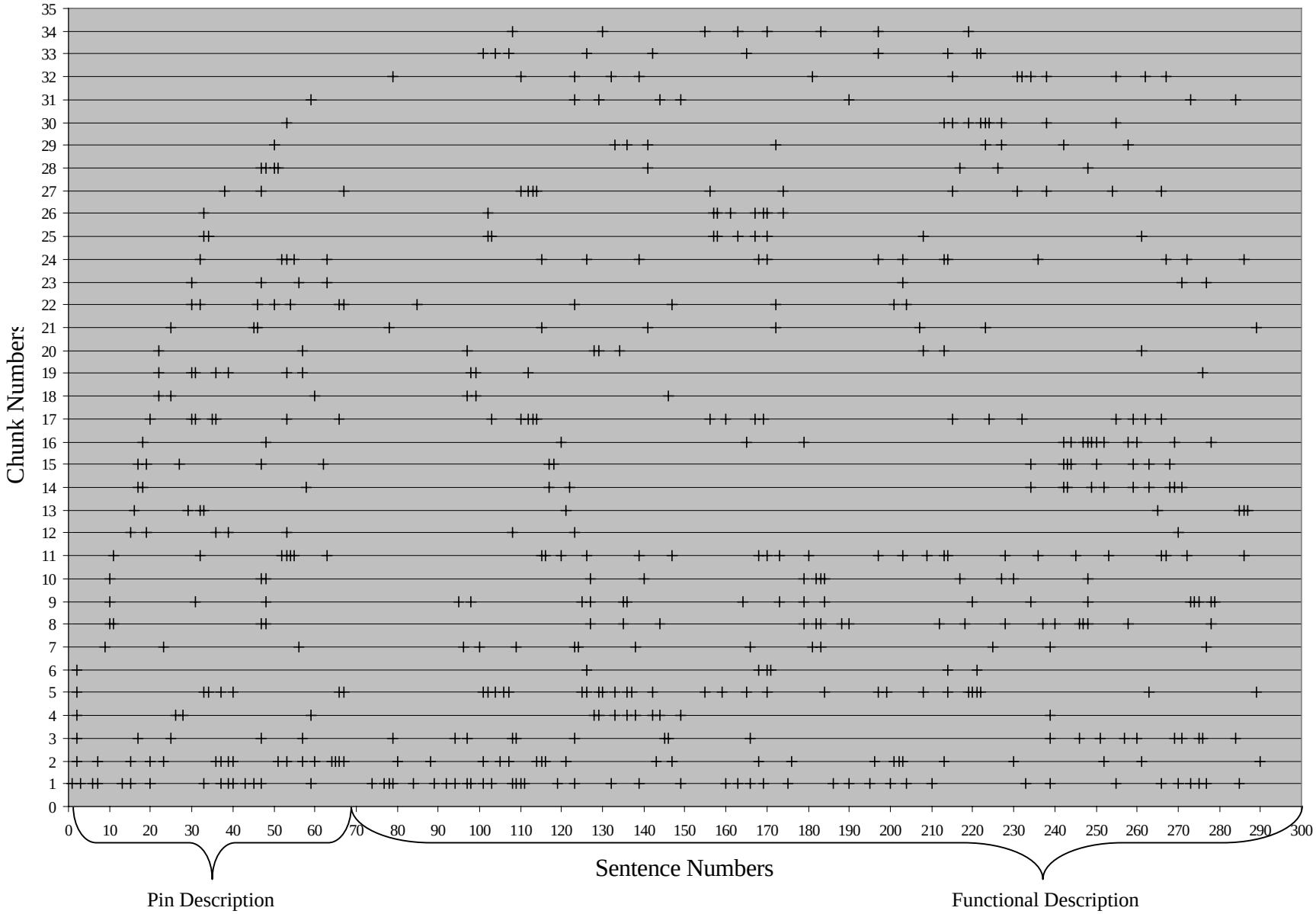

Figure C.1 Occurrence Graph for DMA 8237A

Table C.3 Chunk List for Occurrence Graph for US Patent 5381538

Chunk Number	Chunk
1	provide
2	data information
3	store
4	memory
5	transfer
6	control
7	control information
8	receive
9	allow
10	load
11	memories 16
12	i/o bus 18
13	processor portion 20
14	determine
15	write
16	memory controller 50
17	bus interface unit 54
18	read
19	dma controller 52
20	retrieve
21	pass
22	perform
23	control signal generator circuit 102
24	circuit 103
25	backup circuit 110
26	dma control
27	generate
28	fifo register circuit 104
29	enable
30	fifo
31	indicate
32	backup memory 108
33	control circuits
34	set
35	said register circuit

Cut-off Frequency of Occurrence: 14

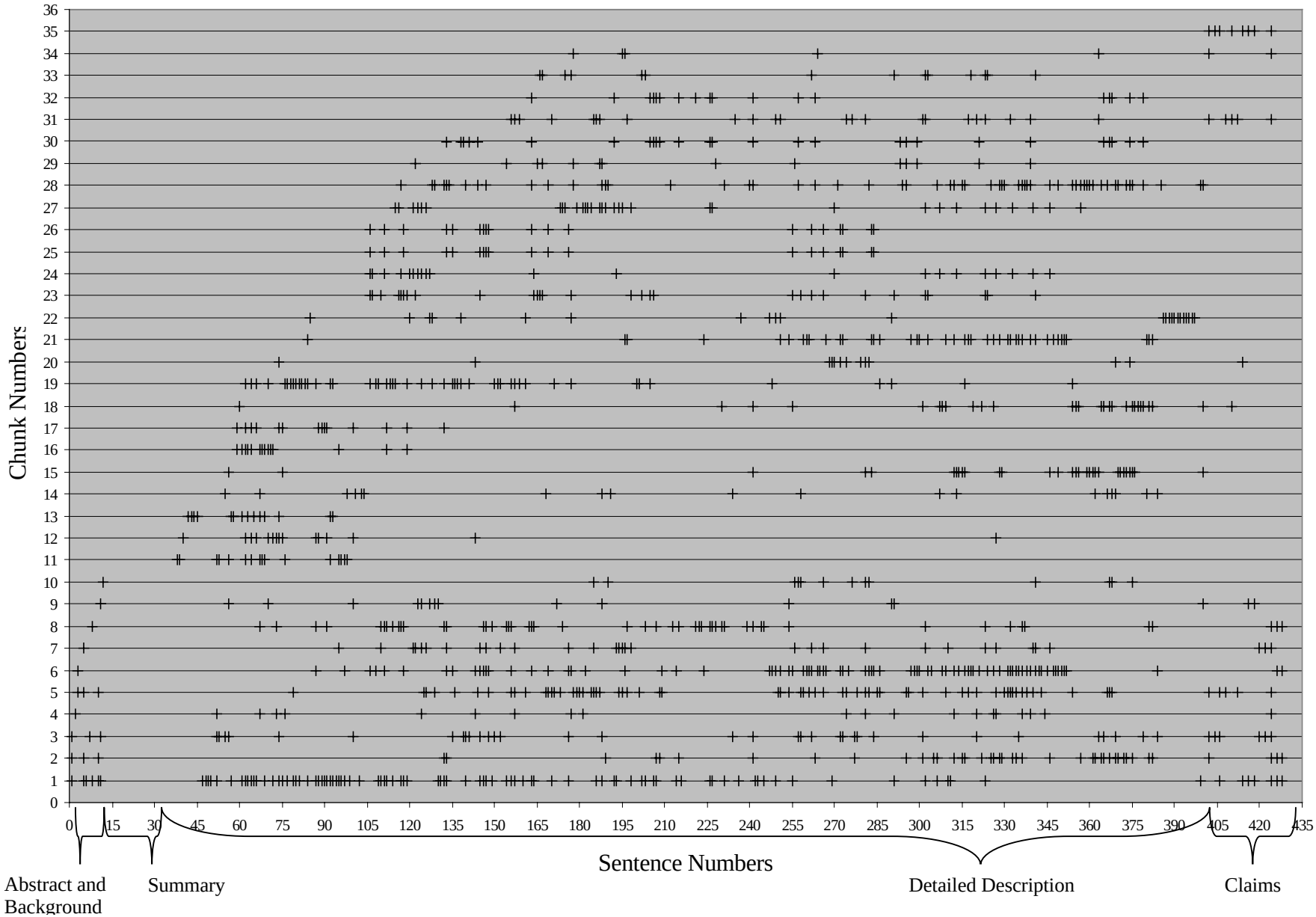

Figure C.2 Occurrence Graph for US Patent 5381538

Table C.4 Chunk List for Occurrence Graph for US Patent 4847750

Chunk Number	Chunk
1	peripheral dma controller
2	include
3	couple
4	peripheral address bus
5	peripheral data bus
6	memory
7	transfer
8	store
9	peripheral devices
10	operate
11	connect
12	dma
13	enable
14	write
15	program
16	generate
17	occur
18	cause
19	data bus
20	acknowledge
21	dual port memory
22	response
23	signal
24	receive
25	dma transfer
26	select
27	set
28	host computer 3
29	processor 3a
30	memory 15
31	host dma controller 3d
32	indicate
33	dma sequencer 29
34	dma controller

Cut-off Frequency of Occurrence: 16

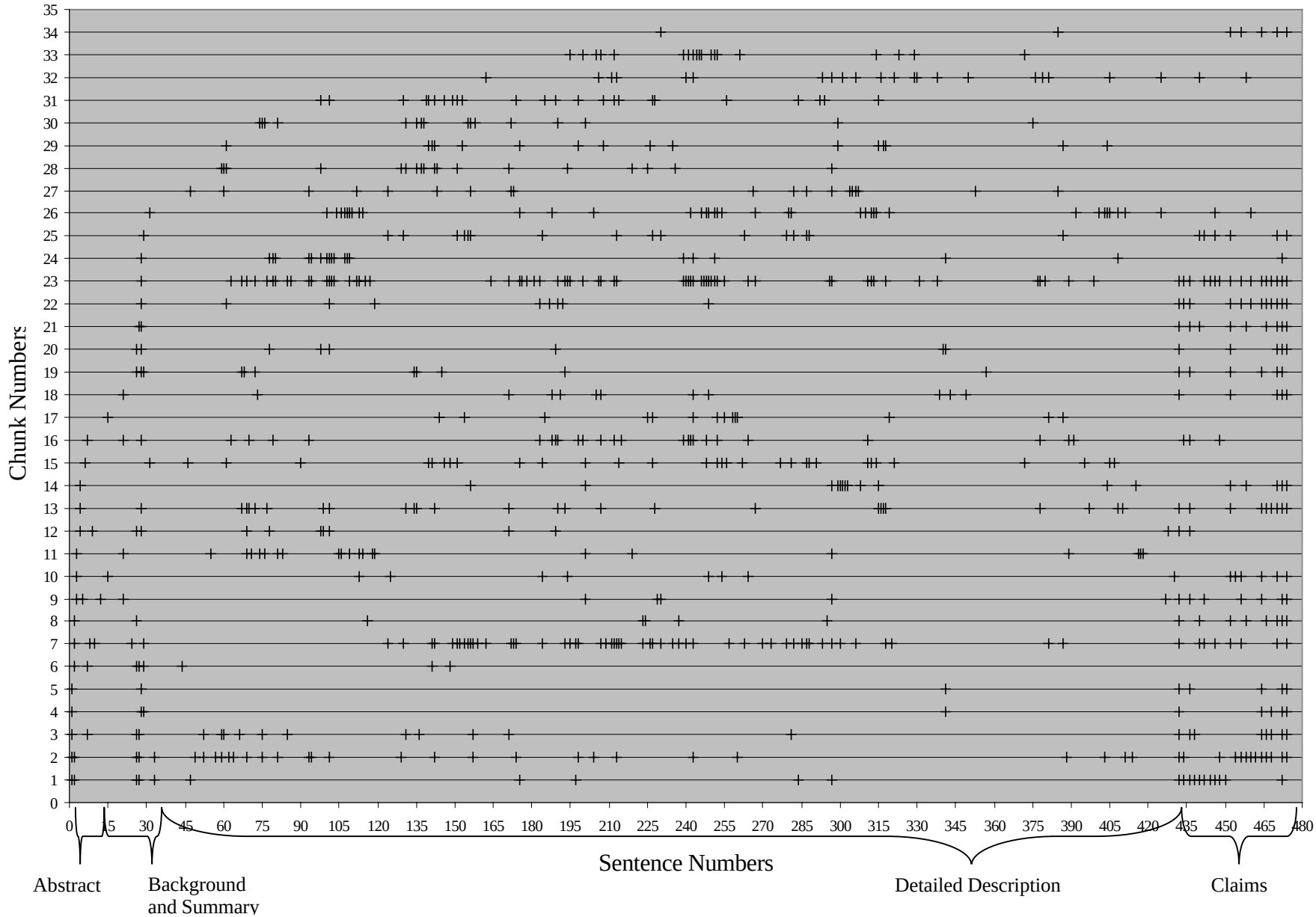

Figure C.3 Occurrence Graph for US Patent 4847750

Table C.5 Chunk List for Occurrence Graph for US Patent 5283883

Chunk Number	Chunk
1	read
2	write
3	memory
4	i/o device
5	dma controller
6	transfer
7	write operations
8	buffer lines
9	store
10	throughput
11	system bus
12	couple
13	comprise
14	perform
15	retrieve
16	pre-fetched
17	receive
18	drained
19	system bus interface
20	send
21	set
22	dma controller 20
23	select
24	state
25	state machine
26	complete
27	buffer line
28	state machine b
29	wait
30	said memory
31	said first input-output device
32	said second input-output device

Cut-off Frequency of Occurrence: 15

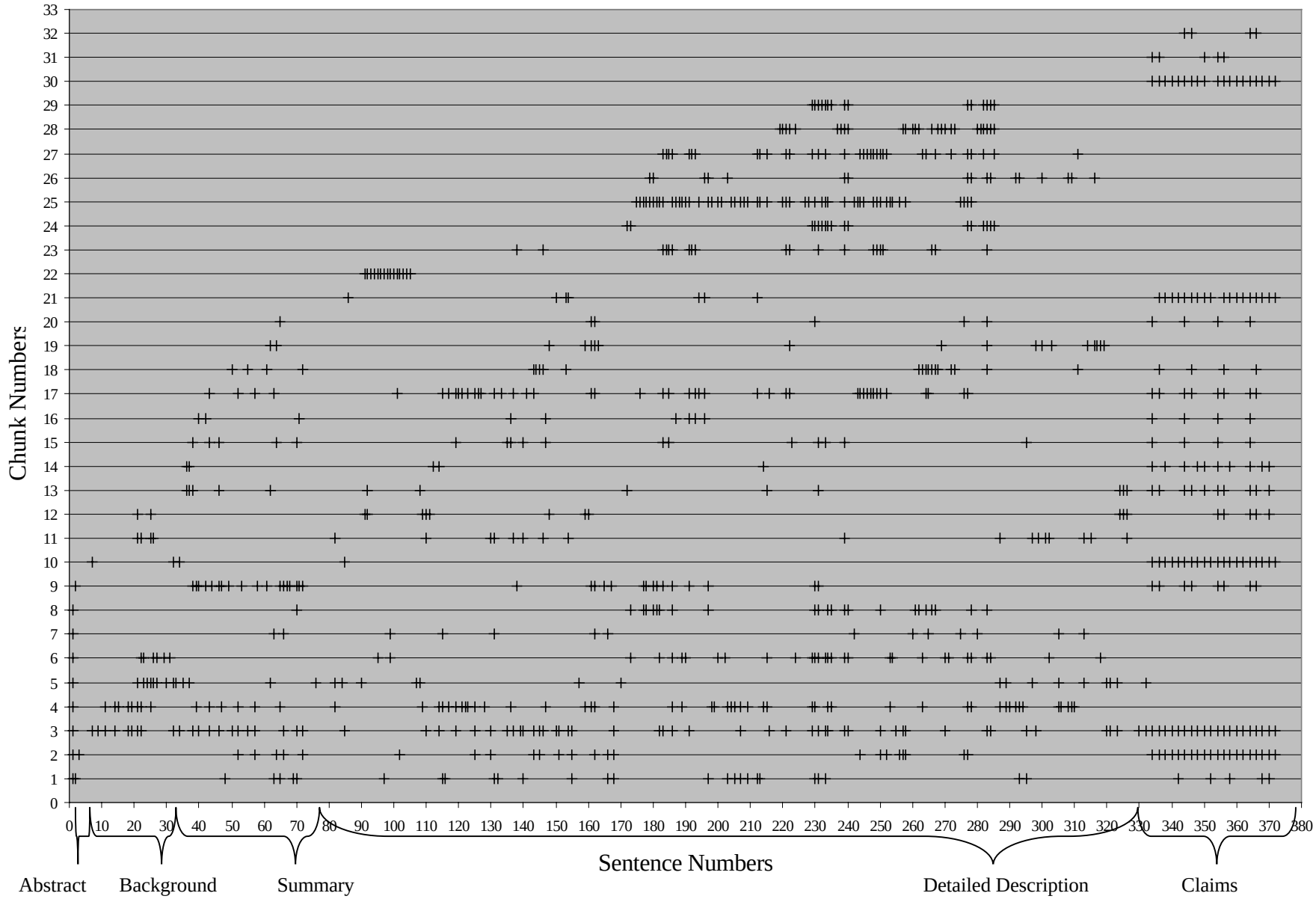

Figure C.4 Occurrence Graph for US Patent 5283883

Table C.6 Chunk List for Occurrence Graph for US Patent 5465340

Chunk Number	Chunk
1	dma controller
2	check
3	perform
4	provide
5	move
6	write
7	destination address
8	memory cycle
9	transfer
10	data transfer
11	read
12	source
13	store
14	operation
15	set
16	destination addresses
17	indicate
18	couple
19	latch
20	write cycle
21	output
22	combinational logic block arrangement
23	enable
24	signal
25	asserted
26	dram 14
27	dma controller 18
28	bus 20
29	receive
30	source address
31	latch 100
32	latch 112
33	dsten 1
34	n signal

Cut-off Frequency of Occurrence: 15

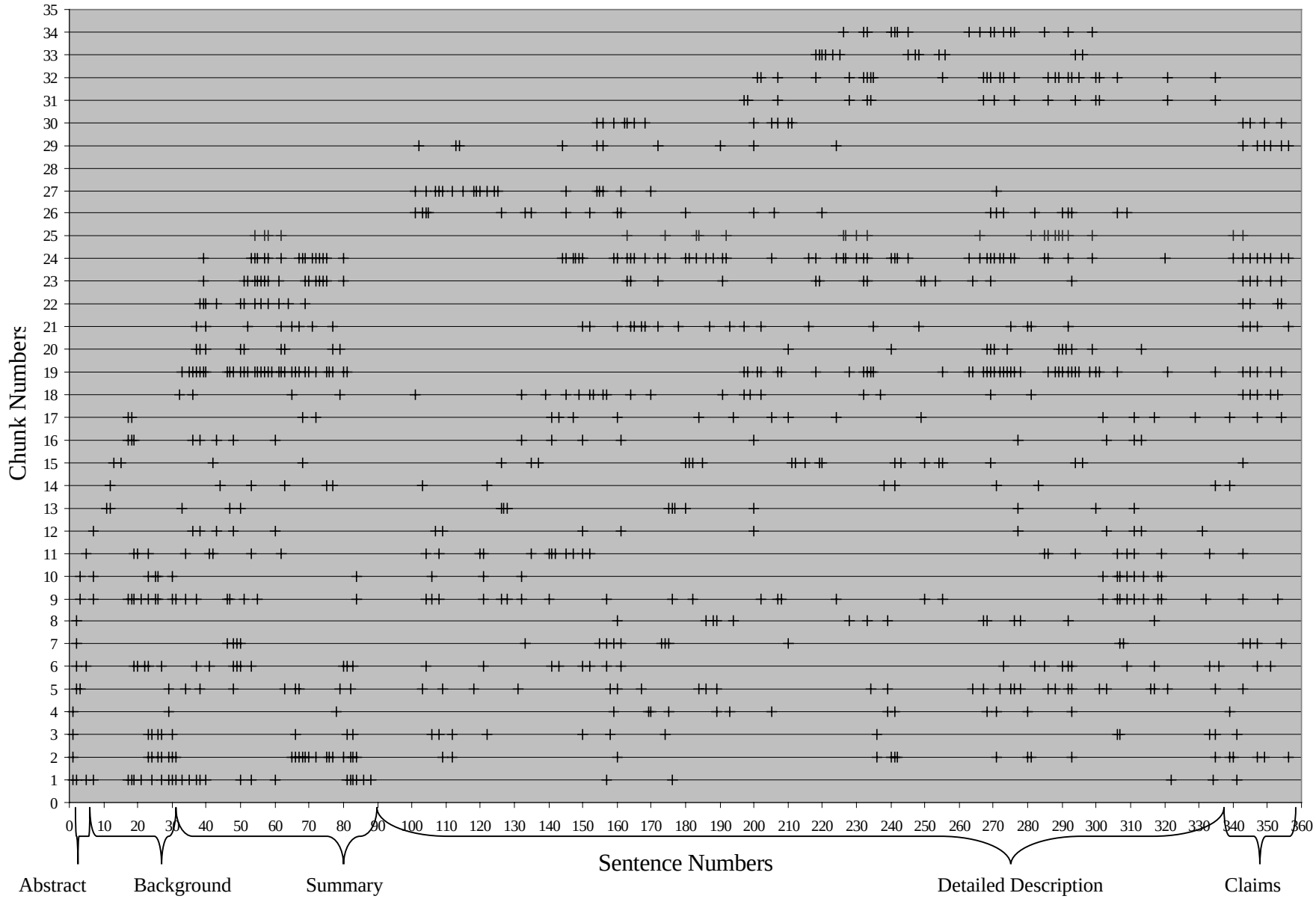

Figure C.5 Occurrence Graph for US Patent 5465340

Table C.7 Chunk List for Co-occurrence Graph for DMA 8237A

Chunk Number	Chunk
1	/eop
2	8237a
3	address
4	channel
5	cpu
6	data
7	data bus
8	dma service
9	dreq
10	hrq
11	idle cycle
12	memory
13	microprocessor
14	register
15	tc
16	transfer
17	acknowledge
18	autoinitialize
19	clear
20	decrement
21	enable
22	generate
23	hold
24	output
25	perform
26	program
27	read
28	request
29	reset
30	set
31	use
32	write

Cut-off Frequency of Co-occurrence: 5

Table C.8 Chunk Pairs for DMA 8237A

Co-ordinate Pair	Chunk	Chunk
2-10	8237a	hrq
2-11	8237a	idle cycle
2-13	8237a	microprocessor
2-27	8237a	read
2-28	8237a	request
2-31	8237a	use
3-16	address	transfer
3-20	address	decrement
3-21	address	enable
4-18	channel	autoinitialize
4-26	channel	program
4-30	channel	set
6-12	data	memory
18-26	autoinitialize	program
19-29	clear	reset
27-31	read	use
27-32	read	write

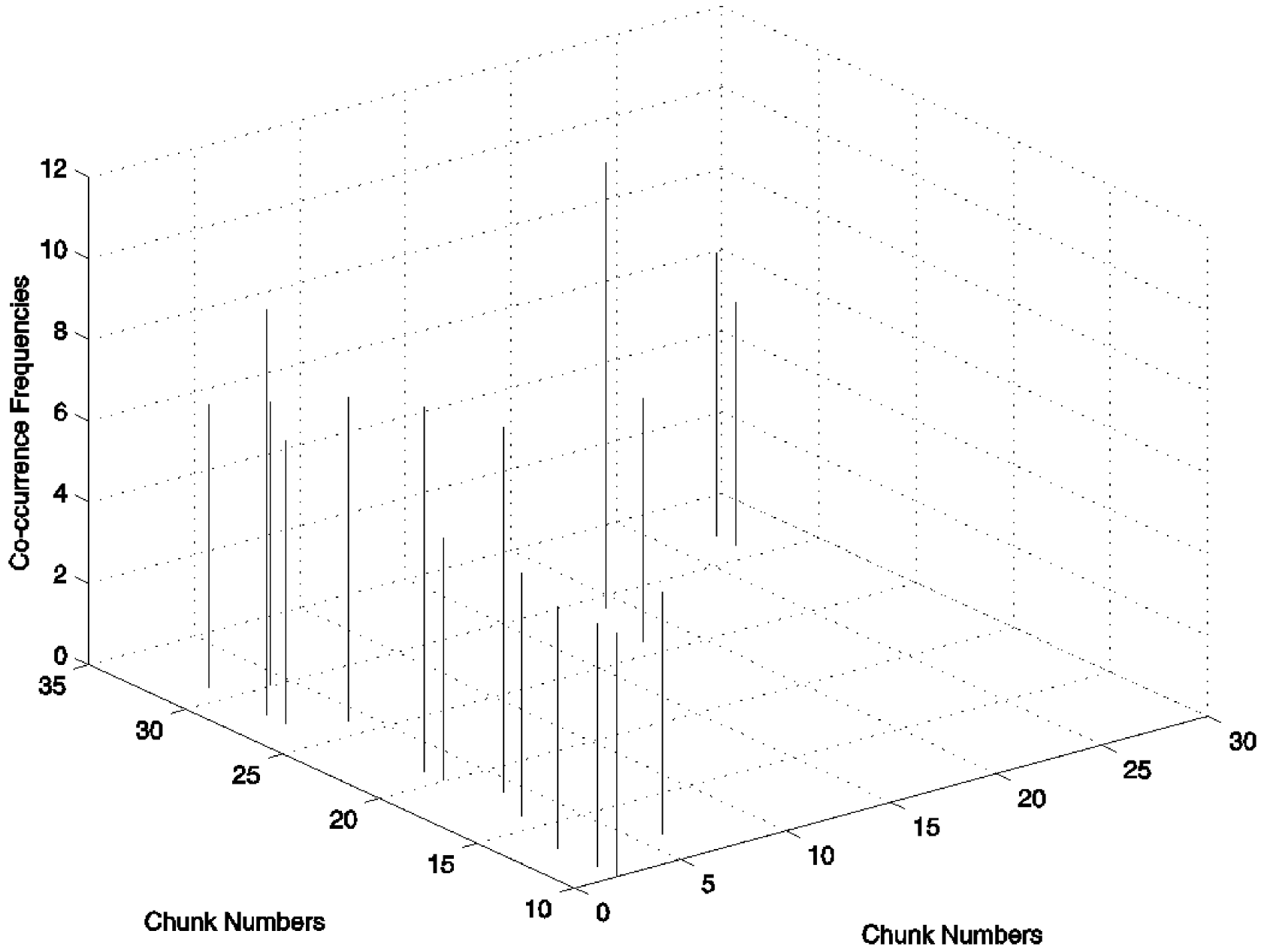

Figure C.6 Co-occurrence Graph for DMA 8237A

Table C.9 Chunk List for Co-occurrence Graph for US Patent 5381538

Chunk Number	Chunk
1	backup circuit 110
2	backup memory 108
3	bus interface unit 54
4	circuit 103
5	control circuits
6	control information
7	control signal generator circuit 102
8	data information
9	dma control
10	dma controller 52
11	fifo
12	fifo register circuit 104
13	i/o bus 18
14	memories 16
15	memory
16	memory controller 50
17	processor portion 20
18	said register circuit
19	allow
20	control
21	determine
22	enable
23	generate
24	indicate
25	load
26	pass
27	perform
28	provide
29	read
30	receive
31	retrieve
32	set
33	store
34	transfer
35	write

Cut-off Frequency of Co-occurrence: 20

Table C.10 Chunk Pairs for US Patent 5381538

Co-ordinate Pair	Chunk	Chunk
1-20	backup circuit 110	control
2-11	backup circuit 110	fifo
6-28	control information	provide
8-12	data information	fifo register circuit 104
8-18	data information	said register circuit
8-24	data information	indicate
8-28	data information	provide
8-30	data information	receive
8-32	data information	set
8-33	data information	store
8-34	data information	transfer
9-20	dma control	control
10-28	dma controller 52	provide
14-28	memories 16	provide
20-26	control	pass
20-34	control	transfer
24-32	indicate	set
24-34	indicate	transfer
28-30	provide	receive
32-34	set	transfer

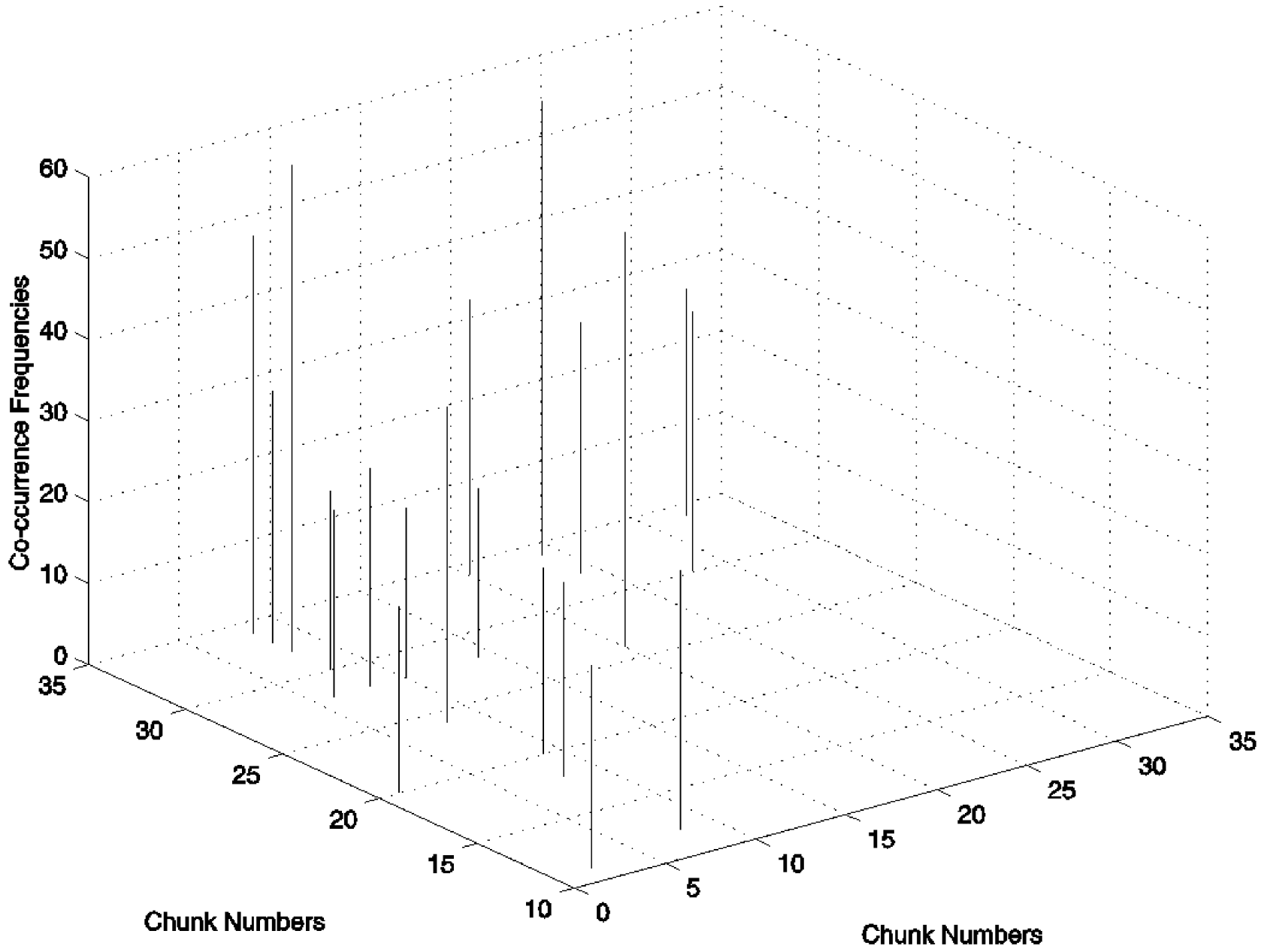

Figure C.7 Co-occurrence Graph for US Patent 5381538

Table C.11 Chunk List for Co-occurrence Graph for US Patent 4847750

Chunk Number	Chunk
1	data bus
2	dma
3	dma controller
4	dma sequencer 29
5	dma transfer
6	dual port memory
7	host computer 3
8	host dma controller 3d
9	memory
10	memory 15
11	peripheral address bus
12	peripheral data bus
13	peripheral dma controller
14	processor 3a
15	acknowledge
16	cause
17	connect
18	generate
19	include
20	indicate
21	occur
22	operate
23	program
24	receive
25	select
26	set
27	store
28	transfer
29	write

Cut-off Frequency of Co-occurrence: 20

Table C.12 Chunk Pairs for US Patent 4847750

Co-ordinate Pair	Chunk	Chunk
1-2	data bus	dma
1-6	data bus	dual port memory
1-11	data bus	peripheral address bus
1-12	data bus	peripheral data bus
1-15	data bus	acknowledge
2-6	dma	dual port memory
2-11	dma	peripheral address bus
2-12	dma	peripheral data bus
2-13	dma	peripheral dma controller
2-15	dma	acknowledge
2-19	dma	include
3-6	dma controller	dual port memory
3-15	dma controller	acknowledge
3-22	dma controller	operate
3-28	dma controller	transfer
3-29	dma controller	write
6-11	dual port memory	peripheral address bus
6-12	dual port memory	peripheral data bus
6-15	dual port memory	acknowledge
6-27	dual port memory	store
6-29	dual port memory	write
11-12	peripheral address bus	peripheral data bus
11-15	peripheral address bus	acknowledge
11-19	peripheral address bus	include
12-13	peripheral data bus	peripheral dma controller
12-15	peripheral data bus	acknowledge
12-19	peripheral data bus	include

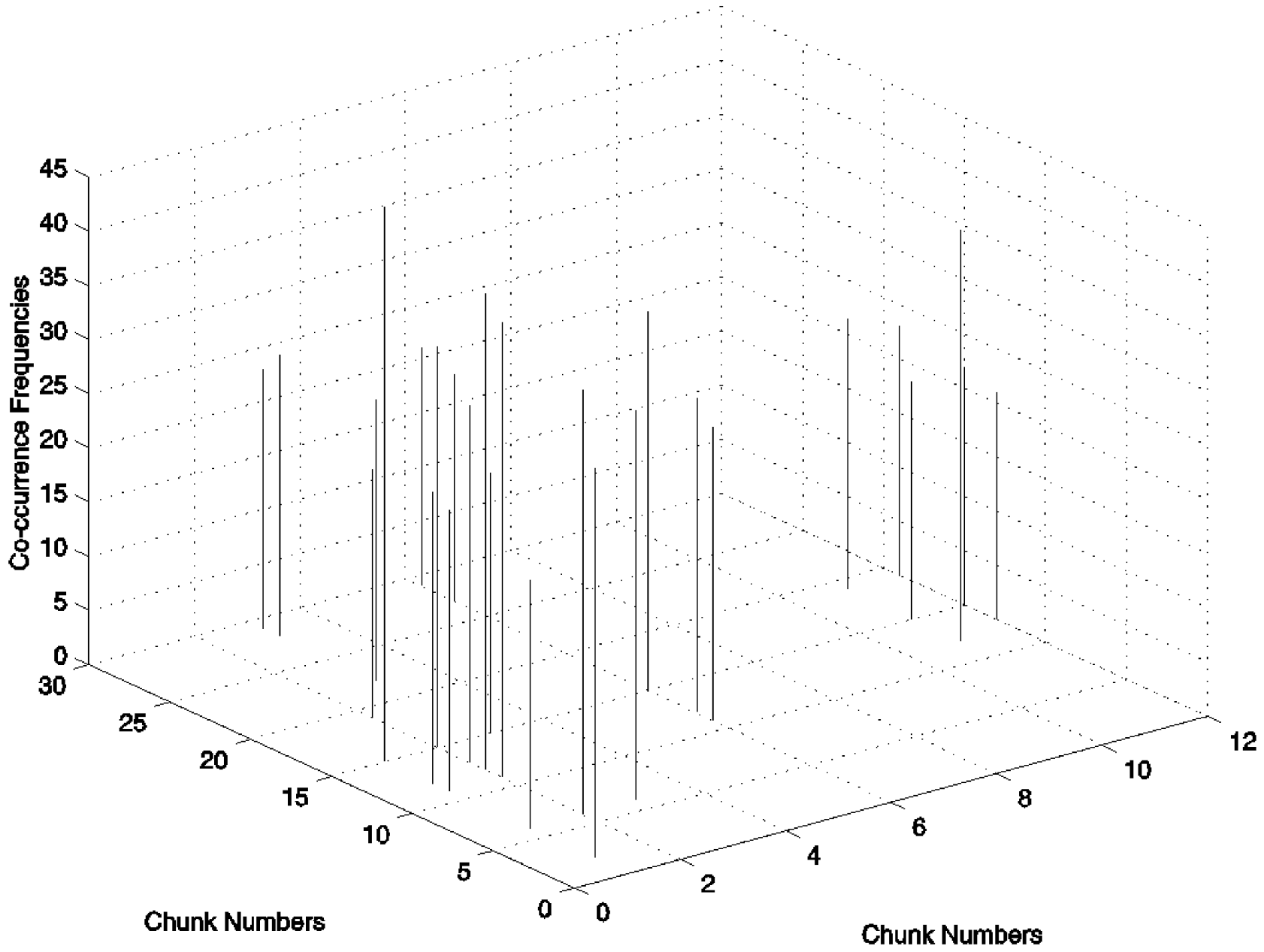

Figure C.8 Co-occurrence Graph for US Patent 4847750

Table C.13 Chunk List for Co-occurrence Graph for US Patent 5283883

Chunk Number	Chunk
1	buffer line
2	buffer lines
3	dma controller
4	dma controller 20
5	drained
6	i/o device
7	memory
8	pre-fetched
9	state
10	state machine
11	state machine b
12	system bus
13	system bus interface
14	throughput
15	write operations
16	complete
17	comprise
18	couple
19	perform
20	read
21	select
22	send
23	set
24	transfer
25	wait
26	write

Cut-off Frequency of Co-occurrence: 20

Table C.14 Chunk Pairs for US Patent 5283883

Co-ordinate Pair	Chunk	Chunk
1-10	buffer line	state machine
1-24	buffer line	transfer
2-24	buffer lines	transfer
5-17	drained	comprise
6-7	i/o device	memory
6-24	i/o device	transfer
7-14	memory	throughput
7-17	memory	comprise
7-23	memory	set
7-24	memory	transfer
7-26	memory	write
9-24	state	transfer
9-25	state	wait
10-24	state machine	transfer
14-17	throughput	comprise
14-26	throughput	write
17-18	comprise	couple
17-19	comprise	perform
17-22	comprise	send
17-26	comprise	write
18-22	couple	send
19-22	perform	send
23-26	set	write
24-25	transfer	wait

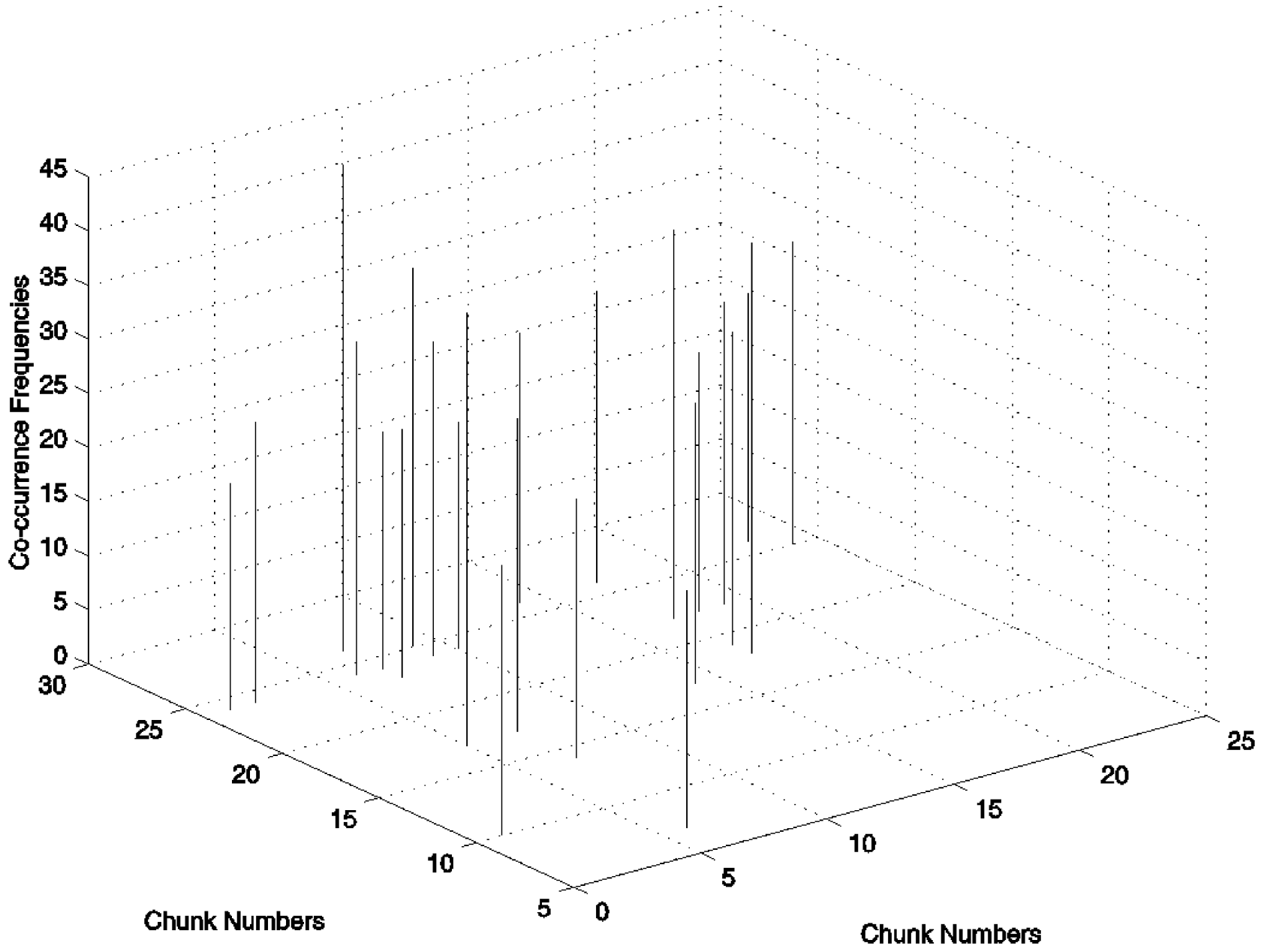

Figure C.9 Co-occurrence Graph for US Patent 5283883

Table C.15 Chunk List for Co-occurrence Graph for US Patent 5465340

Chunk Number	Chunk
1	asserted
2	bus 20
3	combinational logic block arrangement
4	data
5	destination address
6	destination addresses
7	dma controller
8	dma controller 18
9	dram 14
10	dsten 1
11	latch 100
12	latch 112
13	memory cycle
14	n signal
15	operation
16	output
17	source
18	write cycle
19	couple
20	indicate
21	move
22	perform
23	provide
24	read
25	receive
26	set
27	store
28	transfer

Cut-off Frequency of Co-occurrence: 10

Table C.16 Chunk Pairs for US Patent 5465340

Co-ordinate Pair	Chunk	Chunk
1-5	asserted	destination address
4-28	data	transfer
5-16	destination address	output
5-25	destination address	receive
5-28	destination address	transfer
6-17	destination addresses	source
7-28	dma controller	transfer
11-12	latch 100	latch 112
12-21	latch 112	move
13-21	memory cycle	move
16-19	output	couple
16-25	output	receive
19-25	couple	receive
22-28	perform	transfer
24-28	read	transfer
25-28	receive	transfer

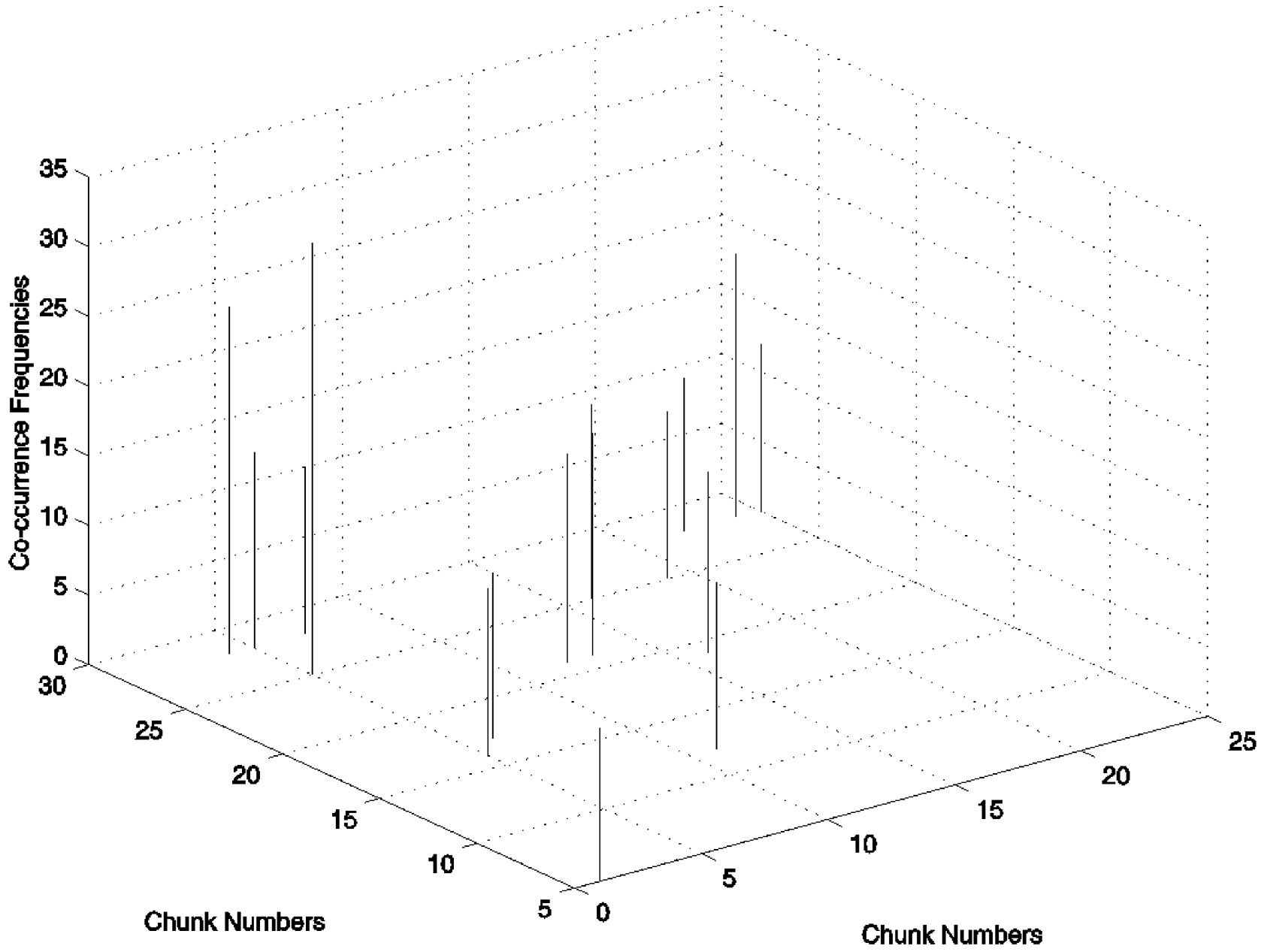

Figure C.10 Co-occurrence Graph for US Patent 5465340

VITA

Ritesh D. Sojitra was born in Surat, India in 1974. He completed his high school education at Gnyan Dham High School, Vapi, India in March 1991 and began his undergraduate study at Birla Vishvakarma Mahavidyalaya, Vallabh Vidyanagar, India. Ritesh graduated with a Bachelor's degree in Electrical Engineering in October 1995. Then, he worked as a Lecturer in Electrical Engineering Department of the same institute. In August 1996, he enrolled at Virginia Polytechnic Institute and State University to pursue his Masters in Electrical Engineering. At Virginia Tech his area of concentration was Computer Engineering with major interest in Digital and VLSI Design.

Ritesh's immediate plan is to begin his employment at Fujitsu Network Communications, Dallas, Texas in September 1998. He will be working as a Hardware Engineer in the ASIC Design Group.