JOURNAL OF COMPUTER AND SYSTEM SCIENCES 9, 256–278 (1974)

Approximation Algorithms for Combinatorial Problems*

DAVID S. JOHNSON†

Project MAC, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139

Received August 27, 1973

Simple, polynomial-time, heuristic algorithms for finding approximate solutions to various polynomial complete optimization problems are analyzed with respect to their worst case behavior, measured by the ratio of the worst solution value that can be chosen by the algorithm to the optimal value. For certain problems, such as a simple form of the knapsack problem and an optimization problem based on satisfiability testing, there are algorithms for which this ratio is bounded by a constant, independent of the problem size. For a number of set covering problems, simple algorithms yield worst case ratios which can grow with the log of the problem size. And for the problem of finding the maximum clique in a graph, no algorithm has been found for which the ratio does not grow at least as fast as n^{ϵ}, where n is the problem size and $\epsilon > 0$ depends on the algorithm.

1. INTRODUCTION

Cook and Karp in [1, 6] have shown the existence of a class of combinatorial problems, the "polynomial complete" problems, such that if any of these problems can be solved in polynomial time, then they all can. Many of these problems, such as tautology testing and the traveling salesman problem, have become notorious for their computational intractability, and it is widely conjectured that none of them can be done in polynomial time. Many problems of this class can be viewed as optimization problems. Some examples of such optimization problems are:

SUBSET-SUM: Given a finite set of positive numbers and another positive number called the "goal," find that subset whose sum is closest to, without exceeding, the goal.

* The research reported here was supported in part by Project MAC, an M.I.T. research program sponsored by the Advanced Research Projects Agency, Department of Defense, under Office of Naval Research Contract Number N00014-70-A-0362-0006 and the National Science Foundation under contract number GJ00-4327.

† Present address: Bell Laboratories, Murray Hill, NJ 07974.

BIN-PACKING: Given a finite list of numbers between 0 and 1 and a sequence of unit-capacity bins, find a packing of the numbers into the bins such that no bin contains a total exceeding 1 and the number of nonempty bins is minimized.

MAXIMUM SATISFIABILITY: Given a set S of disjunctive form clauses, all of whose disjuncts are either literals or their negations, find a truth assignment to the literals which satisfies (makes true) the most clauses.

SET COVERING I: Given a finite cover of a finite set, find that subcover which uses the fewest sets.

SET COVERING II: Given a finite cover of a finite set, find that subcover which has the least overlapping.

GRAPH COLORING: Given a graph, find a coloring of the nodes so that no two adjacent nodes have the same color, and the total number of colors used is minimized.

MAXIMUM CLIQUE: Given a graph, find the maximum subgraph all of whose points are mutually adjacent.

Work on the above-mentioned BIN-PACKING problem [2, 4, 5] has suggested an approach that may be applied to the others. Since no fast algorithm for finding an optimal solution could be found, simple heuristic algorithms that seemed likely to generate fairly good packings were studied. It was found that certain of these simple algorithms could guarantee near-optimal results. For instance, the FIRST FIT DECREASING algorithm guarantees that, asymptotically, no more than 11/9 times the optimal number of bins will be used. This type of result also has appeared in earlier work on a dual problem having to do with multiprocessor scheduling, for which an extensive bibliography is given in [3].

This paper examines polynomial-time heuristic "approximation algorithms" for the other optimization problems listed above, in an attempt to get further results of this type. We discover that a wide variety of worst case behaviors are possible.

Some algorithms, such as the ones we present for SUBSET-SUM and MAXIMUM SATISFIABILITY, have the same type of behavior as FIRST FIT DECREASING in that they guarantee that the solution they generate will be no worse than a constant times the optimal, and indeed that constant can be quite close to 1.

For other algorithms, such as the ones we present for the two SET COVERING problems, there is no fixed bound on the ratio of the worst case results to the optimal, but that ratio can grow no faster than the log of the problem size.

For still others, such as some of the coloring algorithms we give and *all* of the algorithms for MAXIMUM CLIQUE that have been suggested, the ratio grows at least as fast as $O(n^{\epsilon})$, where n is the problem size and $\epsilon > 0$ depends on the algorithm.

We also consider the possibility of using the reductions which show that problems are polynomial equivalent to translate our results from one optimization problem to another. Karp's reductions in [6] are all between language recognition problems, but we can often extend them directly to the associated optimization problems. Difficulties arise, however, since a "good" approximate solution for one problem may map to a solution for the other which is not at all good according to the measure for the second problem.

In Section 2 we introduce a formal framework within which to discuss the above problems, defining what we mean by "optimization problem," "approximation algorithm," "worst case behavior," etc. Each of the next six sections is devoted to one of the problems, and finally Section 9 presents some general observations and conclusions.

2. Approximation Algorithms

An optimization problem P consists of

1. a set INPUT_P of possible *inputs*,
2. a map SOL_P which maps each $u \in \mathrm{INPUT}_P$ to a finite set *approximate solutions*,
3. a measure m_P: $\mathrm{SOL}_P(\mathrm{INPUT}_P) \to Q^+$ defined for all possible approximate solutions.

In addition, the problem P is specified as a maximization problem or a minimization problem, depending on whether the goal is to find an approximate solution with maximal or minimal measure. For each $u \in \mathrm{INPUT}_P$, the optimal measure $u_P{}^*$ is defined by

$$u_P{}^* = \mathrm{BEST}\{m_P(x): x \in \mathrm{SOL}_P(u)\},$$

where BEST stands for MAX or MIN depending on whether P is a maximization or minimization problem. Since $\mathrm{SOL}_P(u)$ is finite, there must be at least one solution $x \in \mathrm{SOL}_P(u)$ such that $m_P(x) = u_P{}^*$, and such a solution will be called an optimal solution.

An approximation algorithm for problem P, or simply an algorithm, is any method for choosing approximate solutions, given $u \in \mathrm{INPUT}_P$. (Since the algorithms we will study are not always completely determined, more than one solution may be choosable for a given input). If A is an approximation algorithm for problem P, then the performance $A(u)_P$ of A for input u is defined

$$A(u)_P = \mathrm{WORST}\{m_P(x): x \in \mathrm{SOL}_P(u) \text{ and } x \text{ is choosable by } A \text{ on input } u\},$$

where WORST is MIN if BEST is MAX, and vice versa. As a measure of the relative worst case behavior of algorithm A on input u, we shall use the ratio

$$r_P(A, u) = \begin{cases} u_P^*/A(u)_P, & \text{if } P \text{ is a maximization problem,} \\ A(u)_P/u_P^*, & \text{if } P \text{ is a minimization problem.} \end{cases}$$

Note that this ratio has been defined so as to always be greater than or equal to 1.

For a given input u, the problem size $|u|$ is the number of symbols required to describe u in some standard notation. Our measure of the overall worst case behavior of algorithm A with respect to problem P will be a function $R[A, P]: N^+ \to R$ defined by

$$R[A, P](n) = \text{MAX}\{r_P(A, u): u \in \text{INPUT}_P \text{ and } |u| \leqslant n\}.$$

As defined, $R[A, P]$ is an increasing function of n.

When there is no possibility of confusion, we will often delete the subscripts P from our notation, and let $R[A]$ stand for $R[A, P]$. Of particular interest are algorithms A for which $R[A]$ has a fixed upper bound independent of n. Examples of these are given in the next two sections.

3. SUBSET-SUM

This maximization problem is a simple form of the well-known "knapsack problem." We shall denote it by SS, and restate it in terms of our formal definitions:

$$\begin{aligned} \text{INPUT}_{\text{SS}} = \{&\langle T, s, b\rangle: T \text{ is a finite set, } s: T \to Q^+ \text{ is a map} \\ &\text{which assigns to each } x \in T \text{ a "size" } s(x), \\ &\text{and } b > 0 \text{ is a single rational number}\}. \end{aligned}$$

$$\text{SOL}_{\text{SS}}(\langle T, s, b\rangle) = \left\{T' \subseteq T: \sum_{x \in T'} s(x) \leqslant b\right\}.$$

$$m_{\text{SS}}(T') = \sum_{x \in T'} s(x),$$

The optimal measure for input $\langle T, s, b\rangle$ is (with subscripts omitted)

$$\langle T, s, b\rangle^* = \text{MAX}\{m(T'): T' \subseteq T \text{ and } m(T') \leqslant b\}.$$

We shall study a series of approximation algorithms $\{A_k: k \geqslant 1\}$, where algorithm A_k is defined as follows.

1. Let SUB be that subset of $\{x \in T: s(x) > b/(k+1)\}$ whose measure is closest to, without exceeding b. Let SUM be this measure, and set LEFT $= T -$ SUB.
2. If for all $x \in$ LEFT, $s(x) +$ SUM $> b$, halt and return SUB.
3. Let y be an element of LEFT for which $s(y) +$ SUM is closest to, without exceeding, b.
4. Set LEFT $=$ LEFT $- \{y\}$. SUB $=$ SUB $\cup \{y\}$, SUM $=$ SUM $+ s(y)$.
5. Go to 2.

Remark. Algorithm A_1 is closely related to the FIRST FIT DECREASING algorithm for BIN-PACKING [5], since the set chosen by it consists of those elements which would be assigned to the first bin if T were being packed into bins of size b by that algorithm. However, the BIN-PACKING results are apparently of no use to us here, since they dealt with the number of bins used, not the sum of the pieces in the first bin.

The majority of the effort in these algorithms, at least for $k \geqslant 2$, is concentrated in the first step. Since the set found in this step has $|\text{ SUB }| \leqslant k$, the step can be accomplished using at most $O(n^k)$ comparisons and additions, where $n = |\ T\ |$, and we do not know how to do any better. The A_k, thus, are a series of seemingly increasingly expensive algorithms. However, they yield increasingly better results as follows.

THEOREM 1. *For $k \geqslant 1$ and $n > 0$,*

$$R[A_k](n) \leqslant (k+1)/k,$$

$$\lim_{n\to\infty} R[A_k](n) = (k+1)/k.$$

Proof. For the upper bound, suppose we are given an input $\langle T, s, b\rangle$ and $T_1 \subseteq T$ is choosable by algorithm A_k on this input. We actually prove the somewhat stronger result that either $m(T_1) = \langle T, s, b\rangle^*$ or else $m(T_1) \geqslant [k/(k+1)] \cdot b$. Let T_0 be an optimal solution for the given input. We now partition T_i for $i \in \{0, 1\}$ as follows.

$$T_i = T_i^{\text{BIG}} \cup T_i^{\text{SMALL}},$$

where $T_i^{\text{BIG}} = \{x \in T_i: s(x) > b/(k+1)\}$, and $T_i^{\text{SMALL}} = T_i - T_i^{\text{BIG}}$. We thus have, for $i \in \{0, 1\}$,

$$m(T_i) = m(T_i^{\text{BIG}}) + m(T_i^{\text{SMALL}}).$$

By Step 1 of algorithm A_k, $m(T_1^{\text{BIG}}) \geqslant m(T_0^{\text{BIG}})$. If $T_1^{\text{SMALL}} \supseteq T_0^{\text{SMALL}}$, we would also have $m(T_1^{\text{SMALL}}) \geqslant m(T_0^{\text{SMALL}})$, and so T_1 would be optimal.

On the other hand, if some $x \in T_0^{\text{SMALL}}$ were excluded from T_1^{SMALL}, then by Step 2 of A_k, $s(x) + m(T_1) > b$, and so

$$m(T_1) > b - s(x) \geqslant b - b/(k+1) = kb/(k+1) \geqslant [k/(k+1)]\langle T, s, b\rangle^*.$$

Thus in all cases we have $r(A_k, \langle T, s, b\rangle) \leqslant (k+1)/k$, and the upper bound is proven.

For the lower bound on the limit, consider the input $\langle T, s, b\rangle$ where

$$T = \{a_1, \ldots, a_{k+2}\},$$

$$s(a_i) = \begin{cases} 1 + \epsilon, & \text{for } i = 1, \\ 1, & \text{otherwise.} \end{cases}$$

$$b = k + 1.$$

Clearly $\langle T, s, b\rangle^* = k + 1$ and $A_k(\langle T, s, b\rangle) = k + \epsilon$. Thus, $r(A_k, \langle T, s, b\rangle) = (k+1)/(k+\epsilon)$. Since ϵ can be made arbitrarily small by making the problem size n large enough, we thus have $\lim_{n\to\infty} R[A_k](n) \geqslant (k+1)/k$ and the theorem follows.

The above result shows that the SUBSET-SUM optimization problem has the desirable property: For any $\epsilon > 0$, there is a polynomial-time algorithm X_ϵ such that $R[X_\epsilon] \leqslant 1 + \epsilon$.

Sahni [8] has extended our results to show that a more complicated version of SUBSET-SUM also has this property. In the ONE-DIMENSIONAL KNAPSACK problem, each $x \in T$, in addition to having a *size* $s(x)$, has a utility $u(x)$. The goal is then to maximize $\sum_{x\in T'} u(x)$ over the same set of approximate solutions as in SUBSET-SUM. Sahni essentially proves that Theorem 1 holds for the new problem when A_k is replaced by a slightly more expensive algorithm A_k'. Unfortunately, these problems and some minor variants are the only such problems we have found.

In the next section we present a problem from which we can obtain in a natural way a sequence of subproblems which are all polynomial complete. We exhibit approximation algorithms B1 and B2 for this problem with the property that for any $\epsilon > 0$ and $i \in \{1, 2\}$, there exists a subproblem $P_{i,\epsilon}$ with $R[Bi, P_{i,\epsilon}] \leqslant 1 + \epsilon$.

4. Maximum Satisfiability

This maximization problem is not a naturally occurring optimization problem, but instead was derived from the language recognition problem called SATISFIABILITY in Karp's paper [6]. Although such derived problems often seem to have no practical applications, it is hoped that their analysis can help us learn more about the nature of polynomial complete problems in general. MAXIMUM SATISFIABILITY

has been chosen as an example because our results illustrate another type of behavior for approximation algorithms.

Let $L = \bigcup_{i>0} \{x_i, \bar{x}_i\}$ be a set of literals. A clause will be any finite subset $C \subseteq L$. A truth assignment will be any subset $T \subseteq L$ such that for *no* $i > 0$ is $\{x_i, \bar{x}_i\} \subseteq T$. Truth assignment T satisfies clause C if $C \cap T \neq \varnothing$. The MAXIMUM SATISFIABILITY problem, denoted by MS, is given by

$\text{INPUT}_{\text{MS}} = \{S: S \text{ is a finite set } \{C_1, C_2, ..., C_p\} \text{ of clauses}\}$,

$\text{SOL}_{\text{MS}}(T) = \{S' \subseteq S:$ there exists a truth assignment T which satisfies every clause $C \in S'\}$,

$m_{\text{MS}}(S') = |S'|$.

MS(k) will denote the subproblem with inputs restricted to sets of clauses, each clause of which contains at least k distinct elements. MS(k) is polynomial complete for all $k \geq 1$.

Each of our algorithms will work by generating a truth assignment TRUE, while keeping track of the associated set SUB of satisfied clauses. The first, B1, is a simple heuristic that can be implemented in time $O(n \log n)$.

1. Set SUB $= \varnothing$, TRUE $= \varnothing$, LEFT $= S$, LIT $= L$.
2. If no literal in LIT is contained in any clause of LEFT, halt and return SUB.
3. Let y be the literal in LIT which is contained in the most clauses of LEFT, and YT be the set of clauses in LEFT which contain y.
4. Set SUB $=$ SUB $\cup$ YT, LEFT $=$ LEFT $-$ YT, TRUE $=$ TRUE $\cup \{y\}$, and LIT $=$ LIT $- \{y, \bar{y}\}$, (where $\bar{\bar{x}}_i = x_i$).
5. Go to 2.

THEOREM 2. *For all* $k \geq 1$ *and* $n > 0$,

$$R[B1, \text{MS}(k)](n) \leq (k+1)/k,$$

with equality for all sufficiently large n.

Proof. Suppose Algorithm $B1$ is being applied to a set S of clauses, all of which contain at least k literals. Each time a literal is added to TRUE, the number of clauses saved, i.e., added to SUB, is by Step 3 at least as large as the number of clauses remaining in LEFT which are wounded, i.e., have one of their literals removed from LIT without being added to TRUE. When the algorithm halts, the only clauses left in LEFT are those which have been wounded as many times as they contain literals, and hence are dead. This means that when the algorithm halts there are at least $k \cdot |\text{LEFT}|$ wounds, and hence $|\text{SUB}| \geq k \cdot |\text{LEFT}|$. Thus, $S^* \leq |S| = |\text{SUB}| + |\text{LEFT}| \leq [(k+1)/k] \cdot |\text{SUB}|$. The upper bound follows.

For the lower bound, we give an example when $k = 3$. Similar examples can be constructed for any other $k > 0$. Consider the input

$$S = \{\{x_1, x_2, x_3\}, \{\bar{x}_1, x_4, x_5\}, \{\bar{x}_2, x_6, x_7\}, \{\bar{x}_3, x_8, x_9\}\}.$$

Clearly $S^* = 4$ since the truth assignment $T = \{x_1, x_4, x_6, x_8\}$ will satisfy all four clauses. However, each of the literals occurs exactly once, so that $B1$ might choose TRUE $= \{\bar{x}_1, \bar{x}_2, \bar{x}_3\}$, thus killing the clause $\{x_1, x_2, x_3\}$. Hence $B1(S) = 3$ and so $r(B1, S) = 4/3 = (k + 1)/k$.

Obviously this algorithm leaves much room for improvement. Since the worst case examples require that all the wounded clauses eventually die, it apparently could be to our advantage to have our algorithm save as many of the wounded as possible. Algorithm $B2$ incorporates this idea and still only requires time $O(n \log n)$:

1. Assign to each clause $C \in S$ a weight $w(C) = 2^{-|C|}$. Set SUB = TRUE = $\varnothing$, LIT = L, and LEFT = S.
2. If no literal in any clause of LEFT is in LIT, halt and return SUB.
3. Let y be any literal occurring in both LIT and a clause of LEFT. Let YT be the set of clauses in LEFT and containing y, YF the set of clauses in LEFT containing $\bar{y}$.
4. If $\sum_{C \in \mathrm{YT}} w(C) \geqslant \sum_{C \in \mathrm{YF}} w(C)$, set TRUE = TRUE $\cup \{y\}$, SUB = SUB $\cup$ YT LEFT = LEFT $-$ YT, and for each $C \in$ YF, set $w(C) = 2w(C)$. Otherwise, set TRUE = TRUE $\cup \{\bar{y}\}$, SUB = SUB $\cup$ YF, LEFT = LEFT $-$ YF, and for each $C \in$ YT, set $w(C) = 2w(C)$.
5. Set LIT = LIT $- \{y, \bar{y}\}$, and go to 2.

Note that since the choice of which literal to consider at Step 3 is left largely undetermined, there is room in this algorithm for the insertion of heuristics which might improve its average case behavior. Its worst case behavior is already perhaps a surprising improvement on that of algorithm $B1$ as follows.

THEOREM 3. *For $k \geqslant 1$ and $n > 0$, $R[B2, \mathrm{MS}(k)](n) \leqslant 2^k/(2^k - 1)$, with equality for all sufficiently large n.*

Proof. Suppose algorithm $B2$ is applied to a set S, all of whose clauses contain at least k literals. Initially, the total weight of all the clauses in LEFT cannot exceed $|S|/2^k$. During each iteration, the weight of the clauses removed from LEFT is, by Step 4 of algorithm $B2$, at least as large as the weight added to those remaining clauses which receive new wounds. Thus the total weight of the clauses in LEFT can

never increase, and so, when the algorithm halts, it still cannot exceed $|S|/2^k$. But each of the dead clauses in LEFT when the algorithm halts must have been wounded as many times as it had literals, hence must have had its weight doubled that many times, and so must have a final weight of 1. Therefore $|\text{LEFT}| \leqslant |S|/2^k$, and so $|\text{SUB}| \geqslant |S|(1 - 1/2^k)$ and the upper bound follows.

We again give a lower bound example for $k = 3$ which can be generalized for arbitrary k. Consider the input

$$\begin{aligned} S = \{&\{x_1, x_2, x_3\}, \{\bar{x}_1, x_4, x_5\}, \\ &\{x_1, \bar{x}_2, x_3\}, \{\bar{x}_1, x_6, x_7\}, \\ &\{x_1, x_2, \bar{x}_3\}, \{\bar{x}_1, x_8, x_9\}, \\ &\{x_1, \bar{x}_2, \bar{x}_3\}, \{\bar{x}_1, x_{10}, x_{11}\}\}, \end{aligned}$$

where x_1 occurs in clauses with all possible combinations of a literal from $\{x_2, \bar{x}_2\}$ and one from $\{x_3, \bar{x}_3\}$, and $\bar{x}_1$ occurs in an equal number of clauses, each filled out with new literals. $S^* = 8$ since the truth assignment $T = \{x_1, x_4, x_6, x_8, x_{10}\}$ satisfies all eight clauses. However, if algorithm $B2$ chose literal $\bar{x}_1$ first, we would have $\sum_{C \in \text{YT}} w(C) = 1/2 \geqslant \sum_{C \in \text{YF}} w(C) = 1/2$, and so $\bar{x}_1$ would have been added to TRUE, with the consequence that one of the four clauses containing x_1 must eventually die. Thus $B2(S) = 7$, and so $r(B2, S) = 8/7 = 2^k/(2^k - 1)$.

An interesting fact about the proofs of Theorems 2 and 3 is that they are independent of the value of S^*, and hence actually prove stronger results than are given by the statements of the theorems. The proof of Theorem 3, for instance, guarantees that $B2$ will yield a set of size at least $\lceil |S|(2^k - 1)/2^k \rceil$, no matter what S^* is, and the reader may verify that S^* *can* be as small as $\lceil |S|(2^k - 1)/2^k \rceil$.

5. SET COVERING I

This section deals with a minimization problem, denoted by SC, from which we can again obtain a natural sequence of polynomial complete subproblems. However for this problem the restriction on the input is relaxed as the index k of the subproblem SC(k) increases, and although we have an algorithm $C1$ such that, for each k, $R[C1, \text{SC}(k)]$ is bounded by a constant independent of the size of the input, this constant can grow arbitrarily large as k increases.

In our format, the problem is described as follows.

$$\text{INPUT}_{\text{SC}} = \{F: F \text{ is a finite family } \{S_1, S_2, \ldots, S_p\} \text{ of finite sets}\}.$$

$$\text{SOL}_{\text{SC}}(F) = \left\{F' \subseteq F: \bigcup_{S \in F'} S = \bigcup_{S \in F} S\right\}.$$

$$m_{\text{SC}}(F') = |F'|.$$

If we consider the input F to be a finite cover of the set $T = \bigcup_{S \in F} S$, then an optimal solution to the problem is a minimum cardinality subcover. This problem has practical applications in areas such as logic design and fault testing.

SC(k) will denote the subproblem with inputs restricted to families, no set of which has more than k elements. For $k \geqslant 3$, each of these subproblems is itself polynomial complete. Our algorithm $C1$ is again a straightforward heuristic, implementable in time $O(n \log n)$ and given by the following.

1. Set SUB $= \varnothing$, UNCOV $= \bigcup_{S \in F} S$, $N = |F|$, SET[i] $= S_i$, $1 \leqslant i \leqslant N$.
2. If UNCOV $= \varnothing$, halt and return SUB.
3. Choose $j \leqslant N$ such that |SET[j]| is maximized.
4. Set SUB $=$ SUB $\cup \{S_j\}$, UNCOV $=$ UNCOV $-$ SET[j], SET[i] $=$ SET[i] $-$ SET[j], $1 \leqslant i \leqslant N$.
5. Go to 2.

THEOREM 4. *For all* $k \geqslant 1$ *and* $n > 0$, $R[C1, \mathrm{SC}(k)](n) \leqslant \sum_{j=1}^{k} (1/j)$, *with equality for all sufficiently large* n.

Proof. First we show that the bound is attainable for sufficiently large n. Let the set to be covered, T, consist of $k \cdot k!$ points, as shown in Fig. 1. F will consist of $k!$ sets making up the optimal subcover F_0, and $k!(\sum_{j=1}^{k} (1/j))$ additional sets which will make up a subcover F_1 that algorithm $C1$ might choose. Divide T into k segments of $k!$ points each, labeled from 1 to k, as shown.

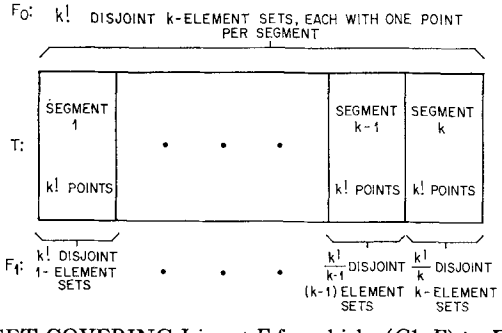

FIG. 1. SET COVERING I input F for which $r(C1, F) \geqslant \sum_{j=1}^{k} (1/j)$.

The optimal subcover F_0 consists of $k!$ disjoint sets, each containing one point from each of the k segments. The choosable subcover F_1 is made up of $k!/k$ disjoint k-element sets forming a cover of segment k, $k!/(k-1)$ disjoint $(k-1)$-element sets forming a cover of segment $k-1$,..., and $k!$ disjoint single element sets forming a cover of segment 1. $C1$ could choose the $k!/k$ sets covering segment k first, since each will cover k new points. After these have been chosen, no remaining set covers more

than $k-1$ new points, so $C1$ could next choose the $k!/(k-1)$ sets covering segment $k-1$. This process could continue until $C1$ had chosen the entire subcover F_1 with its $k!(1/k + 1/(k-1) + \cdots + 1)$ sets, and so $r(C1, F) \geqslant \sum_{j=1}^{k} (1/j)$.

For the upper bound, let us consider the operation of the algorithm. The only internal variables which are interrogated when decisions are made are N, UNCOV, and SET[i], $1 \leqslant i \leqslant N$. We shall thus define a configuration K of the algorithm to be a triple

$$\langle N_K, \text{UNCOV}_K, \langle \text{SET}_K[1], \ldots, \text{SET}_K[N_K] \rangle \rangle,$$

where N_K is a value for N, etc., and $\bigcup_{i=1}^{N_K} \text{SET}_K[i] = \text{UNCOV}_K$.

A run R from a configuration K will be a sequence

$$R = \langle K(1), j(1), K(2), j(2), \ldots, K(t-1), j(t-1), K(t) \rangle$$

of alternating configurations and integers such that $K(1) = K$, $\text{UNCOV}_{K(i)} \neq \varnothing$, $i < t$, $\text{UNCOV}_{K(t)} = \varnothing$, and such that if the algorithm enters Step 2 in configuration $K(i)$, $i < t$, it can choose $j(i)$ at Step 3, and the resulting configuration, after choosing $j(i)$ and updating at Step 4, is $K(i+1)$. We may think of a run as a possible pass through the algorithm, and say for instance that when $j(i)$ was chosen, the value of SET[k] was $\text{SET}_{K(i)}[k]$, $1 \leqslant k \leqslant N_K$.

Note that all the j's in the sequence must be distinct, since once j is chosen, SET(j) is set to $\varnothing$, and so j cannot be chosen again. Let Numbers(R) be the set of the j's in the run R. We say that a set M of integers is selectable from configuration K if there is a run R from K such that $M = \text{Numbers}(R)$. The following lemma is an obvious consequence of these definitions.

LEMMA 1. *Suppose $F_1 \subseteq F$ is a subcover and $M1 = \{i: S_i \in F_1\}$. Let K be the configuration of algorithm $C1$ after it has been given input F and initialized itself via Step 1. Then $F1$ is choosable by $C1$ given F if and only if $M1$ is selectable from K.*

The desired upper bound will be a consequence of a second, less trivial, lemma.

LEMMA 2. *Let K be any configuration, and write $n(K, i)$ for $|\text{SET}_K[i]|$. If $M1$ is selectable from K and $M0$ is* any *set such that $\bigcup_{i \in M0} \text{SET}_K[i] = \text{UNCOV}_K$, then*

$$|M1| \leqslant \sum_{i \in M0} \left(\sum_{j=1}^{n(K,i)} (1/j) \right).$$

Proof of Lemma. We proceed by induction on $h = \text{MAX}\{n(K, i): i \in M0\}$. The lemma holds trivially for $h = 0$, because if $|\text{SET}_K[i]| = 0$, $i \in M0$, and yet $\text{UNCOV}_K =$

$\bigcup_{i\in M0} \mathrm{SET}_K[i]$, we must have $\mathrm{UNCOV}_K = \varnothing$, and hence for any $M1$ selectable from K,

$$|\, M1\,| = 0 = \sum_{i\in M0} \left(\sum_{j=1}^{0} (1/j)\right),$$

since the inner sums on the right are sums of 0 terms.

Suppose now the lemma holds for $h = k - 1$. We shall show it also holds for $h = k$. So let K be a configuration, $M0$ be a set such that $\mathrm{MAX}\{n(K, i): i \in M0\} = k$ and $\bigcup_{i\in M0} \mathrm{SET}_K[i] = \mathrm{UNCOV}_K$, and suppose the set $M1$ is selectable from K. Since $M1$ is selectable, there is a run $R = \langle K(1), j(1),..., j(t-1), K(t)\rangle$ with $K(1) = K$ and such that $M1 = \mathrm{Numbers}(R)$. Let r be the least i such that $|\,\mathrm{SET}_{K(i)}[j(i)]\,| < k$, that is, the index of the first j in the run R for which $|\,\mathrm{SET}[j]\,|$ was less than k when j was chosen, (if none exists, let $r = t$).

If we let $M = \{j(i): i < r\}$ and $\mathrm{COV} = \mathrm{UNCOV}_{K(1)} - \mathrm{UNCOV}_{K(r)}$, then, since every j chosen prior to $j(r)$ had $|\,\mathrm{SET}(j)\,| \geqslant k$ at the time was chosen, we must have

$$|\, M\,| \leqslant |\,\mathrm{COV}\,|/k. \tag{5.1}$$

Now let us turn to configuration $K' = K(r)$. The set $M1' = \{j(i): i \geqslant r\} = M1 - M$ is selectable from K', since $R' = \langle K(r), j(r),..., j(t-1), K(t)\rangle$ is a run from K', inheriting this property from R. Furthermore, consider the sets $\mathrm{SET}_{K'}[i]$, $i \in M0$. None of these sets can have more than $k - 1$ elements, since $j(r)$ can be chosen from $K' = K(r)$ at Step 3, and $|\,\mathrm{SET}_{K'}[j(r)]\,| < k$ by definition (unless $r = t$, in which case $|\,\mathrm{SET}_{K'}[i]\,| = 0$ for all i). Moreover, since $\mathrm{SET}_{K'}[i] = \mathrm{SET}_K[i] - \mathrm{COV}$, we have that

$$\bigcup_{i\in M0} \mathrm{SET}_{K'}[i] = \bigcup_{i\in M0} \mathrm{SET}_K[i] - \mathrm{COV} = \mathrm{UNCOV}_{K'}.$$

Thus the induction hypothesis for $k - 1$ applies to K', $M1'$, and $M0$, and we have

$$|\, M1'\,| \leqslant \sum_{i\in M0} \left(\sum_{j=1}^{n(K',i)} (1/j)\right). \tag{5.2}$$

Now for each $i \in M0$,

$$\sum_{j=1}^{n(K,i)} (1/j) = \sum_{1}^{n(K',i)} (1/j) + \sum_{n(K',i)+1}^{n(k,i)} (1/j).$$

Since $\mathrm{SET}_{K'}[i] \subseteq \mathrm{SET}_K[i]$, and $|\,\mathrm{SET}_K[i]\,| \leqslant k$ for $i \in M0$ by assumption, and since $(1/j) \geqslant (1/k)$ for $1 \leqslant j \leqslant k$, the above quantity thus exceeds

$$\sum_{j=1}^{n(K'i)} (1/j) + |\,\mathrm{SET}_K[i] - \mathrm{SET}_{K'}[i]\,| \cdot (1/k).$$

Then, since $\sum_{i\in M0} |\, \mathrm{SET}_K[i] - \mathrm{SET}_{K'}[i]| \geqslant |\, \bigcup_{i\in M0} (\mathrm{SET}_K[i] - \mathrm{SET}_{K'}[i])| = |\, \mathrm{UNCOV}_K - \mathrm{UNCOV}_{K'} \,| = |\, \mathrm{COV} \,|$, we have

$$\sum_{i\in M0}\left(\sum_{j=1}^{n(K,i)} (1/j)\right) \geqslant \sum_{i\in M0}\left(\sum_{j=1}^{n(K',i)} (1/j)\right) + |\, \mathrm{COV} \,|/k$$

$$\geqslant |\, M1' \,| + |\, M \,| = |\, M1 \,|,$$

by (5.1) and (5.2). Thus the lemma holds for $h = k$ and, by induction, is proved.

To conclude the upper bound proof, let F be any family, no set of which contains more than k elements, F_0 an optimum subcover, and F_1 a choosable subcover with $C1(F) = |\, F_1 \,|$. Let K be the configuration of C1 after it has been given input F and initialized itself via Step 1. Let $M0 = \{i\colon S_i \in F_0\}$ and $M1 = \{i\colon S_i \in F_1\}$. Since F_1 is choosable given F, $M1$ must be selectable from K, by Lemma 1. Moreover, since F_0 is a subcover, $\bigcup_{i\in M0} \mathrm{SET}_K[i] = \mathrm{UNCOV}_K$. Thus Lemma 2 applies and we have

$$C1(F) = |\, F_1 \,| = |\, M1 \,|$$

$$\leqslant \sum_{i\in M0}\left(\sum_{j=1}^{n(K,i)} (1/j)\right) = \sum_{S_0\in F}\left(\sum_{j=1}^{|S|} (1/j)\right)$$

$$\leqslant \sum_{S_0\in F}\left(\sum_{j=1}^{k} (1/j)\right) = |\, F_0 \,| \sum_{j=1}^{k} (1/j),$$

and the upper bound follows.

As a consequence of Theorem 4, we can conclude that, for the general problem SC with no restriction on inputs, $R[C1, \mathrm{SC}](n) \leqslant \ln(n) + 1$, where n is the size of the input. This is because if it takes n symbols to describe the family F, surely no set in F can have more than n elements, and

$$\sum_{j=1}^{K} (1/j) < 1 + \ln(k) < \sum_{j=1}^{K} (1/j) + 1/2.$$

That $R[C1, \mathrm{SC}]$ actually *is* $O(\ln(n))$ can be seen from the generic example given in Fig. 2 [9].

Here T consists of $3 \cdot 2^k$ points. The optimal cover F_0 consists of three disjoint sets of 2^k points each, so $F^* = 3$. The subcover F_1 found by algorithm $C1$ however consists of $k + 1$ sets of size $3 \cdot 2^{k-1}$, $3 \cdot 2^{k-2}$,..., $3 \cdot 4$, $3 \cdot 2$, 3, and 3, respectively. Since F can be described by giving a bit vector of length $|\, T \,|$ for each element of F, we may say $n \sim |\, F \,| \cdot |\, T \,| = (k + 4)3 \cdot 2^k$, and so $\log_2(n) \sim k + \log_2(k) \sim k$. Thus

$$r(C1, F) = (k + 1)/3 \sim (\log_2 n)/3 \sim (.48)(\ln(n)).$$

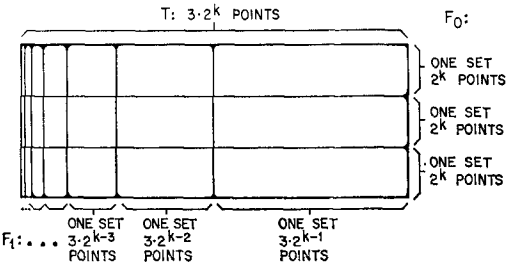

FIG. 2. SET COVERING I input F for which $r(C1, F) \sim O(\log_2 n)$.

Since k can be taken to be arbitrarily large, it follows that $R[C1, \mathrm{SC}]$ must be at least $O(\ln(n))$.

As an example of how results from one polynomial complete optimization problem can be directly applied to another if the reduction between the two is simple enough, we now briefly consider the problem NC of finding a minimal node cover. This minimization problem has inputs which are graphs $G = (N, A)$, and approximate solutions are subsets $N' \subseteq N$ of the nodes such that for each arc in A, one of its endpoints is in N'. The measure is $m(N') = |N'|$. NC(k) refers to the subproblem with inputs restricted to graphs, no node of which is connected to more than k other nodes.

NC is equivalent to SC with inputs F restricted so that every point in $\bigcup_{S \in F} S$ is in precisely two sets of F, for in this case the points may be made to correspond to arcs of a graph and the sets to nodes. Since our examples in Figs. 1 and 2 obeyed this restriction, all our lower bound results for SC and SC(k) apply to NC and NC(k). (The upper bounds apply because NC is a subproblem of SC.)

However, if we still further restrict the inputs, and let SC′(k) and NC′(k) be the problems where *exactly* k elements are in each set of the input family F, then although the lower bound results for SC(k) can still be shown to hold for SC′(k), we have for NC′(k) an entirely different (and stronger) result as follows.

THEOREM 5. *For all* $k \geq 1$ *and* $n > 0$, $R[C1, \mathrm{NC}'(k)](n) \leq 2$, *and for all sufficiently large* n, $R[C1, \mathrm{NC}'(k)](n) \geq (2k - 1)/k$.

The upper bound given here is better than the one guaranteed by the SC(k) result for all $k \geq 4$. However, note that if all points of T occur in precisely two sets of F and all sets have precisely k elements, $|F| = (2|T|/k) \leq 2F^*$. Although this constitutes a short proof of the theorem's upper bound, its major significance is that it shows that the algorithm which simply says, "Given input F, return F," can guarantee this same upper bound without doing any work at all!

6. SET COVERING II

The following set covering problem differs from SC only in the measure chosen. It is a minimization problem based on the EXACT COVER recognition problem presented in [6], which asks whether an input family F has a disjoint subcover. Our optimization problem will hence be denoted by EC:

$$\text{INPUT}_{\text{EC}} = \{F: F \text{ is a finite family } \{S_1, S_2, \ldots, S_p\} \text{ of finite sets}\}.$$

$$\text{SOL}_{\text{EC}}(F) = \left\{F' \subseteq F: \bigcup_{S \in F'} S = \bigcup_{S \in F} S\right\}.$$

$$m_{\text{EC}}(F') = \sum_{S \in F'} |S|.$$

An optimal solution is a subcover of the set $T = \bigcup_{S \in F} S$ with the least possible overlapping. EC(k) will be the subproblem with inputs restricted to families, no set of which contains more than k points. For $k \geqslant 3$, each of these subproblems is again polynomial complete.

Now if all the sets in the family F had exactly k elements, then for each subcover F' we would have $m_{\text{EC}}(F') = k \cdot m_{\text{SC}}(F')$, and the optimal solutions for both SC and EC would be the same. However, for less restricted inputs, the two problems diverge, and optimal solutions for one can be bad for the other. For instance, if OSC is an algorithm which always chooses an optimal solution for the SC problem, then (proof omitted),

$$\lim_{n \to \infty} R[\text{OSC}, \text{EC}(k)](n) = k.$$

$$R[\text{OSC}, \text{EC}] = O(n).$$

Conversely, an algorithm OEC which always chooses an optimal solution to the EC problem will have reciprocal inapplicability:

$$\lim_{n \to \infty} R[\text{OEC}, \text{SC}(k)](n) = k.$$

$$R[\text{OEC}, \text{SC}] = O(n).$$

Surprisingly, however, there exists an algorithm $C2$ which, when applied to problem EC, will have behavior quite similar to that of $C1$ applied to SC. $C2$ also can be implemented in time $O(n \log n)$ and is described as follows.

1. Set SUB $= \varnothing$, LEFT $= F$, UNCOV $= \bigcup_{S \in F} S$.
2. If UNCOV $= \varnothing$, halt and return SUB.
3. Let $S' \in$ LEFT be that set S which minimizes

$$\text{Ratio}(S) = |S - \text{UNCOV}| / |S \cap \text{UNCOV}|.$$

4. Set SUB = SUB $\cup \{S'\}$, UNCOV = UNCOV $- S'$, LEFT = LEFT $- \{S'\}$.
5. Go to 2.

THEOREM 6. *For all* $k \geqslant 1$ *and* $n > 0$,

$$R[C2, \mathrm{EC}(k)](n) \leqslant 1 + \ln(k) \leqslant \sum_{j=1}^{k} (1/j) + 1/2,$$

and for all sufficiently large n, $R[C2, \mathrm{EC}(k)](n) \geqslant \sum_{j=1}^{k} (1/j)$.

Proof. The lower bound follows from an example much like that given in Fig. 1, except that all the sets in F_1 are filled out with points from segment k so that each has exactly k elements.

For the upper bound, let F be any input, all of whose sets have k or fewer elements, and let F_0 be an optimal subcover. If we set $T = \bigcup_{S \in F} S$, we then have $m(F_0) = F^* = a \mid T \mid$, for some a, $1 \leqslant a \leqslant k$.

Now suppose F_1 is choosable by $C2$ on input F, and let us restrict our attention to some particular run of $C2$ during which F_1 is chosen. For each $S \in F_1$, let the overlap $\mathrm{ov}(S)$ be the value of $\mid S\text{-UNCOV} \mid$ when S was added to SUB by $C2$. The cumulative overlap $\mathrm{OV}(F_1)$ is defined to be $\sum_{S \in F_1} \mathrm{ov}(S)$, and we have

$$m(F_1) = \mid T \mid + \mathrm{OV}(F_1).$$

Now let us consider the action of $C2$ on F, and its relation to the sets of the optimum cover F_0. If at a given time a set S' is chosen with $\mathrm{Ratio}(S') = \mid S' - \mathrm{UNCOV} \mid / \mid S' \cap \mathrm{UNCOV} \mid \geqslant y$, then at this time each $S \in F_0$ can have at most $1/(y+1)$ of its points in UNCOV. For if $S \in F_0 \cap \mathrm{SUB}$, S has *no* points in UNCOV; otherwise we must have $\mathrm{Ratio}(S) \geqslant \mathrm{Ratio}(S') \geqslant y$, so $\mid S \cap \mathrm{UNCOV} \mid / \mid S \mid \leqslant 1/(y+1)$. Thus

$$\begin{aligned} \mid \mathrm{UNCOV} \mid &\leqslant \sum_{F \in S_0} \mid S \cap \mathrm{UNCOV} \mid \\ &\leqslant F^*/(y+1) = a \mid T \mid /(y+1). \end{aligned}$$

And therefore, at least $\mid T \mid (1 - a/(y+1))$ points, and quite possibly more, must be outside of UNCOV, i.e., already covered. Thus we cannot choose an S' with $\mathrm{Ratio}(S') \geqslant y$ until $1 - a/(y+1)$ of the points of T have been covered. Conversely, if $x = \mid T - \mathrm{UNCOV} \mid / \mid T \mid$, then $\mathrm{Ratio}(S')$ for the next set S' chosen cannot exceed $a/(1-x) - 1$.

$\mathrm{Ratio}(S')$ gives, for each new point covered by the set S', the number of points in the set which overlap points already covered. This fact leads to interpretation of $\mathrm{OV}(F_1)$ as the area under a curve. See Fig. 3.

We divide the X-axis into a sequence of intervals of length $1/\mid T \mid$ corresponding to the points of T, laid out along the X-axis in the order that the points were covered

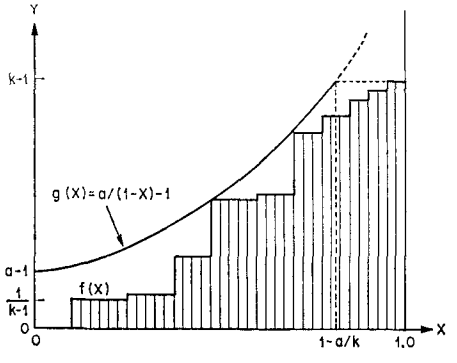

FIG. 3. $\mathrm{OV}(F_1)$ as the area under a curve.

by the algorithm. The Y-value for each interval is the number of overlaps contributed by the corresponding point, as measured by Ratio(S') for the first set S' that covered it. The function f so defined is an increasing step function on the interval (0, 1], with maximum value $k - 1$, and

$$\mathrm{OV}(F1) = |T| \int_0^1 f(x)\,dx.$$

Now an X-value x in the graph can be thought of as representing the value of $|T - \mathrm{UNCOV}|/|T|$ at the time the point was chosen which corresponds to the interval containing x. Thus by our reasoning a few paragraphs above, $f(x) \leqslant a/(1 - x) - 1$, for all $x \in (0, 1]$. Thus an upper bound on $\mathrm{OV}(F_1)$ will be $|T|$ times

$$\int_0^{1-a/k} \left[\frac{a}{1-x} - 1\right] dx + \int_{1-a/k}^1 [k-1]\,dx,$$

where we have replaced $g(x) = a/(1 - x) - 1$ by $k - 1$ for $x > 1 - a/k$, since we have $g(x) > k - 1$ for such x, even though $f(x)$ never can be. Transforming and evaluating, we find that the above equals

$$\begin{aligned}
&\int_{a/k}^1 \frac{a}{z}\,dz - [z]_0^{1-a/k} + (k-1)\frac{a}{k} \\
&\quad = a[0 - \ln(a) + \ln(k)] - 1 + a/k + a - a/k \\
&\quad \leqslant a[\ln(k) + 1] - 1.
\end{aligned}$$

Thus $\mathrm{OV}(F_1) \leqslant |T|(a[\ln(k) + 1] - 1)$, and so

$$\begin{aligned}
m(F_1) &\leqslant |T| + |T|(a[\ln(k) + 1] - 1) \\
&= a\,|T|[\ln(k) + 1] = F^*[\ln(k) + 1].
\end{aligned}$$

The upper bound follows.

Thus, despite the differences in measures and the differences in algorithms, our results for $R[C2, \mathrm{EC}(k)]$ are practically the same as the results for $R[C1, \mathrm{SC}(k)]$. Continuing the parallel, we also have $R[C2, \mathrm{EC}] = O(\ln(n))$. The lower bound example is a modification of that given in Fig. 2, where F_0 is as before, and each of the smaller sets of F_1 has points added to it from the largest one to bring its total cardinality up to $3 \cdot 2^{k-1}$. Thus $C2(F) = (k+1)(3 \cdot 2^{k-1})$, and so $r(C2, F) = (k+1)/2 = O(\ln(n))$, where the problem size n is the same as in the original example.

7. GRAPH COLORING

The results of the next two sections are negative in character, as we can show only lower bounds on the worst case behavior of the approximation algorithms we study. However, these lower bounds will be sufficiently large to serve as a warning that such simple heuristics may well have drawbacks. In our format, GRAPH COLORING, denoted by GC, is a minimization problem given by the following.

$$\mathrm{INPUT}_{\mathrm{GC}} = \{G = (N, A): G \text{ is a finite undirected graph with nodes } N \text{ and arcs } A\}.$$

$$\mathrm{SOL}_{\mathrm{GC}}(G) = \{h: N \to \{1, 2, \ldots, |N|\}: \text{if there is an arc between nodes } x \text{ and } y, h(x) \neq h(y)\}.$$

$$m_{\mathrm{GC}}(h) = |\{z: z = h(x), \text{ for some } x \in N\}|.$$

An approximate solution can be interpreted as a coloring of the nodes of the graph, such that no two adjacent nodes have the same color. An optimal solution is a coloring with the minimum possible number of colors. If $k \geqslant G^*$, we say that G is k-colorable. This problem is often called the "timetabling problem," and occurs in practical situations such as scheduling exams. Welsh and Powell [10], Wood [11], and Matula, Marble, and Isaacson [7] have described what are essentially approximation algorithms for the problem, although they did not analyze their worst case behavior. The Welsh and Powell algorithm, which we shall call D1, is the same sort of simple heuristic we have studied above:

1. Set UNCOLORED $= N$, COLORED$[i] = \varnothing$, $1 \leqslant i \leqslant |N|$, $I = 1$.
2. If UNCOLORED $= \varnothing$, halt and return COLORED$[i]$, $1 \leqslant i \leqslant I$.
3. If each node in UNCOLORED is connected to some node in COLORED$[I]$, set $I = I + 1$.
4. Let x be a node in UNCOLORED with maximum degree among those not connected to any node in COLORED$[I]$.
5. Set COLORED$[I]$ = COLORED$[I] \cup \{x\}$, UNCOLORED = UNCOLORED $- \{x\}$, and go to 2.

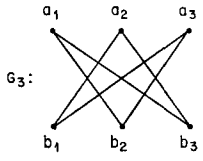

FIG. 4. Two-colorable graph G_3 with $D1(G_3) = 3$.

This algorithm behaved reasonably for Welsh and Powell on the small examples they considered; however, there is a sequence $\{G_k\}$ of two-colorable graphs such that graph G_k has only $2k$ nodes but $D1(G_k) = k$. Figure 4 shows G_3. Graph $G_k = (N_k, A_k)$ has nodes and arcs:

$$N_k = \{a_1, \ldots, a_k, b_1, \ldots, b_k\},$$
$$A_k = \{(a_i, b_j): i \neq j\}.$$

G_k has two-coloring $h(a_i) = 1$, $h(b_i) = 2$, $1 \leqslant i \leqslant k$. Algorithm $D1$, since all nodes have the same degree, could color the nodes in the order $a_1, b_1, a_2, b_2, \ldots, a_k, b_k$, and so set $h(a_1) = h(b_1) = 1$, $h(a_2) = b(h_2) = 2, \ldots$, and $h(a_k) = h(b_k) = k$, thus using k colors. Hence $r(D1, G_k) = k/2$. Since the number of arcs in G_k is proportional to k^2, and we can describe a graph by listing its nodes and arcs, we thus can conclude that $R[D1] \geqslant O(n^{1/2})$.

One way to try to improve this algorithm is to use a different rule for choosing x in Step 4. For instance, consider algorithm $D2$ which replaces Step 4 in the above by the following.

4. Let POSS be that subset of UNCOLORED made up of all the points not connected to any member of COLORED[I]. Let x be the node in POSS connected to the fewest other nodes in POSS.

It is easy to see that $R[D2] \geqslant O(\log n)$: There is a sequence $\{H_k\}$ of two-colorable graphs such that H_k has 2^k points and $D2(H_k) = k + 1$. Figure 5 shows H_1–H_4. H_{k+1} is obtained from H_k by adding 2^k new points and arcs, each new point connected by a new arc to a different one of the 2^k points of H_k. The new points are the circled points in the figure. Algorithm $D2$ applied to H_{k+1} would choose these new points first, coloring them all the same color, and then would have to start with a second color

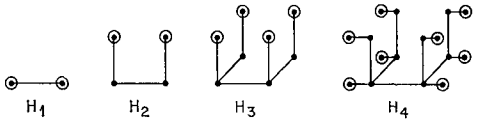

FIG. 5. Graphs H_k, $1 \leqslant k \leqslant 4$.

on the remaining graph, which is just H_k. Since H_1 requires two colors, we thus can conclude that $D2(H_k) = k + 1$, $r(D2, H_k) = (k + 1)/2$, and $R[D2] \geqslant O(\log n)$.

In fact, these examples can be improved upon, so that the size of the graph requiring $k + 1$ colors grows more slowly than a^k for any $a > 1$ (but faster than any polynomial in k), and so the above inequality is a strict one. We can also show that the Wood algorithm [11], and even the most sophisticated of the algorithms in [7], are at least $O(\log n)$. Since we have no upper bound proofs, all these algorithms may be considerably worse.

However, one algorithm, which we shall call $D3$, has been proposed for which $R[D3]$ definitely *is* $O(\log n)$. Let an independent subset of the nodes of G be any subset, no two nodes of which are adjacent in G. $D3$ can be described as follows.

1. Set UNCOLORED $= N$, $I = 1$, COLORED$[i] = \varnothing$, $1 \leqslant i \leqslant |N|$.
2. If UNCOLORED $= \varnothing$, halt and return COLORED$[i]$, $1 \leqslant i < I$.
3. Let IND be the maximum sized independent subset of UNCOLORED.
4. Set COLORED$[I]$ = IND, UNCOLORED = UNCOLORED − IND, $I = I + 1$, and go to 2.

$D3$ is really only our set covering algorithm $C1$ in disguise, applied to the cover of N by the family of its independent subsets, and since an optimal coloring corresponds to a minimum cardinality subcover of this family, our upper bound results for SC apply, and tell us that $R[D3, \text{GC}] \leqslant O(\log n)$. The graphs H_k given in Fig. 5 can be used to show that equality holds.

Unfortunately, it is unlikely that $D3$ can be implemented in polynomial time. Step 3 is equivalent to the problem of finding the maximum sized clique in a graph, and hence is itself a polynomial complete problem. And note the results of the following section.

8. MAXIMUM CLIQUE

For this maximization problem, we have not found any polynomial-time algorithms which even *might* be as good as $O(\log n)$. Denoting the problem by MC, we have

$\text{INPUT}_{\text{MC}} = \{G = (N, A)$: G is a finite undirected graph with nodes N and arcs $A\}$,

$\text{SOL}_{\text{MC}}(G) = \{N' \subseteq N$: there is an arc in A between each pair of nodes in $N'\}$,

$m_{\text{MC}}(N') = |N'|$.

The approximate solutions are the cliques of the graph. An optimal solution is a maximum sized clique. A straightforward heuristic for this problem, implementable in time $O(n \log n)$, is algorithm $E1$ given below:

1. Set SUB $= \varnothing$, REST $= N$.
2. If REST $= \varnothing$, halt and return SUB.
3. Let $y \in$ REST be that element connected to the most other elements of REST.
4. Set SUB $=$ SUB $\cup \{y\}$. REST $=$ REST $-$ {points not connected to y}.
5. Go to 2.

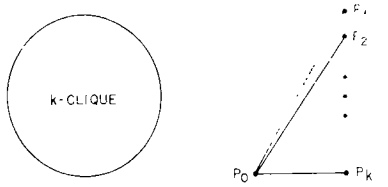

FIG. 6. Graph G with $r(E1, G) = k/2 \sim O(n^{1/2})$.

Figure 6 shows an example of a graph with $2k + 1$ points, k of which form a clique of size k, for which algorithm $E1$ will find only cliques of size two. P would be chosen first as it is the only point connected to k other points, but this foredooms the eventual clique found to have just two points. Thus $r(E1, G) = k/2$. Since the graph has proportional to k^2 arcs, we thus have $R[E1] \geqslant O(n^{1/2})$.

The alternative approach of starting with N and deleting nodes until a clique remains is just as bad. Consider algorithm $E2$:

1. Set SUB $= N$.
2. If SUB is a clique, halt and return SUB.
3. Let $y \in$ SUB be the node connected to the fewest other nodes in SUB.
4. Set SUB $=$ SUB $- \{y\}$ and go to 2.

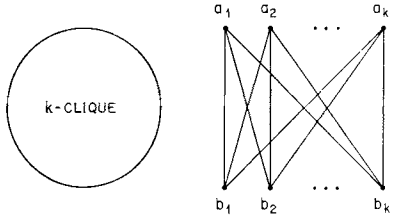

FIG. 7. Graph G with $r(E2, G) = k/2 \sim O(n^{1/2})$.

Figure 7 shows a graph with $3k$ points, k of which form a clique of size k, for which $E2$ will choose a clique of size two. The right-hand component is much like the graph G we presented in the previous section; each of the k points in the top line is connected to each of the k points in the bottom line, but no two points in the same line are con-

nected. Since each of the points in the k clique is connected to only $k - 1$ other points, the first k points deleted by $E2$ will be the points of the k clique, leaving a graph whose largest clique is of size two. Thus $r(E2, G) = k/2$, and again $R[E2] \geqslant O(n^{1/2})$.

Minor attempts at modifying $E1$ and $E2$ may avoid the pitfalls of the particular examples we have given, but to date every proposed n^2-time algorithm E based on such a modifications has been shown to also have $R[E] \geqslant O(n^{1/2})$.

The only way (apparently) to improve on this lower bound is to use algorithms requiring more than n^2 time. For instance, we might try to improve algorithm $E1$ by adding backtracking on a large scale, much as we improved algorithm $A1$ in Section 3. Let F_j be the algorithm which enumerates all the cliques of size j in the graph G, and then runs algorithm $E1$ once for each one, each time initializing SUB to the clique in question. Fj then returns the largest of the cliques found by $E1$ in this process.

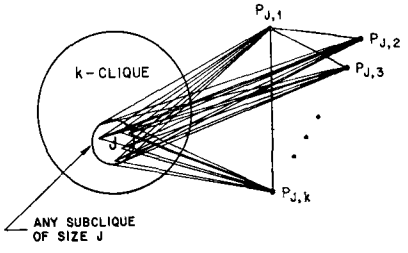

FIG. 8. Graph G with $r(Fj, G) = k/(j + 2) \sim O(n^{1/(j+1)})$.

Our best lower bound on $R[Fj]$ is $O(n^{1/(j+1)})$. Figure 8 shows a graph with $\sim k^{j+1}$ points ($k \gg j$), k of which form a clique of size k, for which F_j will only find a clique of size $j + 2$. For each subclique J of size j in the k clique. there are k additional points $P_{J,1}$–$P_{J,k}$. $P_{J,1}$ and the nodes of the clique J are mutually connected to the $k - 1$ other $P_{J,i}$'s. Since each point P_i outside of J but still in the k clique is only mutually connected with the nodes of J to $k - j - 1$ other points, algorithm $E1$ started on SUB $= J$ will next choose $P_{J,1}$ and thus have to settle for a clique of size $j + 2$. Thus $r(F_j, G) = k/(j + 2)$. Since there are $\sim k^j$ subcliques of size j in a clique of size k, our graph has $\sim k^{j+1}$ points, thus our example yields the lower bound $R[Fj] \geqslant O(n^{1/(j+1)})$.

The cost for this apparent improvement in worst case behavior is a running time proportional to n^{j+2}. If we wish to further increase our expense, we could modify $E1$ so that in Step 3 it looked for a clique of size j instead of a single point. This however, would only lower the constant of proportionality in the lower bound, as can be seen by substituting a clique of size j for each of the points $P_{J,i}$ in the above example.

We have found *no* polynomial-time algorithm E for this problem for which there does not exist an $\epsilon > 0$ such that $R[E] > O(n^{\epsilon})$, and no way to reduce the ϵ without significantly increasing the algorithm's cost.

9. Conclusion

The results described in this paper indicate a possible classification of optimization problems as to the behavior of their approximation algorithms. Such a classification must remain tentative, at least until the existence of polynomial-time algorithms for finding optimal solutions has been proved or disproved. In the meantime, many questions can be asked. Are there indeed $O(\log n)$ coloring algorithms? Are there any clique finding algorithms better than $O(n^{\epsilon})$ for all $\epsilon > 0$? Where do other optimization problems fit into the scheme of things? What is it that makes algorithms for different problems behave in the same way? Is there some stronger kind of reducibility than the simple polynomial reducibility that will explain these results, or are they due to some structural similarity between the problems as we define them? And what other types of behavior and ways of analyzing and measuring it are possible?

Note Added in Proof. Substantial improvements on the results of Section 7 have been made and are to appear.

Acknowledgments

The author wishes to thank Professors Michael J. Fischer and Albert R. Meyer for their encouragement and suggestions as to the presentation of these results.

References

1. S. A. Cook, The complexity of theorem-proving procedures, *in* "Proceedings of the 3rd Annual ACM Symposium on the Theory of Computing," 1971, pp. 151–158.
2. M. R. Garey, R. L. Graham, and J. D. Ullman, Worst-Case analysis of memory allocation algorithms, *in* "Proceedings of the 4th Annual ACM Symposium on the Theory of Computing," 1972, pp. 143–150.
3. R. L. Graham, Bounds on multiprocessing anomalies and related packing algorithms, *in* "Proceedings of the Spring Joint Computer Conference," 1972, pp. 205–217.
4. D. S. Johnson, Fast allocation algorithms, *in* "Proceedings of the 13th Annual IEEE Symposium on Switching and Automata Theory," 1972, pp. 144–154.
5. D. S. Johnson, Near-optimal bin packing algorithms, Ph.D. Dissertation, Massachusetts Institute of Technology, Cambridge, MA, 1973.
6. R. M. Karp, Reducibility among combinatorial problems, *in* "Complexity of Computer Computations" (R. E. Miller and J. W. Thatcher, Eds.), Plenum Press, New York, 1972, pp. 85–104.
7. D. W. Matula, G. Marble, and J. D. Isaacson, Graph coloring algorithms, *in* "Graph Theory and Computing" (R. C. Read, Ed.), Academic Press, New York, 1972, pp. 109–122.
8. S. K. Sahni, On the knapsack and other computationally related problems, Ph.D. Dissertation, Cornell University, Ithaca, NY, 1973.
9. J. H. Spencer, private communication.
10. D. J. A. Welsh and M. B. Powell, An upper bound to the chromatic number of a graph and its application to time-tabling problems, *Comput. J.* **10** (1967), 85–86.
11. D. C. Wood, A technique for coloring a graph applicable to large scale time-tabling problems, *Comput. J.* **12** (1969), 317–319.