SIAM J. COMPUT.
Vol. 3, No. 4, December 1974

COMPUTATIONALLY RELATED PROBLEMS*

SARTAJ SAHNI†

Abstract. We look at several problems from areas such as network flows, game theory, artificial intelligence, graph theory, integer programming and nonlinear programming and show that they are related in that any one of these problems is solvable in polynomial time iff all the others are, too. At present, no polynomial time algorithm for these problems is known. These problems extend the equivalence class of problems known as P-Complete. The problem of deciding whether the class of languages accepted by polynomial time nondeterministic Turing machines is the same as that accepted by polynomial time deterministic Turing machines is related to P-Complete problems in that these two classes of languages are the same iff each P-Complete problem has a polynomial deterministic solution. In view of this, it appears very likely that this equivalence class defines a class of problems that cannot be solved in deterministic polynomial time.

Key words. complexity, polynomial reducibility, deterministic and nondeterministic algorithms, network flows, game theory, optimization, AND/OR graphs

1. Introduction. Cook [3] showed that determining whether the class of languages accepted by nondeterministic Turing machines operating in polynomial time was the same as that accepted by deterministic polynomial time bounded Turing machines was as hard as deciding if there was a deterministic polynomial algorithm for the satisfiability problem of propositional calculas (actually, Cook showed that there was a polynomial algorithm for satisfiability iff the deterministic and nondeterministic polynomial time languages were the same). This problem about equivalence of the two classes of languages is a long-standing open problem from complexity theory. Intuitively, it seems that the two classes are not the same. Consequently there may be no polynomial algorithm for the satisfiability problem. Further empirical evidence that the two classes may not be the same was provided by Karp in [5], where he showed that many other problems like the traveling salesman problem, finding the maximum clique of a graph, minimal colorings of graphs, minimal set covers, etc., had polynomial algorithms iff the two classes of languages were the same. In view of this relationship amongst all these problems, we can say that there is strong evidence to believe that there is no polynomial algorithm for any of the problems given in Karp [5]. However, no formal proof of this (if this is true) is available at this time.

The equivalence class of problems having the property that each member of the class has a polynomial algorithm iff nondeterministic and deterministic polynomial languages are the same is known as P-Complete. In [5], Karp presents 21 members of this class. The purpose of this paper is to extend the class of known P-Complete problems. Specifically, we show that several important problems from

* Received by the editors July 18, 1973, and in revised form April 6, 1974. The research reported here is part of the author's Ph.D. dissertation, Cornell University. An earlier version of these results was presented at the 1972 IEEE Annual Conference on Switching and Automata Theory. This research was supported in part by the National Science Foundation under Grant GJ-33169.

† Department of Computer Science, Cornell University, Ithaca, New York. Now at Department of Computer, Information, and Control Sciences, University of Minnesota, Minneapolis, Minnesota 55455.

areas such as artificial intelligence, game theory, graph theory, network flows and integer optimization are P-Complete. We also introduce the concept of P-Hard.

The rest of this section will be devoted to definitions and establishing our notation. In §2 the new members of the classes P-Complete and P-Hard are presented.

1.1. Definitions. As our computational model we shall use Turing machines. (See Hopcroft and Ullman [4] for a standard treatment of this model.) The reader unfamiliar with deterministic and nondeterministic polynomial time computations should see Karp [5].

DEFINITION 1. P (NP) is the class of languages recognizable by deterministic (nondeterministic) polynomial time bounded one-tape Turing machines.

Open problem. "Is P = NP?" We may rephrase this as, "Is there a deterministic polynomial algorithm for all languages in NP?" Call this *Problem* P1.

DEFINITION 2.[1] A *problem* is a total function $f: \Sigma^* \to 2^{\Sigma^*}$, which takes each finite string to a nonempty subset of strings. (Informally, the finite string represents an encoding of the input or data and f maps this onto a solution set. Thus, for language recognition problems, $f: \Sigma^* \to (0, 1)$, where $f = 0$ iff the input string is not in the language.)

We consider algorithms which, given $x \in \Sigma^*$, produce some $y \in f(x)$. The computing time of the algorithm will be measured as a function of the length of x ($|x|$). (All algorithms will be deterministic unless otherwise stated.)

DEFINITION 3. A problem L will be said to be P-Reducible to a problem M (written $L \propto M$) iff a polynomial algorithm for M implies a polynomial algorithm for L. That is, from a deterministic polynomial algorithm for M we can construct a deterministic polynomial algorithm for L.

DEFINITION 4. A problem L is P-Hard iff a polynomial algorithm for L implies P = NP.

DEFINITION 5. Two problems L and M are P-Equivalent iff $L \propto M$ and $M \propto L$.

Clearly, P-Reducible is a transitive relation and P-Equivalent is an equivalence relation.

DEFINITION 6. P-Complete (PC) is the equivalence class of P-Equivalent problems having a polynomial algorithm iff P = NP.

Our definition of P-Complete differs from that used by Karp [5]. However, it can easily be shown that any problem which is polynomial-Complete under his definition is P-Complete. The reverse, however, may not be true. (No proof of the equivalence or nonequivalence of the two definitions is known.) Note that all P-Complete problems are also P-Hard. In some cases, we may only be able to show the relation P-Hard rather than the stronger P-Complete relation. We shall often write $L \propto$ P1 when we mean "if P = NP, then L is polynomial solvable" and P1 $\propto L$ when we mean "if L is polynomial solvable, then P = NP". No ambiguity should arise from this double use of the symbol $\propto$.

[1] The author is grateful to an anonymous referee for suggesting this definition of a problem which encompasses both language recognition and optimization problems. Σ is the tape alphabet of the Turing machine and Σ^* is the set of all finite length strings or words from the alphabet Σ.

There are several ways to show that a problem L is P-Complete. For instance, one could show L to be P-Equivalent to M, where M is a problem already known to be P-Complete, or show that L has a polynomial algorithm iff $P = NP$, etc. Most of the proofs in the next section will adopt the following approach: (i) show that "if $P = NP$, then L" is polynomial solvable, i.e., $L \propto (P = NP)$, and (ii) show $M \propto L$, where M is a problem known to be P-Complete. M will usually be the satisfiability problem of propositional calculus (see Karp [5] for a formal definition of this problem).

2. P-Complete and P-Hard problems. In this section we shall show that several frequently encountered problems in various areas such as network flows, game theory, graph theory, nonlinear and linear optimization are either P-Complete or at least P-Hard. The reductions are easily seen to be effective. The polynomial factors involved in the reduction are small (usually a constant or a polynomial of degree 1).

2.1. Some known P-Complete problems. To prove some of the reductions, we shall make use of some known members of PC. A brief description of these members is given below. (A more exhaustive list may be found in Karp [5].)

(i) *Propositional calculus.*

(a) *Satisfiability.* Given a formula from the propositional calculus, in conjunctive normal form (CNF), is there an assignment of truth values for which it is "true"?

(b) *Satisfiability with exactly* 3 *literals per clause.* This is the same as (a), except that each clause of the formula now has exactly 3 literals.

(c) *Tautology.* Given a formula, from the propositional calculus, in disjunctive normal form (DNF), does it have the value "true" for all possible assignments of truth values.

(ii) *Sum of subsets of integers.* Given a multiset $S = (s_1, \cdots, s_r)$ of positive integers and a positive integer M, does there exist a submultiset of S that sums to M? (This problem is called the Knapsack problem in [5]. However, here we shall denote by "Knapsack problem" a similar integer optimization problem.) Note that a multiset is a collection of elements that may not necessarily be distinct.

(iii) *Maximum independent set.* Let G be a graph with vertices $v_1, v_2, \cdots, v_n$. A set of vertices is *independent* if no two members of the set are adjacent in G. A *maximum* independent set is an independent set that has a maximum number of vertices.

(iv) *Directed Hamiltonian cycle.* Given a directed graph G, does it have a cycle that includes each vertex exactly once?

THEOREM 2.1. *The following problems are in* PC:

(i) *Satisfiability, satisfiability with exactly three literals per clause, tautology*;

(ii) *Sum of subsets of integers*;

(iii) *Maximum independent set of a graph*;

(iv) *Directed Hamiltonian cycle.*

Proof. (i) is proved in Cook [3]. The rest are proved in Karp [5].

Cook [3] actually shows that satisfiability with at most three literals per clause is P-Complete. From this result one may trivially show that satisfiability with exactly three literals per clause is P-Complete. We show how to convert a two-literal clause into an equivalent pair of three-literal clauses. Let $(x_1 + x_2)$ be the clause and y a variable not occurring in the formula. Then $(x_1 + x_2 + y) \wedge (x_1 + x_2 + \bar{y})$ is satisfiable iff the two-literal clause is. All two-literal clauses may be replaced by pairs of three-literal clauses as above. This at most doubles the number of clauses. Clauses with only one literal can be deleted, the literal determining the truth assignment to that variable.

2.2. Integer network flows. We define the following network problems.

Problem N(i). *Network flows with multipliers.* Let G be a directed graph with vertices $\hat{s}_1, \hat{s}_2, v_1, \cdots, v_n$ and edges (arcs) $e_1, e_2, \cdots, e_m$. Let $w^-(v)$ be the set of arcs directed into vertex v and $w^+(v)$ those arcs directed away from v.

G will be said to denote a *network with multipliers* if:

(a) the source $\hat{s}_1$ of the network has no incoming arcs, i.e., $w^-(\hat{s}_1) = \emptyset$;

(b) the sink $\hat{s}_2$ has no outgoing arcs, i.e., $w^+(\hat{s}_2) = \emptyset$;

(c) to every vertex v_i (excluding the source and sink) there corresponds an integer $h_i > 0$, called its *multiplier.*

(d) to each edge e_i there corresponds an interval $[a_i, b_i]$;

Conditions (a)–(d) are said to define a *transportation* network.

We are required to find a flow vector, with integer entries, $\Phi = (\phi_1, \phi_2, \cdots, \phi_m)$ such that the following conditions hold.

Condition 1. $a_i \leqq \phi_i \leqq b_i$;

Condition 2. $h(v) \sum_{i \in w^-(v)} \phi_i = \sum_{i \in w^+(v)} \phi_i$ for all $v \in V(G)$, $v \neq \hat{s}_1$ $v \neq \hat{s}_2$;

Condition 3. $\sum_{i \in w^-(s_2)} \phi_i$ is maximized.

In what follows, we assume $a_i = 0$.

Problem N(ii). *Multicommodity network flows.* The transportation network is as above, but now $h(v) = 1$ for all v in $V(G)$. We have, however, several different commodities $c_1, c_2, \cdots, c_n$, and some arcs may be labeled, i.e., they can carry only certain commodities. Each arc is assigned a capacity, and we wish to know whether a flow $R = (r_1, r_2, \cdots, r_n)$, where r_i is the quantity of the ith commodity, is feasible in the network.

Problem N(iii). *Integer flows with homologous arcs.* The transportation network remains the same. Also, $h(v) = 1$ and there is only one commodity. Certain arcs are paired, and we require that if arcs i, j are paired, then $\phi_i = \phi_j$. We wish to know if a flow of at least F is feasible in the network.

Problem N(iv). *Integer flows with bundles.* The arcs in the network are divided into sets $I_1, \cdots, I_k$ (the sets may overlap). Each set is called a *bundle*, and with each bundle is associated a capacity C_i. We wish to know if a flow $\geqq F$ is feasible in the network:

$$\sum_{i \in I_j} \phi_i \leqq C_j, \qquad 1 \leqq j \leqq k$$

and

$$h(v) = 1 \quad \forall\, v \in V(G).$$

THEOREM 2.2. *Problems* N(i)–N(iv) *are* in PC.

Proof. (a) N(i), N(ii), N(iii), N(iv) $\propto$ P1. The nondeterministic turing machine (NDTM) just guesses the flows in each arc and then verifies Conditions 1 and 2. In addition, it does the following:

(i) for N(ii) it verifies that the resultant flow is $\geqq R$;

(ii) for N(iii) the "homologous conditions" are checked and $\sum_{i\in w^-(\hat{s}_2)} \phi_i \geqq F$ verified;

(iii) for N(iv) the bundle restrictions are checked and $\sum_{i\in w^-(\hat{s}_2)} \phi_i \geqq F$ verified.

If in N(i) we replace the max $\sum_{i\in w^-(\hat{s}_2)} \phi_i$ requirement to:

$$T\colon \sum_{i\in w^-(\hat{s}_2)} \phi_i \geqq F, \tag{2.2.1}$$

then from the above it follows that $T \propto$ P1.[2] To see N(i) $\propto T$, we note that if the length of the input on a Turing machine's tape is n, then the largest number it can represent is c^n, for some constant c which depends only on the Turing machine. Hence the maximum capacity of an arc is bounded by c^n and so max $\sum_{i\in w^-(\hat{s}_2)} \phi_i \leqq k^n$, for some constant k. Now, assume there is a polynomial $[p(n)]$ algorithm for T. Then, using the method of bisection, we can determine max $\sum_{i\in w^-(\hat{s}_2)} \phi_i$ in at most $\log_2 k^n = n \log_2 k$ applications of T. This, therefore, gives a polynomial algorithm for N(i). Therefore N(i) $\propto T \propto$ P1, and from the transitivity of $\propto$ we conclude N(i) $\propto$ P1. Clearly, this proof technique can be used to show N(iii) and N(iv) to be complete when they are changed to maximization problems.

(b) We now show the reduction for N(i)–N(iv), in the other direction.

(i) Sum of subsets of integers $\propto$ N(i). We construct a network flow problem of type N(i) such that max $\sum_{i\in w^-(\hat{s}_2)} \phi_i = M$ iff there is a submultiset of $S = \{s_1, \cdots, s_r\}$ that sums to M.

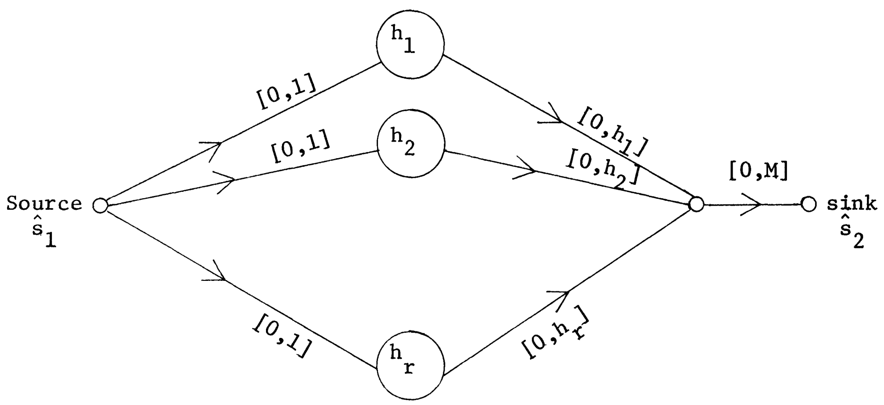


FIG. 2.2.1. *Construction for sum of subsets* $\propto$ N(i)

Consider the construction of Fig. 2.2.1 with $h_i = s_i$, $1 \leqq i \leqq r$. Clearly

$$\max \sum_{i\in w^-(\hat{s}_2)} \phi_i = M$$

iff some submultiset of S sums to M.

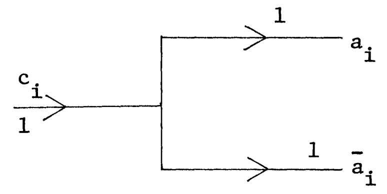
(ii) Tautology $\propto$ N(ii). Suppose that the formula P in DNF has n variables $a_1, a_2, \cdots, a_n$. We shall construct a multicommodity network with n commodities

[2] Recall that P1 was defined in §1.2 to be the decision problem: is NP = P?

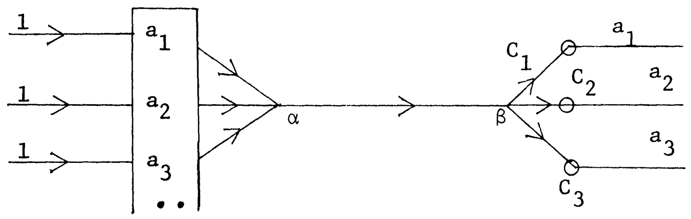
$c_1, c_2, \cdots, c_n$ such that the flow $R(1, \cdots, 1)$ is feasible iff P is not a tautology. The network of Fig. 2.2.2 realizes this.

Discussion.

[A] This section of the network ensures that there is a flow through only one of the nodes a_i or $\bar{a}_i$. In terms of the formula A, a flow through a_i means a truth assignment of 1 to a_i while a flow through $\bar{a}_i$ means an assignment of 0 to a_i.



[B] For each clause (K_i) in P we have a section of the form



If there are j literals in the clause, then arc (α, β) is assigned a capacity of $j - 1$. This requires that the truth assignments be such that clause k_i is false (as at least one term in it is false). Node β is where the "multicommodity" property of the network is used. Here the flow through α is correctly separated into its components, i.e., we are able to get back the truth values of the variables. The components for each flow are connected in series as in Fig. 2.2.2.

We now want to know if a flow $R = (1, 1, \cdots, 1)$ is feasible. It is easy to see that such a flow is possible iff there is a truth assignment to $a_1, \cdots, a_n$ for which each clause is false, i.e., iff P is not a tautology.

(iii) Tautology $\propto$ N(iii). The construction is very similar to that for multicommodity network flows. The network is as in Fig. 2.2.3. Homologous arcs are marked with the same subscripted Greek letter.

The arcs (α, β) have a capacity that is one less than the number of terms in the clause, thereby ensuring that truth assignments that would make the preceding clause "true" cannot occur. The "homologous conditions" permit the separation of the flow at β into the original "truth assignments".

The maximum capacity of the sink is n. Hence there is a flow $\geqq n$ iff there is a consistent assignment of truth values to $a_1, \cdots, a_n$ such that no clause is "true", and hence P is not a tautology.

(iv) Maximum independent set $\propto$ N(iv).[3] Let $G(V, E)$ be an undirected graph for which we want to determine the maximum independent set.

[3] The author is grateful to S. Even for pointing out an error in the original proof and for suggesting the correction.

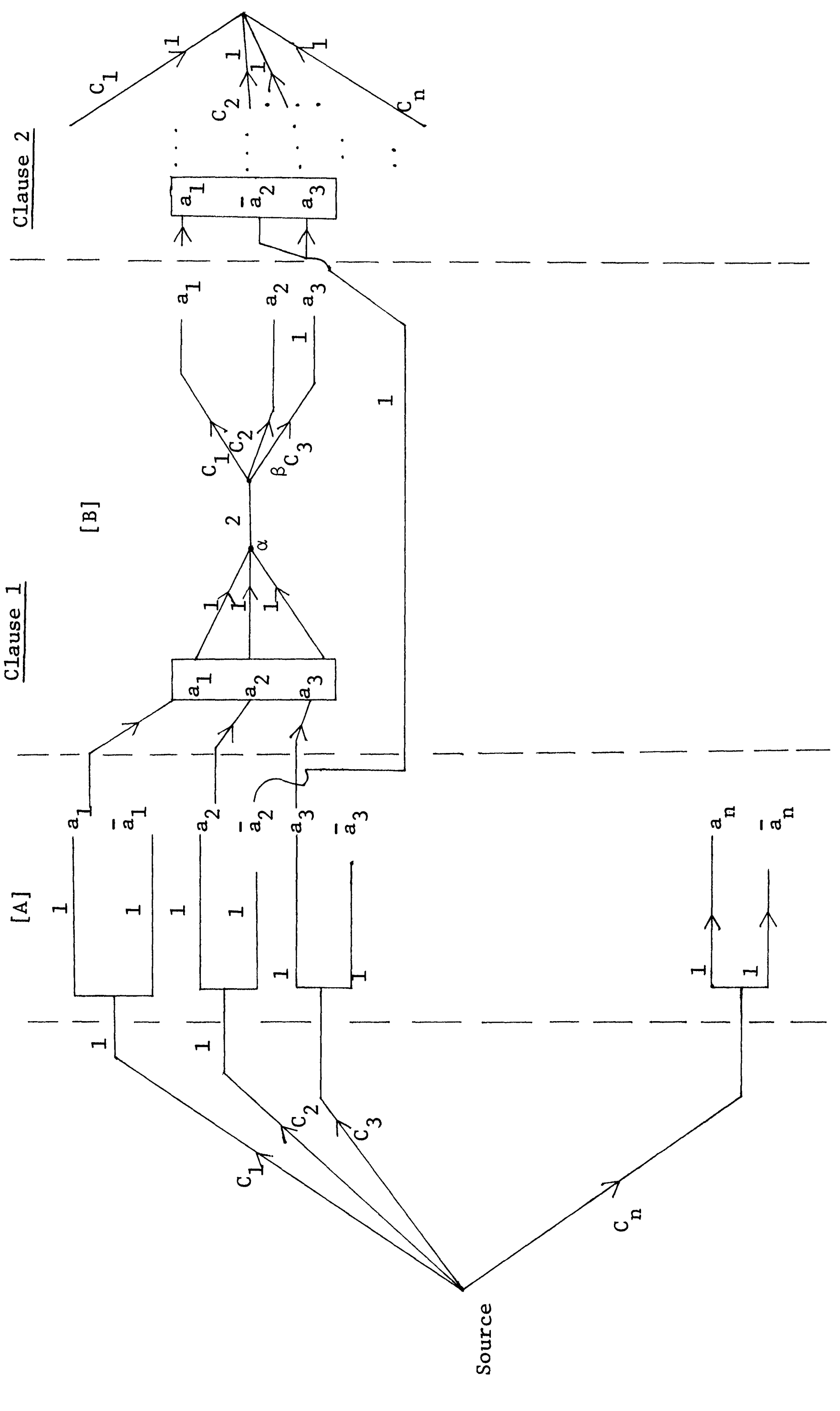


FIG. 2.2.2. *Tautology α multicommodity network flows*

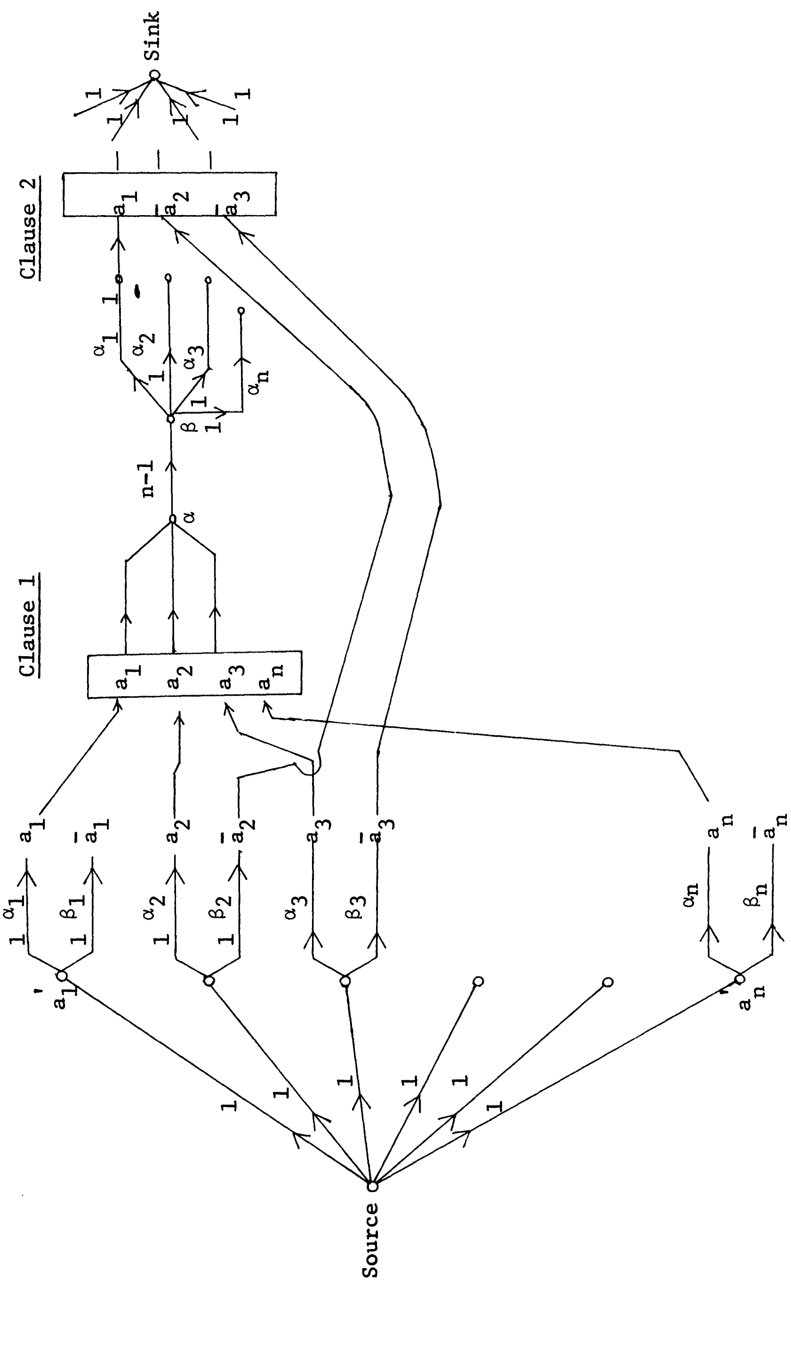

Source
Sink
Clause 1
Clause 2
n-1
α
β
a_1 $\bar{a}_1$ a_2 $\bar{a}_2$ a_3 $\bar{a}_3$ a_n $\bar{a}_n$
α_1 β_1 α_2 β_2 α_3 β_3 α_n β_n
1

FIG. 2.2.3. *Network with homologous arcs*

Construct a network as below:

Let $\hat{s}_1, v_1, \cdots, v_n, \hat{s}_2$ be the nodes of the network $n = |V|$. From the source node, draw an arc of capacity 1 to each of the nodes v_i, $1 \leqq i \leqq n$. From each node v_i, draw an arc a_i to the sink node $\hat{s}_2$. For each edge in G, define a bundle $(a_i, \underset{\sim}{a_j})$ if this edge joins vertices v_i and v_j in G. These are the only bundles in the network. Each bundle is assigned a capacity 1. This ensures that if vertex v_i is chosen in the maximum independent set (i.e., if there is a nonzero flow through it), then there is no flow through vertices adjacent to v_i (i.e., adjacent vertices are not chosen).

Now there is a flow $\geqq F$ iff there is an independent set of cardinality $\geqq F$. We solve the flow problem for $F = n, n - 1, \cdots, 1$, and the first F for which we get a feasible flow defines a maximum independent set.

Example 2.2.1.

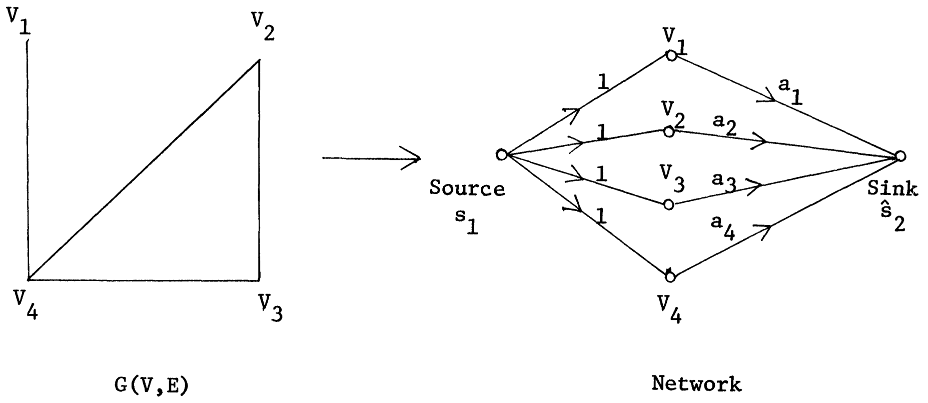


FIG. 2.2.4. *Example for maximum independent set* α N(iv)

The largest k for which there is a feasible flow is $k = 2$, through vertices V_1 and V_2. Thus the maximum independent set of G is of size 2, and one such set is $\{V_1, V_2\}$. The bundles are: (a_1, a_4), (a_2, a_3), (a_2, a_4) and (a_3, a_4).

It is interesting to note that all these problems are related to a similar, polynomial time, flow problem (see [1]).

2.3. Graph theory.

Problem G1. *Minimal equivalent graph of a digraph.* Given a directed graph $G(V, E)$, we wish to remove as many edges from G as possible, getting a graph G_1 such that:

(2.3.1a) In G, there is a path from v_i to v_j iff there is a path in G_1 from v_i to v_j;

(2.3.1b) $E(G_1) \subseteq E(G)$ ($E(G)$ is the set of edges of G), i.e., we want the smallest subset of $E(G)$ such that the transitive closure of G_1 = transitive closure of G.

THEOREM 2.3.1. G1 *is in* PC.

Proof. (a) G1 α P1, Let n = number of vertices in $G = |V(G)|$; then

$$|E(G)| \leqq n(n - 1) < n^2.$$

We can easily construct an NDTM, T, which given G and an integer k, determines if there is a subset of k edges satisfying (2.3.1a,b). T can be constructed so as to work in $O(n^3)$ time. If NP = P, then there is a deterministic algorithm that does this in $p(n)$ time. We find the smallest $k \leqq n^2$ for which such a subset exists. After determining k, the k edges can be determined as below.

Define a sequence $\bar{E}$ of maximum length $|E(G)|$. Set $\bar{e}_i = 1$ if edge i is among the k edges and $\bar{e}_i = 0$ otherwise.

Suppose it is already known that $\bar{E} = (i_1, \cdots, i_j)$ is a correct "partial" choice; then we ask if $\bar{E}(i_{j+1} = 1)$ is.

If yes, then set $\bar{E} = (i_1, i_2, \cdots, i_j, 1)$.

If no, then set $\bar{E} = (i_i, i_2, \cdots, i_j, 0)$.

Do this for $j = 0, 1, 2, \cdots, |E| - 1$.

(b) Directed Hamilton cycle $\propto$ G1.

Note. (i) If the directed graph G has a Hamilton cycle, then its transitive closure is the "complete directed graph" on $|V(G)|$ points. The smallest graph with this transitive closure is the cycle on $|V(G)|$ points. Thus if there is a Hamilton cycle, then this cycle forms the minimal equivalent graph of G.

(ii) Conversely, if the minimal equivalent graph is a cycle on $|V(G)|$ points, then G has a Hamilton cycle.

Therefore G has a Hamiltonian cycle iff the minimal equivalent graph of G is a Hamiltonian cycle.

Problem G2. *Optimal solution to* AND/OR *graphs.* This is a problem frequently encountered in artificial intelligence; see [2], [9] and [10]. We are given a directed graph $G(V, E)$. Each node of G represents a subproblem. In order to solve this subproblem, one might have to solve either all of its successors or only one of them. In the former case the node will be denoted an AND node, while in the latter case it is an OR node. The arcs are weighted, and the weights represent the cost associated with solving the parent node given that the successor (or son) node has been solved. There is one special node, S, which has no incoming arcs. This node represents the total problem being solved. The problem then is to find a minimum solution to S.

As an example, consider the directed graph of Fig. 2.3.1. The problem to be solved is P_1. To do this, one may solve either nodes P_2, P_3 or P_7, as P_1 is an OR node. The cost incurred is then either 2, 2 or 8 (i.e., cost in addition to that of solving one of P_2, P_3 or P_7). To solve P_2, both P_4 and P_5 have to be solved, as P_2 is an AND node. The total cost to do this is 2. To solve P_3, we may solve either P_5 or P_6. The minimum cost to do this is 1. P_7 is free. In this example, then, the optimal

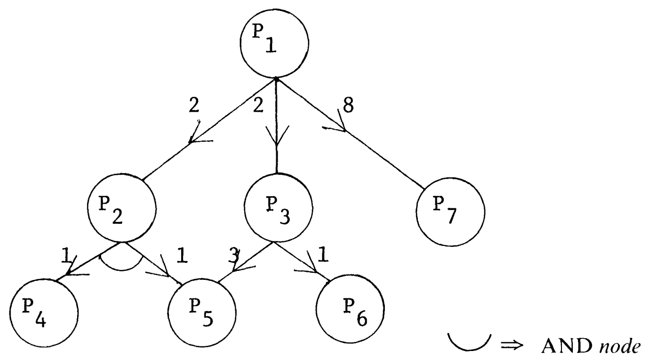


FIG. 2.3.1. AND/OR *graph*

way to solve P_1 is first solve P_6, then P_3 and finally P_1. The total cost for this solution is 3.

THEOREM 2.3.2. G2 $\in$ PC.

Proof. (a) G2 α (P = NP). The proof for this part is very similar to the part (a) of the proofs of each of Theorems 2.3.1 and 2.5.1 (see §2.5).

(b) Satisfiability α G2. We show how to transform a formula P in CNF into an AND/OR graph such that the AND/OR graph so obtained has a certain minimum cost solution iff P is satisfiable.

Let $$P = \bigwedge_{i=1}^{k} C_i, \qquad C_i = \bigvee_{j=1}^{3} l_j,$$

where the l_j's are literals and the variables of P, $V(P)$ are $x_1, x_2, \cdots, x_n$. The AND/OR graph will then have nodes as follows:

1. There is a special node, S, with no incoming arcs. This node represents the problem to be solved.

2. S is an AND node with descendent nodes $P, x_1, x_2, \cdots, x_n$.

3. Each node x_i represents the corresponding variable x_i in the formula P. Each x_i is an OR node with two descendents denoted Tx_i and Fx_i, respectively. If Tx_i is solved, then this will correspond to assigning a truth value of "true" to the variable x_i. Solving node Fx_i will then correspond to assigning a truth value of "false" to x_i.

4. The node P represents the formula P, and is an AND node. It has k descendents $C_1, C_2, \cdots, C_k$. Node C_i corresponds to the clause C_i in the formula P. The nodes C_i are OR nodes.

5. Each node of type Tx_i or Fx_i has exactly one descendent node which is terminal (i.e., has no edges leaving it). These terminal nodes shall be denoted $v_1, v_2, \cdots, v_{2n}$.

To complete the construction of the AND/OR, graph the following edges and costs are added:

1. From each node C_i an edge (C_i, Tx_j) is added if x_j occurs in clause C_i. An edge (C_i, Fx_j) is added if $\bar{x}_j$ occurs in the clause C_i. This is done for all variables x_j appearing in the clause C_i. C_i is designated an OR node.

2. Edges from nodes of type Tx_i or Fx_i to their respective terminal nodes are assigned a weight or cost 1.

3. All other edges have a cost 0.

In order to solve S, each of the nodes $P, x_1, x_2, \cdots, x_n$ must be solved. Solving nodes $x_1, x_2, \cdots, x_n$ costs n. To solve P, we must solve all the nodes $C_1, C_2, \cdots, C_k$. The cost of a node C_i is at most 1. However, if one of its descendent nodes was solved while solving the nodes $x_1, x_2, \cdots, x_n$, then the additional cost to solve C_i is 0, as the edges to its descendent nodes have cost 0 and one of its descendents has already been solved. That is, a node C_i can be solved at no cost if one of the literals occurring in the clause C_i has been assigned a value "true." From this it follows that the entire graph (i.e., node S) can be solved at a cost n if there is some assignment of truth values to the x_i's such that at least one literal in each clause is true under that assignment, i.e, if the formula P is satisfiable. If P is not satisfiable, then the cost is $> n$.

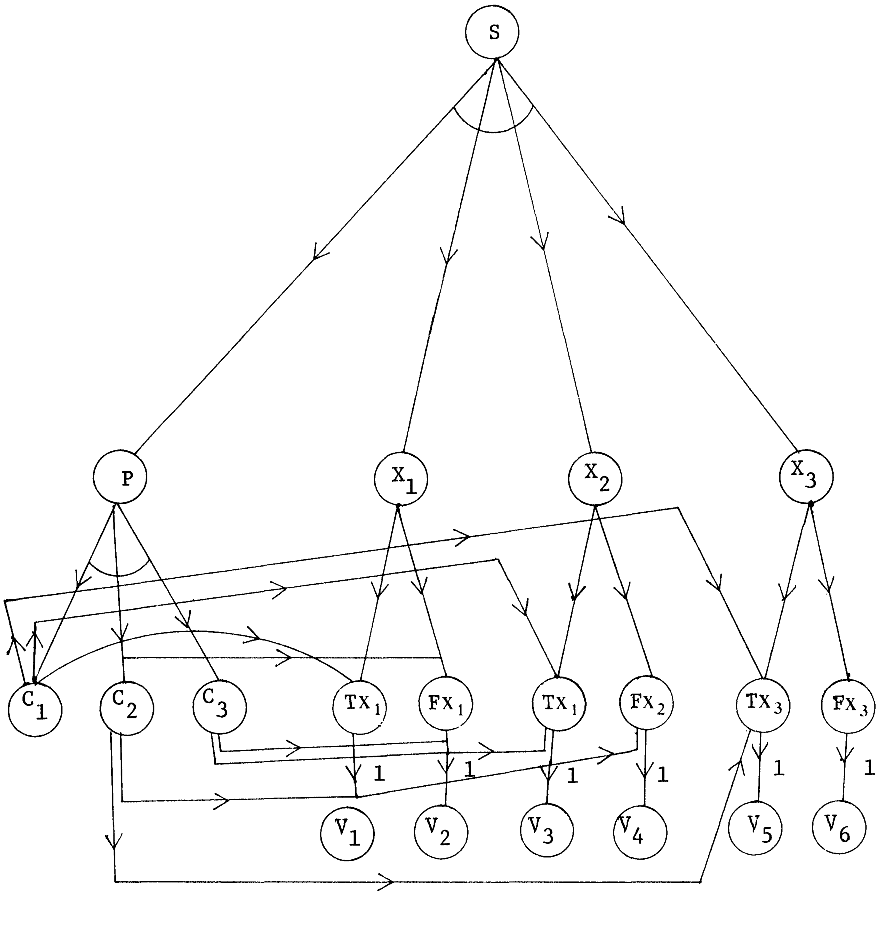
We have now shown how to construct an AND/OR graph from a formula P such that the AND/OR graph so constructed has a solution of cost n iff P is satisfiable. Otherwise the cost is $>n$. Hence from the minimum solution to the AND/OR graph, one can determine if P is satisfiable. The construction clearly takes only polynomial time. This completes the proof.

Example 2.3.1. Consider

$$P = (x_1 + x_2 + x_3)(\bar{x}_1 + \bar{x}_2 + \bar{x}_3)(\bar{x}_1 + x_2), \quad V(P) = x_1, x_2, x_3, \quad n = 3.$$

Figure 2.3.2 shows the AND/OR graph obtained by applying the transformation of Theorem 2.3.2.

The nodes Tx_1, Tx_2, Tx_3 can be solved at a total cost of 3. The node P then costs nothing extra. The node S can then be solved by solving all its descendent nodes and the nodes Tx_1, Tx_2 and Tx_3. The total cost for this solution is 3 (which is n). Assigning the truth value "true" to the variables of P results in P being "true."



AND *nodes marked* ◡
All other nodes are OR

FIG. 2.3.2. AND/OR *graph for Example* 2.3.1

2.4. *n*-person game theory. Following Lucas [7], we have:

An n-person noncooperative game in normal form consists of a set N of n players denoted $1, 2, \cdots, n$, a finite set $N_i = 0, 1, \cdots, n_i$ of $n_i + 1$ pure strategies for each player $i \in N$, and a payoff function F from $N_1 \times \cdots \times N_n$ to R^n.

A *strategy n*-tuple $(S_1^*, \cdots, S_n^*)$ is said to be an *equilibrium n*-tuple iff for all $i, i \in N$ and $S_i \in N_i$,

$$F_i(S_1^*, \cdots, S_n^*) \geqq F_i(S_1^*, \cdots, S_{i-1}^*, S_i, S_{i+1}^*, \cdots, S_n^*), \tag{2.4.1}$$

where F_i is the ith component of F. That is, there is no advantage for a player to unilaterally deviate from an equilibrium point.

Problem GT1. Given a game $G = (F, n, \overline{N})$, does it have an equilibrium point?

THEOREM 2.4.1. GT1 $\in$ PC.

Proof. (a) GT1 $\propto$ P1. The nondeterministic Turing machine just guesses an equilibrium point and verifies that the equilibrium condition (2.4.1) is satisfied.

(b) Satisfiability (3 literals/clause) $\propto$ GT1. Let P be the formula in CNF in n variables. Define an n-person game as below:

Each player has two strategies 0 and 1. Strategy 0 corresponds to assigning a truth value "false" to the corresponding variable and strategy 1 to a "true" assignment.

Let $\quad P = C_1 \wedge C_2 \wedge \cdots \wedge C_k, \qquad C_i = C_{i_1} \vee C_{i_2} \vee C_{i_3},$

where the variables are $x_1, x_2, \cdots, x_n$. Replace each variable in the clause C_i by x_i if $x_i \in C_i$ and by $(1 - x_i)$ if $\bar{x}_i \in C_i$

Replace "$\vee$" by "$+$", getting C_i'.

Example. $C_i = x_i \vee x_2 \vee \bar{x}_3 \Rightarrow C_i' = x_1 + x_2 + (1 - x_3) = x_1' + x_2' + x_3'$. In order that C_i' has a (0, 1) value, replace $x_1' + x_2' + x_3'$ by

$$f_i(\mathbf{x}') = x_1' + x_2'(1 + x_1') + x_3'(1 - x_1')(1 - x_2').$$

Clearly, $f_i(\mathbf{x}') = 1$ iff $C_i(x)$ is "true". Define

$$h_1(x') = 2 \prod_{i=1}^{k} f_i(\mathbf{x}') \quad \text{and} \quad F_1(\mathbf{x}') = \begin{bmatrix} h_1(x') \\ \vdots \\ h_1(x') \end{bmatrix}.$$

From the above definition of $F_1(\mathbf{x}')$, it follows that

$$\max F_1(\mathbf{x}') = \begin{cases} \begin{bmatrix} 2 \\ 2 \\ \vdots \\ 2 \end{bmatrix} & \text{if } P(\mathbf{x}) \text{ is satisfiable,} \\ \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} & \text{otherwise.} \end{cases}$$

Let $G_2(x_1, x_2)$ be a 2-person game with 2 strategies per player and with no equilibrium point:

$$G_2(\mathbf{x}) = \begin{bmatrix} g_1(\mathbf{x}) \\ g_2(\mathbf{x}) \end{bmatrix}, \qquad \begin{bmatrix} g_1 \\ g_2 \end{bmatrix} \leqq \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}.$$

Define

$$F_2(\mathbf{x}) = \begin{bmatrix} g_1(\mathbf{x}) \\ g_2(\mathbf{x}) \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

Then $F_2(\mathbf{x})$ defines an n-person game with no equilibrium point. Set

$$F(\mathbf{x}) = F_1(\mathbf{x}) + F_2(\mathbf{x}) \begin{bmatrix} 2 \\ 2 \\ \vdots \\ 2 \end{bmatrix} - F_1(\mathbf{x}).$$

Then $F(\mathbf{x})$ defines an n-person game in which each player has 2 strategies.

For any choice of strategy vector $\mathbf{x}$, we have either (i) or (ii) below.

$$\text{(i)} \qquad F_1(\mathbf{x}) = 0, \qquad F(\mathbf{x}) = 2F_2(\mathbf{x}) \leqq \begin{bmatrix} 1 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

By changing the strategies for either x_1 or x_2, we can increase the payoff to x_1 or x_2, respectively, as $F_2(\mathbf{x})$ defines a game with no equilibrium point. If such a change results in

$$F_1(\mathbf{x}) = \begin{bmatrix} 2 \\ 2 \\ \vdots \\ 2 \end{bmatrix},$$

then everyone's payoff increases. In any case, such an $\mathbf{x}$ cannot be an equilibrium point.

$$\text{(ii)} \qquad F_1(\mathbf{x}) = \begin{bmatrix} 2 \\ 2 \\ \vdots \\ 2 \end{bmatrix}.$$

Such a point is an equilibrium point, as now

$$F(x) = F_1(\mathbf{x}) = \begin{bmatrix} 2 \\ 2 \\ \vdots \\ 2 \end{bmatrix}.$$

and 2 is the maximum payoff any player can get. So no change from this point, unilateral or otherwise, would be advantageous to any player. Therefore the n-person game defined above has an equilibrium point iff $P(\mathbf{x})$ is satisfiable.

As an example for $G_2(x_1, x_2)$, consider:

Strategy	Payoff
(0, 0)	[0, 1]
(1, 0)	[1, 0]
(1, 1)	[0, 1]
(0, 1)	[1, 0]

$$g_1(\mathbf{x}) = (2 - x_1 - x_2)(x_1 + x_2),$$
$$g_2(\mathbf{x}) = (1 - x_1 - x_2)^2.$$

Clearly, no $\mathbf{x}$ is a stable (equilibrium) point. Set

$$G_2(\mathbf{x}) = \begin{bmatrix} g_1(\mathbf{x})/2 \\ g_2(\mathbf{x})/2 \end{bmatrix}.$$

2.5. Optimization.

Problem K1. *One-dimensional* 0-1 *Knapsack problem.* The problem is:

$$\text{(i) maximize} \quad \sum_{i=1}^{n} x_i p_i,$$
$$\text{subject to} \quad \sum_{i=1}^{n} x_i w_i \leqq M$$
$$x_i = 0, 1, \qquad 1 \leqq i \leqq n,$$
$$p_i > 0, \qquad w_i > 0$$

THEOREM 2.5.1. K1 $\in$ PC.

Proof. (a) K1 $\propto$ P1. Clearly, the problem is reducible to P1 if (i) is replaced by (i') $\sum x_i p_i \geqq Z$. Now if the length of the input is n then each $p_i < k^n$ for some k. So using the method of bisection, we can find the optimal Z in $\log_2 k^n = n \log_2 k$ query steps of (i') for some k, $k \leqq |\Sigma|$ (here $|\Sigma|$ = number of letters in the alphabet for the NDTM above).

(b) Sum of subsets of integers $\propto$ K1. Let $S = (s_1, \cdots, s_n)$ be the multiset of integers. We want to find a subset (if one exists) that sums to M. This may be stated in the form of a K1 problem as below:

$$\text{maximize} \quad \sum x_i s_i,$$
$$\text{subject to} \quad \sum x_i s_i \leqq M,$$
$$x_i = 0, 1.$$

From this we trivially conclude that the general 0-1 integer programming problem with nonnegative coefficients is complete. The 0-1 constraint may be replaced by the inequalities $x_i \leqq 1, 1 \leqq i \leqq n$.

The remarks of the last paragraph naturally lead us to the question of the status of the general integer programming problem (i.e., with both negative and positive coefficients). Here again, we are interested in only nonnegative solutions.

Problem I1. Determining if $Cx = b$ has a nonnegative solution is P-Hard. (Note the entries of C are integer. If C has all entries of the same sign, then the problem is P-Complete.)

To see this, consider the following formulation of the sum of subsets problem:

$$\sum_{i=1}^{n} w_i x_i = M,$$

$$w_i + y_i = 1, \qquad 1 \leqq i \leqq n.$$

Problem I2. Determining if $Cx \geqq 0$ has any integer solution (i.e., the x_i's are not constrained to be nonnegative) is P-Hard.

Application of Knuths' algorithm [6, vol. 2, p. 303] for obtaining integer solutions to $Cx = b$ yields a set of inequalities of the form $Dy \geqq w$. Setting $w = 0$ restricts the x to be $\geqq 0$. Hence $Dy \geqq 0$ has an integer solution iff $Cx = b$ has a nonnegative integer solution. Knuths' algorithm takes only polynomial time, so this problem is P-Hard. If the sign restriction on x is removed, then Knuths' algorithm solves $Cx = b$ in polynomial time. (This result was obtained together with H. B. Hunt III.)

Problem PF. *Permutation functions.* We are given a function $F(\mathbf{i})$ which is defined over all permutations of the elements of the vector $\mathbf{i} = (1, 2, \cdots, n)$. We wish to determine that permutation which minimizes F over all permutations. F is assumed to be polynomially computable.

THEOREM 2.5.2. PF $\in$ PC.

Proof. (a) PF α (P = NP). This part of the proof is very similar to that used in Theorem 2.5.1.

(b) Sum of subsets α PF. Define

$$F_k(\mathbf{i}) = -\left(\sum_{i=1}^{k} w(x_i)\right)\left(2M - \sum_{i=1}^{k} w(x_i)\right),$$

where x_i is the ith element of i.

We compute min F_k over all permutations of $\mathbf{i}$ for $k = 1, 2, \cdots, n$. If there is a subset that sums to M, then it has j elements in it, and min F_j is $-M$. If, on the other hand, for some $k = l$, min F_l is $-M$, then $\sum_{i=1}^{l} w(x_i) = M$. This defines an algorithm to solve the sum of subsets problem in polynomial time if we have a polynomial algorithm for PF.

Problem LB. *Assembly line balancing.* In this problem we are given n jobs $1, 2, \cdots, n$. Each job i requires a certain amount of processing time t_i. We have available machines, each having an available process time T. We want to determine the minimum number of machines needed to process all the jobs (the processing of a job cannot be split up among several machines).

THEOREM 2.5.3. LB $\in$ PC.

Proof. (a) LB $\propto$ (P = NP). This part of the proof is similar to Theorem 2.5.1.

(b) The following known member of PC shall be used (Karp [5]). Given a set of positive integers $s_1, s_2, \cdots, s_n$, is there a partition I such that

$$\sum_{i\in I} s_i = \sum_{i=1}^{n} s_i/2.$$

We show how this problem may be formulated as a line balancing problem. Let

$$t_i = s_i \quad \text{and } T = \sum_{i=1}^{n} s_i/2;$$

then the jobs $1, 2, \cdots, n$ can be processed on 2 machines iff there is a partition I of the jobs such that

$$\sum_{i\in I} t_i = T = \sum_{i=1}^{n} s_i/2.$$

This is the minimum number of machines on which the jobs can be processed as $\sum_{i=1}^{n} t_i = 2T$.

Problem PI. *Quadratic programming.* Here, the constraints are linear while the optimization function is quadratic.

THEOREM 2.5.4. PI *is P-Hard.*

Proof. Sum of subsets of integers $\propto$ PI.

$$\begin{aligned} &\text{maximize} && \sum_i x_i(x_i - 1) + \sum_i x_i s_i = f(x), \\ \text{(i)} \qquad &\text{subject to} && \sum_i x_i s_i \leqq M, \\ & && 0 \leqq x_i \leqq 1. \end{aligned}$$

For $0 < x_i < 1$, $x_i(x_i - 1) < 0$. This, together with (i), implies $f(x) < M$ if for some i, $0 < x_i < 1$. Thus max $f(x) = M$ iff S has a subset that sums to M.

The following variation of this problem may also be shown to be P-Hard: linear programming with one nonlinear constraint. Call this problem PI(b). To show that sum of subsets $\propto$ PI(b), just consider the formulation:

$$\begin{aligned} &\text{maximize} && \sum x_i s_i, \\ &\text{subject to} && \sum_i x_i s_i \leqq M, \\ & && \sum_i x_i(x_i - 1) \geqq 0, \\ & && 0 \leqq x_i \leqq 1. \end{aligned}$$

2.6. Minimal equivalent Boolean form.

Problem B1. Given a formula B from the propositional calculus, we wish to find the shortest formula equivalent to it.

THEOREM 2.6.1. B1 $\in$ PC.

Proof. (a) B1 $\propto$ P1. Define B1k to be the problem: is there a Boolean form of length k equivalent to B? We first show that a polynomial algorithm for P1 implies a polynomial algorithm for B1k. For this, we construct a nondeterministic Turing machine that guesses the Boolean form of length k and then uses the "tautology algorithm" to check that it is equivalent to B. If P1 works in $p(n)$ time, then the "tautology algorithm" works in $p_2(n)$ time (as tautology $\propto$ P1), and so the Turing machine constructed above works in $p_2(n)$ time. Hence B1$k \propto$ P1. The proof for B1 $\propto$ B1k is similar to part (a) of the proof of Theorem 2.3.1. We note that this proof relies heavily on our informal notion of P-Reducibility. The proof does not show that B1 is polynomially related to the other problems in PC. If the time complexity of the tautology problem is $f_1(n)$ and that of P1 if $f_2(n)$, then this reduction gives a $f_2(f_1(n))$ algorithm for B1. If f_1 (and consequently f_2) is exponential, then $f_2(f_1(n))$ is of the form 2^{2^n}. All our other reductions have been of the form $p(n) \cdot f_2(n)$ or $f_2(p(n))$ for some polynomial p.[4]

(b) tautology $\propto$ B1. A formula P is a tautology iff its minimal form is "1".

3. Conclusions. We have extended the class of known P-Complete problems to include some important applications from network flows, game theory, artificial intelligence and integer optimization. We have also introduced the notion of P-Hard. The results indicate that many of the problems for which no polynomial time bounded algorithm is known are related in terms of time complexity. Indeed, all the evidence to date suggests that there is no polynomial algorithm for any of these problems.

Acknowledgments. I am grateful to Professor Ellis Horowitz for many stimulating discussions on this subject. This work was motivated by the work of Cook [3] and Karp [5].

REFERENCES

[1] C. BERGE AND GHOUILA-HOWRI, *Programming, Games and Transportation Networks*, John Wiley, New York, 1964.

[2] C. L. CHANG AND J. R. SLAGLE, *An admissable and optimal algorithm for searching AND/OR graphs*, Artificial Intelligence, 2 (1971), pp. 117–128.

[3] S. A. COOK, *The complexity of theorem proving procedures*, Conference Record of Third ACM Symposium on Theory of Computing, 1971, pp. 151–158.

[4] J. E. HOPCROFT AND J. D. ULLMAN, *Formal Languages and their Relation to Automata*, Addison-Wesley, Reading, Mass., 1969.

[5] R. M. KARP, *Reducibility among combinatorial problems*, Complexity of Computer Computations, R. E. Miller and J. W. Thatcher, eds., Plenum Press, New York, 1972, pp. 85–104.

[6] D. E. KNUTH, *Art of Computer Programming*, vols. 1 and 2, Addison-Wesley, Reading, Mass., 1969.

[7] W. F. LUCAS, *Some recent development in n-person game theory*, SIAM Rev., 13 (1971), pp. 491–523.

[8] D. M. MOYLES AND G. L. THOMSON, *An algorithm for finding a minimum equivalent graph of a digraph*, J. Assoc. Comput. Mach., 16 (1969), pp. 455–460.

[9] N. J. NILSSON, *Problem Solving Methods in Artificial Intelligence*, McGraw-Hill, New York, 1971.

[10] R. SIMON AND R. C. T. LEE, *On the optimal solution of AND/OR series-parallel graphs*, J. Assoc. Comput. Mach., 18 (1971), pp. 354–372.

[4] Note that here we are not saying that the best way to solve this problem takes 2^{2^n} time on a deterministic machine. In fact one can easily solve it in time bounded by 2^{cn}. We are just making the point that this particular reduction does not show that the two problems are polynomially related.