P-COMPLETE PROBLEMS AND APPROXIMATE SOLUTIONS

Sartaj Sahni and Teofilo Gonzales

Computer Science Department
University of Minnesota
Minneapolis, Minnesota

ABSTRACT

We study the class of P-Complete problems and show the following:

i) for any constant $\varepsilon > 0$ there is a P-complete problem for which an ε-approximate solution can be found in linear time
ii) there exist P-Complete problems for which linear time approximate solutions that get closer and closer to the optimal (with increasing problem size) can be found
iii) there exist P-Complete problems for which the approximation problems are also P-Complete.

KEY WORDS

P-Complete, approximation algorithm, polynomial complexity, k-Partition, k-MaxCut, traveling salesman, cycle covers.

1. INTRODUCTION

Our notion of P-Complete corresponds to the one used in [6]. This can easily be seen to be equivalent to that of Cook [2]. A problem, L , will be said to be P-Complete iff the following holds: L can be solved in polynomial deterministic time iff the class of nondeterministic polynomial time languages is the same as deterministic polynomial time languages (i.e. P = NP). Knuth [5] suggests the terminology NP-Complete. However, his notion of "completeness" is that of Karp [4]. Since the equivalence or non-equivalence of the two notions is not known, we will use the term NP-Complete for problems that can be shown complete with Karp's definition and P-Complete for those which require the definition of [6]. The reader unfamiliar with P-Complete problems is referred to [4] and [6]. All problems that are NP-Complete (i.e. complete under Karp's definitions) are also P-Complete [2 and 6]. The reverse is unknown. Since it appears that P $\neq$ NP, the P-Complete problems probably have no polynomial solution. Many of these problems, especially the optimisation problems, are of practical significance. Often, as in the case of the Knapsack problem [7], approximate solutions (i.e. feasible solutions that are guaranteed to be 'reasonably' close to the optimal) would be acceptable so long as they can be obtained 'quickly' (e.g. by an $O(n^k)$ algorithm for small k). Johnson [3] and Sahni [7] have studied some P-Complete problems with respect to obtaining 'good' (i.e. polynomial) approximate algorithms. (For our purposes, an algorithm for a maximization problem will be said to be f(n)-approximate ($f(n) < 1$) iff $\left|\frac{F^* - \hat{F}}{F^*}\right| \leq f(n)$ for all n (F^* is the maximum value of the objective function and $\hat{F}$ the approximation to the maximum, we assume $F^*>0$)). For a more formal definition see [7].) However, interesting questions like:

a) are there polynomial time approximation algorithms for all P-Complete problems (note that the problems in [4] are stated as language recognition problems and in order to speak of approximate solutions we shall have to convert them to optimisation problems. Thus, the clique problem of [4] will become the max clique problem) and
b) are there P-Complete problems for which one can obtain solutions arbitrarily close to the optimal in polynomial time (the algorithms of [7] can get to within 1/k of the optimal in $O(n^k)$ time, here we present problems for which one can get to within 1/k in O(n) time, where n is the space to describe the problem), remain open.

In this paper, we shall attempt to answer these questions.

The following problems known to be NP-Complete (see Karp [4]) shall be used in the reductions:

i) Partition: Given s integers $(c_1,c_2,\ldots,c_s)$ is there a subset $I \subseteq \{1,2,\ldots,s\}$ such that $\sum_{h\in I} c_h = \sum_{h\notin I} c_h$

ii) Cut: Given an undirected graph G(N,A), weighting function $w : A \to Z$, positive integer W, is there a set $S \subseteq N$ such that $\sum_{\substack{\{u,v\}\in A \\ u\in s \\ v\notin s}} w\{u,v\} \geq W$

iii) Sum of Subsets: Given n+1 positive integers $(r_1,r_2,\ldots r_n,m)$, is there a subset of the r_i's that sums to m.

The problems we shall use to answer the questions posed earlier are:

a) k-Partition: Given n integers $r_1,r_2,\ldots r_n$ and an integer $k \geq 2$, are there disjoint subsets $I_1,I_2,\ldots,I_k$ such that $\bigcup_{i=1}^{k} I_i = \{1,2,\ldots n\}$ and

$$\sum_{i\in I_1} r_i = \sum_{i\in I_\ell} r_i \ , \ 2 \leq \ell \leq k$$

(The Partition problem i) above is then just the 2-Partition problem.)

b) k-Cut: Given and undirected graph G = (N,A), integer $k \geq 2$, weighting function $w : A \to Z$, positive integer W, are there disjoint sets $S_1,S_2,\ldots,S_k$ such that $\bigcup_{i=1}^{k} S_i = N$ and $\sum_{\substack{\{u,v\}\in A \\ u\in S_i \\ u\in S_j \\ i\neq j}} w(u,v) \geq W$.

(The Cut problem ii) above is just the 2-Cut problem.)

b') k-MaxCut[1]: Find disjoint sets, S_i $1 \le i \le k$, such that $\bigcup_{i=1}^{k} S_i = N$ and $\sum_{\substack{\{u,v\}\in A \\ u\in S_i \\ v\in S_j \\ i\neq j}} w\{u,v\}$ is maximised.

c) $\lceil n/k \rceil$-Partition: same as a) except that the number of disjoint subsets is now $\lceil n/k \rceil$, $k \ge 2$.

d) $\lceil n/k \rceil$ Cut: same as b) except that the number of disjoint subsets is now $\lceil n/k \rceil$, $k \ge 2$.

d') $\lceil n/k \rceil$-MaxCut: same as b') with k replaced by $\lceil n/k \rceil$.

2. COMPLETENESS PROOFS AND APPROXIMATIONS

We begin by showing that problems a-d of the introduction are P-Complete.

Lemma 2.1

(I) The following problems are NP-COMPLETE
   a) k-Partition
   b) k-Cut
   c) $\lceil n/k \rceil$-Partition
   d) $\lceil n/k \rceil$-Cut

(II) k-MaxCut and $\lceil n/k \rceil$-MaxCut are P-Complete.

Proof

We have to show that i) if P=NP then a)-d) can be solved in polynomial time and ii) if a)-d) can be solved in polynomial time then the class of P-Complete problems is polynomial solvable (this can be shown by reducing any known P-Complete problem to a)-d)).

i) is trivial, so we shall only show ii).

ii) Partition α k-Partition. For any Partition problem $(c_1,c_2,\ldots c_s)$ define a k-Partition problem $(r_1,r_2,\ldots,r_{s+k-2})$ where

$$r_i = \begin{cases} c_i & 1 \le i \le s \\ p & s+1 \le i \le s+k-2 \end{cases} \qquad \text{and } p = \Sigma c_i/2$$

(we may assume that Σc_i is even as otherwise the partition problem clearly has no solution). Now, $\Sigma r_i = kp$ and the k-Partition problem has a solution iff the corresponding Partition problem has one.

k-Partition α k-Cut

If the given k-Partition problem is $(r_1,\ldots,r_n)$ define the corresponding k-MaxCut problem[1] to be $G = (N,A)$ with $N = (1,2,\ldots n)$,

$A = \{\{i,j\} \mid i\in N,\ j\in N,\ i\neq j\}$

$w(\{i,j\}) = r_i\, r_j$

and $W = \frac{(k-1)}{2k} \left(\Sigma r_i\right)^2$

(Note, we may again assume k divides Σr_i.)

Clearly, there is a k-Cut $\ge W$ iff $(r_1,r_2,\ldots,r_n)$ has a k-partition.

[1] The k-MaxCut problem is a generalisation of the 'grouping of ordering data' problem studied in [1]. [1] restricts the sets S_i to be sequential, i.e., if $i,j\in S_\ell$ and $i < j$ then $i+1,i+2,\ldots,j-1 \in S_\ell$. [1] presents an $O(kn^2)$ dynamic programming algorithm for this.

Partition α $\lceil n/k \rceil$-Partition. We prove this only for k = 2. From the partition problem $(c_1,c_2,\ldots,c_s)$, $s > 3$, construct the following $\lceil n/2 \rceil$-partition problem:

$r_i = c_i$ $\quad 1 \le i \le s$

$r_i = p$ $\quad s+1 \le i \le n$

$n = 2(s-2)$

$p = \Sigma c_i/2$ (if Σc_i odd then there is no partition)

Clearly, the partition problem has a solution iff the $\lceil n/2 \rceil$-Partition problem has one.

$\lceil n/k \rceil$-Partition α $\lceil n/k \rceil$-MaxCut. The proof for this is similar to that for k-Partition k-Cut II follows from I and the techniques of [5]. ■

We note that the proofs used in Lemma 1 are minor extensions of the ones used in Karp [4]. The (n-k)-Partition and (n-k)-MaxCut problems are polynomial. We next present an approximation algorithm for the k-MaxCut and $\alpha\lceil n/k \rceil$-MaxCut problems. Consider the algorithm MAXCUT below: (Intuitively, this algorithm begins by placing one vertex of G into each of the ℓ sets S_i $1 \le i \le \ell$; The remaining $n-\ell$ vertices are examined one at a time. Examination of a vertex, j, involves determining the set S_i $1 \le i \le \ell$ for which $\sum_{m\in S_i} w\{m,j\}$ is minimal. Vertex j is then inserted/assigned to this set.)

Algorithm MAXCUT (ℓ,G)

comment. ℓ...number of disjoint sets, S_i, into which the vertices, N = (1,2,...,n), of the graph G(N,A) are to be partitioned, SOL ... the value of the vertex partitioning obtained, $w\{i,j\}$... weight of the edge $\{i,j\}$. SET(i)... the set to which vertex i has been assigned (SET(i) = 0 for all vertices not yet assigned to a set), WT(i) ... used to compute $\sum_{m\in S_i} w\{m,j\}$, $1 \le i \le \ell$

This algorithm assumes that the graph G(N,A) is presented as n lists $v_1,v_2,\ldots,v_n$. Each list v_i contains all the edges, $\{i,j\}\in A$, that are adjacent to vertex i. No assumption is made on the order in which these edges appear in the list. end comment

Step 1 [Initialise] If $\ell \ge n$ then do

SOL ← $\sum_{\{i,j\}\in A} w\{i,j\}$

$S_i \leftarrow \{i\}$ $\quad 1 \le i \le n$

$S_i \leftarrow \emptyset$ $\quad n+1 \le i \le \ell$

Stop

end;

otherwise Set $S_i \leftarrow i$ $\quad 1 \le i \le \ell$

WT(i) ← 0 $\quad 1 \le i \le \ell$,

SET (i) ← i $\quad 1 \le i \le \ell$

SET (i) ← 0 $\quad \ell+1 \le i \le n$

SOL ← $\sum_{\substack{\{i,j\}\in A \\ 1\le i<j\le \ell}} w\{i,j\}$

j ← ℓ + 1

Step 2 [process edge list of vertex j]

for each edge {j,m} on the edge list of vertex j do

if SET(m) ≠ 0 then WT(SET(m)) ← WT(SET(m)) + w{j,m} ;

end

d_j ← degree of vertex j = # of edges adjacent to vertex j

Step 3 [find the set for which $\Sigma_{m \in S_i} w\{j,m\}$ is minimal]

look at WT(a) $1 \leq a \leq \min\{d_j+1, \ell\}$ and determine i such that WT(i) is minimal in this range. (Note that if $d_j + 1 \leq \ell$ then at least one of WT(a) $1 \leq a \leq d_j + 1$ must be 0 and minimal. For $d_j + 1 \geq \ell$ all WT(a) are looked at and the minimal found.)

Step 4 [assign vertex j to set S_i]

SET(j) ← i

Step 5 [update SOL and reset WT]

for each edge $\{j,m\} \in A$ for which SET(m) ≠ 0 do
if SET(m) ≠ i then SOL ← SOL + w{j,m}
WT(SET(m)) ← 0
end

Step 6 [next vertex] j ← j+1 ,

if $j \leq n$ then go to step 2
otherwise terminate algorithm

■

Lemma 2.2

The time complexity of algorithm MAXCUT is $O(\ell+n+e)$ on a random access machine (n is the number of vertices, e the number of edges and ℓ the number of groups into which the vertices are to be partitioned).

Proof

Step	Time Per Execution	Total Time
1	$O(n+e+\ell)$	$O(n+e+\ell)$
2	$O(d_j)$	$O(e)$
3	$O(d_j + 1)$	$O(e + n)$
4	$O(1)$	$O(n)$
5	$O(d_j)$	$O(e)$
6	$O(1)$	$O(n)$

Hence total time = $O(n + e + \ell)$

■

Lemma 2.3

Algorithm MAXCUT is a 1/k - approximate algorithm for the k-MaxCut problem.

Proof

If $n \leq k$ then MAXCUT generates the optimal solution value.

Define the internal weight of the set S_i to be $\sum_{\substack{u \neq v \\ u,v \in S_i}} w\{u,v\}$.

Then the total internal weight (TIW) = $\sum_{i=1}^{k}$ internal weight (S_i). The external weight (EW) = $\sum_{\substack{u,v \\ u \in S_i \\ v \in S_j \\ i \neq j}} w\{u,v\}$.

In Step 4 when vertex j is assigned to set i either WT(i) = 0 (corresponding to $d_j < \ell$) or

$WT(i) \leq \sum_{1 \leq m \leq k} WT(m)/k$.

i.e. if the total internal weight increases by WT(i) then the external weight increases by at least (k-1)WT(i). Consequently, at termination, TIW < EW/(k-1) (note that SOL = EW). But, the optimal value of the solution ≤ TIW + EW. Let F* be the optimal. EW = SOL is the approximation obtained by MAXCUT. The worst case occurs when TIW approaches EW/(k-1). Hence $\left|\frac{F^* - SOL}{F^*}\right| < 1/k$.

■

The next two theorems establish the existence of P-Complete problems for which linear time approximate solutions, that are exceedingly close to the optimal solution value, can be obtained. I.e., there are P-Complete problems that can "almost" be solved in linear time. In this sense, then, these problems are the simplest P-Complete problems.

Theorem 2.1

For any constant ε, $0 < \varepsilon < 1$, there esists a P-Complete problem for which an ε-approximate solution may be obtained in linear time.

Proof

Let k be an integer such the $1/k \leq \varepsilon$. From Lemma 2.1, the k-MaxCut problem is P-Complete. From Lemmas 2.2 and 2.3 it follows that an ε-approximate solution to this P-Complete Problem may be obtained in linear time.

■

Theorem 2.2

There exists a P-Complete problem and a linear time f(n)-approximate algorithm with the property that $f(n) \to 0$ as $n \to \infty$, i.e. the approximate solutions get increasingly accurate as n increases.

Proof

From Lemmas 2.1, 2.2 and 2.3 it follows that there is a linear time $1/\lceil n/k \rceil$ -approximate algorithm for the P-Complete problem: $\lceil n/k \rceil$ -MaxCut.

■

We conclude this section by establishing the existence of a P-Complete problem for which the approximation problem is also P-Complete.

Theorem 2.3

There is a P-Complete problem that has a polynomial time ε-approximate ($0 < \varepsilon < 1$) algorithm iff P= NP.

Proof

From any sum of subset problem $(r_1, r_2, \ldots r_n, m)$ construct the following Knapsack type problem:

$$\left.\begin{array}{ll} \max & k\left(\sum_{1}^{n} \delta_i + \delta_{n+1}\right) \\ \text{subject to:} & \sum_{1}^{n} w_i \delta_i + w_{n+1}\delta_{n+1} = m \\ & \delta_i = 0,1 \qquad 1 \leq i \leq n+1 \end{array}\right\} \ldots . K1$$

where $w_i = r_i$ $1 \leq i \leq n$ and $w_{n+1} = m$. Clearly, K1 is P-Complete. The maximum (i.e. F*) is either 1 (corresponding to $\delta_i = 0$ $1 \leq i \leq n$ and $\delta_{n+1} = 1$) or it is $\geq k$. Also $F^* \geq k$ iff the sum of subset problem has a solution. Assume there is a ε-approximate algorithm for this problem. Let $\hat{F}$ be the value (i.e. the approximate solution value) yielded by this algorithm. Then $\left|\frac{F^* - \hat{F}}{F^*}\right| \leq \varepsilon$. All solutions approximating F*=1 have value $1 - \varepsilon \leq \hat{F} \leq 1 + \varepsilon$. While solutions approximating $F^* \geq k$ have value $k(1-\varepsilon) \leq \hat{F}$. By choosing k suitably large, we can make these two ranges disjoint. Hence, form a knowledge of $\hat{F}$ one can determine whether or not the sum of subsets problem has a solution. Consequently, the ε-approximation problem for K1 is also P-Complete.

■

3. OTHER P-COMPLETE APPROXIMATION PROBLEMS

In this section we look at some other P-Complete problems and show that they have a polynomial time

approximate algorithm iff P = NP. This would then imply that if P ≠ NP, then any polynomial time approximation algorithm for these problems, heuristic or otherwise, must produce arbitrarily bad approximations on some inputs. The problems we look at are:

(i) Travelling Salesman: Given an undirected (directed) complete[1] graph G(N,A) and a weighting function w: A → Z find the shortest cycle containing each vertex exactly once. (The length of a cycle is the sum of the weights of the edges in the cycle.)

(ii) Undirected Edge Disjoint Cycle Cover: Given an undirected graph G(N,A) find the minimum number of edge disjoint cycles needed to cover all the vertices of N (i.e. minimum number of cycles such that each vertex of G is on at least one cycle)

(iii) Directed Edge Disjoint Cycle Cover: Same as (ii) except that G is now a directed graph.

(iv) Undirected Vertex Disjoint Cycle Cover: This problem is the same as (ii) except that now, the cycles are constrained to be vertex disjoint.

(v) Directed Vertex Disjoint Cycle Cover: Same as (iv) except that G is now a directed graph.

(vi) 0-1 Integer Programming with one constraint.

(vii) Multicommodity Network Flows: We are given a transportation network [6] with source s_1 and sink s_2. The arcs of the network are labeled corresponding to the commodities that can be transported along them. Such a network is said to have a flow f iff f units of each commodity can be transported from source to sink. The problem here is to maximize f.

In order to prove some of our results we shall use the following known P-Complete problems:

a) Hamiltonian Cycle: Given an undirected (directed) graph G(N,A) does it have a cycle containing each vertex exactly once, [4].

b) Multicommodity Flows: Given the transportation network of (vii) above does it have a flow of f = 1 [6].

Before continuing, it is necessary to generalize our definition of approximation algorithm to include minimization problems.

Definition 3.1

An algorithm will be said to be an ε-approximate algorithm for a problem P_1 iff either (i) P_1 is a maximization problem and

$$\left|\frac{F^* - \hat{F}}{F^*}\right| \leq \varepsilon \quad 0 < \varepsilon < 1$$

or (ii) P_1 is a minimization problem and

$$\left|\frac{F^* - \hat{F}}{F^*}\right| \leq \varepsilon \quad \varepsilon > 0$$

where F^* is the optimal solution (assumed > 0) and $\hat{F}$ is the approximate solution obtained.

Theorem 3.1

The ε-approximation problem for (i) - (vii) above is P-Complete.

Proof

For each of the problems (i) - (vii), it is easy to see that if P = NP then the ε-approximation problem is polynomially solvable (as the exact solutions would then be obtainable in polynomial time). Consequently, we concern ourselves only with showing that if there is a polynomial time approximation algorithm for any of the problems listed above then P = NP. Our approach is to seperate feasible solutions to a given problem in such a way that from a knowledge of the approximate solution one can solve exactly a known P-Complete problem.

(i) Hamiltonian Cycle α ε-approximate Travelling Salesman

Let G(N,A) be any graph. Construct the graph $G_1(V,E)$ such that V = N and E = {(u,v) | u,v∈V}. Define the weighting function w: E→Z to be $w\{u,v\} = \begin{cases} 1 & \text{if } (u,v) \in A \\ k & \text{otherwise.} \end{cases}$

Let n = |N|. For k > 1, the Travelling Salesman problem on G_1 has a solution of length n iff G has a Hamiltonian cycle. Otherwise, all solutions to G_1 have length ≥ k + n - 1. If we choose k ≥ (1+ε)n then, the only solutions approximating a solution with value n (if there was a Hamiltonian cycle in G_1) also have length n. Consequently, if the ε-approximate solution has length ≤(1+ε)n then it must be of length. n. If it has length >(1+ε) n then G has no Hamiltonian cycle.

(ii) - (v) The proofs for (ii) - (v) are similar. We outline the proof for (iv)

Undirected Hamiltonian Cycle α ε-approximate Disjoint Cycle Cover

Given an Undirected graph G(N,A) construct k copies $G_i(N_i,A_i)$ of this graph. Pick a vertex v∈N. Let $u^1,u^2,\ldots,u^d$ be the vertices adjacent to v in G (i.e. $(u^i,v) \in A$ $1 \leq i \leq d$). Define $H_j(V_j,E_j)$ to be the graph with $V_j = \bigcup_{i=1}^{k} N_i$ and $E_j = \bigcup_{i=1}^{j} A_i \cup \{(u_1^j, v_{i+1}) \mid 1 \leq i < k\} \cup \{(u_k^j, v_1)\}$

Clearly, if G has a Hamiltonian Cycle then, for some j, H_j has a cycle cover containing exactly one cycle (as for some j (v,u^j) are adjacent in the Hamiltonian cycle and using the images of the edges of this cycle in the subgraphs G_i (except for the images of the edge (v,u^j)), together with the edges $(u_i^j,v_{i+1}) \mid 1 \leq i < k\} \cup \{u_k^j, v_1)$ one obtains a Hamiltonian cycle for E_j). If G has no Hamiltonian cycle, then the subgraphs G_i each require at least two disjoint cycles to cover their nodes. Consequently, the disjoint cycle cover for j contains at least k+1 cycles, $1 \leq j \leq k$. For any ε, one may choose a suitable k such that from the approximate solutions to H_j $1 \leq j \leq k$ one can decide whether or not G has a Hamiltonian cycle (i.e. choose k > (1+ε)).

Note that the above proof also works for the case of edge disjoint cycle covers.

(vi) Just consider the reduction:

Sum of Subsets α ε-approximate 0-1 Integer Programming

ie $\min 1 + k(m - \Sigma w_i \delta_i)$

subject to $\Sigma w_i \delta_i \leq m$

$\delta_i = 0$ or 1

(The minimum = 1 iff there is a subset with sum = m otherwise the minimum is ≥ 1 + k)

(vii) Multicommodity flows α ε-approximate Multicommodity Flows

In [6] it was shown that multicommodity flows with f = 1 was P-Complete. Given a multicommodity network N as in [6] we construct k

[1] A graph G(N,A) is said to be Complete iff for every pair of vertices u,v∈N u≠v the edge (u,v)∈A.

copies of it and put them in parallel. Another network with a flow f = 1 is also coupled to the network as in figure 3.1.

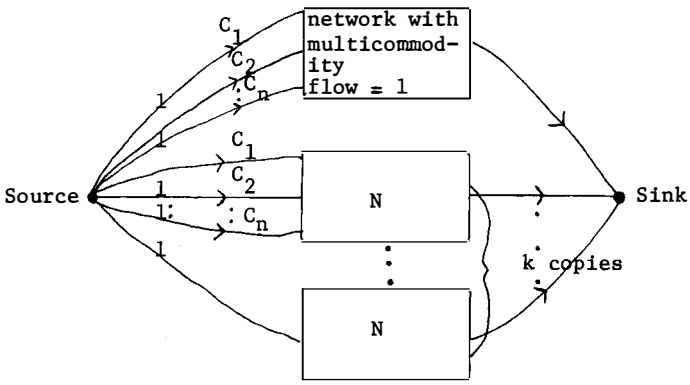

Figure 3.1

Clearly, the multicommodity network of figure 3.1 has a flow of k + 1 iff N has a flow of 1. If N does not have a flow of 1 then the maximum flow in the network of figure 3.1 is 1. Hence the approximation problem for multicommodity flows is P-Complete. ■

4. CONCLUSION

We have answered some existence problems concerning approximation algorithms for P-Complete problems. Our results show that all P-Complete problems are not equally hard. Some are harder than others in the sense that some P-Complete problems can 'almost' be solved in linear time while for others, the problem of determining even an ε-approximate solution is P-Complete. In terms of computational difficulty of the approximation problem, we have exhibited the simplest and the hardest P-Complete problems. We note that some of our results could have been obtained using "padding" techniques. Further, padding techniques also lead to such results as a) there are P-Complete problems with an O(n) average computing time and b) there are P-Complete problems with a sub-exponential (e.g. $O(2^{\sqrt{n}})$) worst case computing time. However, it remains open whether a) and b) are true for any "natural" P-Complete problem, such as those of [4] and [6].

REFERENCES

[1] Bodin, L. D. "A Graph Theoretic Approach to the Grouping of Ordering Data", Networks, 2, 1972, pp. 307-310.

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

[3] Johnson, D. "Approximation Algorithms for Combinatorial Problems", Conference Record of Fifth ACM Symposium on Theory of Computing, 1973.

[4] Karp, R. M.,"Reducibility Among Combinatorial Problems", in Complexity of Computer Computations, R.E. Miller and J.W. Thatcher, eds., Plenum Press, N.Y., 1972, pp. 85-104.

[5] Knuth, D. E., "A Terminological Proposal" SIGACT News, Jan. 1974, Vol. 6, No. 1, pp. 12-18. See also SIGACT News, April, 1974.

[6] Sahni, S., "Computationally Related Problems", to appear in SICOMP.

[7] Sahni, S., "Approximate Algorithms for the 0/1 Kanpsack Problem",to appear in JACM.