TABLE OF CONTENTS

VIEW IMPLEMENTATION OF DEDUCTIVE AXIOMS WITH NEGATION

Dr. Yuksel UCKAN
Dept. of Systems Analysis
Miami University
Oxford, OH 45056

ABSTRACT

A logic-based data base system combines data manipulation capabilities of traditional data base management systems (DBMS) and theorem-proving capabilities of logic systems. If only one programming system is used for both data base access operations and inferential functions in a logic-based system, it is called a deductive data base system. This study investigates capabilities of traditional relational DBMSs as deductive DBMSs. It is shown that views in relational systems can be used to represent and implement deductive axioms that are Horn clauses. Also, provided they are correctly interpreted, deductive axioms involving negated nonground formulas can be translated to views. Proper interpretation of such axioms entails the use of set difference in the corresponding views instead of complementation. In this study, we also describe an automatic Datalog-to-SQL translator that can be used as a user interface in converting axioms represented in Datalog, a logic programming language for data base systems, to views expressed in SQL, a standard relational data base language.

KEY WORDS: Logic system, deductive data base, relational view, negated axiom, Datalog, SQL.

1. INTRODUCTION

Logic-based systems are becoming increasingly important in data base research. A logic-based data base system exhibits some form of intelligence and, through reasoning, can generate additional knowledge that is not explicitly stored in the data base. A traditional relational data base management system provides efficient access to large volumes of data. On the other hand, a logic system can be used to express and process deductive axioms that generate knowledge from facts stored in a data base. A logic-based system should have at least the efficiency of the present-day relational DBMSs such as SQL or INGRES, and the theorem-proving capabilities of artificial intelligence (AI) tools, such as Prolog or Lisp. In addition to these AI tools, other rule definition languages more suitable for data base systems are currently being investigated. These include LDL1 [1] and Datalog [2].

Presently, there is no commercially available logic-based DBMS. There are, however, a number of experimental systems that may evolve into commercial products. For design of logic-based DBMSs, two basic approaches are being considered: heterogeneous and homogeneous. In the **heterogeneous approach**, two distinct systems, a conventional DBMS, and a theorem-proving system, are coupled through an interface. In the **homogeneous approach**, data manipulation and theorem-proving functions are integrated into a single system, with only one programming language for both data manipulation operations and inferential functions. A homogeneous logic-based system is also known as a deductive data base system. There have been some studies aiming to use Prolog as a deductive data base system [3]. Although Prolog is suitable for representing data base facts and deductive rules, and therefore is capable of deductive reasoning, it requires facts and deductive rules to reside within the virtual memory. For a very large data base, duplicating facts in Prolog notation in the virtual memory may be impossible, or at least impractical. Also, unlike a DBMS, Prolog does not support query optimization, concurrent data base access, and security and recovery features. To alleviate these shortcomings, several extensions of Prolog have been proposed [4]. The major problem with such extensions is that they all entail development of new DBMSs.

On the other extreme of the spectrum, there is ongoing research aiming to extend a traditional DBMS by including among its capabilities deductive axiom representation and processing. These are also homogeneous systems. One such system is POSTGRES [5, 6], under development at the University of California, Berkeley. Its language, POSTQUEL, is an extension of the QUEL language of the INGRES data base management system. The objective of the POSTQUEL project is to render INGRES a deductive DBMS.

In Section 2 of this paper, we introduce the fundamental concepts for deductive data bases. In Section 3, we discuss the semantics of Datalog as a logic language for data base systems. In Section 4, we consider an extension of Datalog that can be used to express deductive axioms involving negation. In Section 5, we take the view that a traditional DBMS has substantial deductive capabilities, and investigate the expressive power of SQL, a major data base language, as a rule-based system. Since deductive axioms in a logic system display some similarities to virtual relations (views) in a relational DBMS, we propose to represent deductive axioms using SQL views [7]. In Section 6, we give a set of templates that can be used to translate Datalog axioms to SQL views, and describe briefly an automatic translator from Datalog to SQL. In Section 7, we discuss the strengths and weaknesses of the idea of using a traditional DBMS as a deductive DBMS, and point out the necessary extensions to render a DBMS fully and efficiently deductive.

2. FUNDAMENTAL CONCEPTS OF DEDUCTIVE DATA BASES

A traditional relational data base consists of relations. Each relation contains tuples that represent data. Generally, relations are stored as files in the physical data base. Such relations are called **base relations**. All base relations in a data base constitute the **extensional data base** (EDB).

Knowledge is information that can be derived from data. Through the use of formal logic, it may be possible to derive knowledge from extensional data base relations. The most commonly used form of reasoning for knowledge derivation is deductive reasoning. In **deductive reasoning**, new facts are inferred from a given set of facts using the rules of inference. Compared to physically recording derivable facts in a data base, it is definitely preferable to represent them in an implicit fashion, and generate them whenever needed. For this, we need an appropriate knowledge representation scheme such as predicate logic, semantic networks, and production rules.

A **production rule** for deduction has the form

IF P THEN Q

where P and Q are formulas of the propositional logic. Production rules are also called axioms. Using the Datalog notation, a rule can be written as

Q :- P.

In a production rule, if P and Q involve variables, they are called **nonground formulas**. For example, in

parent(x, y) :- father(x, y).

parent(x, y) and father(x, y) are nonground formulas. If, on the other hand, variables in formulas are replaced by constants, the resulting formulas are called **ground formulas**. For example, the formulas parent(Paul, Mary) and father(Paul, Mary) in

parent(Paul, Mary) :- father(Paul, Mary).

which is an instantiation of the above production rule, are ground formulas.

In a production rule, P is also referred to as the **antecedent**, and Q as the **consequent**. Antecedents can be facts represented in an EDB. Deduced consequents become part of virtual relations. Virtual relations that can be derived from the EDB using production rules form the **intensional data base** (IDB), also called the knowledge base.

A **deductive data base** is composed of an EDB and an IDB that can be derived from the EDB using a set of logical rules. Rules can be generalized to the form below:

Q1 V Q2 V ... V Qn :- P1 & P2 & ... & Pm.

where V denotes disjunction, & denotes conjunction, and Pi and Qj are atomic nonground formulas. An antecedent in this form is said to be in **conjunctive normal form**. If there is at most one atomic formula in the consequent, the rule is referred to as a **Horn clause**. The following are Horn clauses:

P1 & P2 & ... & Pm.
Q :- P1 & P2 & ... & Pm.

A collection of Horn clauses is called a **logic program**. In a deductive data base, the IDB can be defined using a logic program. A logic program is expressed in a logic language. Datalog is an example of a logic-based data model and a logic language for relational data bases. It is a version of Prolog with some differences.

A **deductive DBMS** (also called an **expert DBMS**,) is a DBMS that can deduce IDB facts from an EDB by applying deductive rules in a logic program.

3. <u>STRATIFIED</u> <u>DATALOG</u> <u>PROGRAMS</u>

A Datalog program is a set of facts corresponding to an EDB, and a set of deductive axioms forming an IDB. Facts can be represented as ground formulas, and axioms are Horn clauses. As an example, consider the relational data base of Figure 1. This data base has three relations, the relation "people", and the two unary relations "peop", and "rich". The relation "people" corresponds to a genealogical pedigree chart as shown in Figure 2.

PEOPLE:

NAME	FATHER	MOTHER	SEX
ALICE	-	-	F
BILL	TOM	ELLEN	M
BRIAN	BILL	JEANNE	M
ELLEN	-	-	F
GEORGE	BILL	JEANNE	M
IRA	BILL	JEANNE	M
JASON	JOHN	JOANNE	M
JEANNE	-	-	F
JIM	-	-	M
JOANNE	JIM	ALICE	F
JOHN	-	-	M
KATHY	JIM	ALICE	F
KIM	BILL	JEANNE	F
LINDA	BRIAN	KATHY	F
SALLY	BRIAN	KATHY	F
TOM	-	-	M

PEOP:

NAME
ALICE
BILL
BRIAN
ELLEN
GEORGE
IRA
JASON
JEANNE
JIM
JOANNE
JOHN
KATHY
KIM
LINDA
SALLY
TOM

RICH:

NAME
ALICE
BILL
GEORGE
JASON
JEANNE

Figure 1. Sample Extensional Data Base

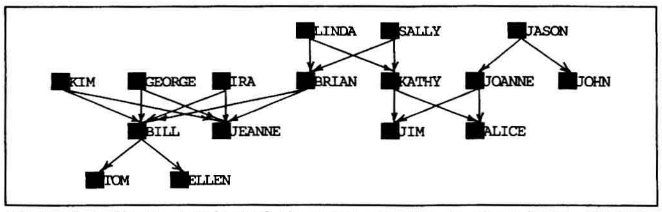


Figure 2. Pedigree Genealogical Chart Corresponding to the Relation "people"

It is possible to represent the facts in the relational data base of Figure 1 in Datalog. In this case, the ground formulas

```
father(Bill, Tom).
mother(Bill, Ellen).
sex(Bill, 'M').
rich(Bill).
...........
```

form a Datalog program corresponding to the EDB. However, in this paper, we will avoid using this representation for extensional data bases, and instead use the conventional relational representation as in Figure 1.

Based on an EDB, deductive axioms can be expressed in Datalog to define intensional data bases. Figure 3 shows nine deductive axioms that define family relationships as a stratified Datalog program of two strata. A **stratified Datalog program** is an ordered set of strata such that each stratum enriched with the least model of the previous stratum, if any, has a unique least model. The **least model of a Datalog program** is the set of all occurrences of the IDB relations defined by it. In each stratum of a Datalog program, the order of deductive axioms is irrelevant. In Figure 3, deductive axioms 1 through 4 define a stratum. Each axiom in Stratum S1 is based on only the EDB relations. Thus, they can be executed in any order. The least model of Stratum S1 includes all instances of the relations parent, brother, sister, and sibling in addition to the EDB of Figure 1. On the other hand, deductive axioms 5 through 9 for spouse, grandparent, uncle, aunt, and cousin form Stratum S2. They are defined in terms of the relations in Stratum S1.

```
(1) parent(x, y) :- father(x, y).
    parent(x, y) :- mother(x, y).

(2) brother(x, y) :- father(x, z1) & father(y, z1) & mother(x, z2) &
                     mother(y, z2) & sex(y, 'M') & x <> y.

(3) sister(x, y) :- father(x, z1) & father(y, z1) & mother(x, z2) &
                    mother(y, z2) & sex(y, 'F') & x <> y.

(4) sibling(x, y) :- sister(x, y).
    sibling(x, y) :- brother(x, y).
                          STRATUM S1

(5) spouse(x, y) :- parent(z, x) & parent(z, y).
(6) grandparent(x, y) :- parent(x, z) & parent(z, y).
(7) uncle(x, y) :- parent(x, z) & brother(z, y).
(8) aunt(x, y) :- parent(x, z) & sister(z, y).
(9) cousin(x, y) :- parent(x, z1) & parent(y, z2) & sibling(z1, z2).
                          STRATUM S2
```

Figure 3. A Stratified Datalog Program for an Intensional Data Base

4. <u>DEDUCTIVE</u> <u>AXIOMS</u> <u>INVOLVING</u> <u>NEGATION</u>

Datalog can be extended to include negated nonground formulas in axioms [8]. Such an extension is desirable to improve the expressive power of the language. Figure 4 shows a Datalog program of nine axioms involving negation. These axioms constitute Stratum S3 with respect to Strata S1 and S2 of Figure 3. For example, since one's mother is one's parent but not one's father, the relation mother can be defined alternatively as Axiom 3 of Figure 4:

mother(x, y) :- parent(x, y) & $\neg$father(x, y).

Although this rule is not a Horn clause, it can be used in deriving the mother relation. The tuples of the mother relation are those of the parent relation provided they are not also tuples of the father relation. This

```
(1) notfather(y) :- sex(y, 'M') & ¬father(x, y).
(2) haveNoFather(x) :- name(x) & ¬father(x, y).
(3) mother(x, y) :- parent(x, y) & ¬father(x, y).
(4) notParent(y) :- name(y) & ¬parent(x, y).
(5) haveNoParent(x) :- name(x) & ¬parent(x, y).
(6) bachelor(x) :- name(x) & sex(x, 'M') & ¬spouse(x, y).
(7) poorUncle(x, y) :- uncle(x, y) & ¬rich(y).
(8) brother(x, y) :- sibling(x, y) & ¬sister(x, y).
(9) poorBrother(x, y) :- sibling(x, y) & ¬sister(x, y) & ¬rich(y).
                            STRATUM S3
```

Figure 4. A Datalog Program for Deductive Axioms Involving Negation

interpretation is equivalent to the set difference operation in relational algebra, and can easily be handled by a logic language processor.

The major difficulty with negation in deductive rules is due to the interpretation of negation in data base terms. With regular Horn clauses, we do not have a problem of interpretation. This is because any Horn clause can be expressed in terms of the relational algebra operations. For example, the IDB relation uncle defined as:

uncle(x, y) :- parent(x, z) & brother(z, y)

is a natural join of the relations parent and brother over the common domain z. However, if a deductive axiom involves negation, since a negated nonground formula is intuitively equivalent to the set complement operation, and complementation is not one of the operators in relational algebra, we must be especially careful in interpreting such an axiom in relational algebra. Let us consider an example, and define the relation bachelor as:

bachelor(x) :- sex(x, 'M') & ¬spouse(x, y).

This is equivalent to saying that "a bachelor is an unmarried male." If we propose to interpret the negated formula ¬spouse(x, y) as the complement of spouse(x, y), we come across two problems. First, in view of such an interpretation, ¬spouse(x, y) defines an infinite set of combinations of all possible males and females who are not married to each other. This problem can be resolved by using the **closed world assumption** (CWA) [9, 10]. The CWA says that if a ground formula cannot be deduced from the EDB or IDB relations, we can assume that it is false. In other words, stated in data base terms, we can assume that the relation does not contain the tuple that corresponds to that ground formula. For example, for the data base of Figure 1, the ground formula spouse(Dick, Anne) is false, because neither Dick nor Anne belongs to the pedigree chart of Figure 2. Consequently, the complement of spouse(x, y) is the set of all ground formulas obtained by combining male and female names from the instance of the relation "people" (or the relation "peop", assuming that it is the relation that defines all persons represented in the data base,) provided such males and females are not married to each other. Thus, the relation bachelor can be rewritten as in Axiom 6 of Figure 4:

bachelor(x) :- name(x) & sex(x, 'M') & ¬spouse(x, y).

The second problem is due to the interpretation of negation as complementation. The CWA has rendered the relation ¬spouse(x, y) finite. Despite this, we get incorrect results if we use complementation. This is because the complement of spouse will include tuples such as (Jim, Joanne), since Jim is married to Alice and not to Joanne. A subsequent selection for male will inaccurately return Jim as a bachelor. To get around this problem, we propose to consider the projection of the negated relation over the variable that also appears in another formula, and compute the difference instead of the complement. For example, for the relation bachelor, it is possible to use the interpretation "the set of bachelors is the difference of the set of all males and the set of all x values obtained by projecting spouse(x, y) over the attribute x." As we will see in the ensuing discussion, this approach can easily be implemented in data base languages such as SQL or QUEL, and it eliminates the need for defining extraneous intermediate relations.

5. USE OF VIEWS AS DEDUCTIVE AXIOMS

In a relational data base, a **view** is a virtual relation, since it is a definition of a relation that can be derived from the existing base relations and other views. A view definition can be supplied by the user, and is stored in the system catalog. Once a view is defined, it can be used as if it were a data base relation. Whenever a view is referenced in a query, the DBMS accesses its definition, substitutes it in the user query, generates the canonical form for the query, and processes it. Many relational DBMSs support views, and offer facilities for declaring, using, and deleting them.

Clearly, the concept of views is similar to that of IDB relations. Like views, an IDB relation is also virtual, and its definition is based on existing EDB relations. Therefore, it is easy to see that deductive axioms can be declared and implemented as views using traditional data base languages. In the present section, we will take this approach, and express Datalog axioms as SQL views. Figure 5 shows SQL view definitions for the Datalog program of Figure 3. These views have been tested using a SQL-based DBMS running on a mainframe, and it has been demonstrated that queries based on them are, in general, as efficient as queries that are based directly on EDB relations.

```
-- (1) parent(x, y) :- father(x, y).
--     parent(x, y) :- mother(x, y).

create view parent (name, parentname)
as select x.name, y.name
   from people x, people y
   where x.father = y.name or x.mother = y.name
```
```
-- (2) brother(x, y) :- father(x, z1) & father(y, z1) & mother(x, z2) &
--                      mother(y, z2) & sex(y, 'M') & x <> y.

create view brother (name, brothername)
as select x.name, y.name
   from peop x, peop y, people f1, people f2, people m1, people m2
   where f1.name = x.name        and f1.father = f2.father
   and   f2.name = y.name        and m1.name = x.name
   and   m1.mother = m2.mother   and m2.name = y.name
   and   f2.sex = 'M'            and m2.sex = 'M'     and x.name ^= y.name
```
```
-- (3) sister(x, y) :- father(x, z1) & father(y, z1) & mother(x, z2) &
--                     mother(y, z2) & sex(y, 'F') & x <> y.

create view sister (name, sistername)
as select x.name, y.name
   from peop x, peop y, people f1, people f2, people m1, people m2
   where f1.name = x.name        and f1.father = f2.father
   and   f2.name = y.name        and m1.name = x.name
   and   m1.mother = m2.mother   and m2.name = y.name
   and   f2.sex = 'F'            and m2.sex = 'F'     and x.name ^= y.name
```
```
-- (4) sibling(x, y) :- sister(x, y).
--     sibling(x, y) :- brother(x, y).

create view sibling (name, siblingname)
as select x.name, y.name
   from peop x, peop y, sister, brother
   where sister.name = x.name  and sister.sistername = y.name
   or    brother.name = x.name and brother.brothername = y.name
```
```
-- (5) spouse(x, y) :- parent(z, x) & parent(z, y).

create view spouse (name, spousename)
as select x.name, y.name
   from peop x, peop y, parent p1, parent p2
   where p1.name = p2.name        and p1.parentname = x.name
   and   p2.parentname = y.name and x.name  = y.name
```
```
-- (6) grandparent(x, y) :- parent(x, z) & parent(z, y).

create view grandparent (name, gparentname)
as select x.name, y.name
   from peop x, peop y, parent p1, parent p2
   where p1.name=x.name and p1.parentname=p2.name and p2.parentname=y.name
```
```
-- (7) uncle(x, y) :- parent(x, z) & brother(z, y).

create view uncle (name, unclename)
as select x.name, y.name
   from peop x, peop y, parent p, brother b
   where p.name = x.name and p.parentname = b.name and b.brothername=y.name
```
```
-- (8) aunt(x, y) :- parent(x, z) & sister(z, y).

create view aunt (name, auntname)
as select x.name, y.name
   from peop x, peop y, parent p, sister s
   where p.name = x.name and p.parentname = s.name and s.sistername = y.name
```
```
-- (9) cousin(x, y) :- parent(x, z1) & parent(y, z2) & sibling(z1, z2).

create view cousin (name, cousinname)
as select x.name, y.name
   from peop x, peop y, parent p1, parent p2, sibling s
   where x.name = p1.name        and y.name = p2.name
   and   s.name = p1.parentname and s.siblingname = p2.parentname
```

Figure 5. SQL View Implementations of the Intensional Data Base of Figure 3

Provided they are correctly interpreted, Datalog axioms with negation can also be translated to SQL views. Figure 6 shows the view definitions for the Datalog program of Figure 4. In obtaining these views certain templates that ensure correct interpretation for negation have been used. We will discuss them in the next section.

As is seen, all kinds of deductive axioms that are expressible in Datalog are also expressible as SQL views.

```
-- (1) notfather(y) :- sex(y, 'M') & ¬father(x, y).

create view notFather (name)
as select y.name
   from peop y, people pe
   where y.name = pe.name and pe.sex = 'M'
   and not exists (select *
                   from peop x, people father
                   where father.father = y.name and father.name = x.name)

-- (2) haveNoFather(x) :- name(x) & ¬father(x, y).

create view haveNoFather (name)
as select x.name
   from peop x
   where not exists (select *
                     from peop y, people father
                     where father.name = x.name and father.father = y.name)

-- (3) mother(x, y) :- parent(x, y) & ¬father(x, y).

create view mother (name, mothername)
as select x.name, y.name
   from peop x, peop y, parent p
   where x.name = p.name  and   y.name = p.parentname
   and not exists (select *
                   from people father
                   where father.name = x.name and father.father = y.name)

-- (4) notParent(y) :- name(y) & ¬parent(x, y).

create view notParent (name)
as select y.name
   from peop y
   where not exists (select *
                     from peop x, parent p
                     where p.name = x.name and p.parentname = y.name)

-- (5) haveNoParent(x) :- name(x) & ¬parent(x, y).

create view haveNoParent (name)
as select x.name
   from peop x
   where not exists (select *
                     from peop y, parent p
                     where p.name = x.name and p.parentname = y.name)

-- (6) bachelor(x) :- name(x) & sex(x, 'M') & ¬spouse(x, y).

create view bachelor (name)
as select x.name
   from peop x, people sex
   where x.name = sex.name and sex.sex = 'M'
   and not exists (select *
                   from peop y, spouse sp
                   where sp.name = x.name and sp.spousename = y.name)

-- (7) poorUncle(x, y) :- uncle(x, y) & ¬rich(y).

create view poorUncle (name, unclename)
as select x.name, y.name
   from peop x, peop  y, uncle
   where x.name = uncle.name and y.name = uncle.unclename
   and not exists (select *
                   from rich
                   where rich.name = y.name)

-- (8) brother(x, y) :- sibling(x, y) & ¬sister(x, y).

create view brother_a (name, brothername)
as select x.name, y.name
   from peop x, peop y, sibling sib
   where x.name = sib.name and y.name = sib.siblingname
   and not exists (select *
                   from sister sis
                   where x.name = sis.name and y.name = sis.sistername)

-- (9) poorBrother(x, y) :- sibling(x, y) & ¬sister(x, y) & ¬rich(y).

create view poorBrother (name, brothername)
as select x.name, y.name
   from peop x, peop y, sibling sib
   where x.name = sib.name and y.name = sib.siblingname
   and not exists (select *
                   from sister sis
                   where x.name = sis.name and y.name = sis.sistername)
   and not exists
       (select *
        from rich
        where y.name = rich.name)
```

Figure 6. SQL View Implementations of Axioms Involving Negation

The only shortcoming of this approach, that of using SQL as a logic programming language, is with respect to recursive axioms. Let us consider the following example.

```
ancestor(x, y) :- parent(x, y).
ancestor(x, y) :- parent(x, z) & ancestor(z, y).
```

This says, one's ancestors include one's parents and the ancestors of each parent. Stated somewhat differently in set-theoretic terms, the set of ancestors for a person is the union of the set of that person's parents, the set of the parents' parents, the set of the parents' grandparents, and so on. Because the relation ancestor is transitive, the set of ancestors for a given person can be computed using a special operator, known as the **transitive closure operator**. The transitive closure operator for the relation ancestor generates the set of ancestors by adding to a known set of ancestors all the tuples successively deduced by transitivity, until such additions do not result in a new tuple in the relation. The relation thus obtained is the **transitive closure set** for ancestor. The transitive closure operation is a succession of the combination of join, projection, and union operations in relational algebra. However, as repetition cannot be represented in relational algebra, in the current versions of SQL there is no operator corresponding to transitive closure. This is the most important weakness of SQL as a logic programming language for data bases.

It is certainly possible to extend SQL such that recursive rules can be expressed as views. There is some ongoing research to modify relational DBMSs such that they can handle recursion. The POSTGRES project at Berkeley [6] is an example. Its aim is to extend INGRES for deductive data base applications.

6. TRANSLATION OF DATALOG AXIOMS TO SQL VIEWS

As can be seen, Datalog is a convenient language to express deductive axioms. In transforming a relational data base to a deductive one, it is easier to express deductive axioms in Datalog than in SQL as a corresponding view. We have proposed a Datalog-to-SQL translator [11] as a user interface to a SQL-based DBMS. This translator uses a total of eight templates to convert Horn clauses and axioms involving negation into corresponding SQL views.

Let us suppose that the extensional data base consists of relations of the type

```
R1 (K1, A1, A2, A3, ...)
```

where K1 is the primary key of R1. At this point, we limit our scope to data bases in which all implied relationships are binary. Clearly, the relation R1 is equivalent to the following set of predicates:

```
K1 (K1)
A1 (K1, A1)
A2 (K1, A2)
A3 (K1, A3)
.........
```

Such predicates are all of the form U(C) or W(D, E). Also, we assume that the IDB contains some previously defined unary and binary relations, conforming to the same forms, U (C), and W (D, E). Consider, for example, the relation

```
PEOPLE (NAME, FATHER, MOTHER, SEX).
```

This can be expressed as the set of predicates

```
NAME (NAME)
FATHER (NAME, FATHER)
MOTHER (NAME, MOTHER)
SEX (NAME, SEX).
```

Additionally, we may have implemented some IDB relations such as

```
PARENT (NAME, PARENTNAME)
BROTHER (NAME, BROTHERNAME).
```

In a Horn clause involving unary and binary predicates, we have to deal with nonground formulas of the types U(x), W(x, y), W(x, y rho cy), and x rho y, where cy is an admissible constant value for the variable y, and rho is a comparison operator. The translation templates corresponding to these cases are given in Figure 7.

```
TEMPLATE 1:  V(x) :- U(x).

    CREATE VIEW V (X, Y)
    AS SELECT X.K1
        FROM   R1 X, U
        WHERE  U.C = X.K1

TEMPLATE 2:  V(x, y) :- W(x, y).

    CREATE VIEW V (X, Y)
    AS SELECT X.K1, Y.K2
        FROM   R1 X, R2 Y, W
        WHERE  W.D = X.K1
        AND    W.E = Y.K2

TEMPLATE 3:  V(x) :- W(x, y rho cy).

where cy is a constant value for the attribute W.E.

    CREATE VIEW V (X)
    AS SELECT X.K1
        FROM   R1 X, R2 Y, W
        WHERE  W.D = X.K1
        AND    W.E = Y.K2
        AND    Y.K2 rho cy

TEMPLATE 4:  V(x, y) :- x rho y.

    CREATE VIEW V (X, Y)
    AS SELECT X.K1, Y.K2
        FROM   R1 X, R2 Y
        WHERE  X.K1 rho Y.K2
```

Figure 7. Translation Templates for Axioms Without Negation

In deductive axioms with negated unary or binary nonground formulas, there are four distinct cases:

1. V(x) :- ¬U(x).
2. V(x) :- ¬W(x, y).
3. V(y) :- ¬W(x, y).
4. V(x, y) :- ¬W(x, y).

In translating each to a SQL view, the negation should be interpreted correctly, as discussed previously in this paper. In Figure 8, the proper interpretation for negation in each case is given. In all cases, negation is reflected in the corresponding SQL view as set difference (the SQL clause NOT EXISTS).

TEMPLATE 5: V(x) :- ¬U(x).

Interpreted as: V(x) :- K1(x) & ¬U(x).

```
CREATE VIEW V(Y)
AS SELECT X.K1
   FROM   R1 X
   WHERE  NOT EXISTS (SELECT *
                      FROM   U
                      WHERE  U.C = X.K1)
```

TEMPLATE 6: V(x) :- ¬W(x, y).

Interpreted as: V(x) :- K1(x) & ¬W(x, y).

```
CREATE VIEW V (X)
AS SELECT X.K1
   FROM   R1 X
   WHERE  NOT EXISTS (SELECT *
                      FROM   R2 Y, W
                      WHERE  W.D = X.K1
                      AND    W.E = Y.K2)
```

TEMPLATE 7: V(y) :- ¬W(x, y).

Interpreted as: V(y) :- K2(y) & ¬W(x, y).

```
CREATE VIEW V (Y)
AS SELECT Y.K2
   FROM   R2 Y
   WHERE  NOT EXISTS (SELECT *
                      FROM   R1 X, W
                      WHERE  W.D = X.K1
                      AND    W.E = Y.K2)
```

TEMPLATE 8: V(x, y) :- ¬W(x, y).

Interpreted as: V(x, y) :- K1(x) & K2(y) & ¬W(x, y).

```
CREATE VIEW V (X, Y)
AS SELECT X.K1, Y.K2
   FROM   R1 X, R2 Y
   WHERE  NOT EXISTS (SELECT *
                      FROM   W
                      WHERE  W.D = X.K1
                      AND    W.E = Y.K2)
```

Figure 8. Translation Templates for Axioms Involving Negation

Algorithms based on the eight translation templates of Figures 7 and 8 have been designed to translate deductive axioms to SQL views. All SQL views in Figures 5 and 6 have been obtained by applying these algorithms to Datalog axioms. In some cases, the translation algorithms produce SQL view definitions with extraneous relations. Such views can be further simplified to improve the efficiency of deductive queries based on them. We have designed SQL view simplification algorithms to simplify the outcome of view translation algorithms [11]. Currently, the translator is being implemented as a user interface. The scope of the translator will be enhanced to include IDB relations of degree higher than two.

7. CONCLUSIONS

In this paper, we investigated deductive capabilities of traditional relational DBMSs as homogeneous logic-based systems. We proposed to use view definition languages of relational systems such as SQL as intensional data base definition languages for deductive axioms. We concluded that deductive axioms that are Horn clauses can be formulated as SQL views without difficulty.

Next, we investigated deductive axioms involving negated nonground formulas. We demonstrated that provided the set difference operator is used instead of complementation for negated formulas, such axioms can also be implemented as SQL views. Clearly, from a data base user's point of view, relational DBMSs with view processing facilities can be regarded as reasonably powerful deductive systems. Presently, the main weakness in this approach is with respect to their inability to compute transitive closure of relations, and hence implement recursive axioms. Traditional relational DBMSs should, and can, be enhanced to process recursive axioms. Also, efficient view processing is all the more important in deductive systems based on relational DBMSs. Better view processing algorithms are needed. This way, the performance of relational DBMSs that are intended for use as deductive systems can be improved substantially.

Finally, we outlined the components of a Datalog-to-SQL translator, designed as a user interface for SQL-based DBMSs. Datalog, a logic programming language for data base systems, is suitable for formulating deductive axioms. The proposed translator transforms Datalog axioms to equivalent SQL views using a minimal set of translation templates. From a practical point of view, such a translator would be helpful for data base users who are well-versed in Datalog (or Prolog) but not in SQL. In its current version, the scope of the translator is limited to binary predicates, and in addition to translation algorithms, it includes SQL view simplification algorithms that change the output of the translation algorithms to more efficient SQL views.

8. REFERENCES

[1] S. A. Naqvi and S. Tsur, A Logic Language for Data and Knowledge Bases, Computer Science Press, Rockville, Maryland, 1988.

[2] J. D. Ullman, Principles of Database and Knowledge-Base Systems, Vol. I, Computer Science Press, Rockville, Maryland, 1988.

[3] M. Missikoff and G. Wiederhold, "Toward a Unified Approach for Expert and Database Systems", Proc. of the Int. Conf. on Expert Database Systems, 1984.

[4] L. Naish and J. Thom, "The MU-PROLOG Deductive Database", Technical Report 83/10, Department of Computer Science, University of Melbourne, 1983.

[5] J. D. Ullman, Principles of Database and Knowledge-Base Systems, Volume II, Computer Science Press, Rockville, Maryland, 1989.

[6] M. Stonebraker and L. A. Rowe, "The Design of Postgres", Proc. of the ACM SIGMOD International Conference on Management of Data, 1986.

[7] Y. Uckan, "Knowledge Representation Using Views in Relational Deductive Data Bases", submitted for publication to The Journal of Systems and Software.

[8] G. Gardarin and P. Valduriez, Relational Databases and Knowledge Bases, Addison Wesley, Reading, MA, 1989.

[9] R. Reiter, "On Closed World Data Bases", Readings in Artificial Intelligence & Databases, (J. Mylopoulos and M. L. Brodie, eds.), Morgan Kaufmann Publishers, San Mateo, California, 1989.

[10] J. Minker, "On Indefinite Databases and the Closed World Assumption", Proc. Sixth Conf. on Automated Deduction, (D. Loveland, ed.), Springer-Verlag, New York, 1982.

[11] Y. Uckan, "A Datalog to SQL Translator", under preparation.

The enumeration of $(r+2)-$paths in a Boolean Cube

Shahram Latifi
Electrical and Computer Engineering Department
University of Nevada, Las Vegas
Las Vegas, NV 89154
(702)597-4183
latifi@unlv.edu

ABSTRACT

The n-dimensional Boolean Cube (n-cube) is an attractive network for a large class of parallel processing applications. In an n-cube, between two nodes of Hamming distance r, there exist r disjoint paths of length r and $(n-r)$ disjoint paths of length $(r+2)$. In this paper, we enumerate the total number of paths (disjoint or non-disjoint) of length $(r+2)$ between two nodes of Hamming distance r contained in an n-cube. The result can be used in two-terminal reliability studies of the n-cube while the derivation of the result gives a better understanding of the topological properties of this network.

Key Words- Disjoint Paths, Hamming distance, n-cube.

1. INTRODUCTION

Hypercube multiprocessors have been the focus of many researchers over the past few years. The appealing properties of the hypercube [1] such as vertex and edge symmetry, logarithmic diameter, high fault resilience, scalability, and the ability to host popular interconnection networks have made this topology an excellent candidate for many parallel processing applications. There is a rich literature dealing with the structure of the n-cube [2]. In particular, the survey paper by Harary [3] includes important graph theoretical results pertaining to the n-cube.

The determination of the number of paths between two nodes is important in the two-terminal reliability analyses of communication networks [4]. The number of disjoint paths between two nodes in an n-cube has already been determined [1]. Here, we focus our attention on enumeration of non-disjoint paths between two nodes. This quantification, in addition to being of theoretical interest, may help finding a solution for the two-terminal reliability in the n-cube, which is an open problem thus far.

In particular, if the probability of having at least one operational path no longer than a specified value, say $r+2$, is of interest (in fact this is a conditional two-terminal reliability [5]), then one has to take into account all the r and $(r+2)$ paths in the network and, using one of the methods for two-terminal reliability evaluation [4], derive the answer. The rest of the paper is organized as follows. Section 2 includes the preliminaries. In Section 3 the enumeration of various $(r+2)$-paths is performed. An example is also included in Section 3 to show the enumeration of paths. Section 4 concludes the paper.

2. PRELIMINARIES

An $n-$dimensional hypercube (i.e. Q_n) can be modeled as a graph $G(V,E)$, with $|V| = N = 2^n$, and $|E| = n2^{n-1}$. The graph $G(V,E)$ is both node and link symmetric. Each node in $G(V,E)$ represents a processor and each edge represents a link between a pair of processors. Nodes are assigned binary numbers from 0 to $2^n - 1$ such that addresses of any two neighbors differ only in one bit position. Links are also labeled from 0 to $n-1$ such that link i connects two nodes whose addresses differ in the ith bit, the rightmost bit position being 0. Figure 1 illustrates the n-cube ($n \leq 3$) with the labeled nodes and links. The *Hamming distance* between two nodes s and t, $H(s,t)$, is the number of links between them along the shortest path. The Hamming distance between s and t is also the number of bits in which addresses of s and t differ. In the text, we shall assume $H(s,t) = r$. The distance function $H(s,t)$ satisfies the following three properties [6]:

$$(a)\ H(s,t) = 0; \qquad \text{iff s = t} \qquad (1)$$
$$(b)\ H(s,t) = H(t,s) > 0; \qquad \text{iff s} \neq \text{t} \qquad (2)$$
$$(c)\ H(s,t) + H(t,w) \geq H(s,w); \quad \text{triangle inequality} (3)$$

A path originating from a node s can be uniquely specified by the labels of links composing it in the order of traversal [4]. For instance, if node s has the address 0000, the path 1-3-0 originated from s, has its end-node at 1011, and passes through intermediary nodes labeled as 0010 and 1010. The *length* of a path is the number of links it contains. A path of length r is referred to as an r-path. Two or more paths are *node-disjoint* if, except for the source and destination, they do not have any node in common; otherwise, the path is called non-disjoint.

THE ENUMERATION OF PATHS BETWEEN TWO NODES

Consider two nodes s and t in the n-cube such that $H(s,t) = r$. The following lemma establishes the number of node-disjoint paths between s and t.

Lemma 1: Between s and t there are r disjoint r-paths and $(n-r)$ disjoint $(r+2)$-paths for $0 < r \leq n$ [1].

On the other hand, if the paths are not restricted to disjoint paths, the following lemma holds true.

Lemma 2: The total number of r-paths between s and t is given by $r!$ [1].

Definition 1. Two paths are called *equivalent* if they have the same destination assuming they originate from the same source.

From the above definition, the following corollary results.

Corollary 1: Two paths equivalent to a unique path, are equivalent themselves.

Lemma 3: If a path contains some links with identical labels, it is equivalent to a path obtained by eliminating pairs of identical link labels.

Proof: A pair of identical link labels in a path imply complementing the bit position corresponding to the identical label twice. Therefore, any two identical labels cancel out, and the result would be an equivalent path.□

For instance, the path $\{0-2-0-1-0-2\}$ is equivalent to the path $\{0-1\}$.

Definition 2. A path is *valid* if its address does not include two adjacent identical labels; otherwise it is *invalid.*

As an example, the path $\{0-0-1-2-3\}$ is an invalid path since traversing link 0 twice consecutively corresponds to a cycle which is not to be included in a path.

Let the n-tuples $a(s)$ and $a(t)$ be the addresses of two nodes s and t, respectively. Furthermore, suppose $a(s)$ and $a(t)$ differ in r bit positions, namely $b_0, b_1, .., b_{r-1}$, where $0 \leq b_0 < b_1 < .. < b_{r-1} < n$. An $(r+2)$-path between s and t must traverse every link whose label is in the set $S = \{b_0, b_1, .., b_{r-1}\}$ in addition to two extra links with identical labels (so that they cancel out). Denote the identical label by b_a such that $0 \leq b_a < n$. Depending on b_a we distinguish two classes of $(r+2)$-paths, namely A and B. If $b_a \in S$, the paths are in class A; otherwise, the paths are in class B. For instance, suppose $a(s) = 0100$ and $a(t) = 1010$ in a 4-cube. Hence, $r = 3$ and $S = \{1,2,3\}$. The 5-path $\{0-1-2-3-0\}$ is in class A whereas the 5-path $\{2-3-2-1-2\}$ belongs to class B. Note that these two paths are equivalent by corollary 1.

Next, we shall obtain the number of all $(r+2)$-paths (disjoint and non-disjoint) between s and t. It will be assumed that s and t belong to an n-cube, $H(s,t) = r$, and $r \geq 2$.

The Enumeration of Class A:

A typical $(r+2)$-path in this class contains $(r+2)$ links with r links having labels corresponding to r distinct bit positions in which s and t differ and two extra links with the same labels chosen from any one of the $(n-r)$ remaining bit positions. To enumerate the paths in class A, consider a path with $(r+2)$ empty (or unassigned) positions and start assigning link labels to these empty positions such that the resulting path will be a valid path in class A. Note that there are $\binom{r+2}{2}$ ways to place two identical labels in $(r+2)$ bit positions. However, some of the paths such constructed are not valid. As mentioned previously, the $(r+2)$-paths which contain the same labels in two consecutive positions are invalid. The number of invalid paths corresponds to instances where two consecutive positions among $(r+2)$ positions have the same label. For each b_a, there are $(r+1)$ invalid paths (i.e. the pattern $b_a - b_a$ can be in positions 0 & 1, 1 & 2, .. , or r&$r+1$). Therefore, there are $\left[\binom{r+2}{2} - (r+1)\right]$ ways to place a particular label in two consecutive positions. But, the repeated label b_a can be any one of $(n-r)$ bit positions which does not belong to S. On the other hand, to the remaining r bit positions, the r labels of S can be assigned in $r!$ number of ways. Therefore, the number of $(r+2)$-paths belonging to class A is:

$$\begin{aligned} N_A &= \left[\binom{r+2}{2} - (r+1)\right] \times (n-r) \times r! \\ &= \frac{r(r+1)!(n-r)}{2} \end{aligned} \tag{4}$$

Note that when $n = r$, $N_A = 0$; i.e. the class A is empty for nodes located at opposite corners of the $n-$cube.

The Enumeration of Class B:

As mentioned earlier, for the paths belonging to class B, $b_a \in S$. Therefore, b_a must appear in the path sequence three times. However, there are invalid paths here too. The invalid paths are those in which the same label appears in two or three consecutive positions. Examples of such unacceptable paths are : $0-0-0-1-2$ or $0-0-1-3-0$. To determine the number of invalid paths in class B, consider any one element, say y, of the r distinct elements in S and suppose y is to repeat three times. There are $\binom{r+2}{3}$ ways to place 3 y's in $(r+2)$ bit positions. The number of invalid paths can be determined by partitioning them into three groups.

Group 1- There are 3 consecutive y's (e.g. the path $..yyy..$). To construct such invalid paths, one can place 3 y's in the left-most 3 positions and every time shift the pattern to the right by one bit position. Thus there are r invalid paths in group 1.

Group 2- There are 2 consecutive y's in the leftmost (e.x. $yy..$) or in the right-most positions (e.x. $..yy$). In each case the 3rd y can sit in one of the $r-1$ positions(recall that there cannot be three consecutive y's in this group). Clearly, the total number of invalid paths in group 2 is $2(r-1)$.

Group 3- There are 2 consecutive y's in the middle (neither in the left nor in the rightmost: e.x. $..yy..$). suppose the first y is placed in the jth bit position where $2 \leq j \leq r$. Such paths can be obtained by moving the 3rd y to one of the remaining $r-2$ positions resulting in $(r-2)$ paths for each j and in $(r-2)\times(r-1)$ paths overall.

Therefore, the number of valid paths are:

$$\binom{r+2}{3} - (g_1 + g_2 + g_3) \qquad (5)$$

where g_i is the number of paths in group i. Substituting for g_i's in (5) we get:

$$\binom{r+2}{3} - (r^2) \qquad (6)$$

However, the remaining $(r-1)$ labels can be permuted $(r-1)!$ times; moreover, y can be any one of r labels. Thus:

$$\begin{aligned} N_B &= r \times (r-1)! \left[\binom{r+2}{3} - (r^2) \right] \\ &= r! \left[\binom{r+2}{3} - (r^2) \right] \\ &= r(r!) \left(\frac{r^2 - 3r + 2}{6} \right) \qquad (7) \end{aligned}$$

The number of $(r+2)$-paths can now be obtained by adding N_A and N_B.

$$N_{r+2} = N_A + N_B = \frac{r(r!)(r^2-3r+2) + 3r(r+1)!(n-r)}{6} \qquad (8)$$

Observe that when $r = 2$, $N_B = 0$. This is true since if there is a path like $0-1$, there is no way to have a valid path of length $(r+2) = 4$ which includes either 0 or 1 three times. The $(r+2)$-path enumeration is illustrated in the following example.

Example- Find the number of $(r+2)$-paths between nodes labeled as $a(s) = 00000$ and $a(t) = 00111$ in a 5-cube.

Solution- The labels of s and t differ in bit positions 0, 1, and 2. Clearly, $r = 3$. For brevity, we show all $r! = 3! = 6$ permutations of the set $\{0,1,2\}$ by three X's in the corresponding places. For instance, the $(3X3XX)$ will represent the set of paths:

$$\{3-0-3-1-2\},$$
$$\{3-0-3-2-1\},$$
$$\{3-1-3-0-2\},$$
$$\{3-1-3-2-0\},$$
$$\{3-2-3-0-1\},$$
$$\{3-2-3-1-0\}.$$

According to equation 8, for $r = 3$, there are 78 5-paths as follows.

The paths in class A are:

$3-X-3-X-X$	$4-X-4-X-X$
$3-X-X-3-X$	$4-X-X-4-X$
$3-X-X-X-3$	$4-X-X-X-4$
$X-3-X-3-X$	$X-4-X-4-X$
$X-3-X-X-3$	$X-4-X-X-4$
$X-X-3-X-3$	$X-X-4-X-4$

The paths in class B are:

0-1-0-2-0
0-2-0-0-1
1-2-1-0-1
1-0-1-2-1
2-0-2-1-2
2-1-2-0-2

Note that $N_A = 72$ and $N_B = 6$.

CONCLUSION

The enumeration of all the paths between two nodes in a network is important for terminal-reliability analysis. The paper has considered the enumeration of $(r+2)$-paths between two nodes in a hypercube. Two different classes of $(r+2)$-paths, namely A and B, were distinguished. For each class, the number of paths was determined. It was shown that N_A, the number of paths in class A, is much more than N_B, the number of paths in class B. Moreover, N_A is linearly proportional to n; whereas N_B is independent of n. The future direction of this work may include the characterization of $(r+2)$-paths, using the properties of hypercube topology such as symmetry, in order to obtain the two-terminal reliability of this network when the length of the paths are restricted to $(r+2)$. Note that limiting the length of paths to a specific value is a legitimate restriction since it puts an upper bound on the communication delay between two nodes.

REFERENCES

[1] Y. Saad and M.H. Schultz, "Topological properties of hypercubes," *IEEE Transactions on Computers*, vol. 37, no. 7, pp. 867-872, July 1982.

[2] S. Latifi, "Hypercube-based topologies with incremental link redundancy," *Ph.D. Dissertation*, Louisiana State University, August 1989.

[3] F. Harary, "A survey of hypercube graphs," *Comput. Math. Applications*, 1989.

[4] C.J. Colbourn, "The Combinatorics of Network Reliability," New York : Oxford, Oxford University Press. 1987.

[5] S. Latifi and S. Rai, "Evaluating the distance reliability in hypercube multiprocessors," submitted to the *Internation Phoenix Conference on Computers and Communications.*

[6] L.N. Bhuyan and D.P. Agrawal, "Generalized Hypercube and Hyperbus Structures for a Computer Network," *IIEEE Transactions on Computers*, vol. C-33, no. 4, pp. 323-333, April 1984.

Planning Monotone Watchman Paths

Laxmi P Gewali
Department of Computer Science
University of Nevada, Las Vegas
Las Vegas, NEVADA 89154

Abstract: Computing an optimum watchman route in the presence of two dimensional obstacles is known to be NP-hard [3]. We introduce a variation of this problem called *The Monotone Watchman Path Problem (MWP)* that asks for a monotone collision-free path connecting two given points amidst a collection of disjoint simple obstacles so that the number of obstacles visible from P is maximized. We present an $O(n^3 logn)$ algorithm to plan such a path.

I. Introduction

Planning collision-free paths in the presence of obstacles is a central problem in robotics and computational geometry [8]. Given a collection of obstacles, a start point s, and a target point t the **find path** problem in robotics asks for a collision-free path connecting s and t. Usually obstacles are modeled by polygon/polyhedra and the object to be moved is considered to be a point. The quality of a collision-free path is often measured by its length: "the shorter the length better the path". The two dimensional version of this problem is rather well studied and several polynomial time algorithms are reported. Voronoi diagrams [10] and visibility graphs are popular tools used for planning collision-free paths in 2-d. The problem is significantly harder in three dimensions. Finding a shortest collision free path in 3-d is shown to be NP-hard in general [5].

Many interesting problems arise when visibility properties ate considered for planning paths. In [3], one such problem called the **watchman route problem** is introduced. A watchman route is such that each point on the scene is visible to some point along the route. An optimum watchman route is a watchman route of minimum length. It has been proven that finding an optimum watchman routes inside a polygon with holes in NP-hard [3]. An linear time algorithm for computing an optimum watchman route inside a simple triangulated orthogonal polygon is reported in [3]. An $O(n^4 loglogn)$ time algorithm for finding an optimum watchman route in a simple polygon is given in [4].

In many situations it is useful to compute monotone paths. A path is said to be **monotone** along a given line l if the projections of the segments of the path along l do not overlap. Monotone paths have important applications for deciding the separability of obstacles [11,1]. An $O(nlogn)$ algorithm for computing a monotone path (if it exists) along a given direction is reported in [ACM89].

In section II we introduce a variation of the watchman route problem called the monotone watchman path problem and present an $O(n^3 logn)$ time algorithm to solve it. We conclude in section III by discussing possible extensions of this problem.

II. Algorithm for Monotone Watchman Path

Consider a two dimensional scene consisting of disjoint polygonal obstacles. We assume obstacles to be simple and are defined to be convex obstacles with bounded number of edges. Two points are said to be **visible** if the straight line segment joining them does not intersect any obstacle. An obstacle O is visible to a point p if some point in O is visible to p. A path P connecting points s and t is called a **s-t-monotone path** if it is monotone along the direction s-t.

The Monotone Watchman Path Problem (MWP): *Given a two dimensional scene consisting of disjoint simple obstacles, a start point s, and a target point t find a collision-free s-t-monotone path P such that the number of obstacles visible from P is maximized.*

Figure 1 shows an example of such a path. A problem closely related to MWP is the monotone visit problem (MVP). MVP asks for a s-t-monotone path that visits the maximum number of obstacles and an $O(n^2)$ algorithm for solving it is given in [7].

Our approach to solve MWP is similar to the technique used in [7]. Let $\{O_1, O_2, O_3, ..., O_k\}$ be the total number of obstacles in the scene. The **region of illumination** R_i corresponding to the obstacle O_i is the region illuminated when the boundary of O_i acts as light source (Figure 2). Define **illumination graph** $G(V, E)$ of obstacle scene to be the planar graph formed by the intersection of edges on the boundaries of R_i's and obstacles together with points s and t (Figure 3). From each vertices of $G(V, E)$ extend vertical line segments in both direction till they meet some obstacle boundary. The resulting graph $G'(V', E')$ is called the **vertical adjacency graph (VAG)** of $G(V, E)$ (Figure 4). Observe that $G'(V', E')$ is trapezoidization of the obstacle scene. A **directed acyclic graph (DAG)** is obtained from $G'(V', E')$ by replacing each vertical edge by two edges, as shown in Figure 5, and assigning left to right direction to all edges. Two s-t-monotone paths are **equivalent** if they see the same set of obstacles. Now observe that equivalent s-t-monotone paths pass through the same sequence of trapezoids.

Lemma 1: [7] Corresponding to each s-t-monotone path there is an equivalent s-t-monotone path consisting of edges in DAG.

Proof: Let $T_{i_1}, T_{i_2}, ..., T_{i_k}$ be the sequence of trapezoids througł which a given s-t-monotone path goes. Replace the portion of path passing through each trapezoid by an equivalent sub-path consisting of left-edge, bottom-edge (or top-edge), and right-edge of corresponding trapezoids one by one. All such edges are in DAG.

Q.E.D.

In order to search for a s-t-monotone path from which maximum number of obstacles are visible we find a longest s-t-monotone path in *DAG*. Here the length of the path is measured by the number of edges on the path. Algorithm for computing longest path in a directed acyclic graph is similar to topological sort [2] and can be done in $O(n)$ time. However the longest s-t-monotone path in *DAG* may not correspond to the one that sees the maximum number of obstacles. The reason is that the path may visit the same obstacle more than once. We can fix this problem by keeping track of obstacles seen from the path during the computation of longest path. The idea is to increment the length of the path only when the newly visited vertex was not belonging to the illumination region already visited. This rule makes sure that an obstacle is counted only once even though the path may see the same obstacle more than once. A precise description of the algorithm is given in Figure 5.

Theorem 1: The monotone watchman path problem can be solved in $O(n^3 logn)$ time.

Proof: Illumination regions corresponding to one obstacle can be done in $O(n^2 logn)$ time by using maintenance of perspective view algorithm given in [6]. Thus step 1 takes $O(n^3 logn)$. Step 2, step 3, and step 4 can be done in $O(nlogn)$ time by using sweep line techniques. Step 5 can be done by using breath first search in $O(n)$ time (since graph is planar). Step 6 to step 9 is the longest path algorithm for an acyclic directed graph and can be done in $O(n)$ time provided the function *on_the_path* in line 9 can be done in constant time. But *on_the_path* can take $O(n)$ time (we may have to check the path all the way back to s). Thus time of step 1 dominates time for all other steps.

Q.E.D.

III. Conclusions

We presented an $O(n^3 logn)$ time algorithm for computing s-t-monotone watchman path from which maximum number of obstacles can be seen. It is very likely that one may be able to improve on the time complexity of the algorithm by finding better ways of computing regions of illumination. Although our algorithm is for simple obstacle it can be easily extended to include the case of general 2-d obstacles in $O(n^4 logn)$ time. Our algorithm can have important applications for detecting the separability of objects [11] under visibility properties.

References

[1] Arkin, E. M., "R. Connelly and J. S. B. Mitchell, "On Monotone Paths Among Obstacles, With Applications to Planning Assemblies", Proc. 5th Annual Symposium on Computational Geometry, 1989.

[2] Aho, A. V., J. E. Hopcroft and J. D. Ullman, "The Design and Analysis of Computer Algorithms", Addison Wesley Publishing Company, 1974

[3] Chin, Y. P. and S. Ntafos, "Optimum Watchman Routes", Information Processing Letters 28(1988) 39-44.

[4] Chin, Y. P. and S. Ntafos, "Watchman Routes in Simple Polygons", Discrete and Computational Geometry, To Appear.

[5] Canny, J. and J. Reif, "New Lower Bound Techniques for Robot Motion Planning Problems", Proc. of 28th FOCS, pp. 46-60, Oct 1987.

[6 Edelsbruner, H., M. Overmars, and D. Wood, "Graphics in Flatland: A case study in: Advances in Computing Research 1, JAI, Greenwich, 1983, CT, pp. 35-59.

[7] Gewali, L., "Planning Monotone Paths to Visit a Set of Obstacles", Technical Report No. CRS-90-36, Department of Computer Science, University of Nevada, Las Vegas, 1990.

[8] Schwartz,J., J. Hopcroft and M. Sharir, "Planning, Geometry and Complexity of Robot Motion Planning", Allex Publishing Company, New Jersey, 1987.

[9] Sharir, M.and A. Schorr, "On Shortest Path in Polyhedral Spaces", Siam Journal of Computing, Vol. 15, No. 1, pp. 193-215, Feb. 1986.

[10] Schwartz, J. and C. K. Yap, "Algorithmic and Geometric Aspects of Robotics", Lawrence Erlbaum Associates, 1987.

[11] Toussaint, G. F., "Movable Separability of Sets", Computational Geometry, Ed. G. T. Toussaint, North Holland

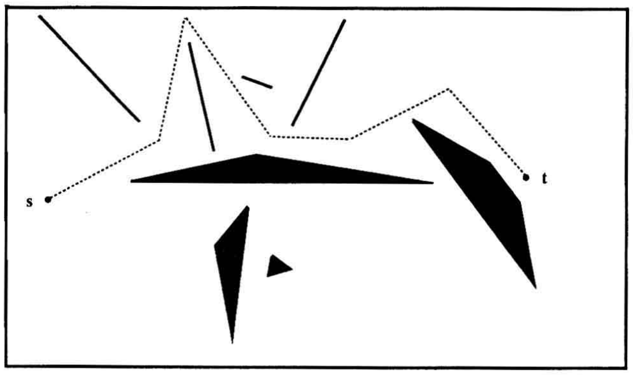


Figure 1: Illustrating an example of s-t-monotone path from which maximum number of obstacles can be seen.

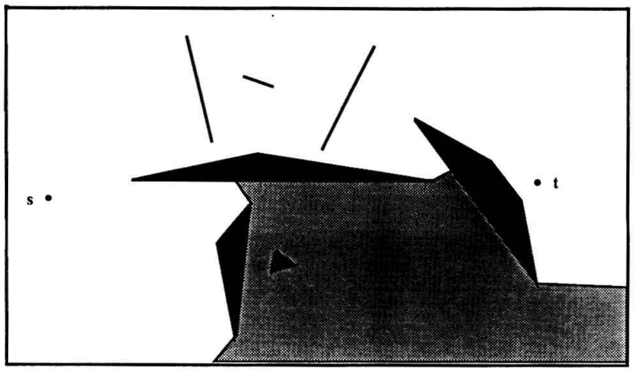


Figure 2: Showing the region of illumination (gray area) of an object.

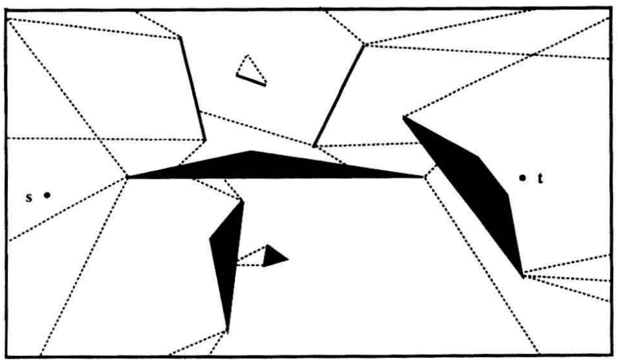


Figure 3: Illumination graph of obstacle scene.

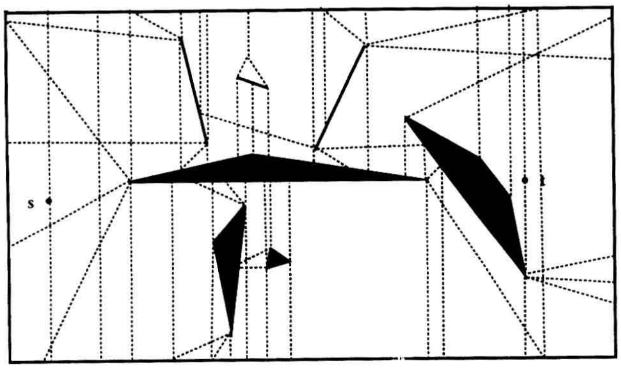
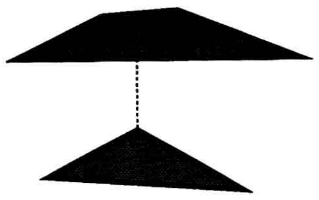


Figure 4: Vertical adjacency graph (VAG) of the obstacle scene.



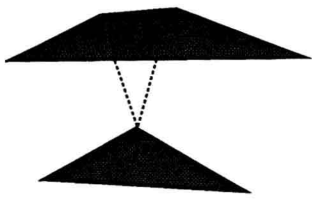


Figure 5: Illustrating the splitting process.

Algorithm Monotone_Watchman
LongDist.v_t is the distance of the longest path from s to v_t. PrevNode.v_t is the node before v_t in the longest path from s to v_t

```
1. Construct the region of illumination for all obstacles.
2. Compute the planar graph G(V,E) formed by the intersection of the edges on the boundaries of obstacles and their illumination regions.
3. Compute the vertical adjacency graph (VAG) from G(V,E).
4. Convert VAG to directed acyclic graph (DAG).
5. create stack; LongDist.$v_0$:=0; PrevNode.$s$ := $s$;
   initialize in-degree for all vertices; push $s$ on stack;
6. while stack not empty do
7.    $v_t$ := pop(stack);
8.    for each edge $(v_t, v_j)$ do
9.      if on_the_path($v_j$) then Temp := LongDist.$v_t$
10.      else Temp := LongDist.$v_t$ + 1
11.     if LongDist.$v_j$ < Temp then
12.        PrevNode.$v_j$ := $v_t$
13.        LongDist.$v_j$ := Temp
14.     in-degree.$v_j$ := in-degree.$v_j$ - 1
15.     if in-degree.$v_j$ = 0 then push $v_j$
16.   end{for}
18. end{while}
```

function on_the_path(v_j);
{this function returns 'true' if the longest path from s to v_j contains at least two vertices belonging to the obstacle corresponding to vj}

```
Let Q be the obstacle corresponding to vj;
on_the_path := false; v := PrevNode.vj;
while v $\neq$ s and ( not on_the_path) do
      begin
         v := PrevNode(v);
         if v belongs to Q then On_the_path := true
      end.
```

Figure 6: Algorithm for computing s-t-monotone path from which maximum number of obstacles can be seen.

A New Raster to Vector Algorithm

Chu, Y. P., Tasi, M. H. and Yu, C. C.*

Institute of Applied Mathematics
National Chung Hsing University
*Shu-Teh Junior College of Technology
Taichung, Taiwan, R.O.C.

ABSTRACT

In engineering drawing, only some lines, curves or symbols (which can also be regarded as lines or curves) are of essential meanings to the users. Using raster files to edit or store images is both space-consuming and difficult. If, however, some of these lines within raster images are transformed into vectors for storage, we can not only save a lot of space, but also make the jobs of edition, enlaragement, shrinkage and even resolution much easier.

An efficient and effective new method of algorithm will be presented here. First, a basic vectorizing concept is used to devise a new piecewise linear approximation of digital curves in 2-dimensional space, which will in turn be applied to raster-to-vector algorithm.

I. INTRODUCTION

Generally speaking, there are two types of data images: one is gray level image, and the other is bilevel image. Algorithm varies with the data types. Because of its gray value, the image in gray level image is usually used as a basis for trace. In using such method [1] [2], the input device must be equipped with over 256 gray levels [3] to achieve better effect, otherwise, line breakage will occur easily. If the gray level of a camera is not high enough, the technique of segmentation can be used to change the gray level image into bilevel image. Scanner can also be used as input device to obtain bilevel image directly.

There are two typical approaches to vectorizations: one based on Runlength code, the other based on thinning. Basically, the former uses Runlength code to show digital image, and connects each individual run before changing them into vectors. The process can be divided into five steps: (1) scanning and thresholding, (2) noise removal and void filling, (3) edge detection, (4) outline following, (5) vectorizing [4].

The advantages of adopting Runlength code to trace vectors can be summed up as follows: Runlength code can retain the width of line, and because of its faster processing speed, it is suitable for real time system, and does not take up too many memory [5] during the course of process. It has its drawbacks, however, such as unsuitable for handling curves, cross-lines (because the lengths of cross point Run are quite uneven), and slight slanting line. These disadvantages have been mollified.

The thinning process allows users to obtain skeleton of the original image as the object for processing. The thinning process can be divided into four steps: (1) thinning, (2) classification, (3) chainning, (4) vectorizing. Thinning has the advantages of doing as litttle damage as possible to the original shape of the geometric diagram, while the width of each line is kept to just one pixel wide. The image after going through thinning remains a bilevel image except that its number of black pixels (which gray value are equal to 1) has been reduced to the minimum number necessitated to show the lines of the original image. In other words, there remains a lot of excessive information (the background of image) within the file. Therefore, we hope to be able to remove the background, and to show the lines by chain code. Black pixels are usually classified before processing. Uutil the classification is done, the whole image looks indeed like some geometric lines. To the image itself, however, it consists of only black pixels and white pixels. The purpose of classification is to assign roles to the black pixels within the image. What follows is to chain up the pixels into curves according to the role of each pixel. Finally, curve approximation is conducted.

II. IMAGE TRANSFORMATION FROM RASTER TO VECTOR

We will now explore and implement the image transformation method. The entire handling process can be divided into three major stages as shown in figure 1. The sections that follow will discuss in detail the method of each stage.

(I) Thinning

An effective thinning algorithm, i.e., Chen-Hsu method [6], is adopted. A binary image is generally defined as a square matrix IM, where each element IM(i,j) is called a pixel which has the value 1 or 0. 1 represents the black pixels which is the lines. Another 3×3 square matrix is then defined as a 8-neighbors matrix with central pixel P1 and is shown in figure 2. In this method, each iteration is divided into two subiterations. In the first subiteration, the central pixel P1 will be deleted from the binary image if it can satisfy the following conditions:

(a) $2 \leq B(P1) \leq 7$

(b) $A(P1) = 1$; if $A(P1) = 2$ then goto (d)

(c) $(P4 = 0)$ or $(P6 = 0)$ or $(P2 = P8 = 0)$; end.

(d) $(P2 = P4 = 1$ and $P6 = P7 = P8 = 0)$ or $(P4 = P6 = 1$ and $P8 = P9 = P2 = 0)$; end.

where B(P1) is the number of nonzero neighbors of pixel P1, and A(P1) is the number of 0-1 pair in the P2, P3, P4,..., P9, P2 sequence.

In figure 3, for example, B(P1) is equal to 5 and A(P1) is equal to 2. But the central pixel still cannot be deleted from the binary inage because condition (d) is not satisfied. In the second subiteration, the central pixel P1 will be deleted from the binary image if it can satisfy the following conditions:

(a) $2 \leq B(P1) \leq 7$

(b) A(P1) = 1; if A(P1) = 2 then goto (d)

(c) (P2 = 0) or (P8 = 0) or (P4 = P6 = 0); end.

(d) $(P2 = P8 = 1$ and $P4 = P5 = P6 = 0)$ or $(P6 = P8 = 1$ and $P2 = P3 = P4 = 0)$;end.

The flowchart of this method is described in figure 4.

(II) Split

Because the file captured by the scanner is too much for the computer to process completely at one time, it is necessary to split raster file first to facilitate processing. Basically, to save splitting time, equal division splitting is adopted, that is, regardless of how the lines are scattered, the file is always splitted into definite size.

The advantages of proper splitting are as follows: First, the size is suitable for program processing; Second, it is suitable for parallel processing. The advantages of equal division splitting are that it is simple and swift. The disadvantages are as follows: when distributed system or parallel processing is used to process the splitted files, the unevenly distributed lines in each splitted file may result in too much difference in the processing time of the processors. To narrow the difference, a more ingenious splitting method, that is, unequal division splitting, is adopted. This method is to split the file into several small files of unequal size to make the linear density within the splitted files more uniform. The steps involved are as follows:

(a) claculate the numver of black pixels in each row of the original file, and keep the score.
(b) Total the number of black pixels in the entire image, which is then divided by the number of processors n to arrive at a reference value.
(c) Split each row into n small files to make the number of black pixels in each small file to come closest to the reference value.

Because it is necessary to analyze the image first, unequal division splitting is more time-consuming than equal division splitting. Under the circumstances of distributed system or parallel processing, and if the lines in an image scatter unevenly, unequal division splitting is prefered, as this method tends to lessen the load of the processors and to save time. But if the lines in an image distribute more evenly, it is more suitable to use equal division splitting to save the time on analyzing.

(III) Vectorizing

This section will first introduce a new method, and then pseudocode will be used to describe the calculating method. Some definitions of terms are as follows: "m" is the array for storing image, and each element of array represents a pixel. "L" is the maximum length of the line section, whose function is to prevent falling into an infinite loop during the course of processing. "S" is the starting point. "C" is the current point. "N" is the next point. "P" is the plus point, recording the point coordinate which is in the positive direction and farthest from line 1. "M" is the minus point, recording the point coordinate which is in the negitave direction and farthest from line 1.

Besides, if a certain point has three or more than three adjacent points around it, it is defined as cross point. If it has two adjacent point, it is defined as center point. If it has only one adjacent point, it is defined as endpoint. If it has no adjacent point, it is defined as isolated point.

Next, a new piecewise linear approximation of digital curves in 2-dimensional space will be discussed. Assuming $\triangle SXN$ as figure 5.

Let $\overrightarrow{XS} = \vec{a}, \overrightarrow{XN} = \vec{b}$.

The area of $\triangle SXN = \frac{1}{2}\sqrt{|\vec{a}|^2|\vec{b}|^2 - (\vec{a}\cdot\vec{b})^2} = \frac{1}{2}hd$.

Further let $\vec{a} = (a_1, a_2)$, $\vec{b} = (b_1, b_2)$,

then we obtain $h = \dfrac{\begin{pmatrix} a_1 & a_2 \\ b_1 & b_2 \end{pmatrix}}{d}$.

Here, h is not the absolute value, so h can be plus or minus. We then use this simple mathematical equation to calculate the distance between point X and line section $\overline{SN}$. First, let the initial value "L" be 1, beginning from m in the upper left corner of the figure, and follow the colum major to locate the first nonzero points. Following the direction defined in figure 6, beginning with 0 in clockwise direction $(0, 1, \cdots, 7)$, locate the first nonzero point within the 3×3 adjacent matrix with S as center, let this point be C. Increment L by 1, record C relative to the direction value of S (for example, if C is in the direction 3 of S, then the direction value is 3), and calculate the number of the asjacent points of S to judge the pattern of S.

Let both P and M be equal to C. Because S and C do not form a triangle, C is taken as the center of 3×3 adjacent matrix. Beginning with the upper direction value and follow clockwise direction $(3, 4, \cdots, 2)$ to locate the first nonzero point. Let this point be N. Increment L by 1, record N relative to the direction value of C, calculate the number of the adjacent points of C to judge the pattern of C. Replace C with N, and follow the aforementioned method to locate the next point N of C, increment L by 1, calculate the heights of $\overline{SN}$ within $\triangle SCN$, $\triangle SPN$, and $\triangle SMN$ as H, Hp and Hm respectively. If H is larger than 0, then it means C is in the plus direction of line section $\overline{SN}$. In comparison with Hp, if $H \geq Hp$, it means C is further deviated from $\overline{SN}$ than P. Replace C with P, if $H < Hp$, then P is not to be replaced. Similarly, if $H < 0$, it means C is in the minus direction of line section $\overline{SN}$; if $H \leq Hm$, it means C is further deviated from $\overline{SN}$ than M. Replace M with C; if $H > Hm$, then M is not to be replaced. If any of H, Hp and Hm exceeds the defined standard CR, it means the margin of errror is too large in approaching $\overline{SN}$ to curve SN, so $\overline{SC}$ is used to approxinate curve SC. If H, Hp and Hm are all smaller than CR, then follow the aforementioned sequence to locate the next point until H, Hp and Hm are larger than CR, or untill L exceeds the maximum length MAXL, or until the end of the line.

Figure 7 is a simple example. Let CR be 1. First locate P1 as S, then locate P2 as C and as P, M; follow the afomentioned sequence, when P3 is C, P4 is N, $H < 0$, and $H < Hm$, so P3 is M. When P4 is C, and P5 is N, becausse $|Hm|$ is larger than CR, $\overline{P1P4}$ is a line section. Next, P4 is S, locate P5 as C and as P, M, and locate P6 as N; when P6 is C, and P7 is N, $H > 0$ and $H = Hm$, so P6 is P, when P7 is C, and P8 is N, because $|Hp|$ is already larger than CR, $\overline{P4P7}$ is a line section. Finally, $\overline{P7P8}$ is a line section, so, in this example, curve P1P8 is approximated by the three line sections of $\overline{P1P4}$, $\overline{P4P7}$ and $\overline{P7P8}$

Figure 8 is an example of curve. CR is still 1, first locate P1 as S, then follow the aforementioned sequence to locate the first line section as $\overline{P1P2}$. Next, let P2 be S, continue the steps to obtain P3 as M, P4 as P. When P5 is C, and P6 is N, because$|Hp| > CR$ and $|H| > |CR|$, $\overline{P2P5}$ is a line section. Then let P5 be S, continue the steps to obtain P7 as M, P6 as P; when P8 is C, P9 is N, because $|Hm| > CR$ and $|H| > CR$,$\overline{P5P8}$ is a line section. Finally, $\overline{P8P10}$ is a line section, therefore, curve P1P10 is approximated by four line sections of $\overline{P1P2}$, $\overline{P2P5}$, $\overline{P5P8}$ and $\overline{P8P10}$.

The algorithm mentioned above is described in pseudo code as follows:

```
L = 0;
TOOFAR = FALSE;
FOR (all points in matrix M)
    Find S in column major sequence;
    Find next point N of S;
    P = M = N;
WHILE (($S \neq$ isolated point) and ($S \neq$ end point)
      and   (L< maxlength)  and   (not TOOFAR))
    L++;
    C = N;
    Find next point N of C;
    IF ($C \neq$ end point)
      Calculate height H of $\overline{SN}$ in $\triangle SCN$;
      Calculate height Hp of $\overline{SN}$ in $\triangle SPN$;
      Calculate height Hm of $\overline{SN}$ in $\triangle SMN$;
   IF (($H < CR$)  and  ($Hm < CR$)  and  ($Hp < CR$))
       IF (($H > 0$)  and  ($H \geq Hp$))  $P = C$;
       IF (($H < 0$)  and  ($H \leq Hm$))  $M = C$;
      ELSE
        TOOFAR = ture;
    ELSE
      TOOFAR = true;
S = C;
```

III. RESULTS

We use three images (1200 × 1200 pixels, 300 dpi) to test our algorithm. Final results are shown in figure 9, 10, 11. Table 1 shows the precessing time, vector number and saving time about these images.

IV. CONCLUSIONS

Due to the requirements in various applications, many researchers have spared no effort to direct their attentions to devise algorithms for vectotizing image. However, both Runlength method and Thinning method are time consuming, since they involve large amount of data and operations. Therefore, it's imperative to improve our existing software technique by adoptions hardware and parallel processing. For instance, the GTX 5000 system, when expanded to eight 68020 processors is suitable to handle such problems.

We have proposed here a new algorithm for vectorizing image. By utilizing the Hp and Hm techniques, the computing time will not increase with the longer line segment. The curve-approximating method presented here is also effective. In addition, from the viewpoint of distributed processing, it's advisable to balance the loads of all processors by unequal division splitting.

REFERENCES AND BIBLIOGRAPY:

1. Watson, L.T., Arvind, A., Ehrivh, R.W., and Haralick, R.M., "Extraction of lines and Regions from Grey Tone Line Drawing Images", Pattern Recognition, 17, 5, 1984, pp. 493-507.
2. Bixler J.P., and Sanford, J.P., "A Technique for Encoding Lines and Regious in Engineering Drawings", Pattern Recogition, 18, 5, 1985, pp367-377.
3. Pavlidis, T., "Algorithm For Graphics and Image Processing", 1882.
4. Ramachandran, K., "Coding Methos for Vector Representation of Engineering Drawings", Proc. IEEE, 68,7, Jul. 1980, pp. 813-817.
5. Ramer, U., "An Iterative Procedure for The Polygon Approximation of Plane Curves", CGVIP, 1, 1972, pp. 244-256.
6. Chen, Y.S., and Hsu, W.H., "An New Parallel Thinning Algorithm For Binary Image", NCS, 1985, pp. 295-299.
7. Peuquet, D.J., "An Examination of Techniques for Reformmating Digital Cartographic Data/ Part 1: The Raster-to-Vector Process", Cartographica, 18, 1, 1891, pp. 34-38.
8. Parker, J.R., "Extracting Vectors from Raster Images", Compute & Graphics, 12, 1988, pp. 75-79.
9. Sklansky J., and Gonzalez, V., "Fast Polygonal Approximation of Digitized Curves", Pattern Recognition, 12, 1980, pp. 327-331.
10. Landy, M.S., and Cohen, Y., "Vectorgraph Coding: Efficient Coding of Line Drawings", CVGIP, 30, 1985, PP. 331-344.

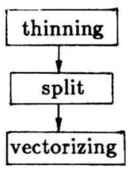


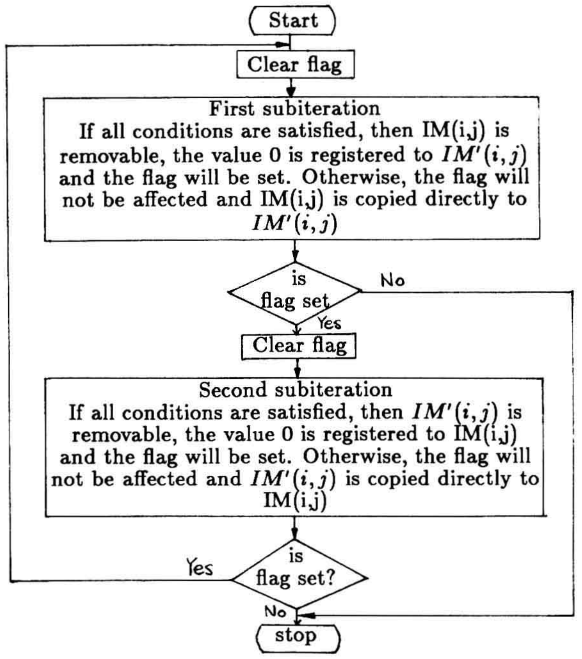
Figure 1. The handling process of image slgorithm

P9 (i-1, j-1)	P2 (i-1, j)	P3 (i-1, j+1)
P8 (i, j-1)	P1 (i, j)	P4 (i, j+1)
P7 (i+1, j-1)	P6 (i+1, j)	P5 (i+1, j+1)

Figure 2. 8-neighbors matrix with the central pixels P1

0	0	1
1	P1	1
1	1	0

Figure 3. B(P1)=5, A(P1)=2



Start
Clear flag
First subiteration
If all conditions are satisfied, then IM(i,j) is removable, the value 0 is registered to $IM'(i,j)$ and the flag will be set. Otherwise, the flag will not be affected and IM(i,j) is copied directly to $IM'(i,j)$
is flag set
No
Yes
Clear flag
Second subiteration
If all conditions are satisfied, then $IM'(i,j)$ is removable, the value 0 is registered to IM(i,j) and the flag will be set. Otherwise, the flag will not be affected and $IM'(i,j)$ is copied directly to IM(i,j)
Yes
is flag set?
No
stop

Figure 4. Flowchart of Chen-Hsu method

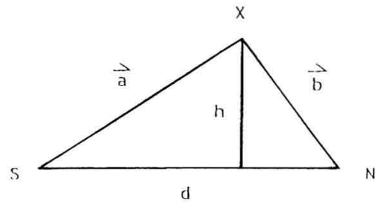


Figure 5. $\triangle SXN$

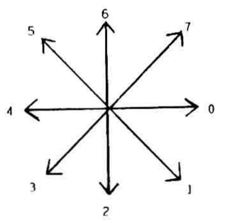


Figure 6. The definition of direction

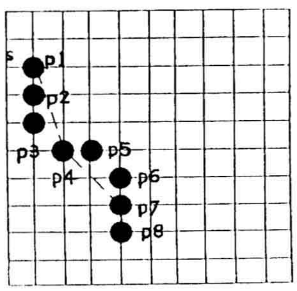


Figure 7. Example 1