RELAXATION PROBLEM SOLVING
(with input to Chinese input problem)

Kam Pui Chow

February 1986
CSD-860058

Relaxation Problem Solving

(with application to Chinese Input Problem)

by

Kam Pui Chow

UNIVERSITY OF CALIFORNIA
Santa Barbara

Relaxation Problem Solving
(with application to Chinese input problem)

A dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosphy

in

Electrical and Computer Engineering

by

Kam Pui Chow
鄒錦沛

Committee in charge:

Professor Terrence R. Smith, Chairman

Dr. Paul R. Eggert

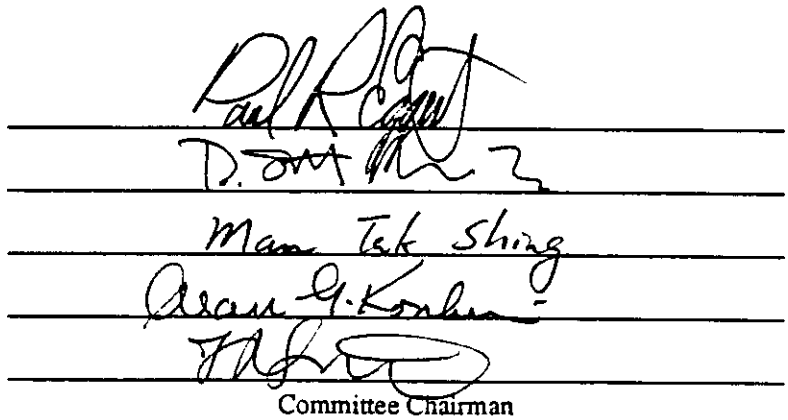
Professor Alan Konheim

Professor Man-Tak Shing (成萬德)

Professor D. Stott Parker

December 1985

The dissertation of Kam Pui Chow is approved.



Committee Chairman

December 1985

Dedicated to

my family

Acknowledgement

I wish to thank my committee members, Paul Eggert, Alan Konheim, Stott Parker, Man-Tak Shing, and Terry Smith. In particular, I wish to thank Terry Smith, my committee chairman, Alan Konheim, the chairman of Computer Science Department of UCSB, and Stott Parker, Professor of Computer Science Department of UCLA, who gave me the flexibility of starting my research in UCSB and finishing in UCLA. Special thanks to Paul Eggert, who provided unlimited supplies of good idea; and Man-Tak Shing, the only Chinese in my committee, which understands all examples in this thesis.

I am also grateful to the other members of the relaxation research group in UCLA, namely Ching-Tsun Chou, Koenraad Lecot, and Rei-Chi Lee, who listened to me and discussed with me at various time.

Thanks to a special young lady on the other side of Pacific, 惠 瓊 (Colinna), who not only sent me the pictures of the Chinese typewriter, but also provided unlimited encourgement during the last part, and the worst part, of the dissertation preparation.

This research is partially supported by the MICRO grant from IBM and Univerisity of California.

My friends at Los Angeles and Santa Barbara made California a wonderful place to stay, especially 偉 明 , 子 貴 , 岑 松 德 , Cindy, and many others.

Vita

October 24, 1959--	Born--Hong Kong.
1979--	Higher Diploma, Hong Kong Polytechnic.
1980--	Computer Programmer, Cable and Wireless Limited, Hong Kong.
1980-1981--	Teaching Assistant, Department of Mathematics, University of California, Santa Barbara.
1981--	M.A., University of California, Santa Barbara.
1982-1984--	Teaching Assistant, Department of Computer Science, University of California, Santa Barbara.
1983-1985--	Research Assistant, Community Organization and Research Institute, University of California, Santa Barbara.
1984--	Computer Scientist, Silogic Inc., Los Angeles.
1984-1985--	Postgraduate Research Engineer, Department of Computer Science, University of California, Los Angeles.

Publications

Chow, K. P. & Cotton, J. W. *Deterministic Model of Concept Identification.* Journal of Mathematical Psychology, 1983.

Chow, K. P. & Eggert, P. R. *Logic Programming Graphics and Infinite Terms.* UCSB Department of Computer Science Technical Report, March 1983.

Abstract

Two fundamental problem solving techniques are introduced to help automate the use of relaxation: multilevel frameworks and constraint generation. They are closely related to iterative relaxation and subproblem relaxation.

In multilevel problem solving, the set of constraints is partitioned vertically into different levels. Lower level constraints generate possible solutions while higher level constraints prune the solutions to reduce the combinatorial explosion. Subproblem relaxation at first relaxes the high level constraints; the solution is then improved by strengthening the relaxed constraints.

The constraint generation technique uses iterative relaxation to generate a set of constraints from a given model. This set of constraints with a constraint interpreter form an expert system. This is an improvement over most existing expert systems which require experts to write down their expertise in rules.

These principles are illustrated by applying them to the Chinese input problem, which is to transform a phonetic spelling, without word breaks, of a Chinese sentence into the corresponding Chinese characters. Three fundamental issues are studied: segmentation, homophone analysis, and dictionary organization. The problem is partitioned into the following levels: phonetic spelling, word, and grammar. The corresponding constraints are legal spellings, legal words, and legal syntactic structures. Constraints for syntactic structure are generated from a Chinese grammar.

Table of Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

Relaxation is a class of heuristic approaches for satisfying constraints. Though it was applied to solving problems before the advent of computers, it still requires careful attention by experienced, specialized programmers for effective use. In this thesis, two fundamental problem solving techniques are introduced to help automate the use of relaxation: multilevel framework and constraint generation. They are closely related to iterative relaxation and subproblem relaxation. Their practical values are demonstrated with application to the Chinese input problem.

Relaxation is a general problem solving paradigm particularly suitable for problems involving constraints. These problems can be divided into two kinds: constraint satisfaction and constrained optimization. Constraint satisfaction is the problem of finding values that satisfy a set of constraints. Constrained optimization is the problem of optimizing an objective function subject to a set of constraints. The problem solving techniques described here are applied to constraint satisfaction problems. Constrained optimization problems will not be discussed.

The constraint generation algorithm is an iterative relaxation algorithm. The multilevel framework can be viewed as a subproblem relaxation process. It partitions the set of constraints into levels. Each level is a constraint satisfaction problem and has its own set of objects. An object at one level is an abstraction of objects at the next lower level. Higher level subproblems are relaxed initially. The lowest level subproblem is solved first. The solution is then improved by strengthening the relaxed subproblems.

The rest of the chapter defines the constraint satisfaction problem, and briefly summarizes related problem solving strategies: iterative relaxation, subproblem relaxation, and multilevel organization.

1. Constraint Satisfaction Problem

Constraint satisfaction problem is the problem of finding values that satisfy a set of constraints. It can be a numerical problem or a symbolic problem. The problem consists of a set of variables and a set of constraints. The goal is to find a solution or a set of solutions such that no constraint is violated. Constraints can be equations, inequalities, or symbolic predicates. They are categorized by the set of variables they describe. There are two kinds of variables: discrete and continuous. Discrete variables assume values from a countable set, finite or infinite; while continuous variables have values from an infinite uncountable set. For example, following are constraints defined on continuous variables $\{x_1, x_2, x_3\}$, and discrete variables $\{x_4, x_5\}$.

$$x_1 + x_2 + x_3 \le 10,$$

$$x_1 \ge 0,$$

$$x_2 \ge 0,$$

$$C_{(x_4,x_5)} = \{(a,b), (a,d)\}.$$

A consistent labeling problem is one kind of constraint satisfaction problem. It is characterized by a finite set of variables, $\{x_1, x_2, \cdots, x_n\}$. Each variable, x_i, has an associated finite domain, from which it can take any of m_i values of labels. Constraints exist on which values are mutually compatible for various subsets of the n variables. The goal is to find one or more sets of assignments of all n variables to values in their corresponding domains, such that for each assignment set all

constraints are simultaneously satisfied.

Many practical problems can be formulated as consistent labeling problems, such as scene analysis and image processing [27, 48], constrained search problems [36], graph theory [59, 20], production systems [64], datatype inference for logic program [9], and pattern matching on relations [29].

2. Relaxation

Relaxation has been used as a computational technique in numerical analysis, operations research, computer vision, and many other areas, but has not been generally recognized as a powerful problem solving method for nonnumeric applications. The basic idea of relaxation is *if the given problem is hard, it is transformed to a simpler related problem.* A problem is called "hard" if the space needed and the time required to solve the problem is huge. Common measures of space and time requirement are the space and time complexity of the problem solving algorithm [43]. They are usually depended on the size of the problem and its inherent difficulties. The basic principle of relaxation can be stated as:

> Given a problem, if the space and time complexity of the problem solving algorithm is high, relax it into a simpler related problem. A problem P is said to be simpler than a problem Q if the time and space complexity of the algorithm solving P is less than Q. Q is related to P if Q's solution is close to P's.

After relaxation, the relaxed version is solved by known techniques and a solution is obtained. This solution can be accepted or rejected. Rejection is usually caused by the solution not satisfying some constraints of the problem. Under rejection, an improvement method is invoked and a better solution is sought. The improvement method usually strengthens the relaxed constraints. During the improvement process, some previously satisfied constraints may be violated. The improvement process is called again. The iterative process continues until a satisfactory solution is obtained. This

technique applies to any kind of problem solving. In general, relaxation can be viewed as the process of removing, or temporarily ignoring, some constraints. Improvement, which is an iterative process, refers to strengthening the relaxed constraints on the solution from the relaxed problem.

A general relaxation algorithm is as follows:

```
procedure relaxation(P)
begin
    C := the set of constraints of the problem P
    while solution of P not found easily do
        relax(C);
    while result R of P not satisfactory do
        improve(R);
    return R
end.
```

There are two basic kinds of relaxation: iterative relaxation and subproblem relaxation. These two methods of relaxation differ in strategy. Iterative relaxation seeks to maintain a set of constraints by modifying *data*. Subproblem relaxation instead alters *constraints*. They are briefly discussed in next two sections.

2.1. Iterative Relaxation

Iterative relaxation is a general computational paradigm for finding values that satisfy a set of constraints. It iteratively assigns values to mutually constrained objects to ensure a consistent set of values. The procedure repeatedly selects an unsatisfied constraint involving a variable x and updates x in such a way as to enforce the constraint. The first application of relaxation method was made in 1935 by Sir Richard Southwell [55] for stress calculation in frameworks. In the relaxation process the forces taken by the constraints are the residual forces of the system not yet carried by the framework. The name *relaxation* originally referred to the reduction of residual forces by the systematic relaxation of constraints. Southwell's iterative relaxation scheme is a powerful tool for solving any

set of simultaneous equations. Assume the following set of constraints (equations) is given:

$$Ax = b, \tag{A}$$

where A is an $n \times n$ matrix, b is an n-vector, and x is an n-vector which is to be solved.

Simultaneous equations can be solved by several methods, which fall into two basic categories: direct and iterative approaches. Iterative relaxation is an iterative process. It starts with an initial estimate, z, and iteratively satisfies the given set of equations by the fixed point equation

$$x = A^* x + b^*,$$

where A^* is an $n \times n$ matrix and b^* is an n-vector, both derived from (A). The iterative relaxation procedure can be written as follow:

$$x_0 := z \ ,$$

$$x_{i+1} := A^* x_i + b^* \ .$$

The process continues until a tolerable answer is obtained, i.e., the highest deviation is within the tolerance limit, or the results of two successive approximations are close to each other. If the matrix A^* satisfies some conditions, the iterative process always terminates and the convergence rate is satisfactory [41, 42]. It can also be applied to solve ordinary and partial differential equations, and systems of differential equations [3].

Relaxation has been a powerful computational tool in the solution of practical engineering and physical problems. In the early 1970s, the relaxation method was extended to handle nonnumeric problems; one example is relaxation labeling, the iterative application of constraints to label any node of a graph. It was first used by Waltz [60] for labeling junctions in line drawings. The method has been extended to image processing, scene analysis, and computer vision [27, 48]. In general, it can be used to solve the consistent labeling problem. A formal analysis of relaxation labeling has been done by Rosenfield *et al* [48]. Both error and rate of convergence have also been analyzed

analytically [5, 47, 63].

In general, relaxation labeling labels a graph consistently with respect to the given constraints. Constraints state possible labels that can exist in adjacent nodes simultaneously. Starting with an initial labeling set that contains all possible labels, of which some may be inconsistent, relaxation is the process of iterative refinement of the labeling set, achieved by successive removal of inconsistent labels from each node.

With iterative relaxation, goals are reached by successive satisfaction or elimination of constraints. As in the labeling algorithm, all constraints are relaxed first, then the solution is improved successively to satisfy each of the constraints. This approach has a wide spectrum of application. Besides the application in scene analysis and image processing, other examples include constrained search problems [36], graph theory [59, 20], production systems [64], data type inference for logic program [9], path problems [57], and pattern matching on relations [29, 30].

Recent research tries to characterize this wide class of situation in which iterative relaxation succeeds [44]. Specifically, whenever a problem can be casts into a set of constraints

$$x_i \leq f_i(x_1, \cdots, x_n)$$

over a semilattice, where f_i's are continuous and "monotone", then the relaxation iteration

$$x_i := f_i(x_1, \cdots, x_n)$$

converges to a solution. Interested readers should refer to the original paper for details.

Iterative relaxation is intrinsically a parallel method, so it is well suited for machines with parallel architecture. Since many problems can be viewed as tree searching problems, combinatorial explosion is always a major difficulty. With relaxation, this can be avoided by local elimination of inconsistency. Most of the redundant paths of a complicated search tree, which may be generated by a backtracking method, will not be examined.

With all these advantages from the relaxation algorithm, why is iterative relaxation not yet well recognized? Its major disadvantage is that it is a local method; a result does not guarantee global properties. But sometimes local optimization is a good enough approximation for the global optimum. Another disadvantage is that the result may not be unique.

2.2. Subproblem Relaxation

By "relaxation", workers in optimization and operations research typically mean reducing difficulty of a problem by transforming it to another one. "Transformation" here means changing the objective function or modifying the set of constraints. One formal definition from Geoffrion [19] is as follows:

> Assume F(A) is the set of feasible solutions of the problem A. A minimizing problem Q is said to be a relaxation of a minimizing problem P if F(Q) contains F(P) and the objective function of Q is less than or equal to that of P in F(P).

Subproblem relaxation has established itself as a key approach in attacking hard problems. Many combinatorially explosive problems, like the Traveling Salesman Problem (TSP) and integer programming, are solved heuristically by relaxing them to easier and computationally feasible problems. The solution from the relaxed problem is then examined and improvement is made if the solution is not satisfactory. Linear programming is usually the target of relaxation since it can be solved efficiently. One fine example is the application of relaxation to TSP [10]. TSP is first formulated as an integer programming problem, then relaxed to a linear programming problem. In fact it is very common to relax an integer programming problem to its linear relaxed version. The result can be improved by branch and bound or cutting plane based on the solution of the relaxed problem, though this kind of relaxation will not be useful if the integer variables are restricted to 0-1.

Another common subproblem relaxation method is Lagrangean relaxation. The term was first introduced by Geoffrion, who applied the Lagrangean technique to integer programming problems. Constraints are relaxed by multiplying them with Lagrange multipliers and added to the objective function. Lagrangean relaxation has been applied to many practical problems. Some representative examples are the generalized assignment problem relaxed to the knapsack problem [49], general integer programming with unbound variables relaxed to the group problem [16], and symmetric TSP relaxed to spanning tree [22, 23]. Other applications and references can be found in the survey by Fisher [17].

Besides Lagrangean relaxation and integer relaxation, relaxation can also be achieved by relaxing constraints, removing constraints, relaxing the objective function, decomposing the problem, and any techniques that can transform a hard problem to an easier one.

A difficulty in subproblem relaxation is to determine which constraint to relax next. Until now, no automated technique has been devised to solve this problem mechanically. All previous examples are results from serious researches and intensive studies.

Subproblem relaxation is also important in other application areas, and applies in a different sense. Sometimes constraints are not "firm", and may be relaxed without penalty. Furthermore, databases may contain inexact and unreliable information that may violate its constraints. This situation happens often in designing expert systems. This is handled in MYCIN [53] by introducing a model of approximate implication, using numbers called certainty factors to indicate the strength of a heuristic rule. The DENDRAL system [6] dealt with rough spectral data, and used confidence ratings to determine which constraints to relax.

3. Multilevel Organization for Problem Solving

In computer science, hierarchical structures, such as trees, are used extensively for data representation. In image processing, hierarchical structures, such as quad trees, organize images with multiple levels of resolution. In artificial intelligence, multilevel structures often represent different levels of abstraction of concepts.

Abstraction summarizes information, achieves a compact representation, and generalizes ideas. Cognitive models based on multilevel structures have been developed. The idea of abstraction is also defined formally in terms of lambda calculus [56]. It is like a procedure in a programming language. Polya [45, 46] emphasized the importance of abstraction for human problem solving.

Multilevel organization research can be classified into two categories. One uses the hierarchical architecture to store homogenous structures, such as quadtree, so that efficient search can be performed. The other uses the hierarchical structure to model human problem solving. Here representation work has been quite successful. Less work has been done on using multilevel structure to control the execution of a system. One possible reason is that it is easy to describe but hard to state formally. There are some exceptions, such as the A^* search [40], and multilevel A^* search [54], whose evaluation function $f(n)$ is the global control on the metalevels.

In this thesis, an attempt is made to formalize multilevel organization not only in problem representation, but also in problem solving. The concept of abstraction, levels, filtering, and control are defined. Measures on the multilevel system's performance are discussed. These measures provide means to visualize the efficiency of the system. However, the formal theory is by no means complete. It is just a first attempt to study multilevel problem solving strategy formally.

The rest of the section discusses some existing multilevel problem solving techniques. They have been applied to hierarchical planning and speech understanding.

3.1. Hierarchical Planning

Hierarchical planning and problem abstraction have been studied before by E. D. Sacerdoti [51] and others. In hierarchical planning, a goal is decomposed into subgoals. The plan to achieve the goal is composed from the plans to achieve the subgoals. Each subgoal is also decomposed into further subgoals, and so on. The hierarchical planner generates a hierarchy of representation of a plan in which the highest is an abstraction, or simplification of the plan and the lowest is a plan detailed enough to solve the problem. This concept is implemented by Sacerdoti in the problem solver ABSTRIPS [51]. The problem solver contains a set of operators. It explores the states that arise from applications of the operators, searching for one that qualifies as a solution to the problem. Each operator has a list of preconditions that must be satisfied before the operator can be applied. Making a precondition true is a subproblem of the current goal. The abstraction space is a simplifying representation of the problem space in which unimportant details are ignored. In ABSTRIPS, the abstraction space contains all the operators given in the initial specification of the problem, but some preconditions of some operators are judged to be more important than others.

Two major characteristics restrict its usage. First, the subproblems generated from the main problem are independent. The assumption is that each goal is decomposable to unrelated subgoals. Second, the abstraction is defined by ignoring some unimportant details. It takes away some constraints on the problem level and calls it the abstraction space. Real abstraction should abstract characteristics completely and correctly.

This is the same as defining all problems that have some common characteristics at a high level and studying the manipulation of the abstraction space. Human problem solving usually manipulates high level concepts while working on the lower, finer details of the problem.

3.2. Speech Understanding

Another application of multilevel organization is in the speech understanding system HEARSAY [15, 14]. It is based on a uniform and integrated multilevel structure, the blackboard, which holds the current state of the system. Key functions are performed by diverse and independent programs called knowledge sources. Knowledge sources cooperate by creating, reading, and changing elements on the blackboard. Each level in the blackboard specifies a different representation of the problem space. Example of levels are database interface, phrase, word sequence, word, syllable, segment, and parameters. This sequence of levels form a loose hierarchy. The elements at each level can be approximately described as an abstraction of elements at the next lower level.

The state of execution is represented by the multilevel structure blackboard. The control of the system is provided by the set of knowledge sources. These knowledge sources cooperate and communicate through the blackboard. The knowledge sources are not restricted to work on one level of the blackboard. For the blackboard, a partial solution at one level constrains search at another level. This architecture of control is excellent and the result is a system that works in experiments. The performance measure is done empirically: for a 1000-word vocabulary, there is a 90% possibility of correct interpretation. However, the loose definition of the levels and the unrestricted relationship between controls make the HEARSAY system hard to formalize.

3.3. Multilevel Framework

The multilevel system proposed here is much more restricted than the HEARSAY architecture. It has a much more rigid definition on level structures. Of course, one can always apply the HEARSAY structure to the Chinese input application discussed here. On the other hand, the Chinese input problem can be formulated by the proposed multilevel system. The formalism to be discussed

later justifies the validity of the multilevel approach.

The multilevel system proposed here partitions the constraints vertically into different levels. Each level contains different sets of objects. An object on one level is an abstraction of objects at the next lower level. The whole idea assumes that constraints can be categorized into different levels of abstraction. The lower level constraints describe relations among the primitive data while the higher level constraints describe relations among the high level abstract concepts.

The processing cycle initially submits all possible solutions of some problem instance to the lowest level. This level removes some impossible solutions, i.e., the solutions that contradict constraints of the level, and generates a smaller set of solutions. This set of solutions is then passed to the next higher level and the filtering process starts again. The process stops when a solution is achieved, an inconsistency is detected, or all levels are explored. This filtering process can be viewed as first relaxing all high level constraints and trying to find a solution satisfying the lowest level constraints. If ambiguities arise at the end of the processing, constraints at the next higher level are enforced to try to achieve a better solution.

Besides filtering, another kind of processing, known as control, also exists. Control is information generated and passed from a higher level to the next lower level. This information helps to reduce the size of the solution set flowing from the lower level to the higher level. One way to reduce the size of the solution is by decreasing the set of constraints; this is called increasing the filtering power of a level. This and other forms of control will be described formally later. The control information is generated using the partial input obtained so far. So there is a cycle of iteration: partial solutions are passed up the levels; the higher levels, using these partial solutions, generate control that is propagated down the levels.

CHAPTER 2

Chinese Input Problem

In this chapter, the Chinese input problem is discussed in details. Different input methods are presented: methods based on the whole Chinese character, methods based on coding scheme, and methods based on phonetic spelling. Input system based on phonetic spelling is argued to be the most appropriate approach. It can be viewed as transforming a phonetic spelling, without word breaks, of a Chinese sentence into the corresponding Chinese characters. Three fundamental issues are studied: segmentation, homophone analysis, and dictionary organization. Segmentation corresponds to inserting spaces to the connected sequence of phonetic symbols so that each subsequence is a Chinese character. The main disadvantage of phonetic input is that homophones cannot be distinguished. They are analyzed here and multilevel framework is applied to reduce some of these ambiguities. The problem is partitioned into following levels: phonetic spelling, word, and grammar.

The data organization for the Chinese input problem is also defined. It corresponds to store, access, and use the constraints. Constraints include the phonetic spelling dictionary, word dictionary, and the syntactic structure.

1. Introduction

During the past decade, computer input and output of oriental languages, such as Chinese and Japanese, have been the subject of intensive research and development in the Orient. Recently, western societies have also shown interest in these areas; for example, a current issue of IEEE COMPUTER [12] is dedicated to work on Chinese/Kanji text and data processing. The problem is

important: there are more than one billion native Chinese speakers. The biggest problem in Chinese text processing is the huge size of the alphabet. Fig 2.1 is a pictorial description of the hierarchy of different kinds of characters.

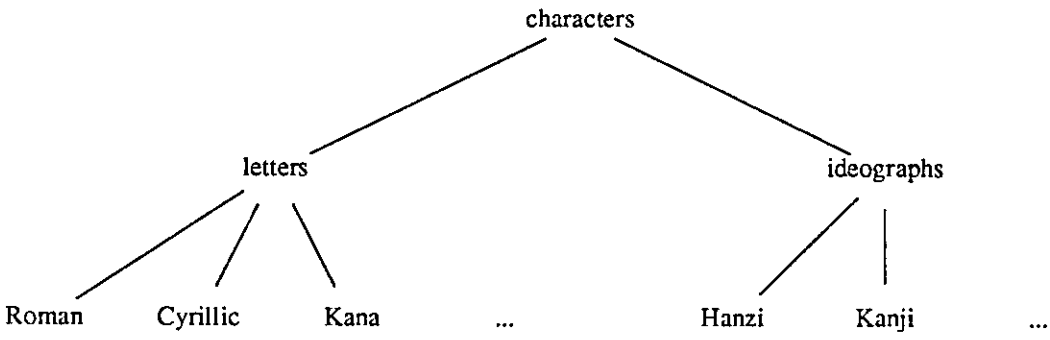


Fig 2.1 Hierarchy structure of characters.

How can 50,000 Chinese and Japanese characters be organized for computer input, output and processing? This thesis concentrates on the processing of Chinese language. The major difference in processing other oriental languages, like Japanese and Korean, will briefly be discussed in chapter 7.

Today, there are more than 50,000 ideographs in the Chinese script. They are called **hanzi** (漢 字). In order to achieve basic literacy, at least 2,000 most frequently used ideographs must be mastered.

The output problem has received satisfactory result. Nowadays, there exists quite a lot of Chinese output system in the commercial market, but none of the input method is satisfactory. Outputting ideographs is easy. The images of the ideographs are stored in the form of dot matrices.

The output devices are able to reproduce the same image, aided by necessary software and hardware. Some researches are done on using metafont [31] to represent Chinese ideographs [24] in order to achieve high quality output. Metafont is able to generate different fonts for Chinese ideographs, such as bold.

The problem addressed here is inputting Chinese to the computer. The rest of the chapter talks about the existing techniques for input, and some characteristics of Chinese languages, such as the word compounding process and basic grammatical patterns.

2. Methods of Chinese Input

Criteria for comparing different input methods are presented before the discussion of them. Cui [13] recently studied methods of evaluating Chinese character keyboards. His evaluation criteria include input speed, learning curves, and error rates. The most important criterion for a good input method is **ease of use.** As computer gains popularity, even intra-office communication is done through the machines. A good input method should appeal to both naive users and professional typists. Most existing methods require professional typists. Another criterion is **avoidance of expensive hardware**; the ASCII keyboard satisfies this criterion. The method should also be **independent of which version of the Chinese alphabet one is using,** such as the normal alphabet, the variant alphabet, or the simplified alphabet.

Current work on inputting systems for hanzi can be divided into three categories: based on the whole character, based on coding scheme, and based on phonetic values.

2.1. Input systems based on the whole character

One implementation is the two-dimensional selection array. In this scheme, the thousands of available characters are laid out in a huge two-dimensional array of keys. Fig 2.2 is a Chinese

typewriter and fig 2.3 is part of the keyboard. The typist visually searches for the desired hanzi, one at a time. For example, in fig 2.3, ri (日) is at column 8 and row 6. Another implementation is each character is input by pressing a single key, or by simultaneously pressing a "character group" key and a corresponding key.

The main disadvantage of this kind of design is the huge keyboard, which is hard to search, and discourages non-professional users. The size of the keyboard also limits the possible number of hanzis. Once the keyboard is set, it is very difficult to expand.

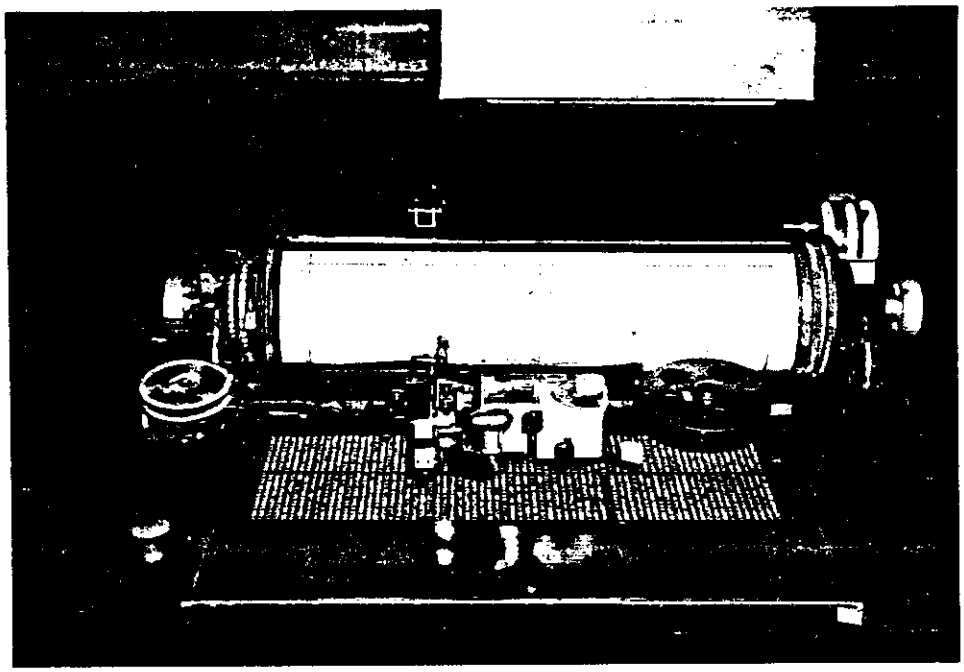


Fig 2.2 A Chinese typewriter.

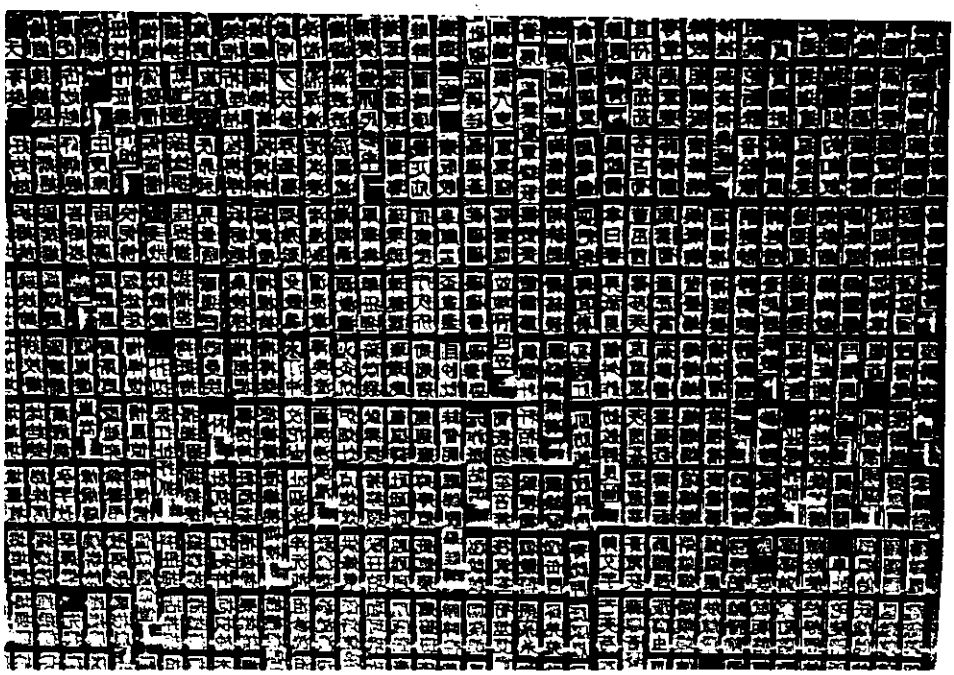


Fig 2.3 A Chinese typewriter keyboard.

1.1. Input systems based on coding schemes

In order to avoid the usage of a huge size keyboard, people devised coding schemes. The goal of coding schemes is to represent each hanzi by a sequence of keys such that all keys can be fitted on a small keyboard. One coding scheme is designed to minimize the number of keystrokes required to input each character. On a standard ASCII keyboard, there are about 40 keys. The shift key will double the number of possibilities to 80. Two key strokes is able to produce 6,400 (80×80) different combinations to represent 6,400 different ideographs. In this scheme, the typist has to memorize the keystroke pairing for each individual hanzi.

Many other encoding methods have been proposed, but none is universally accepted. These include the Three Corner Coding Method [26] and the Pinxxiee Method [58]. In a coding scheme, the typist analyzes each character in turn and figures out an input code derived according to some system of coding rules. For example, the three corner coding method returns following codes:

tian	田	01 00 00
ri	日	01 04 41
huo	火	99 99 99

The coding rules usually depend on the character's shape, sound or both. Special rules are applied when ambiguity appears, such as when the general rules produce the same coding for two different hanzis. Some existing coding systems depend on the arrangement of radicals within a character. Research is being done on automatically generating the character from the radicals input by the user. No success has yet been reported [62].

The basic idea of coding schemes is decomposing a character into its components. The main disadvantage is that there is no reliable definition of what constitutes an element within a hanzi. It is also not clearly understood how hanzis are divided into elements and how many elements are needed to form one hanzi. This can be overcome by artificially introducing some kind of standards and rules, such as the Three Corner Coding Method.

Some work has been done on designing special keyboards to help the typist and to achieve maximum productivity.

Coding methods in general have the following disadvantages:

- They are hard to learn.
- They require memorizing numerous complex rules, exceptions and special cases to handle the ambiguous cases.
- It is extremely difficult to compose text directly at the keyboard.
- They are not easy to use and unsuitable for non-professional typists and occasional users.

2.3. Input systems based on phonetic values

In phonetic systems, the typist spells out the sound of the desired hanzi and the computer locates it in the dictionary database. If several hanzis correspond to the same sound, the system displays all possibilities and the typist indicates the one intended by a pointing device.

One disadvantage of phonetic input is that people who can speak a certain dialect may not be able to input ideographs unless they know how to pronounce them correctly. There are numerous different dialects in Chinese. Twenty five years ago, the Chinese Phonetic Scheme was officially published. Since then, it has been used in the teaching of Chinese children in primary schools where youngsters are obliged to master a phonetic alphabet. In the near future, most Chinese will be able to

use the standard Chinese Phonetic Scheme.

The Chinese Phonetic Scheme is in standard romanization, which is called *hanyu-pinyin* (漢 語 併 音) (''Chinese language pronunciation''). Another common phonetic alphabet used in Taiwan is called *zhuyin-fuhao* (注 音 符 號) (''phonetic symbol''), it is familiarly known by its first four letters *bopomofo*. Either of these standards solves the problem of dialects. It would be nice if there were just one standard, but that is a political problem, not a technical one.

Recently special keyboards were designed for *pinyin* input [52]. They were based on some inherent properties of the pinyin construction; for example, the phonetic symbols can be divided into initial consonants and ending consonants. For example, { *b* , *c* , *d* , *f* , *g* , *h* } are initial consonants, and { *a* , *an* , *ang* , *ao* } are the ending consonants. A pinyin is constructed from following two rules:

1. initial consonants – final consonants,
2. final consonants.

{ *a* , *ba* , *da* , *an* , *ban* } are legal pinyins and { *b* , *c* , *anan* } are illegal pinyins. Much work has been done on designing a new keyboard to make input efficient or possible. Now there are two to three hundred different designs for a Chinese character keyboard, but no single method has gained unanimous acceptance. There is a growing tendency to favor standard ASCII keyboards. The Qwerty arrangement has been criticized by people for years, but has not been replaced yet.

The main disadvantage of phonetic input is that homophones cannot be distinguished. The pronunciation of following three hanzis are identical.

{ 按 , 暗 , 岸 }

There are approximately 400 different phonetic spellings which represent 20,000 hanzis. Following list of hanzis are with the same phonetic spelling, *an* .

{安,按,暗,岸,菴,黯,鞍,俺}

Numerous pronunciations can be generated as the result of tonal variations [52]. Only 1,284 romanized transcription of hanzi are fully distinctive in the sense of corresponding to only one ideograph. The Mandarin dialects has four possible tones, namely, *ping* (平), *shang* (上), *qu* (去), *ru* (入), plus a distinct *qing* (輕) for unstressed syllables. They are usually denoted as follow:

–	*ping*
ˇ	*shang*
`	*qu*
´	*ru*
	qing

For example, the five tones of the phonetic spelling *bo* contain following different hanzis:

bō	波
bǒ	跛
bò	薄
bó	帛
bo	卜

A variation of phonetic conversion is the *word-unit phonetic conversion* method proposed by Becker [4]. Becker pointed out that the fundamental design mistake in some methods is that they

require the typist to enter Chinese text hanzi by hanzi rather than word by word. He argued that most Chinese words are compounds consisting of two or more morphemes. For example, a typical Chinese dictionary lists 15 different hanzi pronounced *dian* and 5 different hanzi pronounced *nao* – but only one compound word pronounced *diannao* which is 電 腦 (''computer'').

Becker stated that this phonetic redundancy built into Chinese vocabulary can radically reduce the ambiguity of successive hanzis. In word-unit phonetic conversion, the system designer develops a Chinese word dictionary that is stored in the computer. Once the computer has been given this dictionary, the process of communicating Chinese text phonetically is analogous to speaking Chinese to a person who knows the language. A study by Becker showed that word-unit phonetic conversion is equal or superior to other typing systems [4].

Next section discusses Chinese words in details and analyzes Becker's approach.

3. Chinese Words

In Becker's approach, word units must be well defined and universally accepted. In this section, the notion of ''words'' is studied with reference to Chinese lingustic literature. We shall see that the concept of *word unit* is not well defined but it does exist. A new approach for input is proposed which makes use of the underlying structure of words and grammar.

3.1. What is a word?

Chinese language is sometimes referred to as monosyllabic, meaning that the vast majority of words are one syllable in length. Morpheme is the smallest meaningful element in language. In classical Chinese, most morpheme units are individual monosyllabic words [7, 33, 34]. For example, the following sentence from the poem 清 平 調 (*qing ping diao*) by 李 白 (*Li Bai*) does not contain any polysyllabic words.

名花傾國兩相歡.

ming hua qing guo liang xiang huan.

In modern Chinese, words are frequently combinations of two or more syllables. A morpheme is free if it can occur as a word, otherwise, it is called a bound morpheme. As many morphemes can occur only in bound form, words are largely disyllabic or polysyllabic morphemes. For example, in the word *fouze* (否則), the second component, *ze* (則), is no longer functioned as a free morpheme in modern Chinese.

So, what is a word? A word should be a unit in the language characterized by syntactic and semantic independence and integrity. It should be able to express one idea. There are several reasons why this definition of word should be adopted [33]. First, this definition is more in line with the way "word" is viewed in other languages and not so narrowly tied to the Chinese writing system. Second, this position tends to agree with the perception of most people attempting to learn the Chinese language. Third, due to the phonological changes that have taken place, many formerly distinct syllables have become homophones in Mandarin. For example, in Mandarin, 要 and 藥 are pronounced as *yao*, where as in Cantonese, 要 is pronounced as *yiu* and 藥 is pronounced as *yeuhk*. The threat of too many homophonous syllables has forced the language to increase dramatically the proportion of polysyllabic words to help reduce the possible ambiguities.

In classical Chinese, no spaces between words are necessary because each ideograph represents one word. This tradition has been kept in modern Chinese. Because strings of ideographs are never broken into strings of words by means of blanks, there is no standard for exactly what constitutes a word as in English.

Despite the lack of a standard, research on linguistic has attempted to find reasonable definition for "word". This work is based on the study of morphology, the internal structure of words. Words consist of one or more morphemes, compounded together.

Compounds are polysyllabic units that can be analyzed into two or more meaningful elements, or morphemes, even if these morphemes cannot occur independently. A compound can be created in the compounding process in different ways. Some compounding processes are:

- reduplication, e.g., *ge-ge* (brother, 哥 哥).
- affixation, e.g., *di-yi* (first, 第 一).
- nominal compounds construction, e.g., *chuan-dan* (bed-sheet, 牀 單).
- verbal compounds construction, e.g., *da-po* (hit-broken, 打 破).
- verb object compounds construction, e.g., *ge-ming* (revolution, 革 命).

There is much disagreement over the definition of *compound*. No matter what criteria one picks, there is no clear demarcation between compounds and noncompounds. There is also no agreement on whether a given form should be regarded as one word or two [33], such as *kan-jian* (看 見 "see"), there is no agreement on whether it should be regarded as one word or two. This illustrates Becker's assumption of a "universally accepted definition of word-unit" is impractical. On the other hand, words do exist in Chinese even if they are not standardized. Thus, while the input method should not have any assumption about the user's knowledge of word units, the system processing the input can use its own opinion of word units to aid processing.

3.2. Categories of Words

A word dictionary contains not only words and phrases, but also bound morphemes and morpheme complexes. They include:

- Determinatives: zhe 這 (this), na 那 (that), san 三 (three), ji 幾 (several);
- Measures: ge 個 (individualization classifier), jian 件 (item, article), cun 寸 (inch);
- Affixes: di 第 (-th);
- Particles: a 阿 (ah!);

Root words are morphemes which underlie primary derived words. They combine with other morphemes to generate compounds.

Though most of these entries are listable, their unrestricted combinations will add up to an enormous number. It is impossible to include in the dictionary such words as:

yitian 一 天 (one day),
santian 三 天 (three days),
shitian 十 天 (ten days),

though these are single words in a syntactic sense. There is an unlimited number of these *transient words.*

Some of these transient words can be generated by regular expressions. One example is the compounds generated by the regular expression

D (digit) = 一 | 二 | 三 | 四 | 五 | 六 | 七 | 八 | 九 | 零
N (number) = D^+

This regular expression generates Chinese numerals similar to the regular expression that generates Roman numerals. A more complete description is given in chapter 3. The total number of possible words in this form is unlimited. The regular expression formalism provides a compact representation for the unlimited transient words.

Note that this regular expression can generate some meaningless words. Since the only function of word analysis is to reduce ambiguities, the extra meaningless words will rarely cause problem. One of my assumptions is the user inputs a syntactically correct sentence.

Thus there are two kinds of words: static and transient. The static words can be listed in a word dictionary while the transient words are generated by regular expressions.

Following section defines an input method based on phonetic conversion and shows how homophones ambiguities will be tackled.

3.3. A Phonetic Input Method

From the arguments in the previous section, words are not as well defined in Chinese as in English. So an input method for Chinese should not depend on any user-supplied word boundaries. But the system can use the knowledge of words to resolve the ambiguities that arise from homophones.

The proposed system, called the *Chinese Input System*, does not require the user to insert spaces between consecutive syllables or words. The user may have his own definition of words. The input to the system is just a sequence of phonetic symbols.

Becker said that most compounds have no homophones even when typed without tone. Table 2.1 below displays the homophones appeared in a Chinese word dictionary of approximately 20,000 entries. There are about 10% of compound words with homophone if there is no tonal indications. This implies tonal indications are necessary in most cases.

Total number of words	Total number of homophones without tonal indication	Total number of homophones with tonal indication
20,000	2,299	471
100%	11.5%	2.4%

Table 2.1 Number of homophones.

The tonal indication is only applied to the vowels, *a, e, i, o, u*. Table 2.2 indicates the number of homophones if tonal indication is used in vowel *a, e, i, o, u* individually. If tonal indication is used on vowel *a*, about 50% of the homophones can be distinguished. Similar situation applies to vowel *i*.

Total number of homophones without tonal indication	Total number of homophones with tonal indication on vowel				
	a	e	i	o	u
1,828 100%	1,080 47%	1,536 67%	1,057 46%	1,612 70%	1,512 66%

Table 2.2 Number of homophones if tonal indication is used.

If both vowel *a* and *i* are used with the tones specified, an estimate of 70% of the homophones can be distinguished. The total number of homophones with tonal indications on vowel *a* and *i* is about 3% ($10\% \times 30\%$). In this case, the user has to type in the tone indications for *a* and *i*.

4. The Chinese Input System

The *Chinese Input System* takes a sequence of phonetic symbols, and displays the corresponding Chinese ideographs. For example, the user can input:

> I am a student of the Computer Science Department of the
> University of California, Santa Barbara.

The corresponding Chinese pinyin sentence is:

woshijiazhoudaxueshengbabalaxiaoyuandiannaoxidexuesheng. (1)

As the user types in the above sequence of phonetic symbols, with spaces arbitrarily inserted, the Chinese Input System should display the following list of hanzi ideographs:

我 是 加 州 大學聖巴巴拉校園電 腦 系的學生 .

The system transforms a sequence of phonetic symbols typed in by the user to a sequence of hanzi ideographs corresponding to the input sequence. Ideally, the output sequence of hanzi should be unique and also what the user wants. This problem consists of the following two components: the control mechanism and the data organization.

4.1. The control mechanism

The control mechanism of a problem solving system decides what the system do next. For the Chinese input problem, the control mechanism is divided into the following three subproblems: the Chinese character break problem, the word break problem, and the hanzi labeling problem.

4.1.1. The Chinese Character Break Problem

Since the input is just a sequence of phonetic symbols, the first processing is to break the sequence into subsequences such that each subsequence corresponds to a legal syllable, i.e., each

subsequence corresponds to some hanzi ideographs. Breaks between syllables are denoted by blanks.

The sequence (1) above corresponds to the following list of syllables:

wo shi jia zhou da xue sheng ba ba la xiao yuan dian nao xi de xue sheng. (2)

Since each syllable corresponds to a hanzi, or a Chinese character, the spaces inserted are called *character breaks*. This problem is called *character break problem.*

Ambiguities may appear. For example, the sequence

xianggong. (3)

has two possible parses, one with three syllables and the other with two syllables:

xi ang gong (4)

xiang gong (5)

After identifying the character breaks, the system can assign hanzi to each syllable. Even if one has a unique parse, the total number of different possible sentences generated from assigning different homophonous hanzis to the substrings is enormous. Table 2.3 displays the number of different hanzis corresponding to each substring in (2) above.

If there are more than one parse, further processing has to be performed on all possible parses. The "best" solution is selected as the final answer. The "best" here means the solution with the least amount of ambiguities. Some other optimality criteria can be used. Another possibility is to display the best couple of solutions such that their amount of ambiguities are close, then the user selects the right one.

syllable	number of homophones without tonal indication	number of homophones with tonal indication
wo	10	1
shi	52	26
jia	21	10
zhou	17	7
da	6	1
xue	6	3
sheng	14	5
ba	14	4
ba	14	4
la	7	2
xiao	22	6
yuan	20	11
dian	15	10
nao	5	3
xi	45	5
de	3	3
xue	6	3
sheng	14	5

Table 2.3 Number of hanzis for syllables in (2).

The total number of possible sentences is about 10^{21}. Even if tonal indication is considered, the number is still about 10^{11}. A selection method based on enumerating all sentences is impractical.

Selection may also be done based on enumerating all possible hanzis for each syllable. For each syllable, all hanzis with that phonetic spelling are displayed and the typist selects the desired one by a pointing device. If we does selection on each character [28], we must still do 18 selections, one for each character, which is impractical for an online input method.

The proposed Chinese Input System uses the word knowledge to filter out some of the impossibilities. Further filtering is done by constraints generated from the grammatical properties of Chinese.

4.1.2. Word Break Problem

Now a list of syllables exists; each syllable corresponds to more than one hanzi. Next the system checks whether adjacent syllables form a word or compound according to the dictionary. As words or compounds may consist of more than two syllables, it is necessary to make sure that these compounds are included in the checking procedure. In the above example (2), the list of syllables consists of the following compounds:

jiazhou , *daxue* , *shengbabala* , *xiaoyuan* , *xuesheng*. (6)

Table 2.4 illustrates the number of homophone compounds correspond to these polysyllables.

polysyllables	no. of homophones without tonal indication	no. of homophones with tonal indication
jiazhou	1	1
daxue	2	1
shengbabala	1	1
xiaoyuan	1	1
xuesheng	1	1

Table 2.4 Number of homophone compounds corresponds to (6).

The total number of sentences that can be generated after identifying these compounds is 4,680 without tonal indication and 78 with tonal indication. The number of monosyllabic words left now is 3: *wo*, *shi* and *de*. Further processing is necessary on these words.

Since this current process tries to identify the words, it is called the *word break problem.*

4.1.3. Hanzi Labeling Problem

The result from the word break problem is a list of strings, each string represents either a single character or a word. Each of these strings contains a list of possible ideographs which may be a possible components of the input sentence. If one is lucky, the number of possible labels in each string is one, the problem is solved. This rarely happens. Single character words always exist in Chinese sentences. The most common kinds of single character words are prepositions, called *coverb.* At this stage, the system will try to eliminate the impossible labels, or ideographs, based on some constraints. For example, some hanzis cannot exist by themselves; they always appear as part of a compound. This elimination process is similar to the labeling problem subject to a set of constraints.

Where do the constraints come from? There are always rules governing the construction of sentences in a language. Work has been done on Chinese grammar [7, 34, 61]. The grammar rules will be used to generate constraints to help resolving ambiguities.

In the output from the Word Break Problem, ambiguities have been greatly reduced. One could attempt to use the whole set of grammar rules to parse the input sentence and help to resolve the ambiguities. Since parsing of Chinese sentence is difficult and I am not interested in parsing, I believe that some constraints generated from the grammar will help to resolve ambiguities and will be inexpensive to apply. One such example is that a final particle,if it exists, always appears at the end of a sentence.

4.2. Data Organization

Data organization for Chinese text processing itself is an interesting subject. There are basically three problems:

1. Store and index the Chinese ideographs.

2. Store and index the Chinese dictionary.

3. Store, access and use the constraints.

Problem (1) has been studied for years by researchers and practitioners in the Orient. Existing techniques include storing the bitmaps that represent the images of the ideographs. The database of Chinese ideographs at UCLA contains 6,000 Chinese and Japanese characters [28]. The bitmap of each ideograph is stored in 72 bytes. Each bitmap is of size 24×24 pixels. Since phonetic conversion is used, each ideograph is indexed by its phonetic spelling. The size of the ideograph database is 432 Kbytes (72bytes $\times$ 6K ideographs).

Since much work has been done on compacting this kind of database, I am not going to pursue any further in this direction. Instead, concentration is put on solving problem (2) and (3).

4.3. The Dictionary Organization

The dictionary organization refers to the problem (2) above. The dictionary consists of two components:

- Hanzi dictionary – an index to the ideograph database.
- Word dictionary – store the words and compounds.

The hanzi dictionary constitutes only a small portion of the dictionary as there are less than 500 different spellings of ideographs. The word dictionary is much bigger. The number of common static words is about 20,000 and there are unlimited number of transient or temporary words. A careful design is required so that each word can be retrieved efficiently and the storage utilization is reasonable.

An implementation of the static word dictionary in the form of a trie based on each individual compounds' phonetic spelling was built. The dictionary used is 漢語併音詞匯 [11]. It contains 12,913 leaves and 22,568 internal nodes. Each of the leaves and internal nodes require 1 index and 1 pointer. There are approximately 35K pointers and indices. As there are about 20,000 words in the trie and each word contains at least 2 ideographs, there are at least 40,000 pointers to the ideograph database. Table 2.5 summarize the storage requirement for the above discussion.

	Approximate number	Number of pointers
2-ideograph compounds	12,000	24,000
3-ideograph compounds	3,000	9,000
Pointers to ideograph database		33,000
Compounds	20,000	
Internal nodes	22,000	
Leaves	13,000	
Internal pointers for the trie		35,000
Internal indices for the trie	35,000	
Size of ideograph database		432,000 bytes
Total number of pointers		68,000
If pointer is 4 bytes, storage required		272,000 bytes
Total number of indices	35,000	
If index is 1 byte, storage required		35,000 bytes
Total storage		739,000 bytes

Table 2.5 Storage requirement for the dictionary.

A better implementation of the dictionary will be presented in the next chapter. The design goal is not only efficient storage utilization, but also fast input recognition.

4.4. Constraints Organization

Constraints are the set of rules that cannot be stored in the dictionary. The number of rules is not very large, so efficient storage implementation is not necessary. The construction of rules is more important. Two methods for constraint construction are discussed, automatic generation of constraints, and heuristic rules.

Constraint generation technique applies to a context free grammar for a language. An iterative algorithm is presented to generate constraints from the given context free grammar. A complete formal grammar for Chinese would have made the work reported here far more rigorous and, perhaps, more interesting. Unfortunately most of the linguistic studies of Chinese grammar have not formalized their results; the major exceptions are the work of Wang [61] and Hashimoto [21]. With a formal set of grammatical rules, one could attempt language parsing, language understanding and others. My goal is just to use the grammatical rules to help deciding what characters the user wants. The grammatical rules are used to generate constraints to discard implausible sentences. For example, the constraints state the possible adjacent parts of speech. If one has a context free grammar that is a super set of the Chinese grammar, one can apply the constraint generation algorithm to generate the set of constraints. This set of constraints is able to remove some impossible solutions. Details of constraint generation will be discussed in chapter 4.

Since there is no universally accepted Chinese grammar, heuristic rules are needed to further reduce the ambiguities. Most ambiguities appear in single hanzi that do not form words. Heuristic rules construction concentrates on rules that resolving these ambiguities. Following grammatical structures generate most of the single hanzi in Chinese writing:

- preposition,
- particles,

- conjunctions.

The set of heuristic rules will be be studied in detail in chapter 4.

CHAPTER 3

The Dictionary

Dictionary organization is a fundamental problem in computer science. In practical use, e.g., a spelling checker, an efficient implementation will save many resources. This chapter describes the organization of a Chinese dictionary. Since there is no published work on efficient Chinese dictionary organization, and since a Chinese dictionary is quite different from an English dictionary, it is worthwhile to spend some time on its design.

As discussed in the last chapter, phonetic spelling is the most reasonable way to type Chinese. In order to handle input by phonetic spelling, the Chinese dictionary is indexed by pinyin. A design of a Chinese dictionary is presented here. It can store more than 6,000 ideographs and 20,000 words.

There are two kinds of words: static and transient. The static words are listable, though there is a large number of them. On the contrary, there are unlimited number of transient words. An efficient implementation for the static word dictionary are presented first. The transient words can be generated from regular expressions, presented last. One important point is the regular expression not only generates the legal transient words, but also generates some illegal words. In the current system, the user input is assumed to be valid and the main goal of the dictionary is to verify the existence of a word.

In order to facilitate the search algorithm presented later, the dictionaries: phonetic spelling and word, are stored as tries. This chapter starts with some terminology, then informally specifies the Chinese dictionary, and analyzes the current system. The informal specification consists of the

definition of the dictionary and the functions available to access it. Storage analysis is based on an existing Chinese word dictionary, 漢語併音詞匯 [11].

1. Terminology

As discussed in the last chapter, pinyin input is the most reasonable way for typing Chinese. In order to handle input by pinyin spelling, the Chinese dictionary is indexed by pinyin. A Chinese dictionary should contain at least 6,000 common ideographs (Chinese characters). These 6,000 ideographs are indexed by approximately 400 different phonetic spellings. Each pinyin spelling thus corresponds to an average of about ten different ideographs. As pointed out in the previous chapter, words are often the basic units of communication. A word dictionary is necessary in addition to an ideograph dictionary. A common Chinese dictionary should contain 20,000 to 30,000 static words for daily usage. There is also an infinite number of transient words.

Let the 6,000 ideographs be the set of Chinese characters available in the dictionary. Each of the ideograph is called *zi* (字). These 6,000 zi form the *zi universe* (*ZIu*). Each zi is indexed by its phonetic spelling, called *pinyin* (併音). Each pinyin may correspond to more than one zi. It is a one to many relation. The collection of these pinyin relations is the pinyin character dictionary, or *zi dictionary* (*ZID*). Another level of the dictionary is the word dictionary, which stores the common words. Each element in the word dictionary is a sequence of pinyins, called *ci* (詞), and the corresponding sequence of ideographs. The word dictionary is also called *ci dictionary* (*CID*). To speed up the processing which will be discussed later, both the pinyin character dictionary and the word dictionary are stored as tries.

2. Specification of the Chinese Dictionary

The following definitions will be used throughout the chapter.

ZIu : Set of ideographs.

Pu : Set of phonetic symbols, { *a, b, ..., z* }.

PINYINu $\subseteq Pu^+$: The set of pinyins.

ZID : $PINYINu \times 2^{ZIu}$, a relation from phonetic spelling to set of ideographs. Each element is an ordered pair of a pinyin and all ideographs with this pinyin.

E.g. { (*gang* , [宮, 弓, 共, 公, 工]), (*zheng* , [政, 整 , 正 , 徵 , 爭 , 証]) }.

CID : Set of relations from sequence of phonetic spellings to ideographs. Each element is an ordered pair of pinyin for a word and all corresponding words of ideographs.

E.g. { ([*gong,zheng*] , [公正 , 公証] }.

zi $\in$ *ZIu* : A ideograph.

ZIs $\subseteq$ *ZIu* : A set of ideographs.

pinyin $\in$ *PINYINu* : A legal phonetic spelling.

$pinyin^+$: A nonempty sequence of *pinyin* s.

ci $\in$ *ZID* : A nonempty sequence of pinyins that form a Chinese word.

ps $\in$ *Pu* : A phonetic symbol.

ps^+ : A nonempty sequence of phonetic symbols, not necessary a pinyin or a word.

Notation	Representation	Size	Approximate total number
a b c	phonetic symbols	1 byte	26
[ang] [ci]	pinyins	2 bytes	500
阿 嗞	ideographs	2 bytes	6,000
[公証]	Chinese words	2 bytes	20,000

Table 3.1 Some notations.

4. The Static Dictionary

The static dictionary has two parts: the dictionary table and the index. The dictionary table is a collection of tables and the index is a trie with pointers to the dictionary table.

4.1. Dictionary Table

The dictionary table consists of a table of words of two ideographs (2-ideograph table), a table of words of three ideographs (3-ideograph table), ... , a table consists of words of n ideographs (n-ideograph table). The number of these tables depends on the length of the word with the maximum number of ideographs. Usually, the maximum length is five. Words rarely contain more than five ideographs.

The 2-ideograph table is a list of ordered pairs of indices to the *ZIu*. Each pair represents a legal word of two ideographs. Words with the same phonetic spelling are grouped adjacent to each other. Only one index is necessary to address a group of ideographs with the same phonetic spelling. The arrangement can be pictured as follows (fig 3.2):

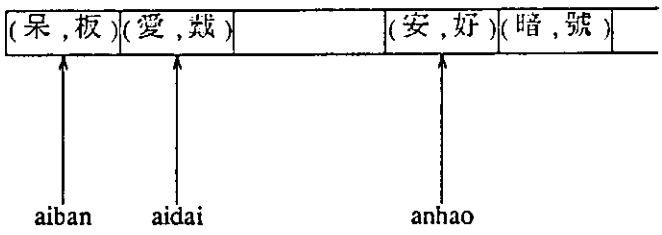


Fig 3.2 The 2-ideograph table.

The 3-ideograph table is a list of ordered triples of indices to the *ZIu*. Each of these triples represents a word of three ideographs. Similar to the 2-ideograph table, the words with the same phonetic spelling are stored adjacent to each other. Following is an example (fig 3.3):

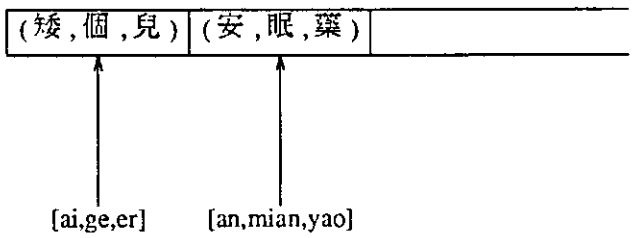


Fig 3.3 The 3-ideograph table.

In general, n-ideograph table is a list of ordered n-tuples of indices to the *ZIu*. Each of these n-tuples represents a word of n ideographs. The words with the same phonetic spelling are stored adjacent to each other.

4.1.1. Indexing

The *ZIu* is indexed by the *ZID*. Since the key values of the dictionary, the phonetic spellings, are of varying sizes, an appropriate implementation for *ZID* is by a trie. This is called the *zi trie*. A trie is a tree in which the branch at any level is determined not by the entire key value but by only a portion of it [2]. In the current implementation, the branching at the *i*th level is determined by the *i*th character of the phonetic spelling. The following list is the beginning of the pinyin dictionary:

{ a, ai, an, ang, ao, ba, bai, ban, bang, ... }

Fig 3.4 is a picture description for this part of the trie. The structure of the nodes will be discussed in details in the implementation section.

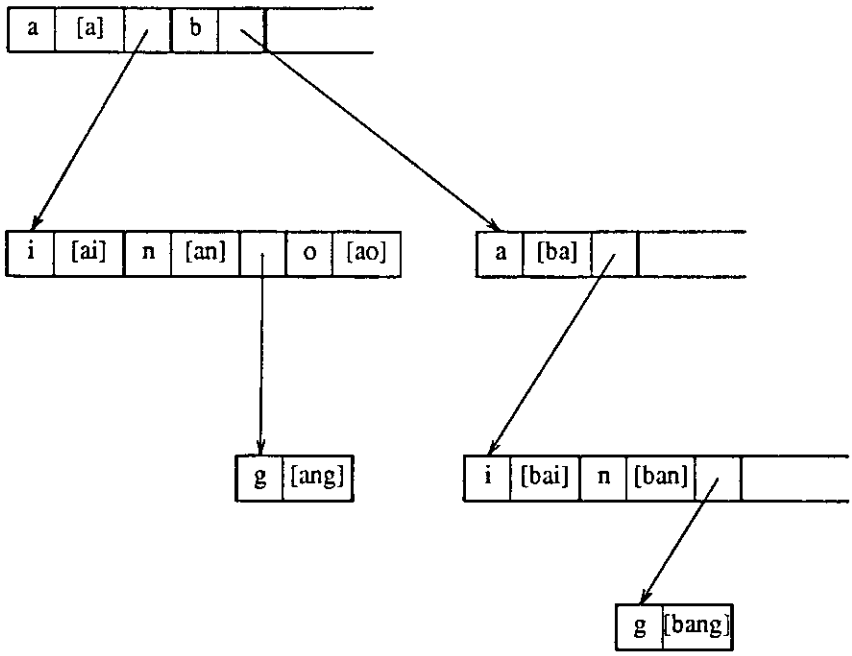


Fig 3.4 The zi trie.

The word dictionary is a list of legal Chinese words. It is indexed by the sequence of pinyins of the words. The word dictionary is implemented as a trie with the nodes at the *i*th level being the *i*th pinyin of a word. This trie is called the *ci trie*. Fig 3.5 is an abstract picture for the ci trie with the words "aiban" [呆 板], and "aidai" [爱 戴].

Followings are the functions available to the ideograph data base, *ZIu*.

add_ zi(*pinyin,zi,ZIu,ZID*) : (*ZIu,ZID*)
pronounce_ zi(*zi,ZIu,ZID*) : *pinyin*

The function add_ zi inserts a new ideograph to the *ZIu*. The corresponding pinyin, which is used as its index, is inserted to the *ZID*. They are linked together by a pointer from *ZID* to *ZIu*. The function pronounce_ zi return the phonetic spelling of a *zi* (ideograph). The major usage of the dictionary is to retrieve stored information. Implementing the dictionary with tries makes information retrieval and processing very efficient. On the other hand, the proposed data organization makes modification very inefficient because the tries are stored sequentially. The add_ zi function here and the add_ ci function below allows user to modify the dictionary if necessary. It is very rare to modify the dictionary once it is set up.

Following is the set of functions used to access the pinyin character dictionary, *ZID*.

is_ zi(*pinyin,ZID*) : boolean
retrieve_ zi(*pinyin,ZID,ZIu*) : *ZI*
is_ zi_ prefix(ps^+,*ZID*) : boolean

is_ zi checks whether the pinyin is a legal phonetic spelling or not, i.e., whether it corresponds to a set of ideographs or not. retrieve_ zi retrieves the set of ideographs corresponding to the spelling pinyin. is_ zi_ prefix checks whether the sequence of phonetic symbols, ps^+, corresponds to a prefix of a legal phonetic spelling.

Following is the set of functions to access the ci dictionary, *CID*.

add_ ci($pinyin^+$,zi^+,*CID*) : *CID*
is_ ci($pinyin^+$,*CID*) : boolean
is_ ci_ prefix(ps^+,*CID*) : boolean

3. The Zi Universe (*ZIu*)

The *ZIu* consists of 6,000 ideographs. Though each ideograph is a bitmap, it is treated as a single unit. The universe can be viewed as a set of ordered pairs of indices and bitmaps, (*N* ,*bitmap*), such as

{ (1, 阿), (2, 哀), (3, 愛), (4, 挨), (5, 癌), ... }.

Each ideograph is uniquely identified by its index. All zis with the same pinyin are stored adjacent to each other. Fig. 3.1 is part of the *ZIu*. Table 3.1 are some notations for figures throughout the chapter.

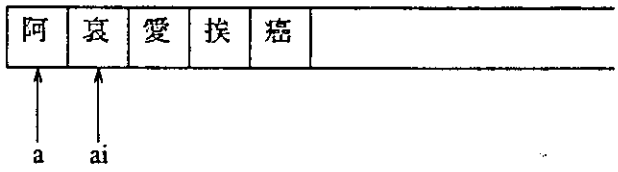


Fig 3.1 Part of the *ZIu*.

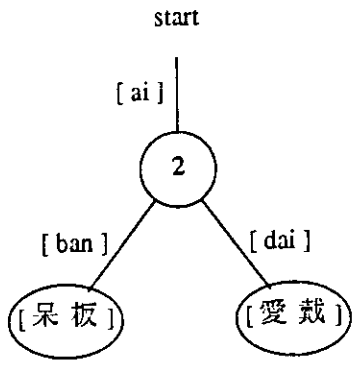


Fig 3.5 The ci trie.

5. Implementation and Storage Analysis

In the following analysis, I assume that one byte consists of eight bits, and 1 K is 1024.

5.1. The Zi Universe (*ZIu*)

In *ZIu*, each zi is a bitmap. Research has been done on how to compress the storage utilization for these ideographs. Some work has been done on implementing this data base in hardware for fast retrieval and low storage requirements. This problem will not be discussed any further. The current implementation at UCLA [28] stores each ideograph by a 24×24 bitmap. Each bitmap can be represented by 72 bytes. The total storage utilization is 432K bytes ($72 \times 6{,}000$). Since there are less than 64K zi under normal circumstances, 2 bytes are enough to index this set of bitmaps. These bitmaps are stored in an ascending order of their pinyin spellings. All ideographs with the same pinyin are stored adjacent to each other. Fig. 3.6 is parts of the *ZIu*.

Fig 3.6 The *ZIu*.

5.2. The Dictionary Table

The dictionary table is implemented sequentially. The 2-ideograph table is an array of pairs of numbers. The 3-ideograph table is an array of triples of numbers. The n-ideograph table is an array of n-tuples of numbers. Each of these numbers is 2 bytes long. With reference to the dictionary discussed in chapter 2, the storage requirement for the dictionary tables is approximately 70K bytes. The detail calculation is as follow:

		Storage Requirement (bytes)
Number of 2-ideograph compound (4 bytes each)	11,993	47,972
Number of 3-ideograph compound (6 bytes each)	3,376	20,256
Number of 4-ideograph compound (8 bytes each)	62	496
Number of 5-ideograph compound (10 bytes each)	2	20
Number of 6-ideograph compound (12 bytes each)	2	24
Total		68,768

Table 3.2 Storage requirement for the dictionary tables.

As words with the same phonetic spelling are grouped together, a tag is necessary to identify the beginning of a group. Since 15 bits is enough to uniquely identify an ideograph, the extra bit can be used as a tag to indicate a break between groups of words.

5.3. The Tries

5.3.1. The zi trie

Each node of the trie is a variable size list of structures. Each structure is a union of three possible types:

- Grey an ordered triple of a phonetic symbol, a pointer to the next node, and a representation of the pinyin.
- Black an ordered pair of a phonetic symbol and a pointer to the next node.

• White an ordered pair of phonetic symbol and a representation for the pinyin spelling.

Fig 3.7 is a pictorial description. In the diagram, "ps" corresponds to a phonetic symbol, and the sign "#" corresponds to a number which represents a pinyin.

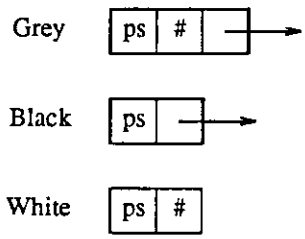


Fig 3.7 Node structure of the zi trie.

The phonetic symbol in the structure is part of the key, and it is used to construct the original key. For the grey nodes and the white nodes, the representation of the pinyin indicates that a valid key can be constructed by combining the partial key at each level, starting from the root to the current node. Two bytes are enough to represent all possible pinyins and these two bytes are called *coded-pinyin*.

The nodes of the trie are varied in size since the number of different phonetic symbols appears as the *i*th character of a spelling is varied. As there are approximately 500 different spellings, two bytes are more than enough to represent them.

An exhaustive counting of pinyin indicates the total number of nodes in the trie is less than 2K. Hence, two bytes is enough to store a pointer to a node. Black node and white node can fit in 4 bytes.

while the grey node requires 6 bytes. A loose upper bound for the storage requirement of the zi trie is 12K bytes (2K nodes × 6 bytes).

Since a node can have at most 26 children (the set of all possible phonetic symbols), a sequential search through the node to locate a correct partial key is not inefficient. Adding an extra byte to the beginning of a node and some modifications to the node structure allow a binary search on the node to locate the partial key. This idea is used in the implementation of the node structure of the ci trie.

The whole trie can be implemented sequentially, without storing any addresses explicitly. Two bits are needed to tell the type of a given node. One bit is used to indicate the beginning of a node. Since 6 bits is enough to represent a phonetic symbol, there exists 10 extra bits out of the 2 bytes that is reserved to store the phonetic symbol.

Though 12K bytes is a very loose upper bound for the zi trie, it will be used as an approximation because this storage requirement is much less than the one used by the ci trie.

5.3.2. The ci trie

Each internal node of the ci trie is a variable size list of structures. Similar to the indexing in the *ZIu*, each structure is a union of three possible types:

- Grey an ordered triple of a coded-pinyin, a pointer to the next node, and a pointer to the dictionary tables.
- Black an ordered pair of a coded-pinyin, and a pointer to the next node.
- White an ordered pair of a coded pinyin and a pointer to the dictionary tables.

The coded-pinyin in the structure is part of a word. For the grey nodes and the white nodes, the pointer to the dictionary table represents the word constructed from the sequence of coded-pinyin along the path from the root of the trie to the current node. Of course, the coded-pinyin has to be decoded. Depending on the level of the node, the pointer points to the appropriate ideograph table, e.g., the node at level 3 points to the 3-ideograph table. Similar to the zi trie, the internal nodes are different in size. Fig 3.8 is a detail description of the ci trie with the words "aiban" (呆 , 板), and "aidai" (愛 , 戴).

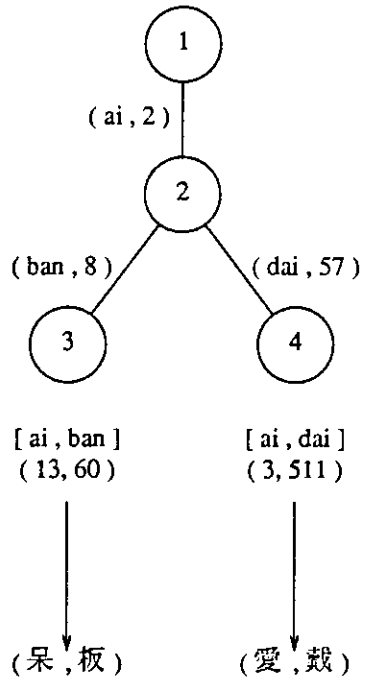


Fig 3.8 Detail of a ci trie.

The actual implementation is similar to the zi trie, except for the following two differences:

1. The phonetic symbol in the zi trie is replaced by the coded-pinyin in the ci trie.
2. The coded-pinyin in the zi trie is replaced by the pointer to the dictionary table in the ci trie.

The ci trie is also implemented sequentially. The basic unit of memory of this trie is 2 bytes: the pointer to the dictionary table is 2 bytes, and the size of the coded-pinyin is 2 bytes. If the whole trie can be implemented with 128K bytes, 2 bytes are enough to store a pointer. With reference to the statistics of the word dictionary in chapter 2, it can be shown that 2 bytes are enough for a pointer.

Since the number of 4 ideograph compounds, 5 ideograph compounds, and the 6 ideograph compounds is so small compared to the others (Table 3.1), they will not be considered in the following analysis.

From a counting of the phonetic spelling dictionary, the number of different phonetic spellings is 416. So the number of structures at the root node is 416. With reference to table 3.1 above, the number of two ideograph compounds is 11,993, implying that the number of structures in the second level is at least 11,993. The number of 3 ideograph compounds is 3,376. If each 3 ideograph compounds needs an intermediate structure at the second level (which is not always the case), then the total number of structures at the second level is 15,369 (11,993+3,376). So 15,369 is an upper bound on the total number of structures of the trie at level 2. The number of structures at level 3 is 3,376, since the 4-, 5-, and 6-ideograph compounds are neglected. An upper bound on the number of structures is then 19,161 (416+15,369+3,376), which is approximately 20K. Since each structure requires at most 6 bytes, an upper bound on the storage requirement is then 120K bytes (20K nodes x 6 bytes). So the whole ci trie can fit in 128K bytes.

A major difference between the zi trie and the ci trie is the size of a node in the ci trie is much bigger than the one in the zi trie. In the zi trie, the maximum number of structures in one node is 26,

while in the ci trie, the maximum number of structures is 416. In order to locate a coded-pinyin in a node efficiently, the node structure is modified to allow binary search on a node. An additional information required is the size of a node, which takes 2 bytes. The structure is also modified to be uniform in size. The grey node is changed to a 4 byte structure, with the first 2 bytes storing the coded-pinyin, and the second 2 bytes storing a ponter to an ordered pair. The ordered pair contains a pointer to the dictionary table, and a ponter to the next node. Fig 3.9 is a pictorial description.

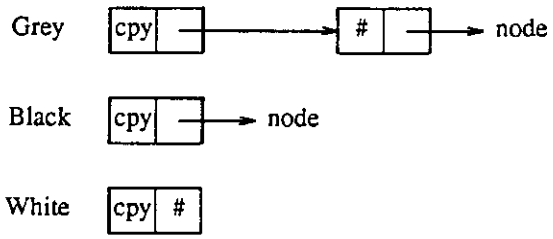


Fig 3.9 Node structure of the ci trie.

Now each structure within a node is 4 bytes. The first 4 bytes of a node store the number of structures within the node. A binary search is possible to perform in the node structure to locate the correct coded-pinyin.

Table 3.3 summarizes the storage requirement for the design.

Data structure	Size (kbytes)
ZIu	432
Dictionary tables	70
ZID	12
CID	120
Total	634

Table 3.3 Summary of storage requirement.

6. The Transient Words

Besides the static words discussed in last section, there exists an infinite number of transient words or temporary words. These transient words are single word syntactically. They are part of the ci dictionary and help resolving some ambiguities. Their usage in Chinese text is very common. Unfortunately, it is impossible to list all of them. For example, an infinite number of words can be constructed from a numeral, followed by *ge* (個), and followed by *ren* (人), such as:

yigeren	(一個人)	(one person)
lianggeren	(兩個人)	(two persons)
sangeren	(三個人)	(three persons)
shigeren	(十個人)	(ten persons)

In this section, methods to generate these transient words are discussed. These methods do not cover all aspect of transient word generation. They just provide some ideas of how to handle these infinite sets of words. Implementation details are not presented.

6.1. Numerals

The first group of transient words to be discussed is the *numerals*. Similar to Roman numerals, there exists an infinite number of Chinese numerals. These Chinese numerals can be generated by a regular expression. Unlike Roman numerals, the regular expression can generate some numerals that are illegal. This will not cause any problem since the dictionary is used in detecting the existence of a legal word instead of checking the correctness of a word. Following is the regular expression.

D (digit) = 一 | 二 | 三 | 四 | 五 | 六 | 七 | 八 | 九 | 零
N (number) = D^+
C = 十 | 百 | 千 | 萬 | 億
N = $(DC)^+$ | $(DC)^+$ | N 零 N
F (fraction) = C_1 分 之 C_2 |
百 分 之 *N* |
N 又 *F* |
N 點 *N* |
N 倍

Since a regular expression can be represented by a deterministic finite automaton, it can be implemented by extending the ci trie to a deterministic finite automaton.

6.2. Numeral-Measure-Noun Combination

In English, the phrase "one pair of chopsticks" consists of the following components: "one" is a numeral, "pair" is a measure, and "chopsticks" is a noun. In Chinese, the measure *ge* (個) is used with the noun *ren* (人), *wenti* (問 題), *xuexiao* (學校), and *jihui* (機 会). There does not exist any general rule that governs the Measure-Noun combination pair. The only possible method is to list all of them. A fairly complete list of these combinations is in Chao's book [7]. Usually the

Measure-Noun pair is proceeded by a Numeral.

A specifier (SP) may exist in front of a Nu-M-N construct. Part of the set of specifiers are listed as follow:

{ zhe (這), na (那), ge (各), di (第), tou (頭) }

Other types of transient words that can be generated with the combination of DFA and exhaustive listing are the *place words* and *time expression.*

6.3. Affixes

Similar to English, there exist affixes in Chinese language. Affixes are bounded morphemes that are added to other morphemes to form larger units. Other affixes are grammatical morpheme indicating number and aspect. Chinese has few affixes. The three kinds of affixes – prefixes, suffixes, and infixes – are discussed in the following sections. The list of affixes is extracted from Li and Thompson's book [33].

6.3.1. Prefixes

Following is the list of prefixes and their constructs:

PREFIXES	CONSTRUCT	EXAMPLE
lao (老)	lao-Surname	lao-Zhang (老 張)
xiao (小)	xiao-Surname	xiao-Zhang (小 張)
di (第)	di-Numeral	di-liu (第 六)
chu (初)	chu-Numeral	chu-er (初 二)
ke (可)	ke-Verb	ke-ai (可 愛)
hao (好)	hao-Verb	hao-kan (好 看)
nan (難)	nan-Verb	nan-kan (難 看)

6.3.2. Infixes

"–de–" (得) and "–bu–" (不) are the only infixes in Chinese. They are called potential infixes of verb compounds. For example,

shuo-de-qingchu (說 得 清 楚)
shuo-bu-qingchu (說 不 清 楚)

6.3.3. Suffixes

Following is a list of common suffixes.

SUFFIXES	CONSTRUCT	EXAMPLE
-men (門)	Human Noun-men Human Pronoun-men	xuesheng-men (學生門) (students) wo-men (我 門) (we)
-xue (學)	Subject Name-xue (equivalent to *-ology*)	xinli-xue (心 理 學) (psychologist)
-jia (家)	Subject Name-jia (equivalent to *-ist*)	wulixue-jia (物 理 學家) (physicist)
-zi (子)	Noun-zi	ti-zi (梯 子) (ladder)
-tou (頭)	Noun-tou	gu-tou (骨 頭) (bone)

7. Summary

In this chapter, an implementation of the static word dictionary and the construction of the transient word dictionary are discussed. The static word dictionary provide a fundamental frame for the Chinese dictionary. The transient word dictionary is augmented to the static word dictionary. It is just a technique to compress the size of the dictionary and store an infinite number of words. The tradeoff is that some illegal words are stored. Since the goal is to store all legal words instead of checking whether a word is legal or not, these minor defect will not cause any problem.

CHAPTER 4

Constraint Generation

This chapter concerns methods for generating the set of high level constraints for the Chinese input problem. High level constraints here correspond to the grammatical structure of Chinese language, whereas the low level constraints are the dictionaries. The analysis consists of two parts: automatic generation of constraints and heuristic rules. The heuristic rules are derived from Chinese language and grammar books [7, 33, 34]. Constraint generation applies only to a context free grammar for the language. There is no universally accepted Chinese grammar, so the automated process is just of theoretical interest. On the other hand, if there exists a context free grammar such that its language is a superset of Chinese language, the constraint generation technique can be used. The result is a set of constraints which is a superset of constraints that would be generated by a Chinese grammar. Any solution that is inconsistent with the superset is inconsistent with the Chinese language. The smaller the difference between the superset and the Chinese language, the better the performance it is.

The constraint generation technique is an iterative relaxation algorithm. The presentation starts with the automatic construction of binary constraints from a context free grammar. The algorithm is then generalized to n-ary constraints. The set of algorithm is represented by a set of fixed point equations. The set of constraints is the solutions of these equations. Negative results are presented if the given grammar is not context free.

Heuristic rules are presented to help resolving ambiguities due to homophones. These rules are of two kinds: syntactic and statistics. The set of syntactic rules provides a framework for constructing Chinese sentences. The set of statistic rules based on statistics collected on some Chinese language texts, such as frequency analysis on the occurences of all hanzis.

1. Automatic Generation of Constraints

Expert systems normally require experts to write down their expertise in rules form. Most of these rules are heuristic in nature. Recent work has tried to generate these rules automatically based on a formal model [32]. Generation of the knowledge base was done automatically using a model of the electrical heart activities [39, 38]. The basic idea is to develop a formal model for the target domain of the expert system. An automatic process then generates the set of constraints that govern the consistency of the formal model. The set of constraints and a constraint interpreter together form the required expert system.

This section presents an iterative relaxation algorithm that generates n-ary constraints for a given context free grammar. Two points are emphasized here: the concept of automated generation of constraints for a given model, and the iterative relaxation algorithm for constraint generation.

Grammatical structure is defined over the parts of speech, such as "noun", "verb", and "preposition". The constraints here mean the possible adjacency relationship between adjacent symbols. As in English, a transitive verb is always followed by a noun phrase. This kind of constraint can be viewed as a set of ordered pairs with the parts of speech as the possible elements. The simple adjacency relationship extend to sequences of three elements, four elements, ..., n elements. The set of constraints then consists of ordered pairs, triples, ..., n-tuples. The problem is then to find the set of all possible ordered pairs, triples, ..., n-tuples, for a given grammar. I shall assume that the given grammar is context free. Some negative results are also presented if the given grammar is context

sensitive.

1.1. Generation of Binary Constraints

I shall start with the simplest case of generating the constraints that are ordered pairs (binary constraints), and then proceed to the general case (n-ary constraints). Generating binary constraints is similar to building a predictive parse table in compiler design [1]. Instead of a direct adaptation of the predictive parse table algorithm, an iterative relaxation algorithm is presented. The iterative relaxation algorithm can be generalized without much change to generate n-ary constraints.

Since every context free grammar (CFG) can be written in Chomsky Normal Form (CNF), I assume that the given grammar is in CNF with the allowance of ε-productions. The algorithms presented below can be modified easily to handle context free grammars in any form.

For a CFG in CNF, the production rules are of the following forms only:

1. $A \rightarrow BC$
2. $A \rightarrow \mathrm{a}$
3. $A \rightarrow \varepsilon$

Here, A, B, and C are nonterminals and a is a terminal.

The following definitions are used in the iterative relaxation algorithm. Let $EXACT_0(A)$ be $\{ \varepsilon \}$ if $A \rightarrow \varepsilon$ is a production of the grammar. Let $EXACT_1(A)$ be the set of terminals that can be derived from A, i.e., $A \rightarrow \mathrm{a}$. Given a context free grammar in CNF, for all nonterminals A, $EXACT_0(A)$ and $EXACT_1(A)$ are formally defined as follows:

$$EXACT_0(A) = \begin{cases} \{ \varepsilon \} & \text{if } A \rightarrow \varepsilon \\ \varnothing & \text{otherwise} \end{cases}$$

$EXACT_1(A)$ is defined by the following rules:

1. If $A \rightarrow a$, then $a \in EXACT_1(A)$.
2. If $A \rightarrow BC$ and $EXACT_0(B) = \{ \varepsilon \}$, then $EXACT_1(C) \subset EXACT_1(A)$.
3. If $A \rightarrow BC$ and $EXACT_0(C) = \{ \varepsilon \}$, then $EXACT_1(B) \subset EXACT_1(A)$.
4. $EXACT_1(A)$ is the smallest set satisfying these rules.

The set concatenation for two sets A and B is defined as follow:

$$AB = \left\{ \alpha\beta \mid \alpha \in A \text{ and } \beta \in B \right\}.$$

If $A = \emptyset$ or $B = \emptyset$, then $AB = \emptyset$.

The above set of rules can be represented by a set of equations. Given a CFG and $EXACT_0(A)$ for all nonterminals A, the sets $EXACT_1(A)$ can be determined by solving the following set of fixed point equations.

If $A \rightarrow B_1C_1 \mid \cdots \mid B_nC_n \mid a_1 \mid \cdots \mid a_m$,

$$EXACT_1(A) = \bigcup_i EXACT_1(B_i)EXACT_0(C_i) \cup \bigcup_i EXACT_0(B_i)EXACT_1(C_i) \cup \left\{ a_1, \cdots, a_m \right\}. \tag{4.1}$$

These set of equations can be solved by the following iterative relaxation algorithm (algorithm 4.1). It generates the set of $EXACT_1(A)$ for all nonterminals A.

Algorithm 4.1

for all nonterminals A, $EXACT_1(A) := \emptyset$.
while there are changes to any of the $EXACT_1(A)$ *do*
begin
  for all nonterminals A *do*
    for each production $A \rightarrow \alpha$ *do*
    begin
      if $A \rightarrow a$ *then*
        $EXACT_1(A) := EXACT_1(A) \cup \{a\}$;
      if $A \rightarrow BC$ *and* $EXACT_0(B) = \{\varepsilon\}$ *then*
        $EXACT_1(A) := EXACT_1(A) \cup EXACT_1(C)$;
      if $A \rightarrow BC$ *and* $EXACT_0(C) = \{\varepsilon\}$ *then*
        $EXACT_1(A) := EXACT_1(A) \cup EXACT_1(B)$;
    end
end.

Since algorithm 4.1 is similar to the algorithm of finding transitive closure and there is a direct correspondence between the equation and the algorithm, the presentation of algorithms will be omitted in the rest of the chapter. They are given in appendix 1.

To illustrate the algorithm, consider the following example. Given the CFG:

$S \rightarrow \Rightarrow T \Leftarrow$

$T \rightarrow aTb \mid \varepsilon$

The Chomsky normal form is:

$S \rightarrow X_1 X_2$

$X_1 \rightarrow X_3 X_4$

$X_2 \rightarrow \Leftarrow$

$X_3 \rightarrow \Rightarrow$

$X_4 \rightarrow X_5 X_6 \mid \varepsilon$

$X_5 \rightarrow X_7 X_4$

$X_6 \rightarrow b$

$X_7 \rightarrow a$

and the set of nonterminals is $N = \{ S, X_1, X_2, X_3, X_4, X_5, X_6, X_7 \}$. $EXACT_0(X_4) = \{ \varepsilon \}$ and the other $EXACT_0$'s are $\varnothing$. The set of terminals is $\{ \Rightarrow, \Leftarrow, a, b \}$, where $\Rightarrow$ is the start of input symbol, and $\Leftarrow$ is the end of input symbol. Table 4.1 is the iterative process in obtaining the $EXACT_1$'s. In the table, $EXACT_1$'s are denoted by E_1's. Line 3 is the solution.

	$E_1(S)$	$E_1(X_1)$	$E_1(X_2)$	$E_1(X_3)$	$E_1(X_4)$	$E_1(X_5)$	$E_1(X_6)$	$E_1(X_7)$
0	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$
1	$\varnothing$	$\varnothing$	$\{\Leftarrow\}$	$\{\Rightarrow\}$	$\varnothing$	$\varnothing$	{b}	{a}
2	$\varnothing$	$\{\Rightarrow\}$	$\{\Leftarrow\}$	$\{\Rightarrow\}$	$\varnothing$	{a}	{b}	{a}
3	$\varnothing$	$\{\Rightarrow\}$	$\{\Leftarrow\}$	$\{\Rightarrow\}$	$\varnothing$	{a}	{b}	{a}

Table 4.1 $EXACT_1(N)$.

In order to generate the possible adjacent pairs, two sets $FIRST_1(A)$ and $LAST_1(A)$ are required for all nonterminals A. $FIRST_1(A)$ is the set of terminals that begin strings derived from A. $LAST_1(A)$ is the set of terminals that end strings derived from A. If $A \rightarrow$ abcde, then

$FIRST_1(A) = \{a\}$, and $LAST_1(A) = \{e\}$.

$FIRST_1(A)$ and $LAST_1(A)$ can be represented by the following set of fixed point equations.

For all production rules of the form $A \rightarrow B_1C_1 \mid \cdots \mid B_kC_k$,

$$FIRST_1(A) = \bigcup_i FIRST_1(B_i) \cup \bigcup_i EXACT_0(B_i)FIRST_1(C_i). \tag{4.2}$$

$$LAST_1(A) = \bigcup_i LAST_1(C_i) \cup \bigcup_i LAST_1(B_i)EXACT_0(C_i). \tag{4.3}$$

Equations (4.2) and (4.3) correspond to algorithm 4.2 and 4.3 in appendix 1.

For the previous example, the iterative processes for generating $FIRST_1(A)$ and $LAST_1(A)$ are given in table 4.2 and table 4.3. In the tables, $FIRST_1$ and $LAST_1$ are represented by F_1 and L_1.

	$F_1(S)$	$F_1(X_1)$	$F_1(X_2)$	$F_1(X_3)$	$F_1(X_4)$	$F_1(X_5)$	$F_1(X_6)$	$F_1(X_7)$
0	$\varnothing$	$\varnothing$	$\{\Leftarrow\}$	$\{\Rightarrow\}$	$\varnothing$	$\varnothing$	{b}	{a}
1	$\varnothing$	$\{\Rightarrow\}$	$\{\Leftarrow\}$	$\{\Rightarrow\}$	$\varnothing$	{a}	{b}	{a}
2	$\{\Rightarrow\}$	$\{\Rightarrow\}$	$\{\Leftarrow\}$	$\{\Rightarrow\}$	{a}	{a}	{b}	{a}
3	$\{\Rightarrow\}$	$\{\Rightarrow\}$	$\{\Leftarrow\}$	$\{\Rightarrow\}$	{a}	{a}	{b}	{a}

Table 4.2 $FIRST_1(N)$.

	$L_1(S)$	$L_1(X_1)$	$L_1(X_2)$	$L_1(X_3)$	$L_1(X_4)$	$L_1(X_5)$	$L_1(X_6)$	$L_1(X_7)$
0	∅	∅	{⇐}	{⇒}	∅	∅	{b}	{a}
1	{⇐}	{⇒}	{⇐}	{⇒}	{b}	{a,b}	{b}	{a}
2	{⇐}	{⇒,b}	{⇐}	{⇒}	{b}	{a,b}	{b}	{a}
3	{⇐}	{⇒,b}	{⇐}	{⇒}	{b}	{a,b}	{b}	{a}

Table 4.3 $LAST_1(N)$.

With the sets $FIRST_1(A)$ and $LAST_1(A)$ for all nonterminals A, the set of possible pairs of adjacent terminals ($CONST_2$) can be generated by following equation.

For all production rules of the form $A \rightarrow B_1C_1 \mid \cdots \mid B_kC_k$,

$$CONST_2 = \bigcup_i LAST_1(B_i)FIRST_1(C_i). \tag{4.4}$$

For the previous example, the possible adjacent pairs can be represented by the following table (table 4.2) with a "×" means the element at the corresponding row followed by the element at the corresponding column is a legal pair.

	⇒	⇐	a	b
⇒		×	×	
⇐				
a			×	×
b		×		×

Table 4.4 $CONST_2$.

1.2. Generation of n-ary constraints

This section generalizes the result in the last section for n-ary constraints. Given a context free grammar in Chomsky normal form, for all nonterminals A, let $EXACT_i(A)$ be the set of sequences of terminals of length i that can be derived from A, i.e., $A \rightarrow \ldots \rightarrow \alpha$ and $|\alpha| = i$. It is defined recursively by the following equation:

For all productions $A \rightarrow B_1C_1 \mid \cdots \mid B_nC_n$,

$$EXACT_i(A) = \bigcup_{j,k} EXACT_j(B_k)EXACT_{i-j}(C_k) \tag{4.5}$$

for i = 2, 3, ...

$EXACT_0(A)$ and $EXACT_1(A)$ are defined in the previous section; for convenience $EXACT_i(A) = \varnothing$ if $i < 0$.

Similarly, $FIRST_i(A)$ and $LAST_i(A)$ are defined. $FIRST_i(A)$ is the set of sequence of terminals of length i that begin strings derived from A. $LAST_i(A)$ is the set of sequence of terminals of length i that end strings derived from A. They are defined by the following fixed point equations:

For all production $A \rightarrow B_1C_1 \mid \cdots B_nC_n$,

$$FIRST_i(A) = \bigcup_k FIRST_i(B_k) \cup \bigcup_{j,k} EXACT_j(B_k)FIRST_{i-j}(C_k) \tag{4.6}$$

$$LAST_i(A) = \bigcup_k LAST_i(C_k) \cup \bigcup_{j,k} LAST_{i-j}(B_k)EXACT_j(C_k) \tag{4.7}$$

for i = 2, 3, ...

The N-ary constraint is denoted by $CONST_N$, and is defined by equation (4.8).

For all productions $A \rightarrow B_1C_1 \mid \cdots \mid B_nC_n$,

$$CONST_N = \bigcup_{i,k} LAST_i(B_k)FIRST_{N-i}(C_k) \tag{4.8}$$

The equations (4.5), (4.6), and (4.7) provide three algorithms (algorithm 4.5, 4.6, and 4.7 in appendix 1) to generate $EXACT_i(A)$, $FIRST_i(A)$, and $LAST_i(A)$.

Table 4.5, 4.6, and 4.7 give the iterative processes for the $EXACT_2(A)$, $FIRST_2(A)$, and $LAST_2(A)$ for the example in the previous section.

	$E_2(S)$	$E_2(X_1)$	$E_2(X_2)$	$E_2(X_3)$	$E_2(X_4)$	$E_2(X_5)$	$E_2(X_6)$	$E_2(X_7)$
0	{⇒⇐}	∅	∅	∅	{ab}	∅	∅	∅
1	{⇒⇐}	∅	∅	∅	{ab}	∅	∅	∅

Table 4.5 $EXACT_2(N)$.

	$F_2(S)$	$F_2(X_1)$	$F_2(X_2)$	$F_2(X_3)$	$F_2(X_4)$	$F_2(X_5)$	$F_2(X_6)$	$F_2(X_7)$
0	{⇒⇐}	{⇒a}	∅	∅	{ab}	{aa}	∅	∅
1	{⇒⇐,⇒a}	{⇒a}	∅	∅	{aa,ab}	{aa}	∅	∅
2	{⇒⇐,⇒a}	{⇒a}	∅	∅	{aa,ab}	{aa}	∅	∅

Table 4.6 $FIRST_2(N)$.

	$L_2(S)$	$L_2(X_1)$	$L_2(X_2)$	$L_2(X_3)$	$L_2(X_4)$	$L_2(X_5)$	$L_2(X_6)$	$L_2(X_7)$
0	{⇒⇐,b⇐}	{ab}	∅	∅	{ab,bb}	{ab,bb}	∅	∅
1	{⇒⇐,b⇐}	{ab}	∅	∅	{ab,bb}	{ab,bb}	∅	∅

Table 4.7 $LAST_2(N)$.

The set of ordered triples are:

{ ⇒aa, ⇒ab, aaa, aab, abb, bbb, ab⇐, bb⇐ }.

1.3. Practicality

1.3.1. An English grammar

How useful is it to generate the n-ary constraints with n > 2? Following is a simple grammar that can generate a limited set of English. Let N_E = { *Start*, *S* , *NP* , *VP* , *RC* } be the set of nonterminals, and { ⇒, ⇐, det , n , name , tv , iv , that } be the set of terminals.

Start → ⇒ *S* ⇐	
S → *NP* *VP*	(sentences)
NP → det n *RC* | name	(noun phrases)
VP → tv *NP* | iv	(verb phrases)
RC → that *VP* | ε	(restrictive clauses)

The above algorithms can be modified easily to handle context free grammars in any form instead of Chomsky normal form. The results for the above grammar are in tables 4.8–4.14.

$E_0(Start)$	$E_0(S)$	$E_0(NP)$	$E_0(VP)$	$E_0(RC)$
$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{\varepsilon\}$

Table 4.8 $EXACT_0(N_E)$.

$F_1(Start)$	$F_1(S)$	$F_1(NP)$	$F_1(VP)$	$F_1(RC)$
$\Rightarrow$	det	det	tv	that
	name	name	iv	

Table 4.9 $FIRST_1(N_E)$.

$L_1(Start)$	$L_1(S)$	$L_1(NP)$	$L_1(VP)$	$L_1(RC)$
$\Leftarrow$	iv	iv	iv	iv
	name	name	name	name
	n	n	n	n

Table 4.10 $LAST_1(N_E)$.

	⇒	⇐	det	n	name	tv	iv	that
⇒			×		×			
⇐								
det				×				
n		×				×	×	×
name		×				×	×	
tv			×		×			
iv		×				×	×	
that						×	×	

Table 4.11 $CONST_2$.

$E_1(Start)$	$E_1(S)$	$E_1(NP)$	$E_1(VP)$	$E_1(RC)$
∅	∅	name	iv	∅

Table 4.12 $EXACT_1(N_E)$.

$F_2(Start)$	$F_2(S)$	$F_2(NP)$	$F_2(VP)$	$F_2(RC)$
(⇒,det)	(det,n)	(det,n)	(tv,det)	(that,tv)
(⇒,name)	(name,tv)	(tv,name)	(that,iv)	
	(name,iv)			

Table 4.13 $FIRST_2(N_E)$.

$L_2(Start)$	$L_2(S)$	$L_2(NP)$	$L_2(VP)$	$L_2(RC)$
(iv,⇐)	(det,n)	(det,n)	(det,n)	(det,n)
(name,⇐)	(tv,name)	(tv,name)	(tv,name)	(tv,name)
(n,⇐)	(iv,iv)	(that,iv)	(that,iv)	(that,iv)
	(name,iv)			
	(n,iv)			
	(that,iv)			

Table 4.14 $LAST_2(N_E)$.

There are 27 ordered triples ($CONST_3$):

(⇒,name,tv)	(⇒,name,iv)	(⇒,det,n)
(iv,iv,⇐)	(name,iv,⇐)	(n,iv,⇐)
(det,n,⇐)	(tv,name,⇐)	(that,iv,⇐)
(iv,tv,det)	(det,n,tv)	(det,n,that)
(iv,tv,name)	(tv,name,tv)	(n,that,tv)
(name,tv,det)	(that,iv,tv)	(n,that,iv)
(name,tv,name)	(det,n,iv)	(tv,det,n)
(n,tv,det)	(tv,name,iv)	(that,tv,det)
(n,tv,name)	(that,iv,iv)	(that,tv,name)

On the other hand, if the ordered pairs are combined to form ordered triples, e.g., combine (det,n) and (n,tv) form (det,n,tv), there are 40 ordered triples:

(⇒,det,n)	(n,⇐,det)	(n,⇐,name)
(name,⇐,det)	(name,⇐,name)	(iv,⇐,det)
(iv,⇐,name)	(det,n,⇐)	(n,iv,⇐)
(⇒,name,⇐)	(⇒,name,tv)	(⇒,name,iv)
(name,iv,⇐)	(tv,name,⇐)	(iv,iv,⇐)
(det,n,tv)	(det,n,iv)	(det,n,that)
(n,tv,det)	(n,iv,tv)	(n,that,tv)
(n,tv,name)	(n,iv,iv)	(n,that,iv)
(name,tv,det)	(name,iv,tv)	(tv,name,tv)
(name,tv,name)	(name,iv,iv)	(tv,name,iv)
(iv,tv,det)	(iv,iv,tv)	(that,tv,det)
(iv,tv,name)	(iv,iv,iv)	(that,tv,name)
(tv,det,n)	(that,iv,tv)	(that,iv,iv)
(that,iv,⇐)		

1.3.2. A Chinese grammar

The Chinese grammar in appendix 2 is constructed from examples in Hashimoto's paper [21] and Wang's paper [61]. It only generates a limited set of Chinese. It can also generate some illegal sentences. There are 13 production rules, 13 nonterminals and 19 terminals in the grammar. There are 361 possible ordered pairs from 19 terminals. The constraint generation process generates 176 ordered pairs. More than half of the 361 possible pairs are considered illegal by the generation process. Out of the 176 ordered pairs generated, some of them are obvious illegal. A better grammar would not generate these impossibilities. This shows that if there exists a Chinese grammar, useful constraints can be generated automatically.

1.4. Other languages

Other languages, such as recursively enumerable languages and context sensitive languages, have been used in language processing. Can the iterative relaxation algorithms in the previous section

be applied to these languages? Unfortunately, the problem of generating the set of adjacent nonterminal pairs for a context sensitive grammar is unsolvable. Before giving the proof, some definitions are presented.

The definitions are adapted from the book by Hopcroft and Ullman [25]. A phrase structure grammar permits productions of the form $\alpha \rightarrow \beta$, where α and β are arbitrary strings of grammar symbols, with $\alpha \neq \varepsilon$. This grammar is also known as type 0 grammar, or unrestricted grammar. A context sensitive grammar permits productions of the form $\alpha \rightarrow \beta$, where α and β are arbitrary strings of grammar symbols, and β is at least as long as α, and $\alpha \neq \varepsilon$.

Theorem 4.1

Given a context sensitive grammar (CSG), the problem of generating the set of all possible adjacent pairs of terminals is unsolvable.

Proof:

Proof is by contradiction. It is known that given a context sensitive language L, and a word w, the question of whether w is in L or not is decidable. It is undecidable whether L is empty or not [25].

Assume there exists an algorithm that can generate all possible adjacent terminal pairs for a context sensitive grammar. Given a CSG G, let P be the set of all adjacent terminal pairs. If the language L generated by G is empty, then the set of adjacent terminal pairs P is empty. So if P is not empty, then the language L generated by G is nonempty. If P is empty, each terminal of G and ε is tested whether it is in L. This is possible since a word w in L is decidable. If none of these tests succeed, L is empty. Otherwise, L is nonempty. This gives an algorithm that can decide whether L is empty or not. It contradicts that fact that it is undecidable whether a context sensitive language is empty or not.

q.e.d.

Corollary 4.2.

Given a phrase structure grammar, the problem of generating all possible adjacent pairs of terminal symbols is unsolvable.

2. Heuristic Rules

Since the goal of the Chinese Input System is to resolve the ambiguities that appear in Chinese homophones instead of parsing a Chinese sentence, I will not derive a grammar and generate the set of n-ary constraints. On the other hand, the low level constraints (i.e. the dictionary) are not able to resolve all ambiguities. Single character words always exist in Chinese sentences. The word dictionary cannot resolve any of these ambiguities. There also exists about 4% of word homophones. Some previous systems display all possible solutions and let the user to select the right one from the terminal. In my proposed system, high level constraints based on heuristic rules are used to solve this problem. Some of the rules presented here are based on particular properties of the language structure. No formal analysis will be presented to justify these rules.

This section describes a set of heuristic rules to help resolving ambiguities. These rules are classified under two categories: syntactic rules and statistic rules. The set of syntactic rules provides a framework for constructing Chinese sentences. They are only heuristic rules since one can always come up with counterexamples that violate these rules. The second set of rules are statistics rules. They are based on statistics collected on some Chinese language texts, such as a frequency analysis on the occurences of all hanzis, and the homophones are ordered according to their frequencies.

Most ambiguities appear in single characters that do not form words. So the heuristic rules concentrate on the detecting and resolving of these ambiguities. The most common hanzis that usually appear as single characters are the prepositions (often called coverbs). Following sections describe the patterns used to construct sentences with prepositions. Then constructions on particles, conjunctions, and verb suffixes are presented. This list is not complete and is just for illustration.

For a given input, the low level constraints are applied first, the high level rules then applied to deal with the unresolved ambiguities. If one has a sufficient large set of rules, the performance of the system will be improved.

2.1. Notations

Table 4.15 contains notations that are used throughout the rest of the thesis.

Notation	
S	Subject
O	Object
Mod	Modifier
Dir	Direction
N	Noun
PN	Personal pronoun
SP	Specifier
QW	Interrogative pronoun
M	Measure word
NU	Numerical expression
TW	Time expression
PW	Place word
L	Localizer
SV	Adjective
A	Adverb
MA	Movable adverb/conjunction
C	Connective
CV	Preposition
FV	Functive verb/transitive verb/intransitive verb
AV	Auxiliary verb
EV	Copula
RV	Resultative verb
PV	Post verb
P	Particle
I	Interjection
VO	Verb-object

Table 4.15 Some notations.

2.2. Preposition Patterns

There is one general pattern for all prepositions (CV) and there are numerous special constructs for individual ones. These patterns can be expressed as regular expression and can be generated by deterministic finite automata.

The general pattern is:

S [A] [AV] CV-N FV [Mod] O.

This pattern tells us that a noun always follows a preposition (CV-N). The simplest sentence pattern that can be generated from this pattern is:

S CV-N FV O.

The other elements, A, AV, and Mod, can be inserted to build more complex patterns. Most prepositions also have specific individual patterns for sentence construction. Following is a specific pattern for *wei* (爲):

• *wei* (爲)

A *wei* N *suo* FV.

A list of common patterns for common prepositions is in appendix 3.

2.3. Particles

Particles follow a word, a phrase, or a sentence to indicate some particular function or aspect. They are divided into two major families: structural particles and modal particles. Particles that occur after the main verb and are used as verb suffixes are structural particles, such as *le* (了), *zhe* (着), *guo* (過), and *de* (的, 得 , 地). Modal particles are free form morphemes placed at the end of sentences. They do not have concrete or substantive meaning, but they do convey certain emotions and moods. Some common modal particles are *a* (呀), *ba* (罷), *de* (的), and *le* (了).

The modal particles are also known as final particles since they always appear at the end of a sentence. If a sentence always terminates with a EOS (end of sentence) marker, the set of modal particles can be represented by a rule like

MP – EOS

where MP is the set of modal particles, { 的, 了 , ... }.

A list of patterns that govern the construction of a phrase or a sentence with the structural particles is in appendix 3.

2.4. Conjunctions

Conjunctions connect two or more clauses to make a coordinate sentence or a subordinate sentence. When a conjunction is used to connect two clauses, it usually contains two components, with one part attached to one clause. For example:

ta yue chi yue pang.
他 越 吃 越 胖.

The *yue...yue* (越 ... 越) is the conjunction.

A list of common conjunctions where at least one component contains a single character is in appendix 3.

2.5. Statistic Rules

The statistic rules are based on statistics collected from Chinese language texts. Two measures are used to help resolving some ambiguities arise from homophones.

- Identification of bound morphemes.
- Frequency analysis.

Identification of bound morphemes refers to the morphemes that cannot appear alone as a single character. This is used loosely here: in most cases, if the hanzi does not appear by itself as a single character unit, then it is called a bound morpheme. For example, the phonetic spelling *ba* corresponds to the following common hanzis:

{巴, 把, 芭, 八, 疤, 拔, 跋, 叭, 伯 }

Table 4.16 is the bound/free properties for these hanzis. Following is the list of notations used in the table:

bounded	a bounded morpheme.
free	an unbounded morpheme.
usually	usually bounded morpheme.

hanzi	property
巴	usually
把	free (preposition)
芭	bounded
八	bounded (numeral)
罷	usually
拔	free
跋	bounded
叭	bounded
伯	free

Table 4.16 Statistical properties for some hanzis.

Frequency analysis refers to a counting on the occurrences of the hanzis in some Chinese texts. Cheng [8] did a frequency analysis on 1,177,985 hanzis. The results suggested only 4,583 hanzis are used in present-day Chinese. The frequency distributions for these hanzis are tabulated. These distributions provide a heuristic to select a hanzi when ambiguities occur. For example, table 4.17 contains the frequencies for the hanzis with phonetic spelling *bu*.

Ideograph	Frequency
不	14,236
部	1,676
歩	651
補	196
捕	103
簿	87
布	< 25
埠	< 25
怖	< 25
哺	< 25

Table 4.17 Frequency distribution of *bu*.

3. Summary

In this chapter, two methods for constraint construction are presented. One is generated automatically while the others are heuristic rules. For the automatic construction technique, the method is represented by a set of fixed point equations. The solution of these fixed point equations determines the set of constraints. If constraints can be generated automatically, the multilevel search can be applied directly. On the other hand, if only heuristic rules are present, a greedy approach is used. This corresponds to the multilevel filtering process. All these will be studied in detail in chapter 6.

CHAPTER 5

Multilevel Model

In this chapter, a formal model for a multilevel problem solving framework is defined. The validity of the Chinese input system is justified by the model. The model here is not general enough to apply to all multilevel problem solving. But the approach to formalize it can be used to study other problems. Methods to measure a multilevel system's performance are proposed.

The presentation starts with some definitions so that the multilevel model can be introduced formally. In a multilevel system, each level is a constraint satisfaction problem. Each level has its own set of variables and constraints. In the current model, the labels at one level are the variables at the next higher level. The labeling of a constraint at one level merges the variables at that level to form an abstraction unit. This abstraction unit not only represents the labeling at the current level, but also a variable at the next higher level. A problem instance is given to the lowest level. The problem instances at the higher levels are generated during the processing stage. A solution to the problem should not violate any constraint of any level. The solution is called globally consistent.

Concepts of join and covering are defined to represent the combining effect of constraints. Constraint factor is used to measure the size of a constraint set, which is a measure on the performance of a level. Intuitively, the smaller the size of a constraint set, the more efficient a search can be performed, since fewer cases need to be considered. In a multilevel system, higher level constraints can reduce the size of a lower level constraint set. This improves the performance of the lower level and is known as the local control.

1. Definitions

A problem consists of a finite set of objects, a finite set of constraints, and a problem instance. The set of objects consists of two categories, a set of variables, $V = \{x_1, \cdots, x_N\}$, and a set of labels. Each variable, x_i, can attain a finite set of values L_i, the label set of x_i. The label set is finite so that a measure can be defined to compare the performance of sets of constraints. Detail is discussed in the section of constraint filtering power. The set of constraints, $C = \{C_1, \cdots, C_M\}$, defines the relationship between the variables.

A *variable-vector* X is an ordered n-tuple of variables, $(x_1, x_2, \cdots, x_n)$, $x_i \in V$ for $i=1,...,n$. A *label-vector* for a variable-vector X, denoted by $l_{(x_1, \cdots, x_n)}$, is an ordered n-tuple of labels $(l_1, \cdots, l_n)$, $l_i \in L_i$ for $i=1,...,n$, where L_i is the label set of x_i. A label-vector denotes a legal labeling for the variable-vector. A problem instance is a variable-vector. The goal is to find a label-vector for the problem instance such that no constraint is violated. A label-vector l_i for a variable-vector X is called a *consistent labeling* if no constraint in the set of constraints is violated. The result may contain more than one label-vector. The *order* of a vector, |v|, is the number of elements in the vector, e.g., $|(x_1, x_2, x_3, x_4)| = 4$.

The set of constraints defines the relationships between variables. A binary constraint on objects x_i and x_j, $C_{x_i x_j}$, is a set of label pairs for the variables x_i and x_j.

For example, let the set of variables be $\{x_1, x_2, x_3, x_4, x_5\}$ and the label set for all variables be $\{a, b, c\}$. $C_{x_1 x_3} = \{(a,a),(b,b),(c,c)\}$ is a binary constraint on x_1 and x_3. If x_1 and x_3 appear together in the problem instance, $C_{x_1 x_3}$ is the set of legal label pairs for them.

A n-ary constraint C_X is a set of label-vectors for the variable-vector X of order n. For example,

$$C_{(x_1,x_2,x_3)} = \{(a,a,a),(b,b,b)\}$$

is a 3-ary constraint. Each label-vector specifies a legal labeling triple for the variable-vector (x_1,x_2,x_3). If variables x_1, x_2, and x_3 appear together in the problem instance, then (a,a,a) and (b,b,b) are the legal labeling for them.

The following notation will be used throughout the rest of the chapter:

$x_i, x_j, y_i, y_j, \ldots$	variables
$a, b, \cdots$	labels
$X, Y, Z, \cdots$	variable-vectors
$l_X, l_Y, l_Z, \ldots$	label-vectors

1.1. Covering

In order to compare constraints, it is necessary to know how to compare variable-vectors and label-vectors first. Intuitively, a variable-vector X is less than ($\leq$) another variable-vector Y if X is a subsequence of Y when X and Y are viewed as sequences of variables. For example:

$(x_1,x_2,x_4,x_5) \leq (x_1,x_2,x_3,x_4,x_5)$

$(x_3,x_4) \leq (x_3,x_4,x_5)$

(x_1,x_3,x_4,x_5) is not comparable with (x_1,x_2,x_3,x_4).

Formally, the less than relation ($\leq$) between two variable-vectors can be defined recursively as follows:

(1) ( ) ≤ any variable-vector;

(2) $(x_1, \cdots, x_{n_1}) \leq (y_1, \cdots, y_{n_2})$

if $(x_1 = y_1)$ and $(x_2, \cdots, x_{n_1}) \leq (y_2, \cdots, y_{n_2})$

or $(x_1, \cdots, x_{n_1}) \leq (y_2, \cdots, y_{n_2})$

The projection $\downarrow$ of a label-vector for a variable-vector X on Y tells the effect of the label-vector of X on the subset of variables Y. It is defined as follows:

$$(l_1, \cdots, l_{n_1})_{(x_1, \ldots, x_{n_1})} \downarrow (y_1, \cdots, y_{n_2}) = \begin{cases} (l_1, \cdots, l_{n_1}) & \text{if } (x_1, \cdots, x_{n_2}) \leq (y_1, \cdots, y_{n_1}), \\ \textit{undefined} & \text{otherwise.} \end{cases}$$

For example:

$(a, b, c, d, e)_{(x_1, x_2, x_3, x_4, x_5)} \downarrow (x_1, x_4, x_5) = (a, d, e)$.

$(a, b, a, b)_{(x_1, x_4, x_5, x_6)} \downarrow (x_1, x_5) = (a, a)$.

The projection of a constraint of X on Y can be defined as follows:

$$C_X \downarrow Y = \left\{ l_X \downarrow Y \mid l_X \in C_X \right\}.$$

For example:

$$C_{(x_1, x_2, x_3, x_4, x_5)} = \left\{ (a, b, a, b, a), (a, a, a, a, a), (a, b, c, d, e) \right\}.$$

$$C_{(x_1, x_2, x_3, x_4, x_5)} \downarrow (x_1, x_2, x_3) = \left\{ (a, b, a), (a, a, a), (a, b, c) \right\}.$$

$$C_{(x_1, x_2, x_3, x_4, x_5)} \downarrow (x_1, x_3) = \left\{ (a, a), (a, c) \right\}.$$

The feasible set of a constraint C_X is the set of all label-vectors that satisfy C_X. Intuitively, a constraint C_X is said to be stronger than another constraint C_Y if the feasible set of C_X is a subset of the feasible set of C_Y. Formally speaking, C_X is stronger than C_Y with respect to Z if $Y \leq X \leq Z$ and $C_X \downarrow Y \supset C_Y$. C_Y is also said to be covered by (or less than ($\leq$)) C_X with respect to Z.

For example:

$$C_{(x_1,x_2,x_3,x_4)} = \{(a,b,c,d), (a,b,c,c), (b,b,c,c)\}.$$

$$C_{(x_2,x_4)} = \{(b,d), (b,c)\}.$$

$$C_{(x_1,x_3)} = \{(a,d)\}.$$

If $Z = (x_1, x_2, x_3, x_4)$,

$C_{(x_1,x_2,x_3,x_4)} \geq C_{(x_2,x_4)}$ *with respect to Z.*

$C_{(x_1,x_2,x_3,x_4)} \geq C_{(x_1,x_3)}$ *with respect to Z.*

$C_{(x_1,x_3)}$ *is not comparable with* $C_{(x_2,x_4)}$

If $Z = (x_2, x_4)$,

$C_{(x_1,x_2,x_3,x_4)}$ *is not comparable with* $C_{(x_2,x_4)}$

A label-vector, l_X, is said to be covered by another label-vector, l_Y, with respect to Z, if $Y \leq X \leq Z$, and $l_X \downarrow Y = l_Y$. In this case, the label-vector l_Y is called redundant with respect to Z. For example, given the following two constraints:

$$l_{(x_1,x_2,x_3,x_4)} = (a,b,a,b)$$

$$l_{(x_2,x_4)} = (b,b)$$

$l_{(x_2,x_4)}$ is redundant with respect to (x_1,x_2,x_3,x_4).

1.2. Join Operation

Since constraints are sets of labels, operations on sets, such as union, can be applied. After a set union is performed on two constraints, some label-vectors may become redundant. So the set union is not a good representation for the combining effect of the two constraints. The operation *join* is defined to represent the combining effect of two constraints. The *join* operation is more complicated than just taking the union of two constraints, and eliminating the redundant label-vectors. For example, the label-vectors, $(a,b)_{(x_1,x_2)}$ and $(b,c)_{(x_2,x_3)}$, are not redundant with respect to (x_1,x_2,x_3). Their combining effect is $(a,b,c)_{(x_1,x_2,x_3)}$.

Given two label-vectors, l_X and l_Y, their join is defined as:

1. If $X \bigcap Y = \varnothing$, then $l_X \; join \; l_Y = \left\{ l_X , l_Y \right\}$.
2. If $X \bigcap Y = Z$, and $l_X \downarrow Z \neq l_Y \downarrow Z$

   then $l_X \; join \; l_Y = \left\{ l_X , l_Y \right\}$.
3. If $X \bigcap Y = Z$, and $l_X \downarrow Z = l_Y \downarrow Z$

   then $l_X \; join \; l_Y = l_W$

   such that $l_W \downarrow X = l_X$, $l_W \downarrow Y = l_Y$ and $W = X \bigcup Y$.

Following are examples.

$$(a,b,c)_{(x_1,x_2,x_3)} \; join \; (a)_{(x_3)} = \left\{ (a,b,c)_{(x_1,x_2,x_3)} , (a)_{(x_3)} \right\}.$$

$$(a,b,c)_{(x_1,x_2,x_3)} \; join \; (a,c)_{(x_3,x_4)} = \left\{ (a,b,c)_{(x_1,x_2,x_3)} , (a,c)_{(x_3,x_4)} \right\}.$$

$$(a,b,c)_{(x_1,x_2,x_3)} \; join \; (c,d)_{(x_3,x_4)} = \left\{(a,b,c,d)_{(x_1,x_2,x_3,x_4)}\right\}.$$

The *outer-join* of 2 constraints, C_X and C_Y, is defined as:

$$C_X \; outer\text{-}join \; C_Y = \left\{l_X \; join \; l_Y \mid l_X \in C_X \text{ and } l_Y \in C_Y\right\}.$$

The *join* of 2 constraints, C_X and C_Y, with respect to Z, is the set of non-redundant label-vectors in the *outer-join* of C_X and C_Y with respect to Z.

For example:

$$C_{(x_1,x_2,x_3,x_4)} = \left\{(a,b,c,d)\right\}.$$

$$C_{(x_2,x_4)} = \left\{(b,d),(a,c)\right\}.$$

If $Z = (x_1,x_2,x_3,x_4)$, then

$$C_{(x_1,x_2,x_3,x_4)} \; outer\text{-}join \; C_{(x_2,x_4)} = \left\{(a,b,c,d)_{(x_1,x_2,x_3,x_4)}, (b,d)_{(x_2,x_4)}, (a,c)_{(x_2,x_4)}\right\}.$$

In this case, the label-vector $(b,d)_{(x_2,x_4)}$ is redundant with respect to Z. So the *join* is

$$C_{(x_1,x_2,x_3,x_4)} \; join \; C_{(x_2,x_4)} = \left\{(a,b,c,d)_{(x_1,x_2,x_3,x_4)}, (a,c)_{(x_2,x_4)}\right\}.$$

2. Multilevel System

In a traditional constraint satisfaction problem, a set of constraints is given to define the relationship between the variables. For a multilevel system, each level is itself a constraint satisfaction problem. Each level consists of its own set of objects (variables and labels), and its own set of constraints. A binary relationship exists between adjacent levels. There are two relations: one is defined between the objects of the two levels; the other is defined between the solutions of the two levels. In the current model, the labels at one level are the variables at the next higher level. The

labeling of a constraint at one level merges the variables at that level to form an abstraction unit. This abstraction unit not only represents the labeling at the current level, but also a variable at the next higher level.

A problem instance P is given to the lowest level, $Level_1$. For the rest of the chapter, level i is denoted by $Level_i$. The problem instances at the higher level are generated during the processing stage. The generated problem instance, P_i, at $Level_i$ ($i>1$), is called the abstraction of the problem at $Level_i$. P_i may contain more than one problem since the set of solutions at $Level_{i-1}$ may have more than one element. P_1 is the given problem instance P. For the abstraction of the problem at $Level_i$, the set of solutions is called the *partial solution set* at $Level_i$. Each element of the partial solution set at $Level_{i-1}$ corresponds to one, and only one, set of variables at $Level_i$. Each set of variables is an element of P_i. Each element of the partial solution set at $Level_i$ corresponds to an element of the partial solution set at $Level_{i-1}$. On the other hand, an element of the partial solution set at $Level_{i-1}$ may correspond to more than one element of the partial solution set at $Level_i$. This is because higher levels are constructed from the lower levels. The partial solution set at $Level_i$ is always generated from a consistent partial solution set at $Level_{i-1}$. Definition of consistency will be presented at the end of the chapter. With the direct correspondence between the partial solution set at $Level_i$ and the partial solution set at $Level_{i-1}$, any inconsistent element at $Level_i$ will eliminate the corresponding element at $Level_{i-1}$. This in turn will eliminate the element at $Level_{i-2}$, and so on. On the other hand, the problem P_i at $Level_i$ is generated from a consistent set of solutions at $Level_{i-1}$, no inconsistency is introduced. So once a consistent partial solution set is obtained for the problem P_i at $Level_i$, all partial solution sets at $Level_j$, $j \leq i$, are consistent.

Each level has its own set of constraints. The constraints of each level define the relationship between the variables at that level. Each element of the generated problem instance has a set of

variables which determines the set of applicable constraints. For the Chinese input problem, the organization of data and the characteristics of the problem allow an efficient retrieval of the set of applicable constraints given a set of variables. These will be illustrated in the next chapter.

Let's define all these concepts formally. In a multilevel system, a problem consists of a set of constraints, a set of objects, and a problem instance. The set of constraints is partitioned into levels. Each level has its own set of variables and labels. For two adjacent levels, $Level_i$ and $Level_{i+1}$, the set of variables at level $Level_{i+1}$, is the set of labels at level $Level_i$. So a level is

$$Level_i = (C_i , V_i , L_{V_i})$$

$$V_i = L_{V_{i-1}}, \quad i > 1$$

where

C_i : *set of constraints,*

V_i : *set of variables,*

L_{V_i} : *set of labels.*

These are just the syntactic relationships between levels. There are also semantic relations between levels. One can view one level as an abstraction of the next lower level. The constraint is the definition of the abstraction unit. Each labeling of a constraint is an abstraction of the current level to the next higher level. It is also a surjection from the current level to the next higher level. Semantically, the set of constraints at the current level defines the set of abstraction of variables at the current level to the next higher one. The semantic relation between levels $Level_i$ and $Level_{i+1}$ is provided by the constraints at $Level_i$. Another definition of constraints is then

$$C_X = \left\{ l_X \mid l_X \in V_{i+1} \right\}$$

where X is a variable-vector from $Level_i$.

The Chinese input system is used as an illustration. The variables at the lowest level, $Level_1$, is the phonetic symbol, { $a, b, ..., z$ }. The label set at $Level_1$ is the set of all legal spellings, { $a, ai, an, ang, ao, \cdots$ }. The set of constraints are the legal spellings:

$$C_{(a)} = \left\{ (a) \right\},$$

$$C_{(a,i)} = \left\{ (ai, ai) \right\},$$

$$C_{(a,n)} = \left\{ (an, an) \right\},$$

...

These are rewritten as follows:

$$C_{(a)} = \left\{ a \right\},$$

$$C_{(a,i)} = \left\{ ai \right\},$$

$$C_{(a,n)} = \left\{ an \right\},$$

...

In this case, a, ai, and an are variables at $Level_2$.

The labels at $Level_2$ are the legal Chinese words,

{呆板，愛戴，...,公正，公証，...}

The variables at $Level_2$ are the legal phonetic spelling,

{ $a, ai, an, ang, ao, ...$ }

The set of constraints is

$$C_{(ai,ban)} = \{(呆,板)\},$$

$$C_{(ai,dai)} = \{(愛,戴)\},$$

...

$$C_{(gong,zheng)} = \{(公,正),(公,証)\},$$

...

The variables at $Level_3$ are the labels at $Level_2$, the legal Chinese words. The labels at $Level_3$ are the possible parts of speech. For example, one possible constraint is that a final particle (FP) always appears at the end of the sentence (end of sentence marker, EOS). Some final particles are:

{ 呀 (*a*), 哩 (*li*), 呵 (*a*), 咯 (*la*), 了 (*liao*) }

The corresponding constraints are:

$C_{(呀,EOS)} = \{(FP, EOS)\}$

$C_{(哩,EOS)} = \{(FP, EOS)\}$

$C_{(呵,EOS)} = \{(FP, EOS)\}$

$C_{(咯,EOS)} = \{(FP, EOS)\}$

$C_{(了,EOS)} = \{(FP, EOS)\}$

3. Control

The control component of a system decides what the system performs next. Sometimes control is centralized, a separate control executive decides how problem solving process should be expended. Sometimes control is diffusely spread throughout the whole system. An efficient control should make the best usage of the available information. If one is working on highly parallel architecture, the system may be executed in a very nondeterministic way. The order of constraint application is unpredictable. The control of the system will then be solely determined by constraints: the more determinism you put in the system, the more communication between the levels of the system.

For the multilevel model, there are basically two kinds of control: global control and local control. Global control determines how the whole system runs, what order the constraints are executed, and where to apply the constraints. For example, the ordering of constraints may be changed over time, which may speed up the problem solving process. It is hard to measure how much improvement is achieved. Sometimes this kind of rearrangement may degrade the system's performance. One possible measure is the average performance of the system over a period of time instead of the instantaneous performance. Even on a simple system, it is very difficult to determine the expected behavior. On a such complicated multilevel system, the expected performance will not be studied. On the other hand, the behavior of some local control in the current model is measurable. Local control is the control information propagated from a level to the next lower level. It only exists between adjacent levels. Its existence is illustrated by the Chinese input application.

When a constraint can be applied at $Level_{i+1}$, it determines some constraints are impossible to apply at $Level_i$. So a constraint at $Level_{i+1}$ can be viewed as a meta-constraint at $Level_i$. A local control is the constraints that applied on $Level_{i+1}$ which removes some irrelevant constraints at $Level_i$. For the multilevel framework, the partial solution is being filtered when going up the levels while the

local control information is propagated down the levels by applying constraints to remove the irrelevant constraints in the lower levels. For example, if the variable is *li*, there are more than 40 legal label-vectors. If the next variable is EOS, then there is only one legal label-vector. The others are illegal.

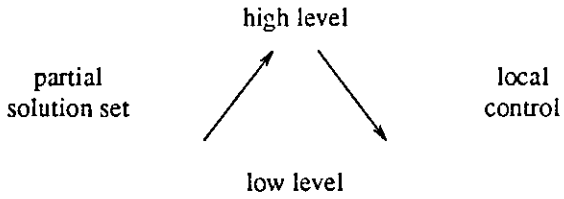


Fig 3.5 Multilevel framework

The notion of constraint filtering power is defined to measure the performance of the system.

4. Constraint Filtering Power

Since the local control generates constraints at level i from level i+1, the set of constraints is changing with time. Some kind of measure is necessary to indicate the possible increase in performance introduced by the local control. Constraint factor (CF) is defined to measure the size of constraint sets. Intuitively, for a set of constraints, the more label-vectors it contains, the bigger its constraint factor, and the smaller its constraint filtering power. The smaller the constraint filtering power, the less time it requires to process the set of constraints. For example, given the following two set of constrains:

$$C_1 = \left\{ (a,b,c,)_{(x_1,x_2,x_3)} \right\}.$$

$$C_2 = \left\{ (a,b,c,)_{(x_1,x_2,x_3)}, (c,d,e,)_{(x_1,x_2,x_3)} \right\}.$$

Given a variable-vector (x_1,x_2,x_3), there are two label-vectors satisfy C_2, while there is only one label-vector satisfy C_1. C_1 has a bigger constraint filtering power than C_2. The constraint filtering power is defined as the reciprocal of the constraint factor. Following is the definition of constraint factor. Let C_i, C_j be sets of constraints.

1. $CF(\varnothing) = 0$;
2. $CF(C_i) \geq 0$ *for all* C_i;
3. $CF(C_i) + CF(C_j) \geq CF(C_i \bigcup C_j)$ *(subadditivity)*;
4. $CF(C_i \bigcup C_j) \geq CF(C_i)$ *and* $CF(C_i \bigcup C_j) \geq CF(C_j)$;
5. *If* $C_i \subset C_j$, *then* $CF(C_i) \leq CF(C_j)$ *(monotone)*;

This definition is similar to the definition of an outer measure [18], except that outer measure requires countable subadditivity instead of subadditivity. Since the set of label-vectors is finite, the total number of set of constraints is finite. In this case, countable subadditivity is the same as subadditivity. If the set of label-vectors is extended to an infinite set, such as the set of real number, countable subadditivity is then necessary. Measure theory is then needed to compare the sizes of infinite sets.

Following is one possible definition for constraint factor, denoted by *CFI*. Let C_i and C_j be sets of constraints.

$$CFI(C_i) = \sum_{l_k \in C_i} | l_i | .$$

$$CFI(C_i \bigcup C_j) = CFI(C_i \; join \; C_j).$$

Since the *join* operation is commutative and associative, *CFI* is well defined on finite unions of constraints. The set of constraints of a level is a finite union of constraints; its *CFI* is also well defined.

For example, in chapter 4 section 1.3.1, the automatic method generates 27 ordered triples. The constraint factor is 81 (27×3). On the other hand, if the ordered pairs are combined to form ordered triples, there are 40 of them. The constraint factor is then 120 (40×3), which overestimate the true value.

Theorem 5.1

CFI is constraint factor.

Proof:

1. $CFI(\varnothing) = 0$ since nothing is in $\varnothing$.
2. $CFI(C_i) \geq 0$ *for all* C_i.
3. $CFI(C_i) + CFI(C_j) \geq CFI(C_i \bigcup C_j)$ because C_i *join* $C_j \subset C_i \bigcup C_j$.
4. $CFI(C_i \bigcup C_j) \geq CFI(C_i)$ is the same as $CFI(C_i joinC_j) \geq CFI(C_i)$.

   Let $C_k = C_i joinC_j$.

   Since none of the element of C_i or C_j is redundant,

   so for all $l_X \in C_i$, there exists $l_Z \in C_k$,

   such that $l_Z \downarrow X = l_X$.
5. If $X \subset Y$, then $CFI(X) \leq CFI(Y)$ (obvious).

q.e.d.

The constraint filtering power *(CFP)* is defined as:

$$CFP(X) = \frac{1}{CFI(X)}.$$

Theorem 5.2

CFP is a non-decreasing function with respect to the local control operation.

Proof:

Since the local control operation remove some constraints, *CFI* is decreasing by the monotone property of *CFI*, This implies *CFP* is a non-decreasing function by definition.

q.e.d.

5. Consistency

As defined in section 1, a label-vector for a variable-vector is called a consistent labeling if no constraint is violated. In a multilevel system, there are two kinds of consistency: local and global. Local consistency refers to one level, while global consistency refers to the whole system.

Assume a problem with levels, $Level_1, Level_2, \cdots, Level_N$, and a problem instance, P, is given. The notation, $P_1, P_2, \cdots, P_N$, is used to represent the abstraction of the problem P at levels $Level_1, Level_2, \cdots, Level_N$. A solution S_i for the problem P_i at level $Level_i$ is called *locally consistent* if no constraint at level $Level_i$ is violated. A solution S for the problem P is called *globally consistent* if the abstraction of the solution S is locally consistent at all levels.

Theorem 5.3

Given a multilevel system, if a solution's abstraction at level $Level_i$ is locally consistent, then the solution's abstraction at $Level_j$, for all $j \leq i$, are locally consistent.

Proof:

Direct consequence from the discussion in the section on multilevel system (section 2).

Corollary 5.4

Given a multilevel system, if a solution's abstraction at level L_N is locally consistent, then the solution is globally consistent.

Proof:

Obvious from the definition and previous theorem.

6. Summary

In a traditional consistent labeling problem, the set of constraints is fixed. The constraint filtering power does not vary with time. This measure gives a number which does not represent much. On the other hand, in a multilevel system, constraint filtering power of a level may be varied over time since some local control information may be propagated from a higher level to a lower level.

Different levels usually have different filtering power. Constraints at different levels have different costs to apply. The efficiency of a level should be defined in terms of the constraint filtering power.

CHAPTER 6

Control Mechanism

Previous chapters discussed data organization, constraint generation, and the theoretical framework. This chapter discusses several different control mechanisms and their disadvantages. First, one restriction is required on the formal model of chapter 5 to make it applicable to the Chinese input problem. Then a search method is presented for the lowest level to solve the character break problem. It identifies the break points for the phonetic spellings. The character break search is a form of dynamic programming, with input symbols processed one at a time from left to right and matched againist the database.

This search strategy is generalized for the multilevel system. In the multilevel generalization, several databases are matched simultaneously during the processing of input symbols. An alternative control discussed, multilevel filtering, is also quite useful in practice. The input problem is presented to level 1. Level 1 processes the input with the constraints and a partial solution is generated. The partial solution is submitted to level 2 as a problem. The level 2 constraints then process the generated problem and generate another partial solution. The process then continues for the next higher level. A combination of the multilevel search and filtering is the best candidate for implementation. The multilevel system solves the character break problem, word break problem, and the hanzi labeling problem simultaneously.

1. Restriction on the Formal Model

One restriction is needed on the multilevel formalism in order to model the Chinese input problem. The restriction is that constraints must be defined on a sequence of variables that are adjacent to each other. Constraints can only be applied to contiguous sequences of variables. For example, $C_{x_1x_2x_3x_4}$ can only be applied to a variable sequence of the form $\alpha x_1x_2x_3x_4\beta$ where α and β are variable sequences. It cannot apply to $\alpha x_1\alpha_1x_2\alpha_2x_3\alpha_3x_4\beta$ if any of α_1, α_2, and α_3 is nonempty.

For example, the constraint

$$C_{(j,i,n)} = \left\{ \text{jin} \right\}$$

can only applied when the input variable sequence contains "jin" as a subsequence.

2. Character Break Search

The character break search is the level 1 search. It solves the character break problem. Input is a sequence of phonetic symbols. The character break search locates break points between phonetic spellings in an input sequence of phonetic symbols. It is called character break search because each phonetic spelling corresponds to a hanzi in the final result. The search strategy is similar to the beam search in HARPY [35]. The difference is that all possible partial solutions are kept here while in the HARPY system, a threshold value called beam width, is used to prune away some solutions. So the character break search can be viewed as a beam search with infinite beam width. No pruning is required since high level constraints are used filter out the impossible solutions. The filtering process is discussed in the next section.

As discussed in chapter 2, the pinyin dictionary is stored as a trie. Each node in the trie indicates whether the sequence of phonetic symbols along the path from the root to the current node constitutes a legal phonetic spelling. The corresponding coded-pinyin is stored if the sequence is a

pinyin.

The input is a sequence of phonetic symbols $x_1 x_2 \cdots x_n$. The last symbol is always the end of sentence symbol, EOS. The EOS is assumed to be part of the alphabet. The goal is to identify the break points such that each subsequence between a pair of break points is a legal pinyin. The final solution set may contain more than one solution. For example, the input

x i a n g g a n g

has the following two solutions:

xiang gang ,
xi ang gang .

The character break search is a form of dynamic programming, with the input symbols processed one at a time from left to right and matched against the database. As the database is a trie, matching can be done very efficiently. The algorithm to be presented below can be modified easily if the database is a deterministic finite automaton. The basic idea is tha, at the ith step, the first $i-1$ symbols are processed and a set of partial solutions is generated. Each element is a possible candidate to be in the final solution set.

For easy visualization, the following notation for break points is used. For a sequence of phonetic symbols, $[x_1, x_2, \cdots, x_n]$, if the break points are after $x_{b_1}, x_{b_2}, \ldots, x_{b_k}, x_n$, the representation is

$$[\ x_1 \cdots x_{b_1-1}\ ,\ x_{b_1} \cdots x_{b_2-1}\ ,\ \cdots\ ,\ x_{b_k} \cdots x_n\ ].$$

For example, the sequence

[x, i, a, n, g, g, a, n, g]

with a break point after the first "g" is written as

[xiang, gang].

Each partial solution contains the following information:

- A list of recognized pinyins obtained so far. It is represented by a list of break points.
- A sequence of phonetic symbols that is a legal prefix for a certain pinyin.
- A pointer to a node in the deterministic finite automaton.

Consider the following more detailed example. Assume the input sequence is

[x, i, a, n, g, g, a, n, g].

and that the set of possible pinyins that can applied to these phonetic symbols is

$$\left\{ \text{a , an , ang , gang , xi , xia , xian , xiang } \right\}.$$

The corresponding trie is shown in fig 6.1.

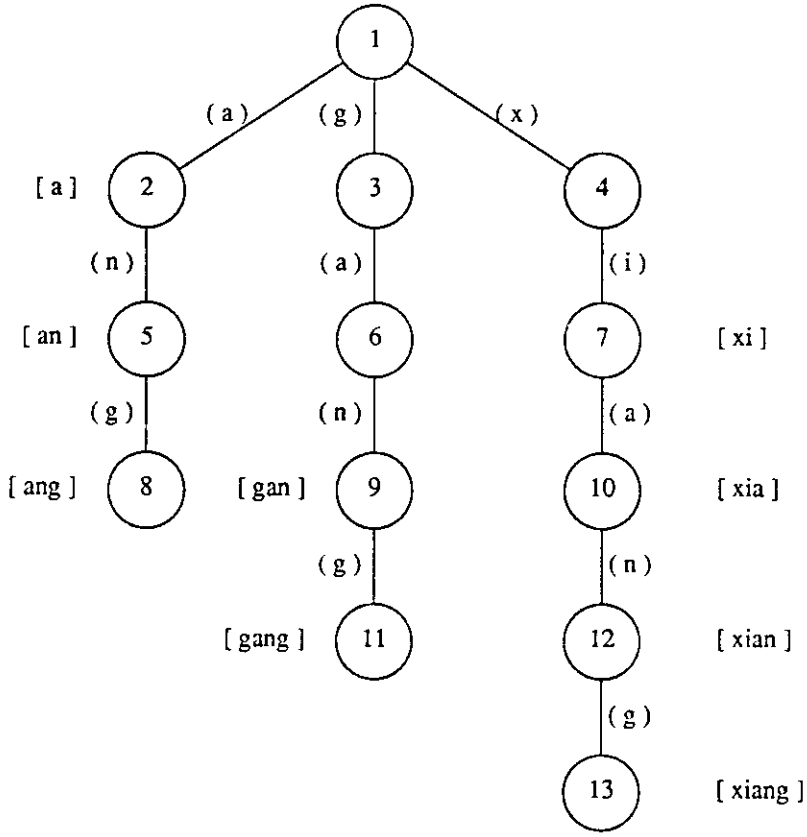


Fig 6.1 A pinyin trie.

The set of possible break point lists is

$$\left\{ [\,\text{xi}\,,\,\text{ang}\,,\,\text{gang}]\,,[\,\text{xiang}\,,\,\text{gang}] \right\}.$$

After the first four symbols are processed, the partial solutions are:

([xi , an] , [ ] , 1)

([xian] , [ ] , 1)

([xi] , [an] , 5)

([xia] , [n] , $PtoDB_n$)

([xi , a] , [n] , $PtoDB_n$)

The third element of the above triples is a pointer to the trie. If it is a number, it corresponds to a node of the trie in fig 6.1. $PtoDB_n$ is a pointer to a node in the trie which is not in fig 6.1. The current state of execution contains the following information:

- The set of partial solutions.
- The list of remaining phonetic symbols.

For the above example, the current state is

([([xi , an] , [ ] , 1), $\cdots$, ([xi , a] , [n] , $PtoDB_n$)] , [g, g, a, n, g])

The search algorithm proceeds as follows. The next character in the list of unprocessed symbols is appended to the sequence of phonetic symbols which is a legal prefix. The result is a list of phonetic symbols. If this is a pinyin, a new partial solution is created with a new break point after the current symbol, and an empty legal prefix. The database pointer is set to the root of the trie (the database). For the above example, the current symbol is ''g''. For the partial solution

([xi] , [an], 1),

a new partial solution is created:

([xi, ang], [ ], PtoRoot).

If the newly created list of phonetic symbols is also a legal prefix for some other pinyins, the database pointer of the current partial solution is advanced to the next node of the trie according to the current symbol. The current symbol is also appended to the legal prefix. If the new list of phonetic symbols does not form a pinyin but form a legal prefix, the partial solution is updated by advancing the database pointer and appending the current symbol to the legal prefix. If the new list of phonetic symbols does not even form a legal prefix, the partial solution is removed from the set of partial solutions. This is done for all elements in the set of partial solutions.

The following notations are used in the algorithm presented below.

[ ]	the empty list.
[$S_1, \cdots, S_n$]	a list with elements S_1, ..., S_n.
[Car|Cdr]	a list with the head Car and the tail Cdr.
head([Car|Cdr])	a function returns the head of the list, Car.
tail([Car|Cdr])	a function returns the tail of the list, Cdr.
L + M	a list created by append the list L to the list M, e.g. [a,b] + [c,d] = [a,b,c,d].
CS	the current state of execution with components PSL and PSS.
EOS	the end of sentence marker. EOS is denoted by "#" in figures.
PSL	the list of unprocessed phonetic symbols. PSL(CS) denotes the PSL for the current state CS.
PSS	a list of partial solutions. Each is a structure of three components BS, PPS, and PtoDB.

BS	a list of break points.
PPS	a list of phonetic symbols that is a legal prefix for a certain phonetic spelling.
PtoDB	a pointer to a node in the deterministic finite automaton (database).
Input	the input to the system, a list of phonetic symbols $[x_1, x_2, ..., x_n]$.
PtoRoot	a pointer to the root of the database (the trie), or the starting node of the deterministic finite automaton.
tmp	a temporary variable to store a sequence of phonetic symbols.
tmpPS,tmpPS1	a temporary variable to hold a partial solution.
tmpPSS	a temporary variable to hold a partial solution set.
advancePt(PtoDB,s)	move the PtoDB to the next node in the trie according to the symbol s. If moving is unsuccessful, a nil pointer is returned.

Algorithm 6.1 is the character break search. The notations defined in chapter 3 are used here. Each node of the trie is a union of three possible types: grey, black, and white. A grey node represents a legal pinyin and a legal prefix. A black node represents a legal prefix. A white node represents a legal pinyin.

Algorithm 6.1 Character break search.

```
procedure charBreakSearch;
begin
    PSL(CS) := Input;
    PSS(CS) := [ ( [ ], [ ], PtoRoot) ];
    while PSL(CS) ≠ [ ] do
    begin
        first := head(PSL(CS));
        PSL(CS) := tail(PSL(CS));
        tmpPSS := [ ];
        for all partial solutions PS in PSS(CS) do
        begin
            tmp := PPS(PS) + [first];
            PtoDB(tmpPS) := advancePt(PtoDB(CS),first);
            if PtoDB(tmpPS) is grey or white then
            begin
                BS(tmpPS1) := BS(PS) + tmp;
                PtoDB(tmpPS1) := PtoRoot;
                PPS(tmpPS1) := [ ];
                tmpPSS := tmpPSS + tmpPS1;
            end;
            if PtoDB(tmpPS) is grey or black then
            begin
                BS(tmpPS) := BS(PS);
                PPS(tmpPS) := PPS(CS) + [first];
                tmpPSS := tmpPSS + tmpPS;
            end
        end
        PSS(CS) := tmpPSS;
    end
end.
```

Trace 6.1 is a trace of the search. Part of the search tree is shown is fig 6.2. In the figure, a square node represents a failure node, i.e., no solution is under that subtree. In the trace, each element corresponds to a circular node in the search tree. Since no failure node is generated in the algorithm, none is shown in the trace. The path in the figure indicates the search path of one solution. Since all

partial solutions are kept simultaneously, all solutions are obtained at the same time.

Trace 6.1 Trace for character break search on "xianggang".

1. { [x] }
2. { [xi] , [x, i] }
3. { [xia] , [xi, a] }
4. { [xian] , [xia, n] , [xi, an] , [xi, a, n] }
5. { [xiang] , [xian, g] , [xi, ang] }
6. { [xiang, g] , [xi, ang, g] }
7. { [xiang, ga] , [xi, ang, ga] }
8. { [xiang, gan] , [xi, ang, gan] }
9. { [xiang, gang] , [xi, ang, gang] }
10. { [xiang, gang, EOS] , [xi, ang, gang, EOS] }

Each element in the trace corresponds to a possible partial solution at that step. For example, step 6 above contains 2 partial solution:

[xiang, g]
[xi, ang, g]

[xiang, g] means a break point exists after the first "g", and the second "g" is a legal prefix for a pinyin. [xi, ang, g] means there exists two break points, one is between "i" and "a", and the other is between the two "g" 's. The second "g" is a legal prefix for a pinyin.

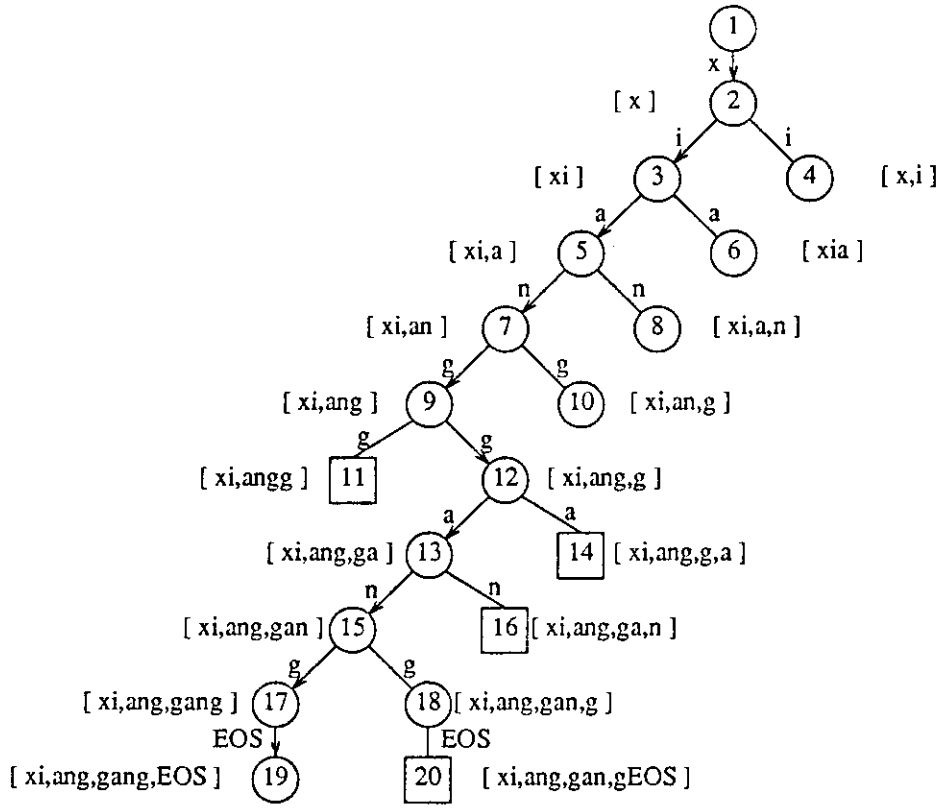


Fig 6.2 Search tree for "xianggang".

3. Multilevel Filtering

The first, and the simplest, control strategy for the multilevel system is multilevel filtering. A problem is presented to the lowest level, level 1. Level 1 processes the input with the constraints and a partial solution set is generated. It usually consists of more than one solution. The solution set is then submitted to the level 2 as a problem. The level 2 constraints then process the problem and

generate a solution set. The process then continues for the next higher level. If one is interested in the solution at level *i*, one needs to do processing up to at least level *i*. The process stops when there is at most one solution at level *i*, or all levels are processed. The input is assumed to be a correct Chinese sentence.

The following assumption is used in the algorithm presented below. It can be implemented easily.

- Partial solution sets at adjacent levels are linked together by the definition of abstraction mapping. Any change at one level will change the corresponding ones at adjacent levels.

In multilevel filtering, a variant of the character break search is used. The character break search is modified so that when the input is a sequence of variables at level *i*, the level *i* database is used. When the input to the character break search is a sequence of phonetic symbols, zi trie is used. If the input is a sequence of pinyins, the ci trie is used.

Algorithm 6.2 Multilevel filtering.

```
procedure levelSearch(i,input);
begin
    PSL(CS_i) := input;
    PSS(CS_i) := [ ( [ ], [ ], PtoRoot_i ) ];
    while PSL(CS_i) ≠ [ ] do
    begin
        first := head(PSL(CS_i));
        PSL(CS_i) := tail(PSL(CS_i));
        tmpPSS := [ ];

        for all partial solutions PS in PSS(CS_i) do
        begin
            tmp := PPS(PS) + [first];
            PtoDB(tmpPS) := advancePt(PtoDB(CS_i),first);

            if PtoDB(tmpPS) is grey or white then
            begin
                BS(tmpPS1) := BS(PS) + tmp;
                PtoDB(tmpPS1) := PtoRoot;
                PPS(tmpPS1) := [ ];
                tmpPSS := tmpPSS + tmpPS1;
            end;

            if PtoDB(tmpPS) is grey or black then
            begin
                BS(tmpPS) := BS(PS);
                PPS(tmpPS) := PPS(CS_i) + [first];
                tmpPSS := tmpPSS + tmpPS;
            end

        end

        PSS(CS_i) := tmpPSS;
    end
end.
```

```
procedure multilevelFilter(targetLevel);
begin
    set solution at level 0 to input problem;
    for i := 1 to targetLevel do
    begin
        curInput := solution at level i – 1;
        levelSearch(i,curInput);
    end;
    i := targetLevel + 1;
    while set of solution at targetLevel is not unique
        and i ≤ maximum number of levels do
    begin
        curInput := solution at level i – 1;
        levelSearch(i,curInput);
        i := i + 1;
    end;
    output the set of solutions at targetLevel;
end.
```

In order to have a clear picture of the filtering process, following example is discussed in details. Fig 6.3 is a pictorial description of the process. Consider the input sequence of phonetic symbols

x i a n g g a n g.

These are level 1 variables. They are the nodes at level 1 in fig 6.3. The corresponding labeling sets are:

x: { xang, xi, xia, xian, xiang, xiao, xie, xin, xing, xiong, xiu, xu, xuan, xue, xun }

i: { ai, bai, bei, bi, bian, biao, bie, bin, bing, cai, chai, chi, ... }

a: { a, ai, an, ang, ao, ba, bai, ban, bang, bao, bian, biao, ca, ... }

n: { an, ang, ban, bang, ben, beng, bian, bin, bing, bong, can, ... }

g: { ang, bang, beng, bing, bong, cang, ceng, chang, cheng, chong, ... }

By applying the character break search, the solution set is

{ [xiang, gang] , [xi, ang, gang] },

which are variables at level 2. They are nodes at level 2 in fig 6.3. Two partial solutions exists, one contains two variables: xiang, gang; and the other contains three variables: xi, ang, gang. These correspond to two problems at level 2. Following are the labeling sets for these variables:

ang: { 昂 };

xiang: { 胸 , 鄉 , 響 , 向 , 香 , 享 , ... };

gang: { 崗 , 綱 , 鋼 , 港 , ... };

xi: { 喜 , 嬉 , 吸 , 溪 , ... };

There is only one constraint defined on these variables:

$C_{(xiang,gang)}$ = { [香 , 港] }

The solution set at level 2 is

{ [xianggang] , [xi, ang, gang] }

such that xi, ang, gang are all possible single characters by themselves. Here, two partial solutions exists, one consists of one variable and the other consists of three variables. These solutions are then variables at level 3. They correspond to nodes at level 3 in fig 6.3. With the constraint $C_{xiang,gang}$ above, the labeling sets for variables xiang and gang become:

xiang: { 香 };

gang: { 港 }.

This forms an unique solution for the variable-vector (xiang, gang). The solution is (香 , 港). On the other hand, the solution corresponds to the break [xi, ang, gang] still have more than 100 combinations. By a greedy decision rule, the solution [(香 , 港)] is selected, and the corresponding character break is [xiang, gang].

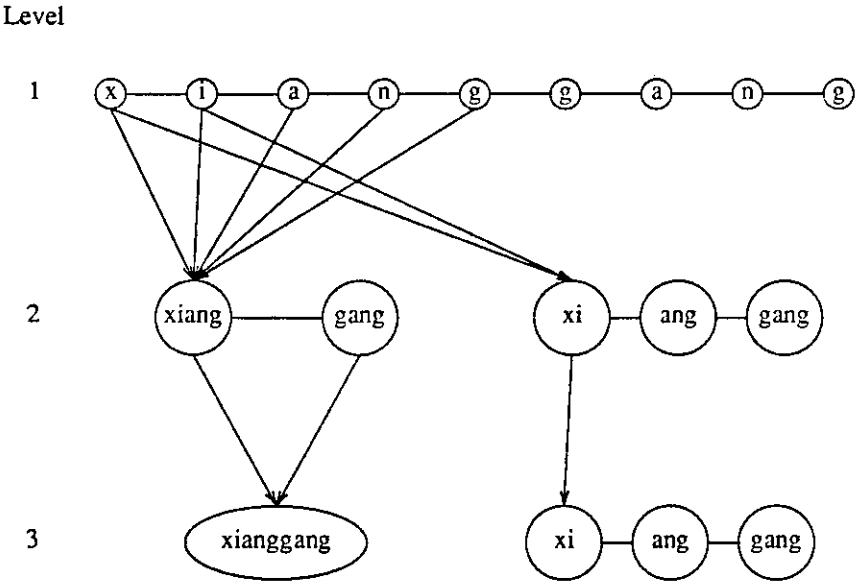


Fig 6.3 Multilevel filtering.

4. Multilevel Search

In the multilevel filtering, every level is an individual problem. A problem is solved level by level. Every level can be viewed as a filter. A level gets a problem and generates a set of solutions. The constraints at a level are used to eliminate all impossible solutions. The solution set is then

passed to the next higher level and the process continues. The disadvantages to this approach is there are many unnecessary solutions passing from one level to another, and a level does not use information from higher levels during processing. Since the problem solving process is from left to right, the process at the ith stage can use information from the previous $i-1$ stages and the higher level constraints to reduce the size of the partial solution set.

In this section, the character break search is extended to work on n levels simultaneously. For the Chinese input problem, only three levels are present. The basic idea is that the search processes at all levels proceed at the same time. At a certain level i, if an element of level $i+1$ is found, it will be sent to level $i+1$ immediately so that the search at level $i+1$ move forward. The assumption for this method is that the constraints at all levels can be checked from left-to-right.

Formally speaking, for the multilevel system with n levels, the current state (CS) is a set of states, $\{CS_1, ..., CS_n\}$. Each CS_i is a snapshot of the system at level i. The database is a set of finite state machines, $\{FSM_1, ..., FSM_n\}$. Each FSM_i defines the set of constraints at level i. In the multilevel model, the solution at level i is the problem at level $i+1$.

Each current state at level i, CS_i, is a list of ordered pairs, (PS_i , $PtoDB_i$), where PS_i is partial solution, and $PtoDB_i$ is a pointer to the database FSM_i. The list is written as

$CS_i = [\ (PS_{i_1}, PtoDB_{i_1}), ..., (PS_{i_m}, PtoDB_{i_m})\ ]$.

For each FSM_i, each node may be of the following three types:

- white: The sequence of elements from the starting node to the current node is a constraint. This sequence forms a legal element at the next higher level.
- black: The sequence of elements from the starting node to the current node is a legal prefix for a certain constraint.
- grey: A node with properties of both a grey node and a white node.

This corresponds to the implementation of the dictionary in chapter 3.

Algorithm 6.3 Multilevel Search.

```
function advancePtr(PtoDB_i,next): boolean;
begin
    advance the pointer PtoDB_i for FSM_i to the next state
        according to the element "next";
    if advance is successful then
        return true;
    else
        return false;
end;


procedure multilevelSearch(i,nextEle);
begin
    for all partial solutions at level i (PS_i_j,PtoDB_i^j) do
    begin
        if advancePtr(PtoDB_i_j,nextEle) then
        begin
            if the new PtoDB_i_j point to a white or black node then
            begin
                generate a new partial solution;
                update the current state;
            end
            if the new PtoDB_i_j point to a white or grey node then
                multilevelSearch(i+1,nodeContent);
        end
    end
end;


procedure mainControl;
begin
    for all levels i do
        CS_i := [ ];
    while Input not empty do
    begin
        next := get the next element from Input;
        multilevelSearch(1,next);
    end;
    output PS at targetLevel;
end.
```

Trace 6.2 is a trace of the multilevel search on input

j i a o d i a n.

The trace displays the variables and the partially formed variables at all three levels. It also shows the current states of the finite state machines. For example, step 3 of the trace is

3: L_1 = { [j, i, a] };
L_2 = { [ji, a], [jia] };
L_3 = { [ji] };

L_1 is the list of input variables. L_2 is a set of 2 elements [ji, a] and [jia]. [ji, a] consists of one variable "ji" and a partial variable "a". A partial variable means it may become a variable as the process continues, i.e., it is a legal prefix for a constraint. [jia] is another partial variable. L_3 = { [ji] } is a partial variable at level 3. The "a" is not propagated since no new variable is formed at level 2.

Trace 6.2 Trace of multilevel search on "jiaodian".

1: L_1 = { [j] };
L_2 = { [j] };
L_3 = { [ ] };

2: L_1 = { [j, i] };
L_2 = { [ji] };
L_3 = { [ji] };

3: L_1 = { [j, i, a] };
L_2 = { [ji, a] , [jia] };
L_3 = { [ji] };

4: L_1 = { [j, i, a, o] };
L_2 = { [ji, ao] , [jiao] };
L_3 = { [jiao] , [ao] };

5: L_1 = { [j, i, a, o, d] };
L_2 = { [ji, ao, d] , [jiao, d] };
L_3 = { [jiao] , [ao] };

6: L_1 = { [j, i, a, o, d, i] };
L_2 = { [ji, ao, di] , [jiao, di] };
L_3 = { [jiao] , [di] };

7: L_1 = { [j, i, a, o, d, i, a] };
L_2 = { [ji, ao, di, a] , [ji, ao, dia] , [jiao, dia] , [jiao, di, a] };
L_3 = { [jiao] , [di] };

8: L_1 = { [j, i, a, o, d, i, a, n] };
L_2 = { [ji, ao, di, an] , [ji, ao, dian] , [jiao, di, an] , [jiao , dian] };
L_3 = { [jiaodian] , [dian] };

9: L_1 = { [j, i, a, o, d, i, a, n, EOS] };
L_2 = { [ji, ao, di, an , EOS] , [ji, ao, dian , EOS] , [jiao, di, an , EOS] , [jiao , dian , EOS] };
L_3 = { [jiaodian , EOS] };

Fig 6.4 is the corresponding search graph generated during the search process. Numbers next to a node give the order of the nodes being generated. These nodes display the variables at the corresponding level, and the solution at the next lower level. For example, the node 3 "ji" is a variable at level 2, it is also a solution for the level 1 variables, j and i. The EOS symbol is denoted by "#" in the figure.

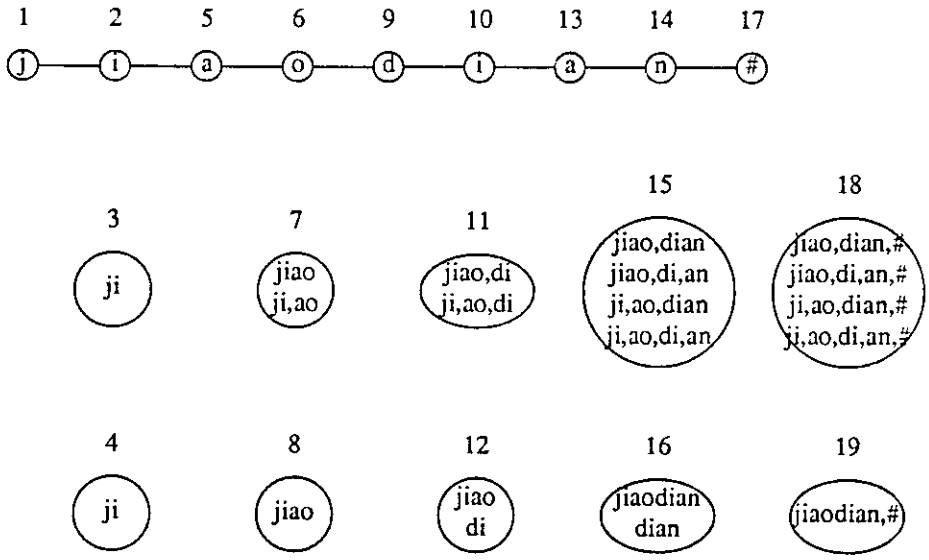


Fig 6.4 Multilevel search for "jiaodian".

The input is a problem at level 1. The input variables are processed one at a time from left to right. Exactly the same as the character break search, each input variable corresponds to a change of the current state, which in turn generates a move at FSM_1. After the move, if the node in FSM_1 corresponds to an element at level 2, the element is submitted as input to level 2. This input will then generate a move at level 2. This move is processed the same way as the move in level 1. This propagation continues until no element at the next higher level is generated. A move at the next higher level can be viewed as local control information propagated to the current level. When the search process moves down the trie, the total number of applicable constraints is reduced. This lowers the constraint factor, which in turn increases the constraint filtering power.

5. Hybrid Control

The main problem with multilevel filtering is that the partial solution set passed from one level to another may be huge. On the other hand, the multilevel search requires the constraints at all levels to be organized as finite state machines. In reality, constraints are usually expressed in rule form, especially the high level ones. As in expert system development, constraints are usually rules and driven by an interpreter. In the Chinese input problem, no Chinese grammar exists that can be used to generate the set of constraints to build a finite state machine. Only a set of patterns exists. In case a finite state machine can be built for the set of constraints, one would like to make full use of them. Under normal circumstances, finite state machines can be built for some levels and heuristic rules exist for other levels. A combination of multilevel filtering and search is useful.

In the Chinese input system, a two level search is performed on level 1 and level 2. The partial solution generated is sent to level 3 where rules can be applied.

Trace 6.3 and trace 6.4 are traces for the input

p e n g y o u a i.

Trace 6.3 is part of the trace of the multilevel search with level 1 and level 2. The solution is then passed to the rules of level 3. Trace 6.4 is the filtering process.

Trace 6.3 Trace for hybrid control on "pengyouai" (Searching).

1: L_1 = { [p] };
L_2 = { [p] };
L_3 = { [ ] };

2: L_1 = { [p, e] };
L_2 = { [pe] };
L_3 = { [ ] };

...

9: L_1 = { [p, e, n, g, y, o, u, a, i] };
L_2 = { [peng, you, ai] };
L_3 = { [pengyou] , [youai] };

10: L_1 = { [p, e, n, g, y, o, u, a, i, EOS] };
L_2 = { [peng, you, ai, EOS] };
L_3 = { [pengyou] , [youai] }.

There are two partial solutions at level 3:

{ [pengyou] , [youai] }.

They correspond to two possible parses of the input:

Parse 1: [pengyou , ai] ;

Parse 2: [peng , youai] .

Each parse has its own label sequence assign to the variables at level 2. The corresponding level 2 variables and their labels are described below. The solution is denoted by a list of pairs, (x_i,l_i), where x_i is a variable, and l_i is the list of possible labels.

Parse 1: [(peng , [朋]) , (you , [友]) , (ai , [哀 , 愛 , ...])];

Parse 2: [(peng , [朋 , 逢 , ...]) , (you , 友) , (ai , 愛)].

Trace 6.4 Trace of hybrid control on "pengyouai" (Filtering).

Rule for final particles:

(ai , EOS) → (呀 , EOS).

Parse 1: [(peng , [朋]) , (you , [友]) , (ai , [哀 , 愛 , ...])];
→ [(peng , 朋) , (you , 友) , (ai , 呀)].

Parse 2: [(peng , [朋 , 逢 , ...]) , (you , 友) , (ai , 愛)].
→ [(peng , [朋 , 逢 , ...]) , (you , 友) , (ai , 愛)].

6. Parallel Execution of the Character Break Search

The character break search in section 2 can be executed in parallel without much change. If one starts the search with any of the input symbols and finds a solution, and there is a break point right before that selected symbol, then the combination of the list of the break points is a solution for the problem. For example, consider the input

l a o p e n g y o u a i.

If the search starts with the symbol "y", the input is then "youai", and the solution set is

[you , ai , EOS].

For the first part of the input, "laopeng", the solution is

[lao , peng].

A final solution is then

[lao , peng , you , ai].

The parallel approach works as follows. Each variable checks whether the current variable and the next variable forms a legal pinyin. If so, the result is stored in a global data area. It then checks whether itself and the next 2 variables form a legal pinyin. The result is stored if it is true. The process continues for the next 3, 4, ... variables. Since the maximum length of a pinyin is six, the number of processing is at most six times. A global control is executed in parallel to locate a legal path from these partial results. Traces for input "xianggang" and "jiaodian" are presented fig 6.5 and fig 6.6.

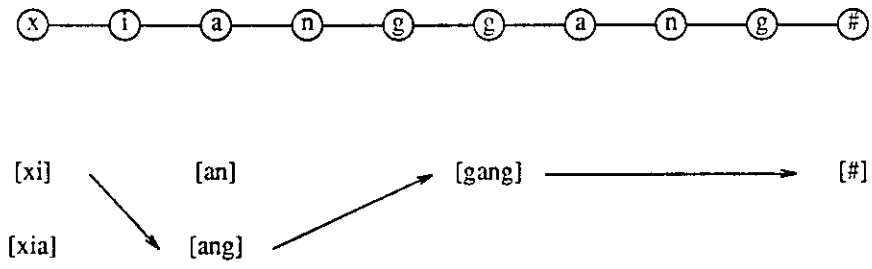


Fig 6.5 Trace for "xianggang".

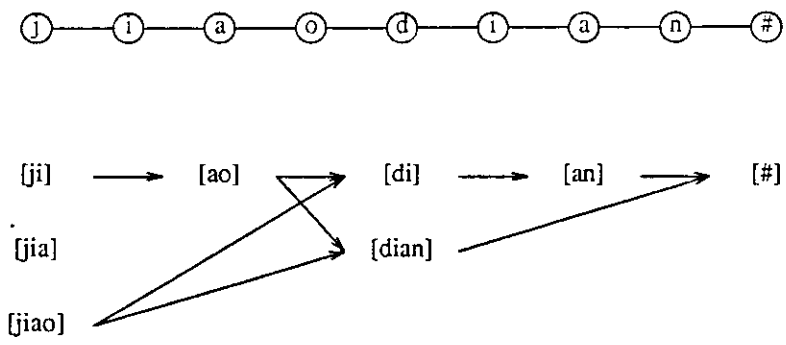


Fig 6.6 Trace for "jiaodian".

Another nice property of the problem is decomposability, that is, the problem can be broken down into smaller subproblems. The solution of the original problem is obtained by combining the solutions of the subproblems. The difficulty of problem solving by decomposition is how to find the subproblems. Since some phonetic symbols cannot appear before another one in a pinyin, e.g., "o" never appears before "d", a break point can be inserted between these two symbols. With this property, an input problem can be divided into smaller subproblems. For the input

x i a n g g a n g.

A break point is inserted between the two "g" 's. Two subproblems are generated:

x i a n g.
g a n g.

These two subproblems can be executed in parallel.

7. Summary

In this chapter, different control strategies are presented. Due to incomplete information available in the real problem, the hybrid control method is usually the best candidate for implementation.

CHAPTER 7

Conclusion

1. Related Problems and Future Research

A system based on phonetic spelling for inputting Chinese has been presented. The basic approach can be applied to other oriental languages, such as Japanese and Korean.

Around 1,500 years ago, the Japanese and Koreans borrowed the hanzi from Chinese to develop their own languages with some changes. Japanese and Korean have grammatical word endings, something like "-ed" or "-ing" in English. Since hanzi provided no way to write these grammatical endings, they devised their own phonetic alphabets, known as kana and hangul. The Japanese kana exists as katakana, formed by all or part of a hanzi; and hiragana, formed by cursive writing figures. The hanzi is called kanji in Japanese. Though the use of phonetic alphabets made hanzi unnecessary for writing Japanese and Korean, they are still written in an amalgam of hanzis and native phonetic letters. In Japanese sentences, kanji is not commonly used by itself but is accompanied by katakana, hiragana, and alphanumeric symbols. In Japan, most nouns are written in kanji while the kana are employed as auxiliary letters to denote grammatical inflection, the transliteration of foreign sounds, onomatopoeia, and exclamation. On the other hand, hanzis are being phased out of Korean gradually.

The existence of these alphabets makes the input problem for Japanese and Korean easier than Chinese. People of these countries are used to phonetic symbols. In the kana system, there are symbols to represent different intonation which reduce ambiguities. During the process of importing hanzi to Japanese and Korean, Chinese words (cis) have been imported in vast number to the point

that the majority of words in Japanese and Korean dictionaries are Chinese style compounds. A similar design can be used for inputting Japanese and Korean. In Japanese, the best input method is the kana to kanji conversion technique [37], in which the user types in the kana and the system finds the corresponding kanji. This is similar to typing Chinese with phonetic spelling.

The application of the problem solving schema to other oriental languages is straightforward. On the other hand, many other interesting and important problems appear as a result of this research:

1. The presented framework is not perfect. There still exists some unresolved ambiguities. Since they cannot be removed by syntactic analysis, I think semantic analysis is necessary to achieve a better performance.

2. Semantic analysis is closely related to language understanding. Even though research on natural language understanding has been actively performed in North America, Europe, and Japan, less work has been done on the Chinese language. The reason may be the lack of an uniform input method and an uniform internal representation. Since the Chinese language is quite different from other languages, the question of how hard it is to parse a Chinese sentence is still open. For example, in most Chinese sentences, the subject is determined by the relative position of the noun and the existence of prepositions. The field of Chinese language understanding is still in its infant stage.

3. The presented system cannot tolerate any error input, such as spelling errors. How hard is it to perform error recovery? How much modification of the algorithm is required? I think some kinds of probabilistic reasoning is necessary.

4. As the input method is based on phonetic spelling, can the idea be adapted to Chinese speech processing? Since speech input usually contains many errors, problem 3 has to be solved first.

5. Can the multilevel framework be used in other problem domains? The HARPY structure has been applied to an image understanding task [50]. Can the multilevel framework used in a similar way?

6. Most existing expert systems require experts to write down their expertise in rules. Previous research was on building user friendly tools which allow experts to build their own expert systems. Can an expert system be generated from an expert system? I think the idea of constraint generation is a step towards this direction.

2. Conclusion

Two problem solving techniques, multilevel framework and constraint generation, are discussed. They are closely related to relaxation. The constraint generation algorithm is an iterative relaxation algorithm. The multilevel problem solving framework can be viewed as a subproblem relaxation process. A problem is partitioned vertically into several subproblems. Higher level subproblems are relaxed initially. The simplest subproblem is solved first. The solution is then improved by strengthening the relaxed subproblems. On the other hand, the partial solution is sent up the levels, and control information is propagated down the levels, which makes it different from pure subproblem relaxation. These problem solving strategies are applied to the Chinese input problem.

Three fundamental problems for inputting Chinese are addressed here: sentence segmentation, homophone analysis, and dictionary organization. A design for a Chinese input system is presented. In the process of solving these problems, three other problems are discussed: multilevel problem solving, constraint generation, and dictionary organization. The solutions to these problems are related to each other. The organization of the dictionary facilitates the search strategy of the multilevel framework. The constraint generation makes the building of such a system easier.

In this thesis, I have pointed out how iterative relaxation can be used for constraint generation and multilevel frameworks as a general control mechanism for problem solving. I do believe that these two fundamental ideas, automatic construction of constraints and multilevel problem solving framework, could have more applications.

References

1. A. V. Aho and J. D. Ullman, *Principles of compiler design*, Addison-Wesley, Reading, Mass. (1979).

2. A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *Data structures and algorithms*, Addison-Wesley, Reading, MA (1983).

3. D. N. de G. Allen, *Relaxation methods*, McGraw-Hill, New York (1954).

4. J. D. Becker, Typing Chinese, Japanese, and Korean, *IEEE Computer* 18, 1 (Jan 1985), 27-361.

5. A. Blake, A convergent edge relaxation algorithm, Memorandum MIP-R-135, Machine intelligence research unit, University of Edinburgh (April 1982).

6. B. G. Buchanan and E. A. Feigenbaum, Dendral and Meta-Dendral: their applications dimension, *Artificial intelligence* 11, 1 (1978), 5-24.

7. Y. R. Chao, *A grammar of spoken chinese*, University of California Press (1970).

8. C. Cheng, Analysis of present-day Mandarin, *Journal of Chinese linguistics* 10 (June 1982), 281-357.

9. C. T. Chou, Relaxation and polymorphic type inference, Master thesis, Computer Science Dept., UCLA, Los Angeles (1985).

10. N. Christofides and C. Whitlock, An LP-based TSP algorithm, Imperial College Report 78-79, Imperial College (1978).

11. Hanyu pinyin cihui, Wenzi Gaige Chubanshe, Peking (1958).

12. IEEE Computer, January 1985.

13. W. Cui, Evaluation of Chinese character keyboards, *IEEE Computer* **18**, 1 (Jan 1985), 54-59.

14. L. D. Erman, F. Hayes-Roth, V. R. Lesser, and D. R. Reddy, The Hearsay-II speech-understanding system: integrating knowledge to resolve understanding, *Computing survey* 12, 2 (June 1980).

15. L. R. Erman and V. R. Lesser, A multi-level organization for problem solving using many, diverse, cooperating sources of knowledge, *Proc. 4th IJCAI*, Tbilissi, USSR (1975), 483-490.

16. M. L. Fisher and J. F. Shapiro, Constructive duality in integer programming, *SIAM J. Appl. Math.* 27 (1974), 31-52.

17. M. L. Fisher, The Lagrangian relaxation method for solving integer programming problems, *Management Science* **27**, 1 (Jan 1981), 1-18.

18. A. Friedman, *Foundations of modern analysis*, Dover publications, inc., New York (1982).

19. A. M. Geoffrion, Lagrangean relaxation and its uses in integer programming, *Mathematical programming study* 2 (1974), 82-114.

20. R. M. Haralick and G. L. Elliott, Increasing tree search efficiency for constraint satisfaction problems, *Artificial Intelligence* 14 (1980), 263-313.

21. A. Y. Hashimoto, The Mandarin syntactic structures., *Unicorn* 8 (1971), 1-149.

22. M Held and R M Karp, The traveling salesman problem and minimum spanning trees, *Operations research* 18 (1970), 1138-1162.

23. M Held and R M Karp, The traveling salesman problem and minimum spanning trees: Part II, *Mathematical programming* 1 (1971), 6-25.

24. J. Hobby and G. Guoan, A Chinese Metafont, STAN-CS-83-974, Standford Computer Science Department (1983).

25. J. E. Hopcroft and J. D. Ullman, *Introduction to automata theory, languages, and computation*, Addison-Wesley, Reading, Mass. (1979).

26. L. R. Hu, Y. W. Chang, and J. K. Huang, *The Three-corner coding method: the digitalized Chinese dictionary*, Northern Gate Book, Taipei (1977).

27. R. A. Hummel and A. Rosenfeld, Relaxation processes for scene labeling, *IEEE trans. systems, man, and cybernetics* **SMC-8**, 10 (October 1978), 765-768.

28. Chokho Ip, A Chinese Japanese word processor, Master thesis, Computer Science Dept., UCLA, Los Angeles (1985).

29. L. Kitchen and A. Rosenfeld, Discrete relaxation for matching relational structures, *IEEE trans. systems, man, and cybernetics* **SMC-9**, 12 (December 1979), 869-874.

30. L. Kitchen, Relaxation applied to matching quantitative relational structures, *IEEE trans. systems, man, and cybernetics* **SMC-10**, 2 (February 1980), 96-101.

31. D. E. Knuth, *TEX and METAFONT, New directions in typesetting*, Digital Press and American Mathematical Society (1979).

32. N. Lavrac, I. Bratko, I. Mozetic, B. Cercek, A. Grad, and M. Horvat, KARDIO-E – an expert system for electrocardiographic diagnosis of cardiac arrhythmias, *Expert systems* 2 (January 1985), 46-50.

33. C. N. Li and S. A. Thompson, *Mandarin Chinese: a functional reference grammar*, University of California Press, Berkeley (1981).

34. H. T. Lin, *Essential grammar for modern chinese*, Cheng & Tsui company, inc., Boston (1981).

35. B. T. Lowerre and R. Reddy, The HARPY speech understanding system, pp 340-360 in *Trends in speech recognition* , ed Lea, W. A., Prentice-Hall, Englewood Cliffs, N.J. (1980).

36. A. K. Mackworth, Consistency in networks of relations, *Artificial intelligence* 8 (1977), 99-118.

37. H. Makino, Beta: an automatic kana-kanji translation system, *IEEE Computer* 18, 1 (Jan 1985), 46-53.

38. I. Mozetic, I. Bratko, and N. Lavrac, An experiment in automatic synthesis of expert knowledge through qualitative modelling, *Proc. logic programming workshop 83*, Albufeira, Portugal (1983).

39. I. Mozetic, I. Bratko, and N. Lavrac, The derivation of medical knowledge from a qualitative model of the heart, *ISSEK workshop 84*, Bled, Yugoslavia (1984).

40. N. J. Nilsson, *Principles of artificial intelligence*, Tioga Press, Palo Alto: Calif. (1980).

41. J. M. Ortega and W. C. Rheinboldt, *Iterative solution of nonlinear equations in several variables*, Academic Press, New York (1970).

42. J. M. Ortega, *Numerical analysis -- A second course*, Academic Press, New York (1972).

43. C. H. Papadimitriou and K. Steiglitz, *Combinatorial optimization: algorithms and complexity*, Prentice-Hall, New Jersey (1982).

44. D. S. Parker, Relaxation problem solving, UCLA working paper, Computer Science Dept., UCLA, Los Angeles (1985).

45. G. Polya, *How to solve it*, Doubleday Anchor, New York (1957).

46. G. Polya, *Mathematical discovery*, John Wiley & Sons, New York (1981).

47. J. A. Richards, D. A. Landgrebe, and P. H. Swain, On the accuracy of pixel relaxation labeling, *IEEE trans. systems, man, and cybernetics* SMC-11, 4 (April 1981), 303-309.

48. A. Rosenfeld, R. A. Hummel, and S. W. Zucker, Scene labeling by relaxation operators, *IEEE trans. systems, man, and cybernetics* SMC-6, 6 (June 1976), 420-433.

49. G T Ross and R M Soland, A branch and bound algorithm for the generalized assignment problem, *Mathematical programming*, 8 (1975), 91-103.

50. S. Rubin, The ARGOS image understanding system, Doctoral thesis, Dept. of Computer Science, Carnegie-Mellon University, Pittsburgh, Pa. (1978).

51. E. D. Sacerdoti, Planning in hierarchy of abstraction spaces, *Artificial intelligence* 5, 2 (1974), 115-135.

52. J. Sheng, A pinyin keyboard for inputting chinese characters, *Computer* (January 1985).

53. E H Shortliffe, *Computer-based medical consultations: MYCIN*, Elsevier Scientific Publishing Company, Inc., New York (1976).

54. T. R. Smith and R. E. Parker, An analysis of the efficacy and efficiency of hierarchical procedures for computing trajectories over complex surfaces, Unpublished manuscript, Department of Computer Science, UCSB, Santa Barbara (1985).

55. R. V. Southwell, *Relaxation methods in engineering science*, Oxford U. Press (1940).

56. J. F. Sowa, *Conceptual structure: information processing in mind and machine*, Addison-Wesley, Reading, Mass. (1984).

57. R. E. Tarjan, A unified approach to path problems, *JACM* 28, 3 (July 1981), 577-593.

58. H. C. Tien, The Pinxxiee Chinese word processor, *IEEE Computer* 18, 1 (Jan 1985), 65-66.

59. J. R. Ullman, An algorithm for subgraph isomorphism, *JACM* 23 (Jan 1976), 31-42.

60. D. Waltz, Generating semantic descriptions from drawings of scenes with shadows, pp 19-92 in *The Psychology of Computer Vision* , ed P. Winston, McGraw-Hill, New York (1975).

61. W. S. Wang, Some syntactic rules for Mandarin, *Proceedings of the ninth international congress of linguists.* (1964), 191-202.

62. Asia Computer Weekly, Dec 1983.

63. S. W. Zucker, E. V. Krishnamurthy, and R. L. Haar, Relaxation processes for scene labeling: convergence, speed and stability, *IEEE trans. systems, man, and cybernetics* SMC-8, 1 (January 1978), 41-48.

64. S. W. Zucker, Production systems with feedback, pp 539-555 in *Pattern-Directed Inference Systems* , ed D. A. Waterman and F. Hayes-Roth, Academic Press (1978).

Appendix 1

Algorithm 4.2

for all nonterminals A, $FIRST_1(A) := EXACT_1(A)$;
while there are changes to any of the the $FIRST_1(A)$ *do*
begin
    for all production rules $A \rightarrow BC$ *do*

    begin
        $FIRST_1(A) := FIRST_1(A) \cup FIRST_1(B)$;
        if $EXACT_0(B) = \{\varepsilon\}$ *then*
            $FIRST_1(A) := FIRST_1(A) \cup FIRST_1(C)$;
    end
end.

Algorithm 4.3

for all nonterminals A, $LAST_1(A) := EXACT_1(A)$;
while there are changes to any of the the $LAST_1(A)$ *do*
begin
    for all production rules $A \rightarrow BC$ *do*
    begin
        $LAST_1(A) := LAST_1(A) \cup LAST_1(C)$;
        if $EXACT_0(C) = \{\varepsilon\}$ *then*
            $LAST_1(A) := LAST_1(A) \cup LAST_1(B)$;
    end
end.

Algorithm 4.5

for all nonterminals A, and productions $A \rightarrow B_1C_1 \mid \cdots \mid B_nC_n$,

$$EXACT_i(A) = \bigcup_{k=1}^{k=n} \bigcup_{j=1}^{j=i-1} EXACT_j(B_k)EXACT_{i-j}(C_k),$$

while there are changes to any of the the $EXACT_i(A)$ *do*
begin
    for all production rules $A \rightarrow BC$ *do*
    begin
        if $EXACT_0(B) = \{\varepsilon\}$ *then*
            $EXACT_i(A) = EXACT_i(A) \cup EXACT_i(C)$;
        if $EXACT_0(C) = \{\varepsilon\}$ *then*
            $EXACT_i(A) = EXACT_i(A) \cup EXACT_i(B)$;
    end
end.

Algorithm 4.6

for all nonterminals A, $FIRST_i(A) = \emptyset$;
for all productions $A \rightarrow BC$,

$$FIRST_i(A) = FIRST_i(A) \cup EXACT_i(B) \cup \bigcup_{j=1}^{i-1} EXACT_j(B)FIRST_{i-j}(C);$$

while there are changes to any of the the $FIRST_i(A)$ *do*
begin
    for all production rules $A \rightarrow BC$ *do*
        $FIRST_i(A) = FIRST_i(A) \cup FIRST_i(B) \cup EXACT_0(B)FIRST_i(C)$;
end.

Algorithm 4.7

for all nonterminals A, $LAST_i(A) = \varnothing$;
for all productions $A \rightarrow BC$,

$$LAST_i(A) = LAST_i(A) \cup EXACT_i(C) \cup \bigcup_{j=1}^{i-1} LAST_{i-j}(B) EXACT_j(C);$$

while there are changes to any of the the $LAST_i(A)$ *do*
begin
for all production rules A → BC *do*

$$LAST_i(A) = LAST_i(A) \cup LAST_i(C) \cup LAST_i(B) EXACT_0(C);$$

end.

Algorithm 4.8

$CONST_n = \varnothing$;
For all productions $A \rightarrow BC$,

$$CONST_n = CONST_n \cup \bigcup_i LAST_i(B) FIRST_{n-i}(C).$$

Appendix 2

Following is a context free grammar for Chinese. The set of terminals is { Start, S, TP, NP, N, NPR, DET, TIME, VP, DEG, MAN, LOC, COMP }. The set of nonterminals is { α, ψ, *particle, place, men, noun-human, noun-nonhuman, pronoun, measure, prep, asp, de, adverb, bi, verb, aux, conjunction, dem, numeral, post-det* }.

Start → ⇒ S ⇐

S → *conjunction* S *particle*
| S *particle*
| NP VP
| NP VP TP
| NP VP TP *particle*
| ε

TP → TIME | *place* | TIME *place*

NP → N
| NPR *men*
| DET N
| DET *post-det* N
| DET *post-det* S N

N → NPR | *noun-nonhuman* | *place*

NPR → *noun-human* | *pronoun*

DET → *dem measure*
| *numeral measure*
| *dem numeral measure*

TIME → *prep* NP

VP → VP NP
| *verb* NP COMP
| *aux verb* NP *prep* COMP

COMP → *asp* COMP
| LOC COMP
| MAN COMP
| DEG COMP
| *prep* COMP
| *measure* COMP
| ε

DEG → *bi* S
| *adverb*
| *bi* S *measure*

DEG → *de* S
| *adverb*

LOC → *prep place*

Appendix 3

Table a.1 is the list of phonetic spellings and the corresponding Hanzis that appear in the appendix. If the entry in the following table is not the required Hanzis, the right one will be printed.

ba	把	bei	被	bi	比	bie	別
bu	不	chi	遲	cong	從	dao	到
dui	對	de	的	duo	多	er	兒
gei	給	gen	跟	geng	更	guan	關
guo	過	hai	還	he	和	jia	加
jiao	叫	kuai	塊	lai	來	le	了
mei	沒	li	離	qi	起	qu	去
rang	讓	shang	上	shao	小	shi	是
suo	所	tong	同	wan	往	wang	往
wei	爲	xia	下	xiang	像	yang	樣
yi	一	you	有	yu	于	zai	在
zao	早	zhi	之	zhong	中		

Table a.1 Some common Hanzis.

3. Patterns for prepositions

The general pattern is:

S [A] [AV] CV-N FV [Mod] O.

Following is the list of all specific patterns.

• *wei* (爲)

A *wei* N *suo* FV.

• *gen* (跟), *tong* (同), *he* (和)

S [A] [AV] *gen*-N *yikuair/yiqi* FV [Mod] O.

S [A] [AV] *tong*-N *yikuair/yiqi* FV [Mod] O.
S [A] [AV] *he*-N *yikuair/yiqi* FV [Mod] O.

• *dui* (對)

S [A] *dui*-N [A] SV.
S [A] [AV] *dui*-N *jiayi* (加 以) FV.
S [A] [AV] *dui*-N *yuyi* (予 以) FV.
S [A] [AV] *dui*-N FV [*le* (了)] [Mod] O..

• *duiyu* (對 于), *guanyu* (關 于)

[*duiyu/guanyu*-N] S [*duiyu/guanyu*-N] [AV] FV O.

• *zai* (在)

S *zai* NP.
S FV-*zai* PW.
O, S [A] [AV] FV-*zai* PW.
S [A] [AV] *ba* O FV-*zai* PW.
S [A] [AV] *zai*-PW FV [Mod] O.
S [A] [AV] FV-*zai* FV [Mod] O.
[*zai*] PW, *you* Mod NP.
[*zai*]-PW, S *shi* comment.
[*zai*]-TW S [*zai*-TW] A CV-N FV O.
[*zai*]-TW, *mei/bu/bie* FV O.
[*zai*] condition *zhi zhong*, S [A] SV [le].
[*zai*] condition *zhi zhong*, S [A] [AV] FV O [le].
[*zai*] condition *zhi shang*, S [A] SV [le].
[*zai*] condition *zhi shang*, S [A] [AV] FV O [le].
[*zai*] condition *zhi xia*, S [A] SV [le].
[*zai*] condition *zhi xia*, S [A] [AV] FV O [le].

• *yu* (于)

Same set of rules as *zai*.

• *dao* (到)

dao PW *qu/lai* [le].
S [*cong* PW_1] *qu/lai* [le] PW_2 [le].
S [A] [AV] *cong*-PW_1 *dao*-PW_2 FV [Mod] O [le].

• *wang* (往)

S [A] [AV] *wang*-N FV.

• *lai* (來), *qu* (去)

S [A] [AV] *lai/qu* FV-N *lai/qu*.
S FV_1 O_1 *lai/qu* FV_2 O_2.

• *cong* (從)

S [A] [AV] *cong*-PW_1 *dao*-PW_2 *lai/qu*.
S [A] [AV] *cong*-PW_1 *zuo*-means *dao*-PW_2 *lai/qu*.
S [A] [AV] *zuo*-means *cong*-PW_1 *dao*-PW_2 *lai/qu*.

• *bi* (比)

NP_1 [A] *bi* NP_2 SV.
NP_1 [A] *bi* NP_2 *hai/geng* SV.
NP_1 [A] *bi* NP_2 SV_1-de-SV_2.
NP_1 [A] *bi* NP_2 SV-Nu-M-N.
NP_1 [A] *bi* NP_2 SV Nu-*bei* (倍).
NP_1 [A] *bi* NP_2 FV-de-SV.
NP_1 [AV] *bi* NP_2 *duo* FV-*le* Nu-M-O.
NP_1 [AV] *bi* NP_2 *shao* FV-*le* Nu-M-O.
NP_1 [AV] *bi* NP_2 *zao* FV-*le* Nu-M-O.
NP_1 [AV] *bi* NP_2 *wan* FV-*le* Nu-M-O.
NP_1 [AV] *bi* NP_2 *chi* FV-*le* Nu-M-O.
S FV O, TW_1 *bi* TW_2 + result of comparison.
S TW_1 FV O *bi* TW_2 + result of comparison.
S *yi*-M *bi* yi-M SV.
S *yi*-M *bi* yi-M AV FV [Mod] O.

• *gen* (跟)

NP_1 *gen* NP_2 [A] *yiyang*.
NP_1 *gen* NP_2 [A] *buyiyang*.
NP_1 *bugen* NP_2 [A] *yiyang*.
NP_1 *gen* NP_2 [A] *yiyang* [*de*] SV.
NP_1 *gen* NP_2 [A] *yiyang* AV FV [Mod] O.

• *xiang* (像)

NP_1 [A] *xiang* NP_2 *yiyang*.
NP_1 [A] *xiang* NP_2 *yiyang* AV FV [Mod] O.

• *you* (有)

NP_1 *you* NP_2 Nu-*bei* (倍).
NP_1 *you* NP_2 SV-*bei* (倍).

• *li* (離)

PW_1 *li* PW_2 A SV.
PW_1 *li* PW_2 [*you/shi*] Nu-M-N.
PW_1 *li* PW_2 *gen li* PW_3 *yiyang* SV.
PW_1 *li* PW_2 *bi* PW_3 *li* PW_4 SV.
PW_1 *gen* PW_2 *li* PW_3 *yiyang/buyiyang* SV.
PW_1 *bi* PW_2 *li* PW_3 SV.

• *ba* (把)

S [A] [AV] *ba* O FV + complement.
S [A] [AV] *ba* O *gei* FV + complement.
S [A] [AV] *ba* O *gei* FV *le*.
S [A] [AV] *ba* O *gei* FV-*zai*-PW.
S [A] [AV] *ba* O *gei* FV-*dao* PW [*lai/qu*].
S [A] [AV] *ba* O *gei* FV_1-FV_2-[PW]-FV_3.
S [A] [AV] *ba* Mod Dir O FV-*gei* Ind O.
S [A] [AV] *ba* [Mod] [*gei*] FV-[*le/guo*]-Nu-M.
S [A] [AV] *ba* Mod O [*gei*] FV-[*le/guo*]-Nu-M.
S [A] [AV] *ba* Mod O [*gei*] FV *de* A SV.
S [A] [AV] *ba* O [*gei*] FV + functional ending.
ba person SV *de* + clause with FV.
S [A] [AV] *ba* NP_1 [*gei*] FV NP_2 [P].

4. Patterns for structural particles

Following is a list of patterns that govern the construction of a phrase or a sentence with the structural particles.

• *de* (的)

V O *de* N.
V O *de*.
S V O *de* N.
zai (在) S FV *de shihou* (時 候).
N_1 *de* N_2
PN *de* N
S V *de* N.
A SV *de* N.
shi (是) ... *de*
[TW] S [TW] *cai* (才) FV O *de*.
Nu-M Nu-M *de*

• *le* (了)

V *le*
S V *le* O
SV *le*
S SV *le* Nu-M
zai (在) S FV [*le*] O [*le*] *yehou* (以 劫).

• *guo* (過)

S [*mei* (沒)] SV *guo*.
S [*mei* (沒)] FV *guo* [Mod] O.
S *conglai* (從 來) *mei* (沒) *V guo O*.
S xianglai (向 來) *mei* (沒) *V guo O*.

• *zhe* (着)

FV *zhe*
V_1 *zhe* V_2

S *zhengzai* (正 在) FV *zhe* [Mod] O *ne* ( ).
S FV_1 *zhe* FV_2 O *zhe* ( ).

• *de* (地)

S A *de* FV Mod [*de*] O.

• *de* (得)

V *de* complement
NP FV *de* AV SV.
S FV *de* S-SV.
S FV *de* S-AV-FV-O.

5. Patterns for conjunctions

Following is the list of common conjunctions with at least one component contains a single character.

yue ... *yue* (越 ... 越)
yu ... *yu* (愈 ... 愈)
ye ... *ye* (也 ... 也)
you ... *you* (又 ... 又)
bu ... *bu* (不 ... 不)
suirean ... *ye* (然 ... 也)
yao ... *jiu* (要 ... 就)
fei ... *bu* (非 ... 不)
fei ... *cai* (非 ... 才)
zhiyao ... *jiu* (只 要 ... 就)
zai ... *ye* (再 ... 也)
duo ... *ye* (多 ... 也)
jiushi ... *ye* (就 是 ... 也)
jishi ... *ye* (即 使 ... 也)
gangcai ... *jiu* (剛 才 ... 就)
gang ... *jiu* (剛 ... 就)
yi ... *jiu* (一 ... 就)
yige ... *jiu* (一 個 ... 就)
you ... *meiyou* (有 ... 沒 有)