Some Related Problems from Network Flows, Game Theory and Integer Programming

Sartaj Sahni[†]

TR 72 - 132

July 1972

Department of Computer Science
Cornell University
Ithaca, New York 14850

[†]To be presented at the 13th Annual Symposium on Switching and Automata Theory, IEEE, Maryland, October 1972.

Some Related Problems from Network Flows, Game Theory and Integer Programming

Sartaj Sahni

Cornell University

Abstract

We consider several important problems for which no polynomially time bounded algorithm is known. These problems are shown to be related in that a polynomial algorithm for one implies a polynomial algorithm for the others.

Some Related Problems from Network Flows, Game Theory and Integer Programming

Sartaj Sahni

Cornell University

1. Introduction

In [2] Cook introduced the concept of P-reducibility. He showed that there exists a polynomial time bounded algorithm to determine if a formula from the Propositional Calculus, in conjunctive normal form, was satisfiable (i.e. was true for some truth assignment to its variables) if and only if the class of languages accepted by polynomial time bounded Turing machines is the same as that acceptable by deterministic Turing machines in polynomial time. The latter problem is a well known "Open Problem" in the theory of computing. Substantial work in identifying other P-equivalent or P-complete problems has been done by Karp [5].

In this paper we show that the following problems are P-complete:

(1) Multicommodity network flows, flows with multipliers, flows with homologous arcs and bundle flows.

(2) Existence of a pure strategy equilibrium point in n-Person Games.

(3) 0-1 Integer programming with only one constraint (Knapsack Problem)

(4) Minimal equivalent digraph

(5) Minimal equivalent Boolean Form

We shall also show that Quadratic Programming and Linear Programming with one nonlinear constraint are P-hard, a condition not as strong as P-completeness.

In Section 2 we give a precise formulation of the notions of P-reducible, P-complete and P-hard. Section 3 contains a brief description of some of the known members of the class PC (Polynomially Complete problems) which will be used in showing other problems equivalent. Section 4 includes detailed proofs and descriptions of the new problems that have been found to be in PC or PH (polynomially hard).

2. Definitions

Definition 1 [Knuth]

a) A computational method is a quadruple (Q,I,Ω,f) satisfying:

(i) Q contains the subsets I and Ω

(ii) f is a function from Q into itself.
f leaves Ω pointwise fixed,
i.e. $f(q) = q \quad \forall q \in \Omega$.

Informally Q, I, Ω, f represent the states of the computation, the input set, output set, and the computational rule.

b) Each input $x \in I$ defines a computational sequence $x_0, x_1, \ldots$ as follows

$$x_0 = x \quad \text{and} \quad x_{k+1} = f(x_k) \quad \text{for} \quad k \geq 0$$

A computational sequence is said to terminate in k steps if k is the least integer for which $x_k \in \Omega$.

(Note that by the definition of a computational sequence, if $x_k \in \Omega$ then $x_{k+1} \in \Omega$ as $x_{k+1} = f(x_k) = x_k$).

An algorithm is a computational method which terminates in finitely many steps for all inputs $x \in I$.

If the computational rule f is single valued then the computational method is deterministic otherwise it is nondeterministic.

We restrict ourselves to computational methods that are effective (i.e. all the operations to be performed are sufficiently basic that they can be done in a finite amount of time).

Assuming that at each step of the computational sequence only one basic operation (add, multiply, compare, etc.) is performed, we may define the time complexity of the sequence to be the least k for which $x_k \in \Omega$. In the case of a non-terminating computation this is not defined.

The time complexity of an algorithm[1] will in general be some function of its inputs, say $f(\underline{N})$ where $\underline{N}$ is a vector that characterizes the inputs (for example in graph algorithms $\underline{N}$ may be the number of vertices and edges).

If $f(\underline{N})$ is a polynomial in $\underline{N}$ then the algorithm is polynomially time bounded. [In some cases it may be meaningful for $\underline{N}$ to also characterize the outputs, for e.g.: if the number of outputs is an exponential function of the inputs we should not expect to find an algorithm that computes them in

[1] Unless otherwise specified 'algorithm' will mean 'deterministic algorithm'.

$P(\underline{N})$; P a polynomial and $\underline{N}$ a characterization of the inputs].

As our computational model we shall use Turing Machines. (For a standard treatment of this model see Hopcroft and Ullman [3].) It should be noted that our results are valid, independent of the computational model chosen (e.g. random access machines).

Definitions 2 through 5 are from Karp [5].

Definition 2 P is the class of languages recognizable by by one-tape Turing machines which operate in polynomial time.

Definition 3 π is the class of functions from Σ^* into Σ^* defined by one-tape Turing machines which operate in polynomial time.

Definition 4 Let L and M be languages. Then $L \alpha M$ (L is P-reducible to M) if there is a function $f \varepsilon \pi$ such that $f(x) \varepsilon M = x \varepsilon L$.

Informally, a problem S is P-reducible to a problem T if a polynomial algorithm for T implies a polynomial for S i.e. given a polynomial algorithm for T we can construct a polynomial algorithm for S . In further discussion we shall use this informal notion of P-reducible. However, a suitable modification of each of the problems considered will easily be seen to be P-reducible in the sense of Definition 4.

Definition 5 NP is the class of languages recognizable in polynomial time by nondeterministic Turing Machines.

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

Definition 6 A problem P2 will be said to be P-complete if P2 α P3 and P3 α P2 for some P3 already known to be P-complete.

Clearly P-reducible is a transitive relation and P-complete is an equivalence relation.

Definition 7 P1 (defined above) is in PC (PC is the equivalence class of P-complete problems and it includes P1)

Definition 8 A problem P2 is P-hard if P3 α P2 and P3 ε PC

Clearly all members of PC are P-hard. In some cases we may only be able to show the relation P-hard rather than the stronger P-complete relation.

3. Some Known Members of PC

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) 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 if have the value True for all possible assignments of truth values.

(ii) Sum of subsets of integers

Given a multiset $S = (s_1, \ldots, s_r)$ of positive integers and a positive integer M does there exist a sub-multiset 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 set of elements that may not necessarily be distinct.

(iii) Maximum Independent Set

Let G be a graph with vertices $v_1, v_2, \ldots, v_n$. 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 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 Hamilton Cycle.

Proof (i) is proved in Cook [2]. The rest are proved in Karp [5]. Karp [5] acuually 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.

4. Complete and P-hard Problems

In this section we shall show that some frequently encountered problems in various areas such as Network flows, Game Theory, nonlinear and linear optimization are either P-complete or at least P-hard. The reductions will easily be seen to be effective. The polynomial factors involved in the reductions are small (usually a constant or a polynomial of degree 1).

4.1 Integer Network Flows

We define the following network problems:

N(i) Network flows with multipliers.

Let G be a directed graph with vertices s_1, s_2, v_1, ..., v_n and edges (arcs) e_1, e_2, ..., 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, s_1, of the network has no incoming arcs, i.e. $w^-(s_1) = \phi$

(b) The sink, s_2, has no outgoing arcs, i.e. $w^+(s_2) = \phi$

(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 $\Phi = (\phi_1, \phi_2, \ldots, \phi_m)$ such that:

1) $a_i \leq \phi_i \leq b_i$

2) $h(v) \sum_{i \in w^-(v)} \phi_i = \sum_{i \in w^+(v)} \phi_i$ for all $v \in V(G)$, $v \neq s_1$, $v \neq s_2$

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

(Note: ϕ_i integer)

In what follows we assume $a_i = 0$.

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 $c_1, c_2, \ldots, c_n$ and some arcs may be labelled i.e. they can carry only certain commodities. Each arc is

assigned a capacity and we wish to know whether a flow $R = (r_1, r_2, \ldots, r_n)$, where r_i is the quantity of the i^{th} commodity, is feasible in the network.

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.

N(iv) Integer Flows with Bundles

The arcs in the network are divided into sets $I_1, \ldots, I_k$ (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 $\geq F$ is feasible in the network.

$(\sum_{i \in I_j} \phi_i \leq C_j, \quad 1 \leq j \leq k)$ and $h(v) = 1, \quad \forall v \in V(G)$.

Theorem N: Problems N(i) - N(iv) are in PC.

Proof a) N(i), N(ii), N(iii), N(iv) α P1.

The Nondeterministic Turing Machine 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 $\geq R$.

ii) For N(iii) the 'homologous conditions' arc checked and $\sum_{i \in w^-(s_2)} \phi_i \geq F$ verified.

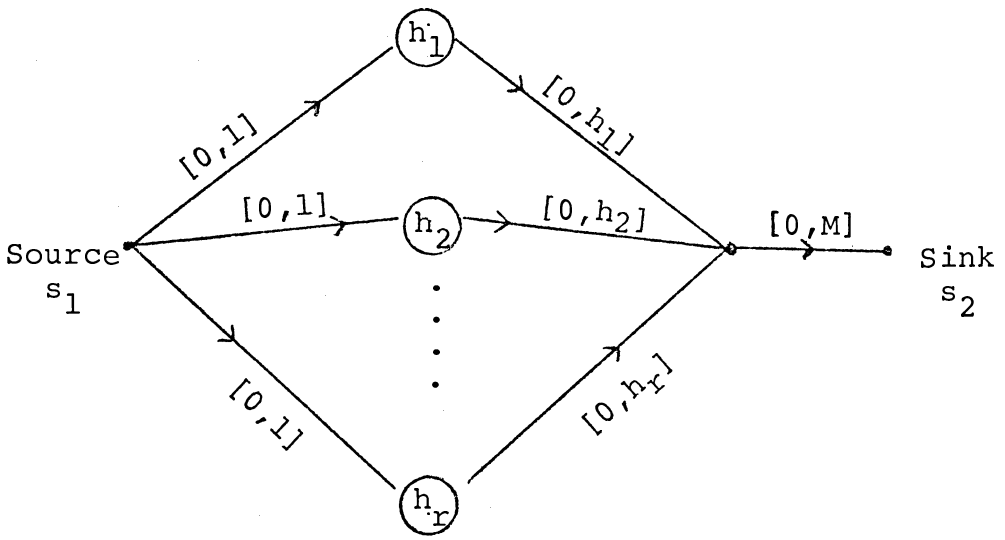
iii) For N(iv) the bundle restrictions are checked and

$\sum_{i \in w^-(s_2)} \phi_i \geq F$ verified.

If in N(i) we replace the max $\sum_{i \in w^-(s_2)} \phi_i$ requirement to T: $\sum_{i \in w^-(s_2)} \phi_i \geq F$

then from the above it follows that T α Pl . To see N(i) α 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 $\sum_{i \in w^-(s_2)} \phi_i \leq k^n$ 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 $\sum_{i \in w^-(s_2)} \phi_i$ 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) α T α Pl and from the transitivity of α we conclude N(i) α Pl . Clearly, this proof technique can be used to show N(iii) and N(iv) complete when they are changed to maximization problems.

b) (i) Sum of subsets of integers α N(i) . We construct a network flow problem of type N(i) such that max $\sum_{i \in w^-(s_2)} \phi_i = M$ iff there is a sub-multiset of S that sums to M . (For notations see Section 3.)



$h_i = s_i \qquad 1 \leq i \leq r$

Clearly $\max \sum_{i \in w^-(s_2)} \phi_i = M$ iff some sub-multiset of S sums to M .

(ii) Tautology α N(ii)

Suppose that the formula P in DNF has n-variables $a_1, a_2, \ldots, a_n$. We shall construct a multicommodity network with n-commodities $c_1, c_2, \ldots, c_n$ such that the flow R(1, 1, ..., 1) is feasible iff P is not a tautology. The network of Figure N2 realizes this.

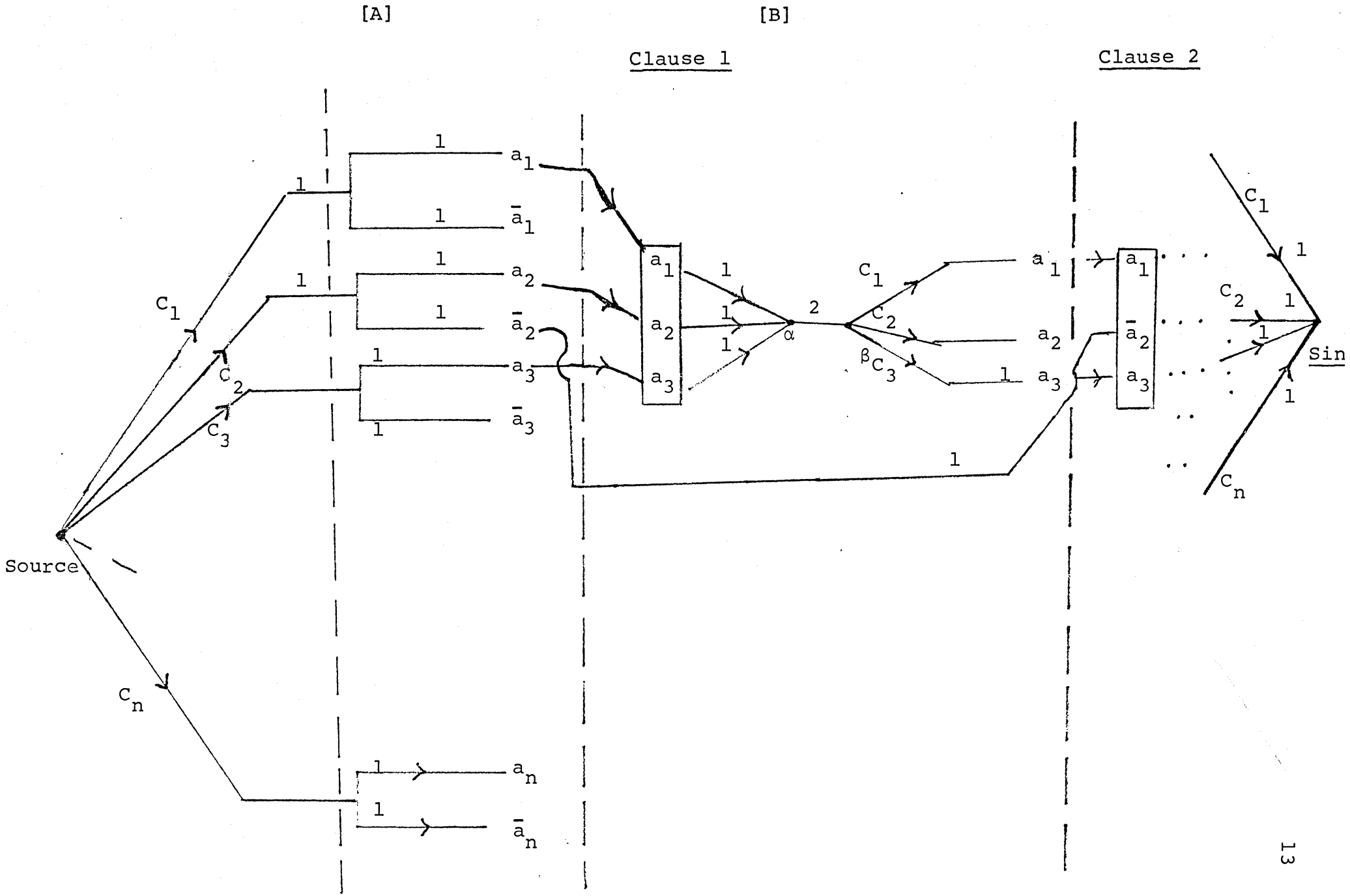
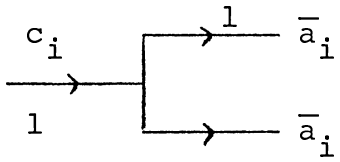


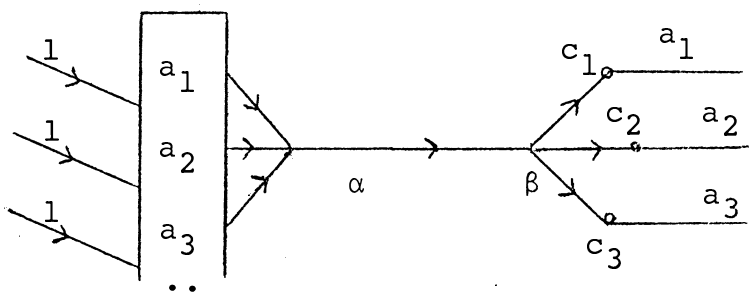
Figure N2 Tautology α Multicommodity Network Flows.

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 "multicomoodity" 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 Figure N2.

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

(iii) Tautology α N(iii)

The construction is very similar to that for multi-commodity network flows. The network is as in Figure N3. 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 $\geq n$ iff there is a consistent assignment of truth values to $a_1, \ldots, a_n$ such that no clause is "True" and hence P is not a Tautology.

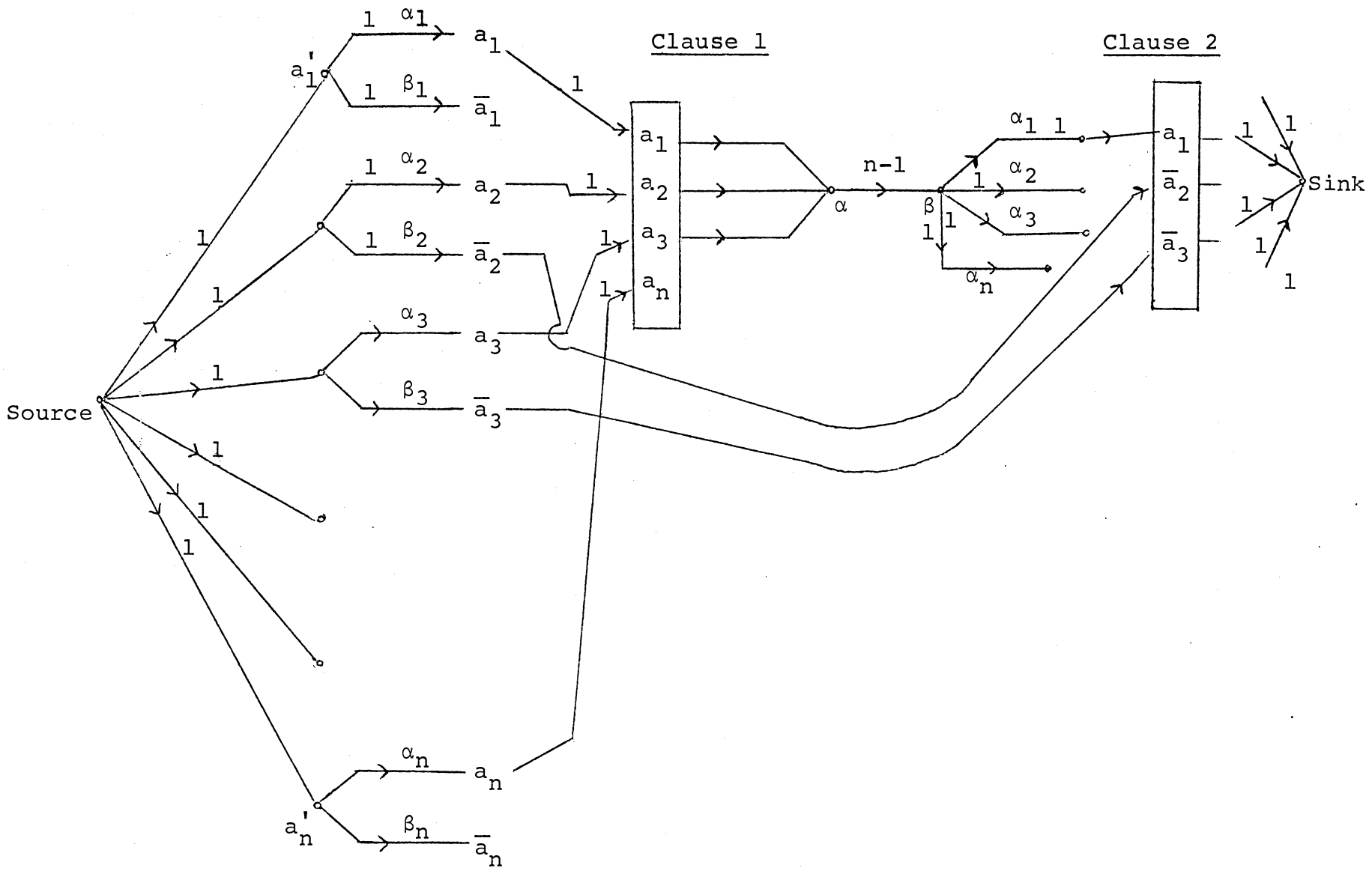


Figure N3: Network with Homologous Arcs.

(iv) Max. Independent Set α N(iv)

Let $G(V,E)$ be an undirected graph for which we want to determine the maximum independent set.

Construct a network as below:

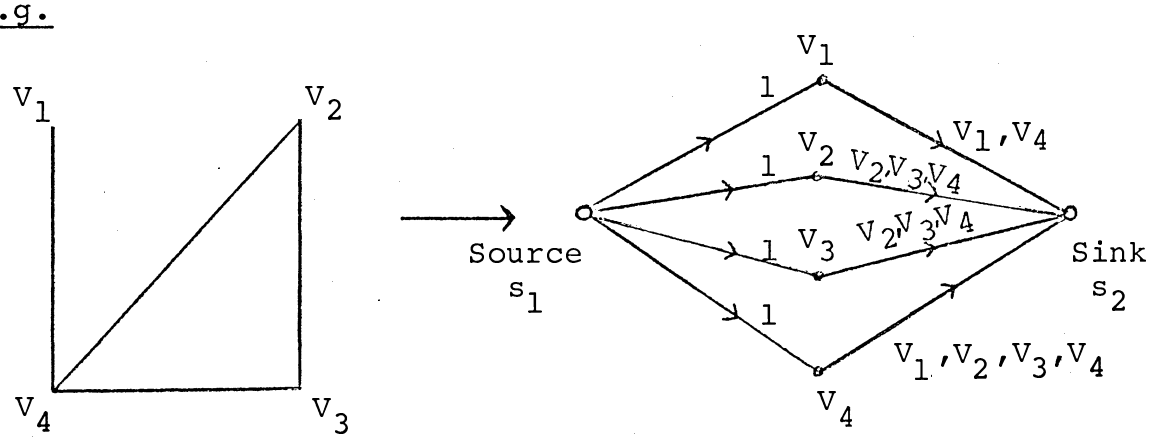
Let $s_1, v_1, \ldots, v_n, s_2$ be the nodes of the network $n = |V|$. From the source node draw an arc of capacity 1 to each of the nodes v_i $1 \leq i \leq n$. From each node v_i draw an arc to the sink node s_2. Mark each such arc v_i. For each node v_i mark the sink arc (v_i,s_2) by v_j if v_j is adjacent to v_i. Define the bundles $I_1, I_2, \ldots, I_n$ as:

$$I_j = \{e_i \mid \text{arc } e_i \text{ is marked by } v_j\}$$

Each bundle is assigned a capacity 1. This ensures that if vertex v_i is chosen in the maximum independent set (i.e. there is a non-zero 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 $\geq F = k$ iff there is an independent set of cardinality $\geq k$. We solve the flow problem for $k = n, n - 1, \ldots, 1$ and the first k for which we get a feasible flow defines a maximum independent set.

E.g.



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 $\{V_1, V_2\}$. (In this case the solution is unique.)

4.2 Graph Theory

G1: Minimal Equivalent Graph of a Digraph

Given a directed graph $G(E,V)$ we wish to remove as many edges from G as possible, getting a graph G_1 such that:

(i) 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

(ii) $E(G_1) \subseteq E(G)$ ($E(G)$ 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 G1: G1 is in PC

Proof (a) Gl α Pl

Let n = # of vertices in $G = |V(G)|$ then

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

We can easily construct a NDTM, T, which given G and an integer k determines if there is a subset of k edges satisfying (*) . T can be constructed so as to work in $O(n^3)$ time. If NP = DP then there is a deterministic algorithm that does this in p(n) time. We find the smallest $k \leq n^2$ 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 $|E(G)|$. $\bar{e}_i = 1$ if edge i is among the k edges 0 otherwise.

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

If Yes then set $\bar{E} = (i, i_2, \ldots, i_j, 1)$

If No then set $\bar{E} = (i, i_2, \ldots, i_j, 0)$

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

(b) Directed Hamilton Cycle α Gl

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 Hamilton cycle <=> the minimal equivalent graph of G is a Hamilton cycle.

4.3 n-Person Game Theory

Following Lucas [7] we have:

An n-Person non-cooperative game in normal form consists of a set N of n players denoted 1, 2, ..., n; a finite set $N_i = 0, 1, \ldots, n_i$ of $(n_i + 1)$ pure strategies for each player $i \in N$; and a payoff function F from $N_1 x \ldots x N_n$ to R^n.

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

$* \; F_i(S_i^*, \ldots, S_n^*) \geq F_i(S_i^*, \ldots, S_{i-n}^*, S_i, S_{i+1}^*, \ldots, S_n^*)$

where F_i is the i^{th} component of F.

i.e. There is no advantage for a player to unilaterally deviate from an equilibrium point.

G_T1: Given a game $G = (F, n, \overline{N})$ does it have an equilibrium point.

Theorem G_T1: GT1 $\in$ PC

Proof (a) GT1 α P1

The nondeterministic Turing Machine just guesses an equilibrium point and verifies that * is satisfied.

(b) Satisfiability (3 literals/clause) α 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 1 to a "True" assignment.

Let $P = C_1 \wedge C_2 \wedge \ldots \wedge C_k$

$C_i = C_{i_1} \vee C_{i_2} \vee C_{i_3}$

the variables are $x_1, x_2, \ldots, x_n$

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' .

E.g. $C_i = x_1 \vee x_2 \vee \bar{x}_3 \Rightarrow C_i' = x_1 + x_2 + (1 - x_3)$

$= x_1' + x_2' + x_3'$

In order that C_i' has a (0,1) value

replace $x_1' + x_2' + x_3'$ by

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

Clearly $f_i(\underline{x}') = 1$ iff $C_i(x)$ is "True".

Define $h_1(\underline{x}') = 2 \prod_{i=1}^{k} f_i(\underline{x}')$

and $F_1(\underline{x}') = \begin{pmatrix} h_1(x') \\ \vdots \\ h_1(x') \end{pmatrix}$

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

$\underline{\max}\ F_1(\underline{x}') = \begin{pmatrix} 2 \\ 2 \\ \vdots \\ 2 \end{pmatrix}$ if $P(\underline{x})$ is satisfiable

$= \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}$ otherwise

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

$$G_2(\underline{x}) = \begin{pmatrix} g_1(\underline{x}) \\ g_2(\underline{x}) \end{pmatrix} \qquad \begin{pmatrix} g_1 \\ g_2 \end{pmatrix} \leq \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}$$

define $$F_2(\underline{x}) = \begin{pmatrix} g_1(\underline{x}) \\ g_2(\underline{x}) \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

Then $F_2(\underline{x})$ defnnes an n-person game with no equilibrium point.

Set $F(\underline{x}) = F_1(\underline{x}) + F_2(\underline{x})\left(\begin{pmatrix} 2 \\ 2 \\ \vdots \\ 2 \end{pmatrix} - F_1(\underline{x})\right)$

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

For any choice of strategy vector $\underline{x}$ we have either

(i) $F_1(\underline{x}) = 0$, $F(\underline{x}) = 2F_2(\underline{x}) \leq \begin{pmatrix} 1 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$

By changing the strategies for either x_1 or x_2 we can increase the payoff to x_1 or x_2 respectively as $F_2(\underline{x})$ defines a game with no equilibrium point. If such a change results in $F_1(\underline{x}) = \begin{pmatrix} 2 \\ 2 \\ \vdots \\ 2 \end{pmatrix}$ then everyone's payoff increases. In any case such an $\underline{x}$ cannot be an equilibrium point.

(ii) $F_1(x) = \begin{pmatrix} 2 \\ 2 \\ \vdots \\ 2 \end{pmatrix}$ Such a point is an equilibrium point as now $F(x) = F_1(\underline{x}) = \begin{pmatrix} 2 \\ 2 \\ \vdots \\ 2 \end{pmatrix}$ 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 $P(\underline{x})$ is satisfiable.

As an example for $F_2(x_1,x_2)$ consider:

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

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

Clearly no $\underline{x}$ is a stable (equilibrium) point.

Set $F_2(\underline{x}) = \begin{pmatrix} g_1(\underline{x})/2 \\ g_2(\underline{x})/2 \end{pmatrix}$

4.4 Optimization

K1: One dimensional (0-1) Knapsack Problem.

The problem is to:

$$\text{maximize:} \quad \sum_{1}^{n} x_i\, p_i \qquad \ldots \qquad \text{(i)}$$

$$\text{subject to} \quad \sum_{1}^{n} x_i\, w_i \leq M$$

$$x_i = 0,1 \qquad 1 \leq i \leq n$$

$$p_i > 0, \quad w_i > 0$$

Theorem K1: K1 $\in$ PC

Proof (a) K1 α P1

Clearly the problem with (i) replaced by (1') .. $\Sigma\, x_i\, p_i \geq Z$ is reducible to P1 . 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 , $k \leq |\Sigma|$ ($|\Sigma|$ = number of letters in the alphabet for the NDTM above).

(b) Sum of subsets of integers α K1

Let $S = (S_1, \ldots, S_n)$ 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

$$\text{max:} \quad \Sigma\, x_i\, S_i$$

$$\Sigma\, x_i\, S_i \leq M$$

$$x_i = 0,1$$

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

[Karp [5] proves a similar result by showing Satisfiability α 0-1 Integer Programming]

PI: Quadratic Programming

Here, the constraints are linear while the optimization function is quadratic.

Theorem PI: PI is P-hard.

Proof Sum of subsets of integers α PI

$$\text{maximize:} \quad \sum_i x_i (x_i - 1) + \sum_i x_i s_i = f(x)$$

$$\text{subject to:} \quad \sum_i x_i s_i \leq M \qquad \ldots \qquad \text{(i)}$$

$$0 \leq x_i \leq 1$$

For $0 < x_i < 1$, $x_i(x_i - 1) < 0$

This, together with (i) implies $f(x) < M$

if for some i, $0 < x_i < 1$

$\max f(x) = M$ 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).

Sum of subsets α PI(b)

Just consider the formulation

$$\text{maximize:} \quad \sum x_i \, s_i$$

$$\text{subject to:} \quad \sum_i x_i \, s_i \leq M$$

$$\sum_i x_i \, (x_i - 1) \geq 0$$

$$0 \leq x_i \leq 1$$

4.5 Minimal Equivalent Boolean Form

B1: Given a formula B from the Propositional calculus we wish to find the shortest formula equivalent to it.

Theorem B1: B1 $\in$ PC

Proof (a) B1 α 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 $p_2(n)$ (as Tautology α P1) time and so the Turing machine constructed above works in $p_2(n)$ time. Hence B1k α P1 . The proof for B1 α B1k is similar to G1(a) . 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 $f_2(n)$ then this reduction gives a $f_2(f_1(n))$ algorithm

for Bl . 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 .

(b) Tautology α Bl

A formula P is a tautology iff its minimal form is '1' .

5 Conclusions

We have extended the class of known P-complete problems to include some important applications from Network Flows, Game Theory and Integer Programming. We have also introduced the concept of P-hard and shown that Quadratic Programming and Linear Programming with one non linear constraint are P-hard. The results obtained indicate that many of the problems for which no polynomial time bounded algorithm is known are related in terms of time complexity. Indeed the complexity of these problems may well be exponential. If such is the case then the next step will be to search for subexponential algorithms and to investigate the use of heuristics to improve the average behavior of these algorithms. In [4] we look at the sum of integers problem and the Knapsack problem. Subexponential algorithms for these problems are obtained. We also study the use of heuristics in solving these problems.

6 Acknowledgements

I am grateful to Professor Ellis Horowitz for his assistance in this research.

References

[1] Berge, C. and Ghouila-Howri, "Programming, Games and Transporation Networks", Spottiswoode, Ballantyne, Ltd., London (1965).

[2] Cook, S.A., "The Complexity of Theorem-Proving Procedures", Conference Record of Third ACM Symposium on Theory of Computing, 151-158 (1970).

[3] Hopcroft, J.E., and Ullman, J.D., "Formal Languages and Their Relation to Automata", Addison-Wesley (1969).

[4] Horowitz, E., and Sahni, S., "Computing Partitions with Applications to the Knapsack Problem", Cornell University, Technical Report No. 72-128.

[5] Karp, R.M., "Reducibility Among Combinatorial Problems", Technical Report No. 3, April 1972, Department of Computer Science, University of California, Berkeley.

[6] Knuth, D.E., "Art of Computer Programming, Vol.1: Fundamental Algorithms", Addison-Wesley (1968).

[7] Lucas, W.F., "n-Person Game Theory", SIAM Review, October 1971, Vol.13, No.4.

[8] Moyles, D.M., and Thomson, G.L., "An Algorithm for Finding a Minimum Equivalent Graph of a Digraph", JACM, Vol.16, No.4, July 1969, 455-460.