Chapter 10

SEARCH METHODS REVISITED

Xiao-Shan Gao

In this chapter, we review the work of automated geometry theorem proving based on heuristic search methods and report some recent advances in this direction, especially the *geometry deductive database* which shows surprising power in terms of proving and discovering difficult geometry theorems. We also discuss several key issues in developing an efficient geometry theorem prover with heuristic search methods.

10.1 Introduction

There are mainly two approaches to automated geometry theorem proving (abbr. AGTP): the synthetic approach based on *search and heuristics* initiated by H. Gelernter (1963) in the late 1950s and the approach based on *algebraic computation* initiated by A. Tarski (1951) in the late 1930s, reshaped and popularized in the 1980s by W. T. Wu (1978). Most of the recent work on AGTP is inspired by Wu's method and focuses on using algebraic methods. Generally speaking, the algebraic approaches are very powerful decision procedures. Does this make the search methods obsolete? The answer is negative. Despite its "weakness," the methods based on search and heuristics are still worth improving, because this may lead to techniques useful to automated reasoning in the general case. Even for AGTP, improving the synthetic methods has many positive aspects:

1. Proofs produced by synthetic methods are generally easier to understand than proofs given by algebraic computations.
2. Using search methods, it is possible to generate multiple and shortest proofs for a given geometry theorem.
3. Using predicates only (no algebraic computation) makes the reaching of fixpoints (see Section 10.3.2) possible. As a result, new theorems may be discovered.
4. Although algebraic methods can prove a much greater number of theorems, there still exist theorems (Example 10.3) which can be solved by the syn-

thetic approaches elegantly but have not been solved with the algebraic approaches, because to prove them we need excessively large computer memory.

In this chapter, we will review some of the main work and techniques for the synthetic approach of AGTP.

Using the search method to prove geometry theorems goes back to as early as the late 50s in the work by Gelernter, Hanson and Loveland (1960), which is considered as a classic work in the field of AI. While several basic concepts of AGTP is introduced, the prover reported in this work seems unable to prove a large number of moderately difficult theorems. It is noted by Feigenbaum and Feldman (1963) that "The fascination with mechanical theorem proving for most of the researchers working in this area lies less with the end (the production of theorems, perhaps new and important) than with the means (a thorough understanding of the organization of information processing activity in mathematical discovery)." In our opinion, both the ends and the means are important, and the less success of Gelernter's work in proving difficult theorems has its reasons. We will discuss some of the main involving issues in Section 10.5.

Gelernter's work lead to many subsequent work on AGTP using similar approaches (Evans 1968; Gilmore 1970; Anderson 1981; Coelho and Pereira 1986; Koedinger and Anderson 1990). Although many improvements are presented, all of the work is within the framework presented by Gelernter. Nevins' theorem prover (1976) is a noticeable exception, in which a combination of forward chaining (see next section for definition) and backward chaining is used with the emphasis on the forward chaining. As a consequence of using forward chaining, numerical diagrams are not used, because each derived fact in the forward chaining is valid.

In (Chou, Gao and Zhang 1998), a prover is implemented based on the theory of deductive database, which escapes Gelernter's framework completely and is the first synthetic prover capable of proving a large number of difficult geometry theorems. This prover can be used to find the *fixpoint* for a geometric configuration, i.e., the system can find all the properties of the configuration that can be deduced using a fixed set of geometric rules.

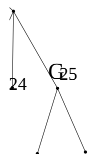
In (Chou, Gao and Zhang 1994), the *area method* for automated generation of human-readable proofs of geometry theorems is introduced. The area method is deterministic in that the method eliminates points in a rigid way. In (Chou, Gao and Zhang 1996a and 1996b), a relaxed search strategy is used in the area method to introduce some kind of non-determinism into the method. Using a relaxed search strategy has two positive aspects. (1) It allows the program to generate multiple and the shortest proofs for the same theorem. (2) The area method can be extended to non-constructive geometry statements. Most of the previous work of AGTP generally produce one proof. In the case of the area method, since the proofs generated are short and readable, generating multiple and shortest proofs for geometry theorems is important for the application of the method to *geometry education*.

The rest of this chapter is organized as follows. Gelernter's work is discussed in Section 10.2. The deductive database approach is discussed in Section 10.3. The method for generating multiple and shortest proofs is discussed in Section 10.4. Various technical issues in synthetic methods are discussed Section 10.5.

10.2 Gelernter's geometry theorem proving machine

Gelernter's geometry theorem proving machine (Gelernter 1963) uses a *backward chaining search*. Let G_0 be the conclusion to be established by the proof. It will be called the problem goal. If G_i is a formal statement with the property that G_{i-1} may be immediately inferred from G_i, then G_i is said to be a subgoal of order i for the problem. The problem solving graph (Figure 1) has as nodes the G_i, with each G_i joined to at least one G_{i-1} by a direct link. The problem is solved when any G_i can be immediately inferred from the premises of the problem and axioms. If, as is generally the case in geometry, a given subgoal is a conjunction of statements, the graph splits at that point, and each parallel subgoal must be separately established.

Without any improvements, this kind of brute force search has been shown to be too much time-consuming for so simple a logic as propositional calculus (Newell, Shaw and Simmon 1963). It is a fortiori out of the question for any of the more interesting logics. A remaining alternative is to have the machine rely upon heuristic methods.



The main heuristic used by Gelernter is to use numerical diagrams as semantic models. There are two benefits the prover derives from a numerical diagram. (1) The diagram is used as a filter to reject goals not consistent with its numerical representation as early as possible. This will reduce the search space drastically, because without this heuristic, most of the deduction paths generated will end up as false results.

(2) The diagram is used to determine order relations among points, or points and lines. These relations are necessary for the prover to find a proof. The first benefit is very important to the backward search and is not useful in the forward chaining (see Section 10.3). The second benefit is related to the problem of producing diagram independent proofs, a discussion of which can be found in Section 10.5.2.

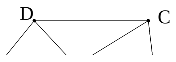
With the backward chaining search and the main heuristic determined, now the search for a proof actually becomes the search for a valid subgoal in the problem solving tree. There are two methods of doing this search: the *breadth-first search* and the *depth-first search.*

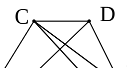
- In the breadth-first search, we will search the subgoals in order $1, 2, \ldots$ respectively. In Figure 1, the search order is: $G_0, G_{11}, G_{12}, G_{21}, \ldots$.
- In the depth-first search, we search the left-most branch of the problem-solving tree until we cannot go further. If the last subgoal is not established to be valid, we delete this subgoal and go back to its parent sub-goal. Now the search continues with this subgoal as the staring point and with the same style. The step of returning to the parent of a sub-goal is called *backtracking.* In Figure 1, the search order is: $G_0, G_{11}, G_{21}, G_{31}, G_{32}, \ldots$.

The breadth-first search is *complete* in the sense that if a valid subgoal does exist in the tree then it will be found by the breadth-first search. On the other hand, if the search tree is infinite then the depth-first search may end up in an infinite branch without a valid subgoal and thus cannot find the valid subgoal in other branches. In Gelernter's proving machine, a breadth-first search is used.

In his proving machine, Gelernter used geometric axioms about congruent triangles as the basic geometric rules. This might be one of the reasons that his proving machine fails to prove many difficult geometry theorems. Without adding techniques about auxiliary points, the rules about triangle congruence can be used to prove a very limited number of high school level theorems involving straight lines only. Most of the basic results in geometry such as the centroid theorem, the orthocenter theorem, and Simson's theorem are beyond the scope of these rules. In Section 10.5, we will discuss the concerns about selecting a set of "good" geometric rules.

Gelernter's proving machine has been used to prove more than fifty theorems from the high school textbooks. These theorems are quite simple compared with the theorems proved by the current theorem provers based algebraic methods (Chap. 5) or the deductive database method (Section 10.3). The following theorem is one of the most difficult ones proved by the proving machine.





Example 10.1:

$ABCD$ is a trapezoid such that $AB \parallel CD$. M and N are the midpoints of AC and BD. Let E be the intersection of MN and BC. Show that E is the midpoint of BC. (Figure 2(a))

The proving machine is unable to find a proof for the theorem at the present form. Then a new point $K = CN \cap AB$ is added by the user. Now, the proving machine is able to find a proof based on the key fact that $\triangle NBK$ is congruent to $\triangle NDC$. Also, many facts, such as $\angle ABD = \angle CDB$, are derived from the numerical diagram without a vigorous proof.

10.3 A deductive database approach

Most of the subsequent work of AGTP based on heuristics uses a similar approach with that of Gelernter's: they use a backward chaining search and axioms about congruent triangles. Nevins' theorem prover (1976) uses a combination of forward chaining and backward chaining. He also uses rules about congruent triangles. In (Chou et al 1998), a geometry theorem prover based on the theory of deductive database is implemented which uses forward chaining to achieve fixpoints and a set of geometric rules about full-angles.

10.3.1 The geometric rules

Although not specified explicitly, Gelernter used a special kind of deduction rules in his proving machine: Horn clauses. A rule is called a *definite Horn clause* if it has the following form:

$$Q(x) :- \quad P_1(x), \cdots, P_k(x) \qquad \text{meaning } \forall x[(P_1(x) \wedge \cdots \wedge P_k(x)) \Rightarrow Q(x)]$$

where the x are points occurring in the predicates $P_1, \cdots, P_k$, and Q. Also $(P_1(x) \wedge \cdots \wedge P_k(x))$ and $Q(x)$ are called the *body* and *head* of the rule respectively.

In (Chou et al 1998), the following predicates: points, coll (collinear), para (parallel), perp (perpendicular), midp (midpoint), cyclic, circle, eqangle, cong (congruent of segment), eqratio, simtri (similar triangle), and contri (congruent triangle) are used. The rules are highly complicated from the viewpoint of a deductive database (Bancilhon and Ramakrishnan 1986): all the predicates are mutually recursive and most of them are not linear. The central concept is eqangle. Here the angle is not the ordinary angle but the full-angle. Intuitively, a *full-angle* $\angle[u,v]$ is the angle from line u to line v. Two full-angles $\angle[l,m]$ and $\angle[u,v]$ are equal if there exists a rotation K such that $K(l) \parallel u$ and $K(m) \parallel v$. If A, B and C, D are distinct points on l and m respectively, then $\angle[l,m]$ is denoted by $\angle[AB,CD]$.

The introduction of full-angles simplifies the predicate of the angle congruence. For instance, we have the following rule about parallel lines.
R1. $AB \parallel CD$ if and only if $\angle[AB,PQ] = \angle[CD,PQ]$ (Figure 3).

Using ordinary angles, we need to specify the relations among eight angles and we need to use order relations (inequalities) to distinguish the cases. For instance, we have: "if points B, D are on the same side of line PQ and point P, C are on the different sides of line AB (the order relations), then $AB \parallel CD \Leftrightarrow \angle PEB = \angle PFD$." This rule is very difficult to use and may lead to branchings during the deduction. The following two rules also show why full-angle is crucial to this approach.
R2. $\angle[PA,PB] = \angle[QA,QB]$:- cyclic(A,B,P,Q).
R3. cyclic(A,B,P,Q) :- $\angle[PA,PB] = \angle[QA,QB]$, $\neg$ coll(P,Q,A,B).

In rule R2, using the ordinary angle, we need two conditions (Figure 4): $\angle APB = \angle AQB$ or $\angle APB + \angle AQ_1B = 180°$ and to distinguish these two cases, we need to know "points P and Q are on the same or different sides of line AB." Using full-angles, the two cases can be treated uniformly. For the treatment of the negation in rule R3, please see (Chou et al 1998).

10.3.2 Forward chaining and fixpoints

Let D_0 be the hypotheses of a geometry statement and R the geometric rule (or axiom) set. Basically speaking, the *forward chaining search* works as follows:

$$\boxed{D_0} \quad \overset{R}{\subset} \quad \boxed{D_1} \quad \overset{R}{\subset} \quad \ldots \overset{R}{\subset} \quad \boxed{D_k} \quad \text{(Fixpoint)}.$$

For each rule r in R, we apply it to D_0 to obtain new facts. Let D_1 be the union of D_0 and the set of new facts obtained in this way. Repeat the above process for D_1 to obtain D_2, etc. If at certain step $D_k = D_{k+1}$, then we say that a *fixpoint* for D_0 and R is reached, i.e., D_k is a fixpoint of reasoning rules R:

$$R(D_k) = D_k.$$

In the case of datalog (Gallaire 1984) (i.e., when we assume that the rules are Horn clauses without function symbols) the fixpoint can always be reached if the rule set R (i.e., the intensional database) and the initial fact set D_0 (i.e., the extensional database) are finite.

Since D_1 contains D_0 as a subset, the derivation of D_2 from D_1 clearly repeats all the previous deductions used to derive D_1 from D_0. The *semi-naive evaluation* is proposed to solve this problem (Bancilhon and Ramakrishnan, 1986). Note that in the forward chaining searches, the main loop of process is to search the rule set R. We call such search strategies *rule-based search strategies*. In (Chou et al 1998), a *data-based search strategy* is presented, in which a new-fact-list is kept and for each fact d in the list we find and apply all the rules whose bodies contain the predicate of d.

The Data-Based Search Algorithm.

Step 1 Set the hypotheses of the statement to be the initial new-fact-list and the initial database. While the new-fact-list is not empty do Step 2.

Step 2. Let d be the first new fact in the list. Delete it from the list, add it to the database, and do Step 3.

Step 3. Let r be a rule whose body contains a predicate P_0 of the fact d. To apply the rule r, we need to instantiate other predicates in r. Since predicate P_0 will be instantiated as the new fact d, other predicates in r need to be instantiated for all the facts in the database. For all the predicate forms of fact d (notice that a fact could have many predicate forms) and for all the facts of the other predicates in r, do Step 4.

Step 4. Apply rule r to obtain a fact d'. If d' is in the database, do nothing. Otherwise, add it to the end of the new-fact-list.

Since the hypothesis set of a geometry statement is finite and we use a finite rule set without function symbols, a fixpoint will always be reached.

10.3.3 Structured database

In the traditional way of representing a relational database, each n-ary predicate is associated with an n-dimensional array. Since most geometric predicates satisfy some special properties, building database according to this way will lead to very large databases. In (Chou et al 1998), the concept of *structured database* is introduced to reduce the size of the database. The following are three such principles:

(1) Use canonical form for predicates. One geometric property can be represented as many predicate forms. For instance, the predicate coll satisfies the following rules:

coll(A, B, C) :- coll(A, C, B), coll(A, B, C) :- coll(B, A, C).

Thus from coll(A, B, C), we can obtain five "new" facts: coll(A, C, B), coll(B, A, C), coll(B, C, A), coll(C, A, B), coll(C, B, A). We represent predicates as canonical forms by assigning an order to the points in the geometry statement. With such an order, predicates can be represented uniquely. Using canonical forms, the above rules are not needed explicitly in the deduction steps. This will reduce the number of rules used in the deduction process.

(2) Use equivalent classes to represent predicates. For instance, the fact that points $A_1, A_2, \cdots, A_n$ are on the same line can be represented by a sequence of points. In predicate form, we need $n(n-1)(n-2)$ different forms like coll(A_i, A_j, A_k) to represent this fact.

(3) Use representative elements for equivalent classes. For instance, the fact that the line containing points $A_1, ..., A_n$ is parallel to the line containing points $B_1, ..., B_k$ can be represented by $l_1 \parallel l_2$ if using l_1 and l_2 to represent the two lines. If using predicates para(A_i, A_j, B_k, B_l) to represent this fact, we need $2n(n-1)k(k-1)$ predicates.

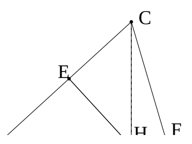
For the 160 geometry theorems solved by the prover, the average size of the databases is 221 if the above structure is used. Using the predicate form, the average size would be 242,117; i.e. one thousand times larger. For many configurations such as Example 10.4, the fixpoint cannot be reached within reasonable time if the above structure is not used.

10.3.4 Experimental results and examples

The following table contains the timing and database size statistics for the 160 geometric theorems solved by the program. The timing is collected on a Sparc-20 workstation.

Time (seconds)		Structured DB		Relational DB	
Time	Theorems	Size	Theorems	Size	Theorems
≤ 0.1	30%	≤ 50	16%	$\leq 10,000$	11%
≤ 1	69%	≤ 100	42%	$\leq 50,000$	43%
≤ 10	94%	≤ 200	66%	$\leq 100,000$	59%
≤ 60	98%	≤ 500	91%	$\leq 1,000,000$	95%
≤ 650	100%	≤ 4021	100%	$\leq 5,041,102$	100%
8.37 (average)		221 (average)		242,117 (average)	

The 160 geometry theorems solved by the prover are mainly chosen from about 600 theorems in (Chou 1988) and (Chou et al 1994). We can find fixpoints for each of the 600 theorems, but only about 160 of them can be proved in this way. Some well-known theorems such as Pappus' theorem and Pascal's theorem are beyond the scope of the program, although the fixpoints can be reached for both of them.



Example 10.2: The Orthocenter Theorem

Show that the three altitudes of a triangle are concurrent (Figure 5).

The hypotheses (extensional database) are: points(A, B, C), coll(E, A, C), perp(B, E, A, C), coll(F, B, C), perp(A, F, B, C), coll(H, A, F), coll(H, B, E), coll(G, A, B), coll(G, C, H).

Reaching the fixpoint costs the program 0.75 second. The size of the fixpoint is 146 if the structured database is used. In predicate form, the size of the fixpoint would be 56,940. The fixpoint contains two of the most often encountered properties of this configuration:
perp(C, G, A, B) (the conclusion) and $\angle[GF, GC] = \angle[GC, GE]$. Another amazing fact is that this simple configuration contains 105 nontrivial ratios!

The following is the proof for the fact perp(C, G, A, B), which is *automatically* generated by the prover. In the proof (hyp) means that the corresponding fact is from the hypotheses.

The Machine Proof

1. perp$[CG, AB]$:- (hyp)perp$[BC, AF]$, (hyp)coll$[GCH]$,
   (2)$\angle[BC, AF] = \angle[AB, CH]$.
2. $\angle[BC, AF] = \angle[AB, CH]$:- (3)$\angle[BC, AB] = \angle[AF, CH]$.
3. $\angle[BC, AB] = \angle[AF, CH]$:- (4)$\angle[BC, AB] = \angle[FE, AC]$,
   (5)$\angle[AF, CH] = \angle[FE, AC]$.
4. $\angle[BC, AB] = \angle[FE, AC]$:- (hyp)coll$[CBF]$, (hyp)coll$[CEA]$,
   (6)$\angle[BF, BA] = \angle[EF, EA]$.
5. $\angle[AF, CH] = \angle[FE, AC]$:- (hyp)coll$[AHF]$, (hyp)coll$[AEC]$,
   (7)$\angle[HF, HC] = \angle[EF, EC]$.
6. $\angle[BF, BA] = \angle[EF, EA]$:- (8)cyclic$[AFBE]$.
7. $\angle[HF, HC] = \angle[EF, EC]$:- (9)cyclic$[CFEH]$.
8. cyclic$[AFBE]$:- (hyp)perp$[FB, FA]$, (hyp)perp$[EB, EA]$.
9. cyclic$[CFEH]$:- (hyp)perp$[FH, FC]$, (hyp)perp$[EH, EC]$.

Example 10.3: The Five Circle Theorem

$P_0P_1P_2P_3P_4$ is a pentagon. $Q_i = P_{i-1}P_i \cap P_{i+1}P_{i+2}$, $M_i = circle(Q_{i-1}P_{i-1}P_i) \cap circle(Q_iP_iP_{i+1})$ (the subscripts are understood to be mod 5). Show that points M_0, M_1, M_2, M_3, M_4 are cyclic (Figure 6).

The fixpoint is reached in 3.89 seconds and contains 541 (220,680 in predicate form) facts. Besides the fact that M_0, M_1, M_2, M_3, and M_4 are cyclic, the program finds the following new result: the ten groups of lines below

$$P_{i+1}M_{i+1}, Q_{i-1}M_{i-1}, Q_{i+2}M_{i-2}; P_{i-1}M_{i-2}, P_iM_{i+1}, Q_{i-1}M_{i+2} : i = 0, ..., 4$$

are concurrent and the ten intersection points of them are on the circle determined by M_0, M_1, M_2, M_3, and M_4, i.e., this circle contains 15 points. The three dotted lines in Figure 7 represent one group of concurrent lines.

Example 10.4:

In the right triangle ABC, $\angle A = 90°$; $AH \perp BC$ and H is on line BC; S is the midpoint of AH; K and N are points on BC and AC such that $KN \parallel AB$; P and L are points on AB and BC such that $PL \parallel AC$; Q and M are points on AB and AC such that $QM \parallel BC$. Show that the six points P, Q, K, L, M, N are on the same circle (Figure 7).

It takes the program 461.06 seconds to reach the fixpoint which contains 1326 (5,041,102 in predicate form) facts. Without using the structured database, the fixpoint is too big to be reached within reasonable time.

10.4 Multiple and shortest proof generation

Generating multiple proofs is an inherent property of search methods. But the pre-condition is that the proving method itself should be very fast to generate a large amount of proofs in a short period of time. Another pre-condition is that the proofs generated should be readable, otherwise there is no reason to generate multiple proofs. The area method (Chap. 7) and the deductive database method are used to generate multiple proofs. In this section, we will introduce briefly how to use the area method to generate multiple proofs.

10.4.1 Basic lemmas about signed areas and ratio of segments

The signed area S_{ABC} in Euclidean geometry is used as a basic (undefined) geometric quantity described by the following properties.

L1 $S_{ABC} = S_{CAB} = S_{BCA} = -S_{BAC} = -S_{CBA} = -S_{ACB}$.

L2 Points A, B, and C are collinear iff $S_{ABC} = 0$.

L3 $PQ \parallel AB$ iff $S_{PAB} = S_{QAB}$.

L4 For points A, B, C, and D, we have $S_{ABC} = S_{ABD} + S_{ADC} + S_{DBC}$.

L5 If points A, B, C, and D are four collinear points such that $A \neq B$, and P is any point not on line AB, then $\frac{\overline{CD}}{\overline{AB}} = \frac{S_{PCD}}{S_{PAB}}$.

Here we introduce another geometric quantity $\frac{\overline{CD}}{\overline{AB}}$, the ratio of two directed segments on the same line, which satisfies L7 below.

The *signed area of a quadrilateral ABCD* is defined to be $S_{ABCD} = S_{ABC} + S_{ACD}$. By lemmas L4 and L1, we have.

Q1 $S_{ABCD} = S_{ABC} + S_{ACD} = S_{ABD} - S_{CBD}$,

Q2 $S_{ABCD} = S_{BCDA} = S_{CDAB} = S_{DABC} = -S_{ADCB} = -S_{DCBA} = -S_{CBAD} = -S_{BADC}$.

L6 (**The Co-side Theorem**) *Let M be the intersection of two lines AB and PQ and $Q \neq M$. Then* $\frac{\overline{PM}}{\overline{QM}} = \frac{S_{PAB}}{S_{QAB}}$; $\frac{\overline{PM}}{\overline{PQ}} = \frac{S_{PAB}}{S_{PAQB}}$; $\frac{\overline{QM}}{\overline{PQ}} = \frac{S_{QAB}}{S_{PAQB}}$.

L7 *For four distinct collinear points P, Q, A, and B, $\frac{\overline{PQ}}{\overline{AB}}$ is a real number which satisfies (1)* $\frac{\overline{PQ}}{\overline{AB}} = -\frac{\overline{QP}}{\overline{AB}} = \frac{\overline{QP}}{\overline{BA}} = -\frac{\overline{PQ}}{\overline{BA}}$; *(2)* $\frac{\overline{PQ}}{\overline{AB}} \cdot \frac{\overline{AB}}{\overline{PQ}} = 1$; *(3)* $\frac{\overline{PQ}}{\overline{AB}} = 0$ *iff* $P = Q$; *and (4)* $\frac{\overline{AP}}{\overline{AB}} + \frac{\overline{PB}}{\overline{AB}} = 1$.

For the relation of these lemmas to other axiom systems of geometries, please refer to (Chou et al 1994).

In the prover, each geometry lemma is represented as a *rule* or a *clause.* For instance, lemma L3 can be written as the following Prolog rule.

el_rule([area,P,A,B],[area,Q,A,B]) :- para(A,B,P,Q),P≠Q,

where [area,P,A,B] is S_{PAB} and the predicate para(A,B,P,Q) means that $AB \parallel PQ$. This lemma means that if $AB \parallel PQ$, [area,P,A,B] will be replaced by [area,Q,A,B].

10.4.2 The Prover

The prover is implemented in Prolog. The geometry lemmas can be put as inference rules in Prolog, and the (backward) logic engine of Prolog is used to do as much inference job as possible. At the highest level, the prover uses a typical *backward chaining* search strategy.

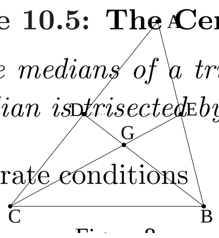
Step1 The prover first transforms the conclusion predicate into an equation of geometric quantities $\alpha = \beta$.

Step2 The prover keeps replacing geometric quantities in α and β with expressions in new geometric quantities by using the rules introduced in the preceding subsection until no new replacement can be made. After applying a rule to α or β to obtain α' and β', we remove the common factors of α' and β'.

Step3 Let the final equation be $\alpha'' = \beta''$. If α'' is literally equal to β'' then the theorem is true. Otherwise, we do not know whether the statement is true or not.

Example 10.5: The Centroid Theore

The three medians of a triangle meet in a
each median is trisected by this point.



As in Figure 8, we need to prove $-2 \cdot \frac{\overline{GD}}{\overline{GB}} = 1$. Non-degenerate conditions are that A, B, C.

The Machine Proof

$(-2) \cdot \frac{\overline{GD}}{\overline{GB}}$

$(\frac{\overline{GD}}{\overline{GB}} = \frac{S_{EDC}}{-S_{ECB}}$, because collinear($B$,$D$,$G$), collinear($C$,$E$,$G$). (L6))

$= \frac{2 \cdot S_{EDC}}{S_{ECB}}$

$(S_{EDC} = \frac{1}{-2}(S_{ECA})$, because midpoint($D$,$C$,$A$). (L5))

$(S_{ECB} = \frac{1}{2}(S_{CBA})$, because midpoint($E$,$B$,$A$). (L5))

$= \frac{2 \cdot S_{ECA}}{(-1) \cdot S_{CBA}}$

$(S_{ECA} = \frac{1}{-2}(S_{CBA})$, because midpoint($E$,$B$,$A$). (L5))

$= \frac{S_{CBA}}{S_{CBA}} = 1.$

(Simplification: removing the factor: S_{CBA}.)

By introducing an order among the points, the area and ratio can be represented canonically. We can thus introduce an order or a rank among the geometric quantities.

Control Strategy 1. A rule can be used only if it reduces an invariant to an expression in invariants with lower ranks.

Control Strategy 1 has a two-fold purpose. First, it guarantees the termination. Second, it makes the production of short proofs possible. Without this strategy, the prover may do endless searches before reaching the shortest proof. This can be seen from Example 10.6.

10.4.3 Multiple proof generation

There are two ways to generate different proofs. First, for an algebraic quantity

$$\alpha = \frac{P_1 \cdot P_2 \cdots P_n}{R_1 \cdot R_2 \cdots R_m}$$

we can use the rules to the polynomials P_i and R_i in different orders. Even by simply changing the order of applying the same set of rules to the algebraic quantities, essentially different proofs may be generated, because when a polynomial, say P_1, becomes Q_1 and R_1 is unchanged, it might happen that $Q_1 = R_1$ could be canceled. If we change R_1 to S_1 at the same time, the above reduction will not happen. The second way of generating different proofs is more important. For each geometric invariant, we may apply different rules to it to obtain different results. The following strategies to control the order of elimination are used.

Control Strategy 2. To apply a set of rules to an algebraic expression

$$\alpha = \frac{P_1 \cdot P_2 \cdots P_n}{R_1 \cdot R_2 \cdots R_m}$$

means either to apply one rule to one of $P_1, \cdots, P_n, R_1, \cdots, R_m$, or to apply one rule to one of the polynomials in the numerator and to apply another rule to one of the polynomials in the denominator.

Control Strategy 3. If two rules are used to eliminate the same point from both the numerator and the denominator of an algebraic expression or used to change the types of the geometric quantities, we will not use them separately to the same algebraic expression, since the separate use of these rules will not lead to canceling of geometric quantities.

Strategies 1,2, and 3 are quite effective to control the number and the length of the proofs.

Example 10.6:

Continuing from Example 10.5. If all the three control strategies (1, 2, and 3) are used, the prover gives 70 proofs with proof lengths ranging from 4 to 7. If we relax the control by dropping control strategy 3, then the prover gives 212 proofs with lengths ranging from 4 to 8. If we further relax the control by dropping the control strategy 1, then the prover gives 24360 proofs before it exhausts the memory of the computer. Much more proofs are expected, because in the 24360 proofs there are only two proofs with length less than or equal to 8.

10.4.4 Shortest proof generation

Using the *breadth-first search*, the first available proof is naturally a shortest proof. Moreover, using the breadth-first search, for some difficult geometry problems, the prover may exhaust the computer space before it obtains the first (shortest) proof. This problem is solved by using the following technique: if the shortest proof is needed, the prover will remember the length of a proof when it is generated and prohibit any backtracking that can generate proofs with lengths longer than or equal to the recorded length.

Another approach is to use the *depth-first iterative deepening search* (Korf 1985) and (Stickel 1985). In this approach, we first try to find a proof with depth 1, then depth 2, and so on by repeated use of depth-first search. The experience shows that both the above approaches to find shortest proofs are quite successful (Chou et al 1996a, 1996b).

Example 10.7:

Continuing from Example 10.6. The following proof for the centroid theorem is the shortest proof found by the prover. This proof cannot be generated by the previous area method.

The Machine Proof

$(-2) \cdot \frac{\overline{GD}}{\overline{GB}}$

$(\frac{\overline{GD}}{\overline{GB}} = \frac{S_{EDC}}{-S_{ECB}}$, because collinear($B$,$D$,$G$), collinear($C$,$E$,$G$). (L6))

$= \frac{2 \cdot S_{EDC}}{S_{ECB}}$

$(S_{EDC} = \frac{1}{-2}(S_{ECA})$, because midpoint($D$,$C$,$A$). (L5))

$(S_{ECB} = -S_{ECA}$, because midpoint(E,B,A). (L5))

$= \frac{S_{ECA}}{S_{ECA}}$

(Simplification: removing the factor: S_{ECA}.)

$= 1$

10.5 Issues concerning search methods

Now it is time to discuss several key issues in developing an efficient geometry theorem prover with the synthetic approaches.

10.5.1 Skolemization and adding of auxiliary points

Since most synthetic geometry reasoning systems use Horn clauses as rules, the systems have no ability to generate auxiliary points. Thus it is important that we can prove at least a large portion of the geometry theorems with the chosen geometric rules. The rules used by the deductive database approach are quite good in this aspect. Most of the 160 theorems proved by these rules without adding auxiliary points cannot be proved with the rules about congruent triangles if no auxiliary points are added. On the other hand, with functions of adding auxiliary points the prover will clearly be able to prove more theorems.

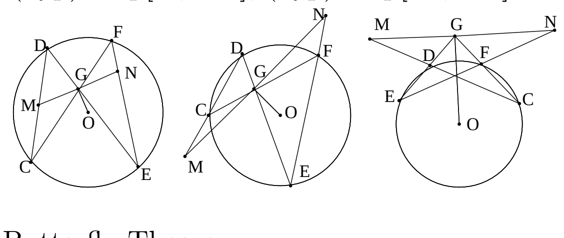
In the theory of logic, constructing new points corresponds to the Skolemization of the existential quantifiers. To use this idea in AGTP is discussed in (Robinson 1983 and Reiter 1976), but seems not implemented.

In (Chou et al 1998), about forty rules are used to add auxiliary points. Two strategies are proposed to control the adding of new points to achieve effectiveness. The first strategy is to separate the process of adding new points from the process of reaching the fixpoints. The program works as follows. For a geometry theorem, we first find a fixpoint without adding auxiliary points. If the conclusion is already in the database then the program terminates. Otherwise, the program will try to construct an auxiliary point and find a new fixpoint. The program will repeat the above process until either the conclusion is in the new database or there exist no new auxiliary points.

The second strategy is to use two heuristics to control the constructing of too many auxiliary points. (1) After an auxiliary point is added and a new fixpoint is reached, we will check whether new properties about the original diagram are found. If they are, we will keep this auxiliary point; otherwise, the auxiliary point will be deleted. (2) No recursive auxiliary points are allowed; i.e., to construct an auxiliary point, we can only use points occurring in the original statement. The second condition guarantees that the program will terminate.

For Example 10.1, the program in (Chou et al 1998) automatically adds an auxiliary point: A_0 which is the midpoint of AD. With this auxiliary point, the proof of the conclusion seems easier:

The Machine Proof
1. midp[E, BC] :- (2)para[CD, EN], (hyp)midp[N, BD].
2. para[CD, EN] :- (3)coll[$ENMA_0$], (4)para[CD, MA_0].
3. coll[$ENMA_0$] :- (hyp)line[MNE], (5)line[MNA_0].
4. para[CD, MA_0] :- (hyp)midp[M, AC], (hyp)midp[A_0, DA].
5. line[MNA_0] :- (6)para[A_0M, A_0N].
6. para[A_0M, A_0N] :- (7)para[A_0M, AB], (8)para[A_0N, AB].
7. para[A_0M, AB] :- (4)para[CD, MA_0], (hyp)para[AB, CD].
8. para[A_0N, AB] :- (hyp)midp[N, BD], (hyp)midp[A_0, DA].



Example 10.8: The Butterfly Theorem

C, D, E, and F are on the same circle with center O. G is the intersection of CF and DE. The line passing through G and perpendicular to OG meets CD and EF in M and N respectively. Show that G is the midpoint of NM. (Figure 9)

The conclusion is not in the first fixpoint. The program automatically adds an auxiliary point A_0 which is the intersection of the line passing through C and parallel to MN and the circle O. With the point A_0, it takes the program 0.4 second to reach the fixpoint which contains the conclusion on a Sparc-20 workstation.

10.5.2 Producing diagram independent proofs

The validity of most elementary geometry theorems involving only equalities is independent of the relative order positions of the points involved. Such geometry theorems belong to the so-called *unordered geometry.* This idea originated from Wu's algebraic method of automated reasoning (Wu 1984). In unordered geometry, the proofs of these theorems can be very simple. However, the ordinary proofs of these theorems involve the order relation; hence they are not only complicated but also not strict. The method based on congruent triangles is not for unordered geometry. In the work using this method, the order relations needed are either derived from a numerical diagram or given in the input. On the other hand, the rules used in the deductive database are for unordered geometry.

For instance, Gelernter's proof depends on the fact $\angle ABD = \angle BDC$ which is true in Figure 2(a). But in a different figure of the same theorem (Figure 2(b)), this fact is not valid anymore. Such facts are not isolated cases in AGTP. On the contrary, they exist in almost all geometry theorems. Nevins (1975) claimed that he has got rid of this drawback by adding the ordering relations to the hypotheses of the statement. Nevins' approach makes the situation much clear, but still does not solve the problem completely. First, to prepare for the order relation in the hypotheses, people still need to consult a diagram. Second, for some geometry theorems it may happen that the order of points in *different diagrams of the same theorem* may be different. For instance, there are at least three diagrams with different order relations among points and lines for the Butterfly theorem (Figure 9). Thus, a proof based on a fixed order relation among points and lines is valid only for some special cases of the statement. On the other hand, using a set of rules for unordered geometry, a uniform proof of this theorem can be given which is valid for all figures.

10.5.3 Avoiding redundant deductions

A *deduction is redundant* if it generates a fact that is already in the database. The redundant deduction is a major hurdle for speeding up the search and eliminating redundancies is proposed as a basic research problem by L. Wos (1988). It is proved that in general cases, the problem of eliminating all redundancies is undecidable (Helm 1990). So the best we can do is to design strategies to reduce the redundancies.

There are three kinds of redundant deductions. First, repeated use of the same rule to the same fact will generate the same result. This kind of redundancies can be solved by the semi-naive search or the data-based search.

Second, some redundant deductions are logically guaranteed. For example, if a rule r in a rule set R is a logical consequence of $R - \{r\}$ then each fact deduced using rule r will also be deduced by other rules in R. For a given rule set, the problem of obtaining a minimal and logically equivalent rule set is undecidable (Sagiv 1988). Many useful partial methods are given in (Buntine 1988; Helm 1990; and Sagiv 1988).

Third, two logically irrelevant deductions may give the same fact if the input facts satisfy certain conditions. We call this kind of redundancies the *conditional redundancies*. There seems no general method to control conditional redundancies.

In (Chou et al 1998), many heuristics are used to control redundant deductions.

This reference list will be replaced by the one of chapter 5.

Anderson, J. R. (1981): Tuning of search of the problem space for geometry proofs, *Proc. of Int, Joint Conf. Artificial Intelligence (IJCAI)*, Vancouver, pp. 165-170.

Bancilhon, F. and Ramakrishnan, R. (1986): An Amateur's Introduction to Recursive Query Processing Strategies, *Proc. of ACM SIGMOD Conference*, edited by C. Zanioilo., pp.16–52.

Buntine, W. (1988): Generalized Subsumption and Its Application to Induction and Redundancy, *Artificial Intelligence*, **36**(2), 149–179.

Chou, S. C. (1988), *Mechanical Geometry Theorem Proving*, D.Reidel Publishing Company, Dordrecht, Netherlands.

Chou S. C., Gao X. S., and Zhang J. Z. (1994): *Machine Proofs in Geometry*, World Scientific, Singapore, 1994.

Chou S. C., Gao X. S., and Zhang J. Z. (1996a): Automated Generation of of Readable Proofs with Geometric Invariants, I. Multiple and Shortest Proof Generation, *J. of Automated Reasoning*, **17**, 325-347.

Chou S. C., Gao X. S., and Zhang J. Z. (1996b): Automated Generation of of Readable Proofs with Geometric Invariants, II. Proving Theorems with Full-Angles, *J. of Automated Reasoning*, **17**, 349–370.

Chou S. C., Gao X. S., and Zhang J. Z. (1998): A Deductive Database Approach To Automated Geometry Theorem Proving and Discovering, accepted by J. Automated Reasoning.

Coelho, H. and Pereira, L. M. (1986): Automated Reasoning in Geometry Theorem Proving with Prolog, *J. of Automated Reasoning*, vol. 2, p. 329-390.

Evans, T. G. (1968): A Heuristic Program to Solve Geometry Analogy Problems, *Semantic Ubform. Proc.*, (ed), Minsky, M.I.T. Press.

Feigenbaum, E.A. and Feldman, J. (1963): *Computers and Thought*, Mcgraw Hill, New York.

Gallaire, H. , Minker, J. and Nicola, J. M. (1984): Logic and Databases: A Deductive Approach, *ACM Computing Surveys*, **16**(2), 153–185.

Gao X. S. , Zhang J. Z., and Chou S. C. (1998): *Geometry Expert*, Nine Chapters Pub. Comp., Taiwan, (in Chinese).

Gelernter, H. (1963): Realization of a Geometry-Theorem Proving Machine, *Computers and Thought*, eds. E.A. Feigenbaum and J. Feldman, pp. 134–152, Mcgraw Hill, New York.

Gelernter, H., Hanson, J. R. and Loveland, D. W. (1960): Empirical Explorations of the Geometry-theorem Proving Machine, *Proc. West. Joint Computer Conf.*, pp.143–147, 1960.

Gilmore, P. C. (1970): An Examination of the Geometry Theorem Proving Machine, *Artificial Intelligence*, **1**, 171–187.

Goldstein, H. (1973): Elementary Geometry Theorem Proving, MIT, AI LAB memo no. 280.

Havel, T. (1988), The Use of Distance as Coordinates in Computer-Aided Proofs of Theorems in Euclidean Geometry, IMA Preprint, No. 389, University of Minnesota, 1988.

Helm, R. (1990): On the Elimination of Redundant Derivations During Execution, *Proc. of 1990 North American Conference on Logic Programming*, pp.551–568, The MIT Press.

Kapur, D. (1986): Geometry theorem proving using Hilbert's Nullstellensatz, *Proc. of SYMSAC'86*, Waterloo, pp.202–208.

Koedinger, K. R. and Anderson, J. R. (1990): Abstract Planning and Perceptual Chunks: Elements of Expertise in Geometry, *Cognitive Science*, **14**, 511–550.

Korf, R. E. (1985): Depth-first Interactive-deepening: an Optimal Admissible Tree Search, *Artificial Intelligence*, **27**(1), 97–109.

Nevins, A. J. (1975): Plane Geometry Theorem Proving Using Forward Chaining, *Artificial Intelligence*, **6**, 1–23.

Newell, A., Shaw, J. C. and Simon H. A. (1963): Empirical Explorations with the Logic Theory Machine: A Case Study in Heuristics, *Computers and Thought*, eds. E.A. Feigenbaum and J. Feldman, pp. 109–133, Mcgraw Hill, New York.

Reiter, R. (1976): A semantically guided deductive system for automatic theorem proving, *IEEE Tras. on Computers*, vol C-25, No.4, 328-334.

Robinson, A. (1983): Proving a Theorem (as done by Man, Logician, or Machine), in *Automation of Reasoning*, ed. by J. Siekmann and G. Wrightson, pp.74–78, Springer-Verlag.

Sagiv, Y. (1988): Optimizing Datalog Programs, in *Foundations of Deductive Databases and Logic Programming*, Ed. J. Minker, pp.659–698, Morgan Kauffmann, 1988.

Stickel, M. (1985): A Prolog Technology Theorem Prover: Implementation by an Extended Prolog Compiler, *J. of Automated Reasoning*, **4**(4), 353–380.

Tarski, A (1951): *A Decision Method for Elementary Algebra and Geometry*, Univ. of California Press, Berkeley, Calif., 1951.

Wang, H. (1957): A Variant to Turing's Theory of Computing Machines, *J. of ACM*, **4**, 1957, pp.63–92.

Wang, D. (1995): Reasoning About Geometric Problems Using an Elimination Method, in *Automated Practical Reasoning*, eds. J. Pfalzgraf and D.M. Wang, pp.148-185, Springer-Verlag.

Wos, L. (1988): *Automated Reasoning: 33 Basic Research Problems*, Prentice-Hall, Englewood Cliffs, New Jersey.

W. T. Wu (1978), On the decision problem and the mechanization of theorem-proving in elementary geometry, *Scientia Sinica*, **21**, 159–172; re-published in *Automated Theorem Proving: after 25 Years*, (Ed. W.W.Bledsoe & D.W.Loveland), 1984, 235–242.

W. T. Wu (1984a), *Basic principles of mechanical theorem proving in geometries (part on elementary geometries)*, (in Chinese), Science Press. English translation by D.M.Wang et al, Springer, (1994).

Wu, W. T. (1984b), Basic principles of mechanical theorem proving in elementary geometries, *J. Sys. Sci.* and *Math. Scis.*, **4**(1984), 207-235. Re-published in *J. Automated Reasoning*, 1986.

References

Brudelin, B. (1986): Constructing Three-Dimensional Geometric Objects Defined by Constraints, in *Proc. Workshop on Interactive 3D Graphics*, pp. 111–129, ACM Press.

Buchanan, S. A. and de Pennington, A. (1993): Constraint Definition System: A Computer Algebra Based Approach to Solving Geometric Problems, *Computer Aided Design*, **25**(12), 740–750.

Buchberger, B. (1985): Gröbner Bases: An Algorithmic Method in Polynomial Ideal Theory, *Recent Trends in Multidimensional Systems theory*, D. Reidel Publ. Comp., 1985.

Chou, S. C. & Gao, X. S. (1990): Ritt-Wu's Decomposition Algorithm and Geometry Theorem Proving, in *Porc. CADE-10*, M. E. Stickel (ed.), pp. 207–220, Lect. Notes in Comp. Sci., Vol. 449, Springer-Verlag, 1990.

Chou, S. C., Gao, X. S., & Zhang, J. Z.. (1995): A Fixpoint Approach To

Automated Geometry Theorem Proving, WSUCS-95-2, CS Dept, Wichita State University, 1995.

Elster, K. H. (1993): *Modern Mathematical Methods of Optimization*, Akademie Verlag, 1993.

Fishler, M. A. and Bolles, R. C. (1981): Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartomated Cartography, *Communications of the ACM*, **24**(6), 381–395, 1981.

GABRI Geometry II, Texas Instruments, Dallas, Texas, 1994.

Gao, X. S. and Chou, S. C., (1998*a*): Solving Geometric Constraint Systems, I. A Global Propagation Approach, *Computer Aideded Design*, **30**(1), 47–54.

Gao, X. S. and Chou, S. C. (1998*b*): Solving Geometric Constraint Systems, II. A Symbolic Computational Approach, *Computer Aided Design*, **30**(2), 115–122.

Gao, X. S., J. Z. Zhang, and Chou, S. C. (1998): *Geometry Expert*, Nine Chapters Pub., 1998, Taiwan (in Chinese).

Gao, X. S., Zhu, C. C. and Huang, Y. (1998): Building Dynamic Mathematical Models with Geometry Expert, I. Geometric Transformations, Functions and Plane Curves, in *Proc. of ATCM'98*, W. C. Yang (ed.), pp. 216–224, Springer-Verlag.

Gao, X. S., Zhu, C. C. and Huang, Y. Building Dynamic Mathematical Models with Geometry Expert, II. Linkages, in *Proc. of ASCM'98*, Z. B. Li (ed.), pp. 15–22, LanZhou Univ. Press, 1998.

Gao, X. S., Cheng, H. F. (1998): On the Solution Classification of the "P3P" Problem, in *Proc. of ASCM'98*, Z. B. Li (ed.), pp. 185–200, LanZhou Univ. Press.

Ge, J. X. , Chou, S. C., and Gao, X. S. (1998): Geometric Constraint Satisfaction Using Optimization Methods, WSUCS-98-1, CS Dept, Wichita State University, 1998, *submitted to CAD.*

Hopcroft, J. and Tarjan, R. (1973): Dividing A Graph into Triconnected Components, *SIAM J. Computing*, **2**(3), 135–157.

Heydon, A. and Nelson, G. (1994): The Juno-2 Constraint-Based Drawing Editor, *SRC Research Report 131a.*

Hoffmann, C. (1995): Geometric Constraint Solving in R^2 and R^3, in *Computing in Euclidean Geometry*, D. Z. Du and F. Huang (eds.), pp. 266–298, World Scientific, 1995.

Jakiw, N. (1994): *Geometer's Sketchpad*, User Guide and Reference Manual, Key Curriculum Press, 1994.

Kempe, A. B. (1887): On a General Method of Describing Plane Curves of the n-th Degree by Linkwork, *Proc. of L.M.S.*, pp. 213–216, 1876; see also, Messenger of Math., T. VI., 143–144.

Kondo, K. (1992): Algebraic Method for Manipulation of Dimensional Relationships in Geometric Models, *Geometric Aided Design*, **24**(3), 141–147, 1992.

Kramer, G. (1992): *Solving Geometric Constraint Systems*, MIT Press, 1992.

Lamure, H. and Michelucci, G. (1996): Solving Geometric Constraints by Homotopy, *IEEE Trans on Visualization and Computer Graphics*, **2**(1), 28–34.

Latheam, R. S. and Middleditch, A. E. (199): Connectivity Analysis: A Tool for Processing Geometric Constraints, *Computer Aided Design*, **28**(11), 917–928.

Lee, J. Y. and Kim, K. (1996): Geometric Reasoning for Knowledge-Based Parametric Design Using Graph Representation, *Computer Aided Design*, **28**(10), 831–841.

Leler, W. (1988): *Constraint Programming Languages*, Addison Wesley.

Light, R. and Gossard, D.(1982): Modification of Geometric Models through Variational Geometry, *Geometric Aided Design*, **14**, 208–214.

Owen, J. (1991): Algebraic Solution for Geometry from Dimensional Constraints, in *Proc. ACM Symp. Found. of Solid Modeling*, ACM Press, pp. 397–407, Austin, TX, 1991.

Steele, G. L. and Sussman, G. L. (1980): CONSTRAINTS – A Language for Expressing Almost-Hierarchical Descriptions, *Artificial Intelligence*, **14**, 1–39.

Sunde, G. (1988): Specification of Shape by Dimensions and Other Geometric Constraints, in *Geometric Modeling for CAD Applications*, M. J. Wozny et al. (eds.), pp. 199–213, North Holland.

Sutherland, I. (1963) Sketchpad, A Man-Machine Graphical Communication System, in *Proc. of the Spring Joint Comp. Conference*, North-Holland, pp. 329–345.

Verroust, A. Schonek, F.and Roller, D. (1992) Rule-oriented Method for Parameterized Computer-aided Design. *Geometric Aided Design*, **24**(3), 531–540, October 1992.

Wang, D. M. (1996): GEOTHER: A Geometry Theorem Prover, in *Proc. CADE-13*, New Brunswick, 1996, pp. 213–239, LNAI, Vol. 1104, Springer-Verlag, Berlin.

Wu, W. T. (1994): *Mechanical Theorem Proving in Geometries: Basic Principles*, Springer-Verlag, Wien New York, 1994.

Yang, L. (1998): A Simplified Algorithm for Solution Classification of the P3P Problem, preprint, 1998.