Eidgenössische
Technische
Hochschule
Zürich

*Institut
für
Informatik*

Karl Lieberherr

Toward Feasible Solutions of NP-Complete Problems

September 1975 *14*

Toward Feasible Solutions of NP-Complete Problems

K. Lieberherr†

Keywords and Phrases :
approximation algorithms for NP-complete problems, decision procedure for satisfiability, incomplete resolution strategy, NP-complete, resolution, satisfiability, worst case behaviour of approximation algorithms.

CR Categories : 5.21,5.25,5.39.

Abstract
****

An algorithm j is considered which finds for each conjunctive normal form (cnf) a relatively good approximation model. An upper bound for the worst case behaviour is proven and is used to derive upper bounds for approximation algorithms for other NP-complete problems (e.g. graph colouring problems). Another algorithm, called Rj2, is obtained from algorithm j by adding a learning mechanism in such a way that for each input cnf a finite number of approximate solutions is produced. This sequence has the property that the last approximate solution is a model if there exists any. An interpretation of a cnf is called maximal if it satisfies a maximal number of clauses in the cnf. It is shown that P=NP if algorithm Rj2 also finds a maximal interpretation for a subclass of the unsatisfiable cnf's.

†
Institut fuer Informatik
Clausiusstrasse 55
CH-8006 Zuerich

Contents
****

Introduction
****

Certain combinatorial problems, such as the travelling salesman problem and the theorem proving in the propositional calculus, have long been notorious for their computational intractability, in this way that, despite the effort of many clever people, no algorithms have been found for them which can be guaranteed to require time bounded by a polynomial in the length of the input. The belief in the inherent difficulty of these problems has been strengthened by results of Cook and Karp [CO71,KA72]. These show that simple forms of the problems mentioned above, together with a wide variety of other combinatorial problems, form a class, the NP-complete problems(''polynomial complete problems'' in the terminology of Karp [KA72]), of which no member is known to have a polynomial time algorithm, but if any of these problems should have such an algorithm then they all have.
These results have stimulated many researchers to examine other combinatorial problems for which no polynomial time algorithms

are known, to determine whether they too are NP-complete. Their efforts have resulted in the discovery of additional members of this class [SA72,SE73,UL73]. Such results have considerable practical significance. If it is proven, that a problem is NP-complete and thus is unlikely to have any polynomial time algorithm, it is possible to concentrate on the following more hopeful alternative approaches.

First approach : Algorithms can be constructed, which, although admittedly exponential in the worst case, seem to work quickly on most practical problems [KE70,LI73].

Second approach : It relates to the fact that the practical situation of concern often imposes additional restrictions on the domain of the problem, which possibly makes it easier to solve the problem [GAR74,RA74].

Third approach : Algorithms are sought, which, although they do not actually find optimal solutions for the problem, are guaranteed to yield solutions which are ''close'' to optimal [GAR72, JO72, JO73, SA74]. This approach partially motivates this paper.

Fourth approach : Algorithms are developed which produce for each input a finite number of approximate solutions. This sequence has the property that the last approximate solution is the exact solution if there exists any. The algorithm can be run until an approximate solution is found which is good enough. This approach will be examined in this paper.

We summarize here the basic definitions, referring the reader to [KA72] for a more complete discussion. Let $B=\{0,1\}$ and let B^* denote the set of all finite strings of elements from B. Any subset L of B^* is called a problem. Let Pi be the class of functions $f:B^*\to B^*$ which are computable in polynomial time by one-tape deterministic Turing machines (or by random access machines). If L and M are problems, we say that L is polynomially reducible to M, written $L<M$, if there is a function f in Pi such that $f(x)$ in M if and only if x in L.

The ''problems'' we shall consider in this paper shall be presented as recognition problems although many are more naturally regarded as optimization problems. The straightforward details of the encoding of entities such as graphs and integers into strings of 0's and 1's are omitted.

The problem ''satisfiability'' (sat) is defined as follows:

Input: Set of clauses $s = \{c(1),c(2), \ldots ,c(p)\}$ in variables $x(1),x(2), \ldots ,x(n)$, each clause being a set of literals, where a literal is either a variable $x(i)$ or its negation $\neg x(i)$.

Property: There is a truth assignment to the variables which simultaneously satisfies all the clauses in s (a clause is satisfied if any of its literals is $x(i)$ for some ''true'' variable $x(i)$, or $\neg x(j)$ for some ''false'' $x(j)$).

A formula F of the propositional calculus is said to be a conjunctive normal form (cnf) iff F has the form $F(1), \ldots ,F(n)$, $n\geq 1$, where each of $F(1), \ldots ,F(n)$ is a disjunction of

literals. Remark that an element of sat is a conjunctive normal form. Given a propositional formula G, let x(1), ... ,x(n) be the variables occuring in formula G. Then an interpretation of G is an assignment of truth values to x(1), ... ,x(n). An interpretation can be described by a set I containing for each variable of G exactly one literal. If a variable x occurs positively in I (i.e. x is in I), then ''true'' is assigned to x, otherwise, if x occurs negatively in I (i.e. ¬x is in I), then ''false'' is assigned to x. If a formula G is true under an interpretation I, we say that I is a model of G. A formula which has a model is called satisfiable. Hence we can define an element of sat as a cnf which has a model or as a satisfiable cnf.

A problem L is in NP iff L<sat; L is NP-hard iff sat<L. (NP-hard means as hard as the most difficult problem in NP.) L is NP-complete iff L is NP-hard and in NP. (NP-complete means representative of the complete class NP with respect to difficulty.)
Instead of the problem sat any other NP-complete problem could be used to define the concepts NP, NP-hard and NP-complete.
The elements in the class of the NP-complete problems have all the same degree of difficulty in the following sense : If a polynomial time recognition algorithm exists for any NP-complete problem then they all have such an algorithm.
Let P be the class of problems recognizable in polynomial time (e.g. by a deterministic Turing machine or by a random access machine). The following theorem holds : P=NP iff there is an algorithm in P for the NP-complete problems.

Two clauses c1 and c2 are said to conflict if there are literals in c1 which appear complemented in c2. The clauses c1 and c2 are said to clash if there is exactly one literal in c1 which appears complemented in c2. If the clauses c1 + {x} and c2 + {¬x} (x is a variable and + means union of sets) clash, then their resolvent is the clause c1+c2. We will say that c1+c2 is obtained from c1+{x} and c2+{¬x} by applying resolution. c1+{x} and c2+{¬x} are called parent clauses and x is the variable resolved upon.

A resolution proof of unsatisfiability of a set s of clauses is a sequence of clauses c(1),c(2), ... ,c(k) such that c(k) is the empty clause and for 1≤i≤k-1, c(i+1) is the resolvent of some pairs from s+{c(1), ...,c(i)}. In [R065] it is shown that a set of clauses is unsatisfiable iff there is a resolution proof of its unsatisfiability.

A clause c1 subsumes a clause c2 if c1 is a subset of c2. A cnf s has the same models as the cnf s' obtained from s by deleting those clauses which are subsumed by other clauses. s' is said to be obtained by applying subsumption to s.

We shall use the following abbreviations :

abs	abs(a) : absolute value of a
card	card(a) : the cardinality of the set a

cnf	conjunctive normal form (= a set of clauses)
in	a in B : a is an element of B
trunc	for nonnegative reals trunc(a) is the greatest integer which is smaller or equal to a
**	exponentiation
{}	the empty set
+	union of sets (and addition)
*	intersection of sets (and multiplication)
≠	end of a proof

1. The Approximation Algorithm j

Let s be a finite set {c(1),c(2), ... ,c(p)} of clauses with set of variables v.

Def: [weight of a clause c(k):weight(c(k))]
weight(c(k)) = 2**(-card(c(k)))

Def: [weight of a set s of clauses : weight(s)]
weight(s) = sum of the weights of the clauses which occur in s.

Johnson, in [J073], gives the following algorithm j. It computes for each variable y a number which indicates whether y must be set true or false, so that many of the remaining clauses can still be satisfied.

Algorithm j

1. satis:={}; true:={}; var:=v; left:=s; assign to each clause c in s the weight w(c):=weight(c);
2. while there is a variable of var in any clause of left do
   begin
   a) Let y be any variable occuring in both var and a clause of left. Let yt be the set of clauses in left containing y and yf the set of clauses in left containing ¬y.
   b) If the sum of the weights of the clauses in yt is greater or equal to the sum of the weights of the clauses in yf then
      begin
         true := true + { y} ; satis := satis + yt ; left := left - yt ; for each c in yf set w(c) := 2 * w(c);
      end else
      begin
         true := true + {¬y} ; satis := satis + yf ; left := left - yf ; for each c in yt set w(c) := 2 * w(c);
      end ;
   c) var := var - {y}
   end

A slight modification of the proof of theorem 3 in [J073] gives the following result.

Theorem [worst case behaviour j]
Let s be a set of clauses and weight(s) its weight. Then algorithm j leaves in the worst case at most trunc(weight(s)) clauses unsatisfied, i.e. at least card(s) - trunc(weight(s)) clauses are satisfied.

In [J073] the above theorem is proven only if in s each clause contains at least k literals and weight(s) is set to 2**(-k).

Remark: The time required by algorithm j (if appropriately implemented on a random access machine) is bounded by c*n*(n+1)*l, where c is some constant, n is the number of variables and l the number of literals in s. With this bound it is even possible to determine in statement 2.a) the variable for which abs(weight(yt)- weight(yf)) is maximized.

Proof
At statement 1 the clauses in left have weight weight(s). During each iteration the weight of the clauses removed from left is, by statement b) in the while loop, at least as large as the weight added to those remaining clauses for which the current literal is not satisfied. Thus the total weight of the clauses in left can never increase and so when the algorithm halts, it still cannot exceed weight(s). But when the algorithm halts, each of the clauses in left must have had its weight doubled as many times as it had literals and so must have final weight 1. ≠

If we have a look at the proof of theorem [worst case behaviour j], we see that algorithm j assumes its worst case behaviour only, if the weight of a clause which is not satisfied by the current literal is at least doubled, and if the total weight does not exceed the original weight weight(s).
The following algorithm jj has the same worst case behaviour as algorithm j. For obtaining algorithm jj replace the two occurences of the statement ''set w(c):= 2*w(c)'' in the statement b of the while loop by the statement ''assign a rational number e(c) to each clause c and set w(c):=2*w(c) + e(c), such that the sum of the w(c) for all clauses in left does not exceed weight(s)''. This means that at most the absolute value of the difference of the weights of the clauses in yt,yf respectively, can be additionally distributed.
The advantage of algorithm jj consists in the possibility of preferring ''important'' clauses (they express conditions which should absolutely be satisfied) while the worst case behaviour is not made worse.

Algorithm j and the proven worst case behaviour can obviously be generalized to sets of clauses where each clause has an integral weight.

1.1. The Application of Algorithm j to Other NP-Complete Problems

****

1.1.1 Introduction

Algorithm j has a surprisingly good worst case behaviour. On the other hand there exists a large number of NP-complete problems for which only approximation algorithms with a bad worst case behaviour are known [J073]. Our aim is to detect NP-complete problems for which good approximation algorithms are available. The following definitions are from [J073].

Def: [optimization problem]
An optimization problem p consists of
1. A set Input[p] of possible inputs
2. A map sol[p] which maps each u in Input[p] to a finite set of approximate solutions
3. A function m[p]:sol[p](Input[p]) -> Q defined for all possible approximate solutions. (Q is the set of rational numbers.) m[p] is called a measure.

In addition, the problem p is specified as a maximization problem or a minimization problem, depending on whether the goal is to find an approximate solution with maximal or minimal measure. For each u in Input[p], the optimal measure is defined by u[p]* = best {m[p](x) : x in sol[p](u)}, where best stands for max or min depending on whether p is a maximization or minimization problem. If sol[p](u) is finite, there must be at least one solution x in sol[p](u) such that m[p](x) = u[p]*, and such a solution will be called an optimal solution.
An approximation algorithm for problem p is any method for choosing approximate solutions, given u in Input[p]. Since the algorithms we will study are not always completely determined, more than one solution may be choosable for a given input. If A is an approximation algorithm for problem p, then the performance A(u)[p] of A for input u is defined by A(u)[p] = worst {m[p](x) : x in sol[p](u) and x is choosable by A on input u}, where worst is min if best is max, and vice versa.

Example
The optimization problem maximum satisfiability (opt max sat)
Input[opt max sat] ={s : s is a finite set {c(1),c(2), ... ,c(p)} of clauses}
sol[opt max sat](s)={s' : s' is a subset of s such that there exists a truth assignment t which satisfies every clause in s'}
m[opt max sat](s')=card(s')

Algorithm j is an approximition algorithm for ''opt max sat'' with the performance
j(s)[opt max sat] = card(s) - trunc(weight(s)).

1.1.2 The Optimization Problem ''Graph Colouring''

Def: [opt graph col i]
Input=Graph G=(N,A); G is a finite undirected graph with nodes N and arcs A ;
sol(G)={functions q:N->{1,2, ... ,i'}, where i' is the smallest power of two which is ≥i ; (i is the number of admissible colours)
m(q)=card({z in A : if z is an arc between nodes n1 and n2 then q(n1)=q(n2)}) + card({n in N / n has a colour the number of which is > i}) . (m(q) counts the number of colouring mistakes of q)

Comment : The problem is to colour a graph with i' colours so that the number of adjacent nodes which have the same colour plus the number of nodes which have a colour >i is minimized.

The following problem ''graph col i'' is NP-complete for i≥3.

Def: [graph col i]
Input : Graph G=(N,A)
Property : There is a solution h with measure m(h)=0.

For applying algorithm j we translate ''opt graph col i'' to ''opt max sat'' but first only for numbers i which are of the form i=2**k for some integer k≥2.
Let z be an arc in A between nodes n1 and n2 (n1≠n2). We assign a set t of clauses to z.

1. k=2
To n1 we assign the set of variables {a,b} and to n2 the set {c,d}. There are 4 truth assignments for {a,b} which correspond to the 4 colours for node n1.
t=
1: ¬a ¬b ¬c ¬d
2: ¬a b ¬c d
3: a ¬b c ¬d
4: a b c d
Clause 1 expresses : if node n1 has the colour (1,1) then n2 cannot have the same colour (1,1); by a formula (a∧b=>¬(c∧d)). t has weight 4*1/(2**4) = 1/4. Observe that for each interpretation of t at most one clause is unsatisfied.

2. General case
t consists of 2**k clauses which contain 2*k literals. Hence the weight of t is (2**k)/(2**(2*k)) = 1/(2**k).

Let h=card(A). If we translate the whole graph G with h arcs we obtain a cnf s which contains h*(2**k) clauses. s has weight h/(2**k). Hence algorithm j guarantees that at most trunc(h/(2**k)) clauses are not satisfied. To each unsatisfied clause corresponds exactly one arc whose endpoints have the same colour.
Therefore algorithm j induces an approximation algorithm B for the optimization problem ''opt graph col 2**k''. B has the performance B(G(N,A))[opt graph col 2**k] =

trunc(card(A)/(2**k)). The time required by algorithm B is bounded by
((n*k)**2) * h*(2**k)*2*k= c(k)* (card(N)**2)*(card(A)),
where c(k) is some constant depending on k and n=card(N). (Observe that n*k is the number of variables of t and h*(2**k)*2*k is the number of literals in t.)

Example
If we want to colour a graph G with 16 colours and G has 320 arcs then algorithm B guarantees that for at most 20 arcs the endpoints have the same colour.

Now we give approximation algorithms for the general graph colouring problem where the number of colours is not a power of two. Let G=(N,A) be a graph and let n=card(N) and h=card(A).

i=3: Let n1,n2 be two nodes which are incident with the same arc. We assign two variables a and b to n1 and two variables c and d to n2. We use the same technique as for 4-colourability but delete one clause, say ¬a,¬b,¬c,¬d. Instead we express for the two nodes that they cannot have the colour (1,1), i.e. we add the two clauses ¬a,¬b and ¬c,¬d. If we translate G we obtain a cnf the weight of which is h*3/16 + n*1/4.
Therefore algorithm j induces an approximation algorithm C(3) for ''opt graph col 3''. C(3) has the performance C(3)(G(N,A))= trunc(h*3/16 + n*1/4).

i=7: With the above method we obtain an algorithm C(7) the performance of which is C(7)(G(N,A))= trunc(h*7/64 + n*1/8).

i=6: We have to forbid two colours for each point. This is possible with a clause which contains two literals. Hence the performance of C(6) is C(6)(G(N,A))= trunc(h*6/64 + n*1/4).

i=5: We forbid three colours for each point with two clauses the weight of which is 3/8. Hence the performance of C(5) is C(5)(G(N,A))= trunc(h*5/64 + n*3/8).

General case :
With the above method the performance of C(i) is :

C(i)(G(N,A))=

trunc(h*i/(2**(2*log(i')))+ n*(i'-i)/(2**log(i'))) =

trunc(h*i/(i'**2)+n*(i'-i)/i') ,

where i' is the smallest power of two which is ≥i and log is the logarithm to base two.
The running time of C(i) is bounded by c*n*n*h, where c is some constant depending on i.

We shall have a look at another method for (polynomially) reducing ''graph col i'' to ''satisfiability''. We introduce a

variable for each colour of each node . We express for each node that only one colour can be assigned to it. This is possible with i*(i-1)/2 clauses of length two (expressing : at most 1) and a clause of length i (expressing : at least 1). With i clauses of length 2 we indicate that two nodes which are incident with an edge cannot have the same colour.

Example :
We choose a graph with 1 arc and 2 nodes which we want to colour with i=3 colours. The following cnf is obtained :

```
 1: ¬a ¬b
 2: ¬a    ¬c
 3:    ¬b ¬c
 4:  a  b  c
 5:          ¬d ¬e
 6:          ¬d    ¬f
 7:             ¬e ¬f
 8:           d  e  f
 9: ¬a       ¬d
10:    ¬b       ¬e
11:       ¬c       ¬f
```

If we translate a graph with n nodes and h arcs, we obtain a cnf the weight of which is

n*(i*(i-1)/8+2**(-i))+h*i/4.

Observe that this weight is greater than n+h if i≥4. Hence we see that not each polynomial reduction of an NP-complete problem to ''satisfiability'' yields an approximation algorithm C such that the upper bound proven for j gives an interesting upper bound for C. Note that, however, nothing is used of the special structure of the input cnf's. Thus a better worst case behaviour of C can perhaps be proven.

1.1.3 The Optimization Problem ''Hitting Set''

Def: [opt hit set 1]
Input = family f= {u(1),u(2), ... ,u(n)} of subsets of a finite set v such that card(u(k))≤1, 1≤k≤n.
sol(f) = {v' : v' is a subset of v}
m(v') = number of sets u(k) (1≤k≤n) such that card(u(k)*v')≠1.
(recall : * means intersection)

Comment : The problem consists of finding a set which is as ''near'' as possible to a selection set. opt hit set 1 is a minimization problem.

We conjecture that the following problem ''hit set 1'' is NP-complete for 1≥3.

Def: [hit set 1]
Input : family f= {u(1),u(2), ... ,u(n)} of subsets of a finite set v such that card (u(k))≤1, 1≤k≤n.
Property : There is a solution v' with measure m(v')=0.

Remark : If l is infinite then the problem is NP-complete as proven in [KA72].

We cannot yet prove that ''hit set l'' is NP-complete for all $l \geq 3$ but look for an approximation algorithm. Therefore we translate ''opt hit set l'' to ''opt max sat''.
Choose a set u(k), $1 \leq k \leq n$. We assign a set t of clauses to u(k).

1. card(u(k))=1
   Let u(k)={a}. Then set t = a.

2. card(u(k))=2
   Let u(k)={a,b}. Then set t=
   1: a b
   2: ¬a ¬b

3. card(u(k))=3
   Let u(k)={a,b,c}. Then set t =
   1: a b c
   2: ¬a ¬b c
   3: ¬a b ¬c
   4: a ¬b ¬c
   5: ¬a ¬b ¬c
   Remark: t is satisfied iff exactly one variable of {a,b,c} is set true. If t is not satisfied then exactly one clause is not satisfied. t contains 1+ C(3,2)+ C(3,3) clauses. (The numbers C(n,k) are the binomial coefficients defined by
   C(n,k)= n*(n-1)* ... *(n-k+1)/(1*2* ... *k).)

4. card(u(k)=4
   Let u(k)={a,b,c,d}. Then set t =
   1: a b c d
   2: ¬a ¬b c d
   3: ¬a b ¬c d
   4: ¬a b c ¬d
   5: a ¬b ¬c d
   6: a ¬b c ¬d
   7: a b ¬c ¬d
   8: ¬a ¬b ¬c d
   9: ¬a ¬b c ¬d
   10: ¬a b ¬c ¬d
   11: a ¬b ¬c ¬d
   12: ¬a ¬b ¬c ¬d
   Remark : t contains 1+C(4,2)+ C(4,3)+ C(4,4) clauses.

5. General case
   Let h=card(u(k)). t consists of 1+C(h,2)+ C(h,3)+ ... +C(h,h) clauses. Since 1+c(h,1)+ C(h,2)+ ... +C(h,h)=2**h, t consists of 2**h-h clauses. Each clause of t contains 2**h literals. Hence t has weight (2**h-h)/(2**h)= 1-h/(2**h)

Let us translate the whole family f. Let h(i)= card(u(i)), $1 \leq i \leq n$. We obtain a cnf s with weight

w=1-h(1)/(2**h(1))+1-h(2)/(2**h(2))+ ... +1-h(n)/(2**h(n)).

Hence algorithm j guarantees that at most trunc(w) clauses are unsatisfied. To each unsatisfied clause corresponds exactly one set u(i) with card(u(i)*v')≠1. (v' is an approximate solution) Therefore algorithm j induces an approximation algorithm B for the optimization problem ''opt hit set 1''. B has the performance B(f)=trunc(w), where w is defined as above.

Example
Let f be a family of 80 sets each set containing exactly 3 elements. The weight of the cnf associated with f is 80*(1-3/8)=50. Hence for at most 50 sets the intersection with v' contains not exactly one element.

2. The Problem ''Maximum Satisfiability''

Let s be a finite set {c(1),c(2),...,c(p)} of clauses.

Def: [max sat]
Input for max sat : s and a natural number k.
Property : There exists an interpretation (truth assignment) t which satisfies at least k clauses.

For k=p this is the classical satisfiability problem. By theorem [worst case behaviour j], for
k=card(s)-trunc(weight(s))
there always exists an interpretation which satisfies at least k clauses. Hence this problem is computable in constant time.

In [GAR74] the following problem max sat2 is proven to be NP-complete.

Def: [max sat2]
Input for max sat2 : s with the restriction that each clause may contain at most two literals; a natural number k.
Property : There exists an interpretation t which satisfies at least k clauses.

Observe that, if k=p, this problem can be solved in polynomial time [CO71].

Def: [sat=i]
sat=i (satisfiability with exactly i literals per clause) is the same problem as sat with the restriction that each clause must contain exactly i literals.

sat=i is NP-complete for i≥3 [KA72].

We describe the reduction sat=3 < max sat2 sketched in [GAR74]. Let s1 be a set of m clauses with exactly three literals in each clause.

s 1 = a(1) b(1) c(1)
a(2) b(2) c(2)
. . .
. . .
a(m) b(m) c(m)

Define s2 as follows : replace each of the m clauses a(i) b(i) c(i) by the 10 clauses

1) a(i)
2) b(i)
3) c(i)
4) d(i)
5) ¬a(i) ¬b(i)
6) ¬a(i) ¬c(i)
7) ¬b(i) ¬c(i)
8) a(i) ¬d(i)
9) b(i) ¬d(i)
10) c(i) ¬d(i).

Let k=7*m.

Theorem [reduction sat=3 < max sat2]
s 1 is satisfied, if and only if s2 has an interpretation such that at least k clauses are satisfied.

Proof
a(i) b(i) c(i) has the following eight interpretations

	a(i)	b(i)	c(i)
a	1	1	1
b	1	1	0
c	1	0	1
d	0	1	1
e	1	0	0
f	0	1	0
g	0	0	1
h	0	0	0

In the following table we examine the influence of these eight interpretations on the 10 clauses 1),....,10).

	a	b	c	d	e	f	g	h
d(i)=	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0
1)	s	s	s	u	s	u	u	u
2)	s	s	u	s	u	s	u	u
3)	s	u	s	s	u	u	s	u
4)	s/u	s/u	s/u	s/u	s/u	s/u	s/u	s/u
5)	u	u	s	s	s	s	s	s
6)	u	s	u	s	s	s	s	s
7)	u	s	s	u	s	s	s	s
8)	s	s	s	u/s	s	u/s	u/s	u/s
9)	s	s	u/s	s	u/s	s	u/s	u/s
10)	s	u/s	s	s	u/s	u/s	s	u/s
nsc	7/6	7/7	7/7	7/7	6/7	6/7	6/7	4/6

legend : s : satisfied
u : unsatisfied
s/u : if d(i)=1 then s else u
nsc : number of satisfied clauses

7/6 : if d(i)=1 then 7 else 6, etc.

Thus 7*m=k clauses in s2 can be satisfied simultaneously, if and only if s1 is satisfiable. For, if we have any satisfying assignment for s1, then either one, two or three of a(i),b(i),c(i) must be set true for each i. In all three cases, there is a truth setting for d(i) causing precisely seven of the clauses in s2 arising from clause i to be satisfied. Furthermore, no setting of d(i) will permit more than seven clauses of the ten clauses to be satisfied, and at most six of the clauses can be satisfied, if all a(i) b(i) and c(i) are false. ≠

If we analyze the proof of theorem [reduction sat=3 < max sat2] we see that P=NP depends on 1/20 for max sat2 in the following sense. s2 contains 10*m clauses, if s1 contains m clauses. The weight of s2 is : m*(4*1/2+6*1/4)=3.5*m. Thus algorithm j satisfies at least 10*m-trunc(3.5*m) clauses. We want to know whether it is possible to satisfy 7*m clauses. Therefore we only have to decide, in the worst case, whether 1/2*m of 10*m clauses, that means whether 1/20 of the clauses of s2 can be satisfied additionally.

The above result can be improved to ''P=NP depends on 1/80 for max sat2''. In the proof of theorem [reduction sat=3<max sat2] d(i) can be chosen such that at least 6 clauses are satisfied. Therefore one unsatisfied clause in s1 ''corresponds'' to at most one unsatisfied clause in s2.
Hence if it is possible to decide in polynomial time whether 1/80 of the clauses((1/8)*m/(10*m)) can be satisfied additionally in s2, then P=NP.
The result ''P=NP depends on 1/80 for max sat2'' must be looked at as a property of the given reduction sat=3<max sat2. Otherwise this result is trivial for the following reason :

If s1 contains m clauses then add to s2 k*m new variables and k*m new clauses. These clauses shall express that a literal l1 of s2 implies a literal l2 of the additional k*m variables so that the new k*m clauses are satisfiable. (This can easily be done.) Therefore, if for a fixed k it is possible to decide in polynomial time whether (1/2*m)/(10*m+k*m)= 1/(2*(10+k)) of the clauses of s2 can be satisfied additionally, then P=NP. k can be arbitrarily large and thus we obtain the trivial result: ''P=NP depends on an arbitrarily small number >0 for max sat2''.

3. Complete Algorithms

We examine learning algorithms for sat. An algorithm for sat is complete if it halts after a finite number of steps with the result ''satisfiable'' if the input cnf is satisfiable, and with the result ''unsatisfiable'' otherwise.

First we describe informally the complete algorithm A presented

in this section. The idea originates in the completeness proof for resolution in [RO65].
Let s be a cnf. Let v(1),v(2),. .,v(k) be all the variables which occur in s. Algorithm A tries to find an interpretation I for s such that, if it fails to find a model, it ''learns'' a resolvent c. This clause c is added to the set s of clauses and the presence of c will guide algorithm A in the next step to a ''better'' interpretation in the following sense. If s is satisfiable then algorithm A will find a model after a finite number of steps, if s is unsatisfiable, then after a finite number of steps an old clause (a clause which occurs originally in s) will be learned. If algorithm A learns an old clause then we can prove that the input cnf must be unsatisfiable.
Now we give the formal definitions.

Def: [interpretation construction method R']
Let s be a cnf. Let v(1),v(2),...,v(k) be all the variables which occur in s. Let I be the interpretation defined as follows. I(0) is the empty set; and for $0<j\leq k$, I(j) is the set $I(j-1)+\{v(j)\}$, unless some clause in s consists entirely of complements of literals in the set $I(j-1)+\{v(j)\}$; in this case I(j) is the set $I(j-1)+\{\neg v(j)\}$. Finally, I is I(k).

Def: [interpretation construction method R]
Let s be a cnf. Let v(1),v(2),...,v(k) be all the variables which occur in s. Let l(1),l(2),...,l(k) be a sequence of k literals containing each one of the k variables v(1), ... ,v(k) exactly once. Let I be the interpretation defined as follows. I(0) is the empty set; and for $0\leq j\leq k$, I(j) is the set $I(j-1)+\{l(j)\}$, unless some clause in s consists entirely of complements of literals in the set $I(j-1)+\{l(j)\}$; in this case I(j) is the set $I(j-1)+\{\neg l(j)\}$. Finally, I is I(k).

Def: [learning variable]
If in the above construction R' (R, respectively), in the case where I(j) is set $I(j-1)+\{\neg v(j)\}$ ($I(j-1)+\{\neg l(j)\}$, respectively) a clause consists entirely of complements of literals in the set $I(j-1)+\{\neg v(j)\}$ ($I(j-1)+\{\neg l(j)\}$, respectively) then v(j) (v(p(j)), respectively) is a learning variable.

Example for R'

s =
1: ¬a ¬b ¬c
2: b ¬c
3: ¬a c
4: a b
5: ¬b c
6: ¬b

$I(0)=\{\}$; $I(1)=\{a\}$; $I(2)=\{a,\neg b\}$ (if b is set true then clause 6 is unsatisfied); $I(3)=\{a,\neg b,\neg c\}$.
c is a learning variable. If c is set true then clause 2 is

unsatisfied, otherwise, if c is set false then clause 3 is unsatisfied.

Example for R

We choose the same cnf as in the previous example. Let ¬a,c,b be the chosen sequence of literals.
I(0)={}; I(1)={¬a}; I(2)={¬a,c}; I(3)={¬a,c,¬b}.
b is a learning variable. If b is set true, then clause 6 is unsatisfied, otherwise, if b is set false ,clauses 2 and 4 are unsatisfied.

Remark : I is a model iff there is no learning variable.

Algorithm A

Let s be a cnf.
repeat
find an interpretation I with method R' for the cnf s ; for the first learning variable v(j) learn the set t of clauses defined as follows : let c(1),c(2), ... ,c(g1) be the g1 clauses which were unsatisfied when v(j) was set ; let d(1),d(2), ... ,d(h1) be the h1 clauses which would have been unsatisfied if v(j) had been set opposite.
t := {resolvents of c(g) and d(h) with 1≤g≤g1 and 1≤h≤h1} ;
s:=s+t (at this point the algorithm learns the clauses in t.)
until there is no clause learned or an old clause is learned;
if I is not a model then s has no model.

Remark
For all g (1≤g≤g1) and for all h (1≤h≤h1) resolution is always possible between c(g) and d(h) since c(g) and d(h) clash because of the learning variable.

Theorem [completeness of A]
Let s be a cnf. If s is unsatisfiable, then A learns the empty clause; otherwise A finds a model.

Proof
A terminates after a finite number of steps because there exists only a finite number of clauses on the given set of variables. If A finds the empty clause then s is unsatisfiable, because there exists a resolution proof for the unsatisfiability of s. Let us suppose that the empty clause has not been learned when the algorithm halts. If t is empty then I must be a model because there does not exist a learning variable. Let us suppose t contains a clause c of s. Then there cannot exist a learning variable, for c would already be unsatisfied when the first learning variable is set. ≠

{Proof structure :
1. empty clause in s when A halts : s is unsatisfiable
2. empty clause not in s when A halts
2.1 t=empty set : then I is a model

2.2 t contains an element of s : contradiction}

If, in each step of the repeat loop, only one clause of the set t (see definition of A) is learned, then this new algorithm, called A1, still is complete. For, if this learned clause is old, a contradiction to the assumption that it was learned with the first learning variable can be deduced as above.

Let A2 be algorithm A1 where R' is replaced by R. A2 is also complete.

There are different forms of the termination condition of algorithms A,A1,A2 which give the same result, but require different numbers of repetitions of the repeat loop. They can be composed of the following conditions.

1. Conditions implying that I is a model
   1.1. there is no clause learned
   1.2. I is a model

2. Conditions implying that the input cnf is unsatisfiable
   2.1. an old clause is learned
   2.2. the empty clause is learned

3. A condition implying that if I is not a model then there is no model
   3.1. no new clause is learned (This means that t is empty or a subset of s)

Some examples of equivalent termination conditions are :
1. until there is no new clause learned
2. until the empty clause is learned or I is a model
etc.

4. Combination of R and j : The Interpretation Construction Method Rj

Let us suppose that there exist sequences of unsatisfiable cnf's such that the length of the refutations by resolution grows more rapidly than any polynomial in the lengths of the cnf's. Then it is unfavorable to use a resolution method for proving the unsatisfiability of cnf's. The many well-known resolution methods are complete proof methods, i.e. for each unsatisfiable cnf they find a (relatively long) resolution proof. If the resolution methods do not find a proof then the cnf is satisfiable.
Perhaps it is better to use complete interpretation methods. An interpretation method is complete if for each satisfiable cnf it finds a model. If it does not find a model then the cnf is unsatisfiable. There are several experiences which indicate that complete interpretation methods possibly are superior to resolution.
1. The proof that a cnf is satisfiable is very short (it

consists in writing out a satisfying interpretation).
2. There exist fast algorithms for finding relatively good interpretations (algorithm j).
3. There exist complete learning algorithms for guiding the algorithm j.
4. Empirical results obtained by implementing Rj.

In this section we shall describe a refinement of the interpretation construction method R.

Let s be a finite set {c(1),c(2), ... ,c(p)} of clauses with set of variables v. Let y be a variable in v which is set at step k of algorithm R. Let yt be the set of clauses containing the literal y and yf the set of clauses containing the literal ¬y. Set wp:= weight(yt) and wn:= weight(yf).
The methods R and j are not ''compatible''. To combine them we need the following definition for describing algorithm Rj.

Def: [conflict variable at step k of R]
A variable y is a conflict variable if it is not a learning variable and
if y is set true and wn<wp then a clause becomes unsatisfied or
if y is set false and wn>wp then a clause becomes unsatisfied.

Example

```
 1:  a
 2:     b
 3:        c
 4:           d
 5: ¬a ¬b
 6: ¬a    ¬c
 7: ¬a       ¬d
 8:    ¬b ¬c
 9:    ¬b    ¬d
10:       ¬c ¬d
```

Let a,b,c,d be the chosen sequence of literals. Variable a is a conflict variable. It is not a learning variable, for if a is set true then no clause is unsatisfied. The literal a occurs only in clause 1. Hence wp=1/2. ¬a occurs in clauses 5,6 and 7. Hence wn=3/4. Therefore wn>wp and if a is set false then clause 1 becomes unsatisfied. Thus a is a conflict variable.
This is an example of an unsatisfiable cnf where each variable is a conflict variable. There are also satisfiable cnf's where each variable is a conflict variable.

The interpretation construction method Rj

```
repeat
1.  satis:= {}; I:= {}; var:= v; left:= s; assign to each clause
    c in the present s the weight w(c) := weight(c);
2.  for k:=1 to card(v) do
    begin
     a) for each element y of var do
        begin
            let  yt  be  the set of clauses in left containing y
            and  yf  the  set  of clauses in left containing ¬y;
            compute  the sum wp(y) of the weights of the clauses
            in  yt  and  the  sum  wn(y)  of  the weights of the
            clauses in yf ; determine whether y is a learning or
            a conflict variable
        end ;
     b) if there are variables which are not conflict variables
                then  choose  among these a variable y for which
                abs(wp(y)-wn(y)) is maximized (the weight of the
                cnf  when  y  is  set  is  diminished as much as
                possible)
        else
                choose  a  variable y for which abs(wp(y)-wn(y))
                is  minimized ; (the weight of the cnf when y is
                set is enlarged as least as possible)
     c) if y is not a conflict variable then
        begin
            if  wp(y)>wn(y)  or  if  wp(y)=wn(y)  and y is not a
            learning  variable and no clause becomes unsatisfied
            if y is set true, then b:=true else b:=false ;
        end else
            if wp(y)>wn(y) then b:=false else b:=true ;
     d) if b then
        begin
            I:=I + { y}; satis:=satis + yt; left:=left - yt; for
            each c in yf set w(c):= 2*w(c);
        end else
        begin
            I:=I + {¬y}; satis:=satis + yf; left:=left - yf; for
            each c in yt set w(c):= 2*w(c) ;
        end ;
      e) var := var - {y} ;
    end
3.  a)  for  the  first learning variable v compute the set t of
        clauses  defined  as  follows : let c(1),c(2), ... ,c(g1)
        be  the g1 clauses which were unsatisfied when v was set;
           let  d(1),d(2),  ...  ,d(h1)  be the h1 clauses which
        would have been unsatisfied if v had been set opposite;
        t  :=  {resolvents  of  c(g)  and  d(h)  with 1≤g≤g1 and
        1≤h≤h1} ;
    b)  choose  an element c of t; s:= s + {c}; (at this point a
        new clause is learned)
until there is no new clause learned or I is a model;
if I is not a model then there exists no model.
```

Theorem [completeness of Rj]
The interpretation construction algorithm Rj is a

complete interpretation construction method.

Proof
Use the fact that A2 with the termination condition ''there is no new clause learned or I is a model'' is complete. Algorithm Rj produces an unsatisfied clause if and only if a learning variable is set. ≠

Rj learns in each step of the repeat loop only one clause. Hence the weights change very slowly and the convergence of I to a model (if one exists) takes a long time. In the sequel we need an algorithm Rj1 (which learns more clauses) for stating a theorem and describing some experimental results. If we replace 2.d), 3.a) and 3.b) of algorithm Rj by the following statements and additionally initialize tunsat(:={}) in statement 1 we obtain algorithm Rj1.

2.d1) if b then
begin
I:=I + {y}; satis:=satis+yt; left:=left-yt; for each c in yf set w(c):= 2*w(c); if y is a learning variable then put each element of yf which is totally unsatisfied in the set tunsat;
end else
begin
I:=I+{¬y}; satis:=satis+yf; left:=left-yf; for each c in yt set w(c):= 2*w(c); if y is a learning variable then put each element of yt which is totally unsatisfied in the set tunsat;
end;

3.a1) for each element c of tunsat put the resolvents of c and all other clauses of s in the set t ;

3.b1) s := s + t ;
while a clause c occurs twice in s, eliminate one occurence of c.

Theorem [completeness of Rj1]
Rj1 is a complete interpretation construction method.

Proof
Rj1 learns at least the clauses which Rj would learn in the same situation. ≠

There are different forms of the termination condition of algorithm Rj1 which give the same result :
1. until there is no new clause learned;
2. until the empty clause is learned or I is a model;
3. until there is no new clause learned or I is a model;
Observe that the condition ''an old clause is learned'' does no longer imply unsatisfiability.

After some steps of the repeat loop of algorithm Rj1, s can contain clauses d1,d2 so that d2 subsumes d1, e.g. d1={a,b,c} and d2={a,b}. If s is satisfiable then it has a model in which d1 is satisfied not only by c. But the occurrence of c in d1 may change the weight information, so that c is set true although this might be wrong. Therefore d1 misleads the weight

information and it is better to delete d1. Hence a version of Rj1, called Rj1.sub1 which uses subsumption is usually faster. It can be used if we want to test for satisfiability. If we replace 3.b1) of algorithm Rj1 by the following statements we obtain algorithm Rj1.sub1.

3.b2) s := s + t ;
eliminate in s those clauses that contain other clauses;

Theorem [completeness of Rj1.sub1]
Rj1.sub1 is a complete interpretation construction algorithm.

Proof
Each model of a cnf s is also a model of the cnf which results when subsumption is applied to s. ≠

Now we define a version of Rj1, called Rj1.sub2, which uses subsumption and the ''resolution history''. We have a look at the statement : ''eliminate in s those clauses that contain other clauses'' in algorithm Rj1.sub1.

1. Suppose that a learned clause c in t subsumes an old clause d in s. d is replaced by c. Suppose that resolution was applied between d and a clause d1; let d2 be the resolvent of d and d1 and let l be the resolved literal. We distinguish two cases :
   1. c does not contain l.
      Then each clause learned with d is subsumed by c.

      Example :
      c= a b
      d= a b c
      d1= ¬c d e
      d2= a b d e

   2. c contains l.
      Now a smaller clause would be learned. Let d3 be the resolvent of c and d1. d2 contains at least one literal more than d3. Without affecting the satisfiability of s it is possible to replace d2 by d3.

      Example :
      c= a b
      d= a b c
      d1= ¬b d e
      d2= a c d e
      d3= a d e

2. Suppose that a learned clause c is subsumed by an old clause d. (Observe that this is not possible in algorithm Rj.) Then the clause c need not be learned.

The cases 1.1 and 2 are detected by algorithm Rj1.sub1. Algorithm Rj1.sub2 is able to treat case 1.2 too. It stores for each clause the resolution history, i.e. where the parent clauses and the resolvents are stored. (A parent clause d of a clause d1 is a clause such that resolution was applied between d

and d1.) If a learned clause subsumes an old clause then the possible abridgements can be executed according to 1.2.

5. The Behaviour of Algorithm Rj1 for Unsatisfiable Cnf's

For satisfiable cnf's algorithm Rj1 finds an interpretation I which satisfies the maximal number of clauses that can be satisfied. We call such an interpretation a maximal interpretation. Let max sat2.1 be the subproblem of max sat2 obtained by the reduction sat=3 < max sat2 in chapter 2. max sat 2.1 has the following definition :

Input: Set s of clauses, each containing at most 2 literals. s must have some additional properties which are described in chapter 2.
Property: There exists an interpretation which satisfies 7/10 of the clauses of s.

We make a slight change of algorithm Rj1. A variable max is introduced which must be set at the beginning of the algorithm. It indicates the maximal number of clauses of the input cnf which can be satisfied. If we want to recognize whether a cnf s belongs to sat we set max:= card(s) and if we want to recognize whether a cnf s belongs to max sat2.1 we set max:= 7/10 * card(s). The termination condition of Rj1 : ''until there is no new clause learned or I is a model'' is replaced by the condition ''until there is no new clause learned or max clauses of the input cnf are satisfied''. The last statement of algorithm Rj1 : ''if I is not a model then there exists no model'' is replaced by ''if I does not satisfy max of the clauses of the input cnf then there is no interpretation which satisfies max of the clauses of the input cnf''. Call this new algorithm Rj2.

Lemma

If algorithm Rj2 finds a maximal interpretation for each element of the problem max sat2.1, then P=NP.

Proof
Since each clause of an element of max sat2.1 contains at most 2 literals, algorithm Rj2 can learn clauses only of length ≤ 2. But the number of clauses of length ≤ 2 is a quadratic function of the number of variables. Therefore algorithm Rj2 halts after $O(n*n)$ steps, if n is the number of variables of the input cnf. If algorithm Rj2 always finds a maximal interpretation then it finds an interpretation I which satisfies 7/10 of the clauses of the input cnf if there is one. Hence Rj2 is a polynomial time algorithm for the NP-complete problem max sat2.1, if it always finds a maximal interpretation. $\neq$

Lemma

Algorithm Rj2 finds a maximal interpretation for each satisfiable cnf s.

Proof
Set max=card(s). Then Rj2 finds a model because it is complete and s is satisfiable. ≠

From the above lemmas we conclude:

Theorem
If algorithm Rj2 finds a maximal interpretation for each cnf in sat + max sat2.1 then P=NP.

So far, using a PASCAL-implementation, we have found no element in max sat2.1 for which an interpretation I exists which satisfies 7/10 of the clauses but so that I is not found by Rj2. But the input cnf's used are not relevant because they contain at most 29 variables.

Observe the interesting fact that the following problem is NP-complete.
Input: Set s of clauses which is saturated by resolution(i.e. each resolvent of two clauses in s belongs to s); integer k and a subset t of s.
Property: There is a truth assignment which satisfies at least k clauses of t.

6. Restricted Learning

We suppose that the input cnf's for the following class of algorithms, called Rj1.sub1.k , k=4,5,6, ... are in sat3. Rj1.sub1.k is the same algorithm as Rj1, except that the set t can contain clauses of at most length k.
Algorithm Rj1.sub1.k generates at most O(m**k) distinct clauses, where m is the number of variables. Hence Rj1.sub1.k runs in polynomial time.
We would like to know :
(1) Does the procedure work correctly for some constant integer k≥4 and if not
(2) Does it work correctly for k=k(n), some slowly growing function of n, where n is the size of the input ? (Obviously k=m is sufficient)

A positive answer to (1) would imply P=NP. A positive answer to (2) would yield a subexponential algorithm for sat (and hence for all NP-complete problems) provided k(n) is asymptotically slower than n**e for every e>0.
It follows from [GA75] that for each k and for infinitely many unsatisfiable cnf's in sat3 Rj1.sub1.k will not generate the empty clause. But this does not imply that for each k algorithm Rj1.sub1.k will not find a model for infinitely many satisfiable cnf's.
If the answer to question (1) is negative we would like to know:
a) For which cnf's does the procedure work correctly ?
b) How good is the best approximation which is found for the cnf's for which Rj1.sub1.k does not work correctly ?

c) What is the improvement if we use Rj1.sub1.k+1 instead of Rj1.sub1.k ?

7. Conclusions and Open Problems

We have considered new procedures for determining whether a set of clauses is satisfiable: algorithms A and Rj and their related versions. For proving the completeness of these algorithms we used methods of resolution proof theory, i.e. we used the fact that for unsatisfiable cnf's the above algorithms generate a resolution proof and therefore decide correctly if a cnf is unsatisfiable. But we believe that for algorithm Rj1 there exist other proof techniques. We conjecture that even incomplete resolution strategies, perhaps even polynomial ones , generate enough clauses such that the weight information is sufficient for finding an interpretation if the input cnf is satisfiable. If, in fact, there exists an incomplete polynomial resolution strategy with this property then P=NP. If our conjecture is true then the lower bounds and similar results for resolution as obtained in [GA74,GA75] and [TS68] will not apply to algorithm Rj1. For example, if we restrict the length of clauses which can be learned by a resolution method for some cnf in sat3 (''satisfiability'' with at most three literals per clause) by a constant $k \geq 4$, then it is known that all these resolution methods are incomplete. But with the same restrictions on the length of clauses learned by algorithm Rj1.sub1, we have so far found no satisfiable cnf for which the restricted algorithm Rj1 decided incorrectly. (We used a sample of about 30 cnf's each containing 25 variables and 200 clauses. These cnf's were constructed such that the ''weight information'' was bad for many variables, i.e. there existed no model if such a variable was set according to the ''weight information'' of the input cnf.)
Finally some interesting open questions are mentioned. Let s be a satisfiable cnf.

(1) Which is the minimal set m of resolvents that must be added to s so that Rj computes a model in one step, i.e. the repeat loop has to be executed only once ? (Remark : All resolvents are sufficient [RO65]. If all resolvents are added we can choose an arbitrary variable which must be set so that no clause becomes unsatisfied. This algorithm must yield a model.)
Which functions of the length of s give an upper bound for card(m) for all cnf's ?

(2) The same question as (1), but abs(wp(y)-wn(y)) is replaced in algorithm Rj by max(wp(y)/wn(y),wn(y)/wp(y)). (if wn(y)=0 or wp(y)=0 then max(wp(y)/wn(y),wn(y)/wp(y))=∞)

(3) Are there other interesting functions than wp and wn defined on the literals ? What is the influence of the functions maximization and minimization in statement 2.b) of algorithm Rj on the running time ?

Let us assume the following answer to question (1). For each satisfiable cnf s of length n there have to be learned at most n**k resolvents for some fixed k, i.e. the minimal number of resolvents to be learned for each cnf of length n is bounded by n**k. Then the minimal set can be computed nondeterministically in polynomial time. This would only yield a new polynomial nondeterministic procedure for sat.
A set f of resolvents of a cnf s is said to be sufficient if Rj finds the model in one step for s+f. The minimal set m mentioned above is the smallest sufficient set. If we are able to compute a sufficient set for each satisfiable cnf s deterministically in polynomial time, then P=NP.

Acknowledgements
****

I wish to thank Professor E.Engeler who made this work possible through his support and encouragement. I am also grateful to E.Marmier and E.Graf for their interest and many helpful discussions and suggestions.

REFERENCES +

CO71. COOK S.A., THE COMPLEXITY OF THEOREM-PROVING PROCEDURES, 3.STOC (1971), 151-158.

GA74. GALIL Z., ON THE COMPLEXITY OF RESOLUTION PROCEDURES FOR THEOREM PROVING, TECHNICAL REPORT, DEPARTMENT OF COMPUTER SCIENCE, CORNELL UNIVERSITY, 1974.

GA75. GALIL Z., ON THE VALIDITY AND COMPLEXITY OF BOUNDED RESOLUTION PROCEDURES, 7.STOC (1975).

GAR72. GAREY M.R., GRAHAM R.L., ULLMAN J.D., WORST CASE ANALYSIS OF MEMORY ALLOCATION ALGORITHMS, 4.STOC (1972), 143-150.

GAR74. GAREY M.R., JOHNSON D.S., SOME SIMPLIFIED NP-COMPLETE PROBLEMS, 6.STOC (1974), 47-63.

JO72. JOHNSON D.S., FAST ALLOCATION ALGORITHMS, 13.SWAT (1972), 144-154.

JO73. JOHNSON D.S., APPROXIMATION ALGORITHMS FOR COMBINATORIAL PROBLEMS, 5.STOC (1973), 38-49.

+ The used abbreviations are explained in the bibliography.

KA72. KARP R.M., REDUCIBILITY AMONG COMBINATORIAL PROBLEMS, IN ''COMPLEXITY OF COMPUTER COMPUTATIONS'', R.E. MILLER AND J.W. THATCHER, EDS., PLENUM PRESS, NEW YORK, 1972, 85-104.

KE70. KERNIGHAN B.W., LIN S., AN EFFICIENT HEURISTIC FOR PARTITIONING GRAPHS, BELL. SYST.TECH.J. 49 (1970), 291-308.

LI73. LIN S., KERNIGHAN B.W., AN EFFECTIVE HEURISTIC ALGORITHM FOR THE TRAVELING SALESMAN PROBLEM, BELL SYST.TECH.J. 52 (1973), 498-516.

RA74. RABIN M.O., THEORETICAL IMPEDIMENTS TO ARTIFICIAL INTELLIGENCE, IFIP 74, 615-619.

RO65. ROBINSON J.A., A MACHINE-ORIENTED LOGIC BASED ON THE RESOLUTION PRINCIPLE, J.ACM, VOL 12, NO 1, 23-41.

SA72. SAHNI S., SOME RELATED PROBLEMS FROM NETWORK FLOWS, GAME THEORY, AND INTEGER PROGRAMMING, 13.SWAT (1972), 130-138.

SA74. SAHNI S., GONZALES T., P-COMPLETE PROBLEMS AND APPROXIMATE SOLUTIONS, 15.SWAT (1974)

SE73. SETHI R., COMPLETE REGISTER ALLOCATION PROBLEMS, 5.STOC (1973), 182-195.

TS68. TSEITLIN G.S., ON THE COMPLEXITY OF DERIVATIONS IN THE PROPOSITIONAL CALCULUS, STRUCTURES IN CONSTRUCTIVE MATHEMATICAL LOGIC, PART II, A.O. SILENKO(ED), 1968, 115-125.

UL73. ULLMAN J.D., POLYNOMIAL COMPLETE SCHEDULING PROBLEMS, 4TH SYMPOSIUM ON OPERATING SYSTEM PRINCIPLES (1973), 96-101 AND IN J.COMPUTER + SYSTEM SCIENCES 10, (1975), 384-393.

APPENDIX
****

BIBLIOGRAPHY OF ''P=NP AND RELATED PROBLEMS''
****

THE BIBLIOGRAPHY IS DIVIDED INTO TEN PARTS.

PART A : COMPLETENESS PROOFS OF PROOF PROCEDURES FOR PROPOSITIONAL LOGIC

BIBLIOGRAPHY

PART B : APPROXIMATION ALGORITHMS FOR NP-COMPLETE PROBLEMS

PART C : NP-COMPLETE LANGUAGES, CLASSIFICATION OF NP

PART D : LENGTH OF PROOFS IN THE PROPOSITIONAL CALCULUS

PART E : POLYNOMIAL TIME ALGORITHMS FOR PROBLEMS WHICH ARE ''NEARLY'' NP-COMPLETE

PART F : POLYNOMIAL-TIME REDUCIBILITIES

PART G : CLASSIFICATION OF P

PART H : MACHINES FOR WHICH OBVIOUSLY P=NP

PART J : ALGORITHMS FOR NP-COMPLETE PROBLEMS WHICH WORK QUICKLY ON PRACTICAL PROBLEMS

PART K : COMBINATIONAL COMPLEXITY OF BOOLEAN FUNCTIONS

ABBREVIATIONS

STOC PROCEEDINGS OF THE ANNUAL ACM SYMPOSIUM ON THEORY OF COMPUTING

SWAT PROCEEDINGS OF THE ANNUAL IEEE SYMPOSIUM ON SWITCHING AND AUTOMATA THEORY (FOUNDATIONS OF COMPUTER SCIENCE)

MI MACHINE INTELLIGENCE (B. MELTZER AND D. MICHIE, EDS.), AMERICAN ELSEVIER, NEW YORK

J.ACM JOURNAL OF THE ASSOCIATION FOR COMPUTING MACHINERY

BELL SYST.TECH.J. BELL SYSTEM TECHNICAL JOURNAL

EIK ELEKTRONISCHE INFORMATIONSVERARBEITUNG

J.COMPUTER + SYSTEM SCIENCES JOURNAL OF COMPUTER AND SYSTEM SCIENCES

PART A

AN68. ANDREWS P.B., RESOLUTION WITH MERGING, J.ACM, VOL 15, NO

3, 367-381.

AND70. ANDERSON R., BLEDSOE W.W., A LINEAR FORMAT FOR RESOLUTION WITH MERGING AND A NEW TECHNIQUE FOR ESTABLISHING COMPLETENESS, J.ACM, VOL 17, NO 3, 525-534.

CH73. CHANG C.L., LEE R.C., SYMBOLIC LOGIC AND MECHANICAL THEOREM PROVING, ACADEMIC PRESS, NEW YORK AND LONDON, 1973.

ME68. MELTZER B., SOME NOTES ON RESOLUTION STRATEGIES, MI 3,71-76.

RO65. ROBINSON J.A., A MACHINE-ORIENTED LOGIC BASED ON THE RESOLUTION PRINCIPLE, J.ACM, VOL 12, NO 1, 23-41.

PART B

GAR72. GAREY M.R., GRAHAM R.L., ULLMAN J.D., WORST CASE ANALYSIS OF MEMORY ALLOCATION ALGORITHMS, 4.STOC (1972), 143-150.

JO72. JOHNSON D.S., FAST ALLOCATION ALGORITHMS, 13.SWAT (1972), 144-154.

JO73. JOHNSON D.S., APPROXIMATION ALGORITHMS FOR COMBINATORIAL PROBLEMS, 5.STOC (1973), 38-49.

KO75. KOHLER W.H., STEIGLITZ K., EXACT, APPROXIMATE, AND GUARANTEED ACCURACY ALGORITHMS FOR THE FLOW-SHOP PROBLEM N/2/F/F, J.ACM, VOL 22, NO 1, 106-114.

ROS74. ROSENKRANTZ D.J., STEARNS R.E., LEWIS P.M., APPROXIMATE ALGORITHMS FOR THE TRAVELING SALESPERSON PROBLEM, 15.SWAT (1974)

SA74. SAHNI S., GONZALES T., P-COMPLETE PROBLEMS AND APPROXIMATE SOLUTIONS, 15.SWAT (1974)

SA75. SAHNI S., APPROXIMATE ALGORITHMS FOR THE 0/1 KNAPSACK PROBLEM, J.ACM, VOL 22, NO 1, 115-124.

PART C

BO72. BOOK R.V., COMPLEXITY CLASSES OF FORMAL LANGUAGES, IN : AUTOMATA, LANGUAGES AND PROGRAMMING, PROCEEDINGS OF A SYMPOSIUM ORGANIZED BY IRIA, EDITED BY M.NIVAT. 1972.

CON74. CONSTABLE R.L., HUNT B.H., ON THE COMPUTATIONAL COMPLEXITY OF SCHEME EQUIVALENCE, TECHNICAL REPORT TR 74-201, DEPARTMENT OF COMPUTER SCIENCE, CORNELL UNIVERSITY, ITHACA, NEW YORK 14850.

BIBLIOGRAPHY

CO71. COOK S.A., THE COMPLEXITY OF THEOREM-PROVING PROCEDURES, 3.STOC (1971), 151-158.

DO74. DOBKIN D., LIPTON R.J., ON SOME GENERALIZATION OF BINARY SEARCH, 6.STOC (1974), 310-316.

DU74. DUNHAM B., WANG H., TOWARD FEASIBLE SOLUTIONS OF THE TAUTOLOGY PROBLEM, RC 4924, IBM THOMAS J. WATSON RESEARCH CENTER, YORKTOWN HEIGHTS, N.Y. 10598.

FA73. GENERALIZED FIRST-ORDER SPECTRA AND POLYNOMIAL TIME RECOGNIZABLE SETS, SIAM-AMS PROCEEDINGS, VOLUME 7, EDITED BY R.M. KARP.

GAR74. GAREY M.R., JOHNSON D.S., SOME SIMPLIFIED NP-COMPLETE PROBLEMS, 6.STOC (1974), 47-63.

KA72. KARP R.M., REDUCIBILITY AMONG COMBINATORIAL PROBLEMS, IN "COMPLEXITY OF COMPUTER COMPUTATIONS", R.E. MILLER AND J.W. THATCHER, EDS., PLENUM PRESS, NEW YORK, 1972, 85-104.

RA74. RABIN M.O., THEORETICAL IMPEDIMENTS TO ARTIFICIAL INTELLIGENCE, IFIP 74, 615-619.

SA72. SAHNI S., SOME RELATED PROBLEMS FROM NETWORK FLOWS, GAME THEORY, AND INTEGER PROGRAMMING, 13.SWAT (1972), 130-138.

SC75. SCHNORR C.P., SATISFIABILITY IS QUASI-LINEAR COMPLETE IN NQL, UNIVERSITAET FRANKFURT, 1975.

SE73. SETHI R., COMPLETE REGISTER ALLOCATION PROBLEMS, 5.STOC (1973), 182-195.

SP75. SPECKER E., STRASSEN V., KOMPLEXITAET VON ENTSCHEIDUNGSPROBLEMEN, SPRINGER, 1975.

UL73. ULLMAN J.D., POLYNOMIAL COMPLETE SCHEDULING PROBLEMS, 4TH SYMPOSIUM ON OPERATING SYSTEM PRINCIPLES (1973), 96-101 AND IN J.COMPUTER + SYSTEM SCIENCES 10, (1975), 384-393.

PART D

CO74. COOK S., RECKHOW R., ON THE LENGTH OF PROOFS IN THE PROPOSITIONAL CALCULUS, 6.STOC (1974), 135-148.

CO75. COOK S.A., FEASIBLY CONSTRUCTIVE PROOFS AND THE PROPOSITIONAL CALCULUS, 7.STOC (1975).

GA74. GALIL Z., ON THE COMPLEXITY OF RESOLUTION PROCEDURES FOR THEOREM PROVING, TECHNICAL REPORT, DEPARTMENT OF COMPUTER SCIENCE, CORNELL UNIVERSITY, 1974.

BIBLIOGRAPHY

GA75. GALIL Z., ON THE VALIDITY AND COMPLEXITY OF BOUNDED RESOLUTION PROCEDURES, 7.STOC (1975).

TS68. TSEITLIN G.S., ON THE COMPLEXITY OF DERIVATIONS IN THE PROPOSITIONAL CALCULUS, STRUCTURES IN CONSTRUCTIVE MATHEMATICAL LOGIC, PART II, A.O. SILENKO(ED), 1968, 115-125.

PART E

BA74. BATNI R.P., RUSSELL J.D., KIME CH.R., AN EFFICIENT ALGORITHM FOR FINDING AN IRREDUNDANT SET COVER, J.ACM, VOL 12, NO 3, 351-355.

HAR72. HARRIS R.V., A POLYNOMIAL BOUND ON THE COMPLEXITY OF THE DAVIS PUTNAM PROCEDURE AS APPLIED TO SYMMETRIZABLE PROPOSITIONS, PHD. THESIS, DEPT. OF COMPUTER SCIENCE CORNELL UNIVERSITY, AUGUST 1972.

HO74. HOPCROFT J.E., WONG J.K., LINEAR TIME ALGORITHM FOR ISOMORPHISM OF PLANAR GRAPHS, 6.STOC (1974), 172-184.

PART F

GI74. GILL J.T., AXIOMATIC INDEPENDENCE OF THE QUESTION NP=P, DEPARTMENT OF ELECTRICAL ENGINEERING, STANFORD UNIVERSITY, STANFORD, CALIFORNIA 94305.

LA74. LADNER R., LYNCH N., SELMAN A.L., COMPARISON OF POLYNOMIAL-TIME REDUCIBILITIES, 6.STOC (1974), 110-122.

LA75. LADNER R.E., ON THE STRUCTURE OF POLYNOMIAL TIME REDUCIBILITY, J.ACM, VOL 22, NO 1, 155-171.

MEH74. MEHLHORN K., POLYNOMIAL AND ABSTRACT SUBRECURSIVE CLASSES, 6.STOC (1974), 96-109.

PART G

JON74. JONES N.D., LAASER W.T., COMPLETE PROBLEMS FOR DETERMINISTIC POLYNOMIAL TIME, 6.STOC (1974), 40-46.

LA75. LADNER R.E., THE CIRCUIT VALUE PROBLEM IS LOG SPACE COMPLETE FOR P, SIGACT NEWS, JANUARY 1975, 18-20.

BIBLIOGRAPHY

PART H

HA74. HARTMANIS J., SIMON J., ON THE POWER OF MULTIPLICATION IN RANDOM ACCESS MACHINES, 15.SWAT(1974).

PR74. PRATT V.R., RABIN M., STOCKMEYER L.J., A CHARACTERIZATION OF THE POWER OF VECTOR MACHINES, 6.STOC (1974), 122-134.

PART J

KE70. KERNIGHAN B.W., LIN S., AN EFFICIENT HEURISTIC FOR PARTITIONING GRAPHS, BELL SYST.TECH.J. 49 (1970), 291-308.

LI73. LIN S., KERNIGHAN B.W., AN EFFECTIVE HEURISTIC ALGORITHM FOR THE TRAVELING SALESMAN PROBLEM , BELL SYST.TECH.J. 52 (1973), 498-516.

PART K

FI74. FISCHER M.J., LECTURES ON NETWORK COMPLEXITY, UNIVERSITY OF FRANKFURT, JUNE 1974.

PA74. PAUL W.J., 2,25*N-LOGER BOUND ON THE COMPUTATIONAL COMPLEXITY OF BOOLEAN FUNCTIONS, TR74-222, DEPARTMENT OF COMPUTER SCIENCE, CORNELL UNIVERSITY, ITHACA, NEW YORK 14853.

UH72. UHLIG, UEBER DIE ANZAHL DER UNTERFUNKTIONEN EINER BOOLESCHEN FUNKTION ZUR CHARAKTERISIERUNG DER KOMPLIZIERTHEIT IHRER REALISIERUNG, EIK 8 (1972)5, 255-268.

Berichte des Instituts für Informatik

Nr. 1	Niklaus Wirth:	The Programming Language Pascal (out of print)
Nr. 2	Niklaus Wirth:	Program development by step-wise refinement (out of print)
Nr. 3	Peter Läuchli:	Reduktion elektrischer Netzwerke und Gauss'sche Elimination
Nr. 4	Walter Gander, Andrea Mazzario:	Numerische Prozeduren I
Nr. 5	Niklaus Wirth:	The Programming Language Pascal (Revised Report)
Nr. 6	C.A.R. Hoare, Niklaus Wirth:	An Axiomatic Definition of the Language Pascal (out of print)
Nr. 7	Andrea Mazzario, Luciano Molinari:	Numerische Prozeduren II
Nr. 8	E. Engeler, E. Wiedmer, E. Zachos:	Ein Einblick in die Theorie der Berechnungen
Nr. 9	Hans-Peter Frei:	Computer Aided Instruction: The Author Language and the System THALES
Nr.10	K.V. Nori, U. Ammann, K. Jensen, H.H. Nägeli:	The PASCAL 'P' Compiler: Implementation Notes
Nr.11	G.I. Ugron, F.R. Lüthi:	Das Informations-System ELSBETH
Nr.12	Niklaus Wirth:	PASCAL-S: A Subset and its Implementation
Nr.13	U. Ammann:	Code Generation in a PASCAL Compiler
Nr.14	Karl Lieberherr:	Toward Feasible Solutions of NP-Complete Problems