ELSEVIER

European Journal of Operational Research 101 (1997) 81–92

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

Theory and Methodology

Computational experience with approximation algorithms for the set covering problem

Tal Grossman [a,1], Avishai Wool [b,2]

[a] *Theoretical Division and CNLS, MS B-213, Los Alamos National Lab, Los Alamos NM 87545, USA*

[b] *Department of Applied Mathematics and Computer Science, The Weizmann Institute, Rehovot 76100, Israel*

Received 1 March 1995; accepted 1 May 1996

Abstract

The Set Covering problem (SCP) is a well known combinatorial optimization problem, which is NP-hard. We conducted a comparative study of nine different approximation algorithms for the SCP, including several greedy variants, fractional relaxations, randomized algorithms and a neural network algorithm. The algorithms were tested on a set of random-generated problems with up to 500 rows and 5000 columns, and on two sets of problems originating in combinatorial questions with up to 28160 rows and 11264 columns.

On the random problems and on one set of combinatorial problems, the best algorithm among those we tested was a randomized greedy algorithm, with the neural network algorithm very close in second place. On the other set of combinatorial problems, the best algorithm was a deterministic greedy variant, and the randomized algorithms (both randomized greedy and neural network) performed quite poorly. The other algorithms we tested were always inferior to the ones mentioned above. © 1997 Elsevier Science B.V.

Keywords: Set covering; Optimization; Neural networks; Approximation algorithms

1. Introduction

1.1. The problem

The problem dealt with in this paper is the Set Covering problem (SCP), defined as follows.

$$\text{SCP}: \quad \min \sum_{j=1}^{n} x_j \text{ s.t. } \begin{cases} A\boldsymbol{x} \geq \mathbf{1}, \\ \boldsymbol{x} \in \{0,1\}^n, \end{cases}$$

where A is an $m \times n$ matrix, $a_{ij} \in \{0,1\}$. We use $k = \max_i \sum_{j=1}^{n} a_{ij}$ to denote the maximum number of ones appearing in any row. When all the rows have exactly $k = 2$ ones, the SCP becomes the Vertex Cover problem. We also refer to SCP as a set of m *inequalities* each containing at most k *variables*.

In the *weighted* SCP the objective function is $\sum_{j=1}^{n} c_j x_j$ for some weights (or costs) $c_j > 0$. Therefore the SCP problem is sometimes referred to as the *unicost* SCP.

The SCP is known to be NP-hard, cf. [13]. This also is a MAX-SNP-hard problem [23,12], i.e., does not have a polynomial time approximation scheme unless P=NP [1].

[1] Deceased.

[2] Present address: Bell Laboratories, 700 Mountain Ave., Murray Hill, NJ 07974, USA.
E-mail: yash@research.bell-labs.com.

PII S0377-2217(96)00161-0

The unicost version of the set covering problem can be presented as the following combinatorial problem. Consider a set of objects (points, or vertices) V and a collection E of sets of these objects. Our goal is to find a subset $C \subseteq E$ of smallest cardinality such that every element $v \in V$ is "covered" by (namely, belongs to) at least one of the subsets in C. The reverse problem is essentially the same: find a subset $S \subseteq V$ such that every element of E (which in itself is a subset of V), is "hit" by S, namely contains at least one element of S. The matrix A in both these cases is the incidence matrix of the two sets V and E. A simple instance of this problem is shown in Fig. 1.

The main goal of this paper is a comparative study of the performance of several different SCP approximation algorithms.

1.2. Related work

The first published approximation algorithms for the SCP with a worst-case analysis [21,22,8], use the greedy heuristic. The approximation ratio of the greedy algorithm (i.e., the worst ratio between the cost of a greedy solution and the optimum) is $\ln m + 1$.

The approximation algorithms of [14,19,24] have worst-case analyses showing approximation ratios of k (the maximal number of ones per row).

A randomized rounding algorithm for SCP appears in [25]. The approximation ratio of this algorithm is $k\left(1 - (c/m)^{1/k}\right)$ for some small constant c.

A number of optimal algorithms for SCP, typically based upon tree-search procedures, have appeared in the literature (e.g., [2,3]). These algorithms have been used to solve problems with up to 50 rows and 500 columns, albeit at considerable computational cost.

The Lagrangian approximation heuristic for SCP of [4] was tested on problems with up to 500 rows and 5000 columns. The approximation algorithm of [30] was tested on problems generated for airline crew scheduling, involving up to 1600 rows and 105000 columns. Genetic algorithms and simulated annealing algorithms for set covering appear in [27,6].

Several neural network based algorithms were suggested or developed for problems related to SCP (like scheduling and diagnostic problems, cf. [26,20,9]), but to our knowledge no neural network based algorithm for the SCP was actually presented and tested so far. One of the algorithms tested in this work is such a neural net based algorithm that was developed recently. It will be described in detail separately [16].

A probabilistic analysis of SCP defined by randomly generated matrices appears in [29].

1.3. New results

We conducted a computational study of several approximation algorithms for SCP. These are the simple greedy algorithm [21], the algorithms of [14,19,24], a randomized rounding algorithm [25], an alternating greedy algorithm, a randomized greedy algorithm, and a neural network algorithm.

The algorithms were tested on a set of random-generated problems with up to 500 rows and 5000 columns, and on two sets of problems originating in combinatorial questions with up to 28160 rows and 11264 columns.

On the random problems and on one set of combinatorial problems, the best algorithm among those we tested was the randomized greedy algorithm, which performed extremely well on all of the problems, and found the lowest cost on almost all of them. The neural network, alternating greedy and simple greedy algorithms were very close in second, third and fourth places. On the other set of combinatorial problems, the best algorithm was the alternating greedy algorithm, with the simple greedy algorithm second, while the randomized greedy and neural network algorithms performed quite poorly. The other algorithms we tested were inferior to the ones mentioned above on all the problems.

A preliminary version of this paper can be found in [17].

2. The algorithms

Nine algorithms were tested, six of which are deterministic and three (R-Gr, RR, and NN) are randomized. Some of the algorithms use the solution vector of the relaxed fractional set covering, denoted by $\boldsymbol{x}^*$, and its cost C^*. The fractional covering is obtained by replacing the integrality requirement on the variables and instead allowing them to take real values in the range $[0, 1]$, thereby transforming the problem to a linear program. Therefore a linear programming (LP)

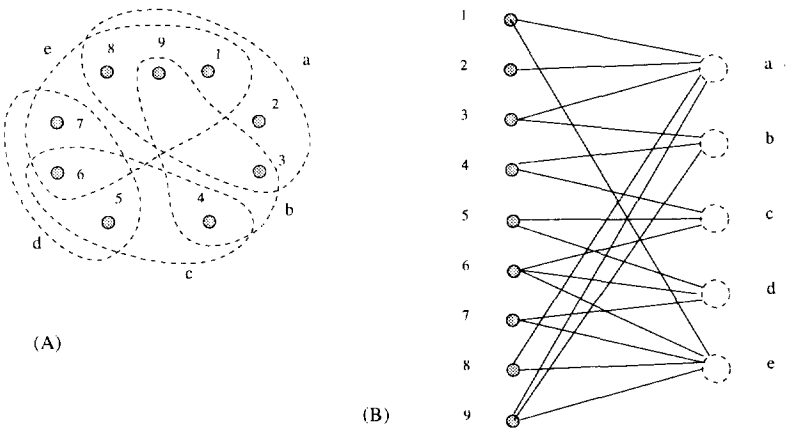

Fig. 1. A set covering instance (A): the points are denoted by numbers, the sets of points are denoted by letters. The goal is to "cover" all the points by choosing the minimum number of sets. The incidence graph of this instance is shown in (B). In the neural network used by algorithm NN (see text), the lines represent symmetric connections of weight 1.

step was performed prior to running the algorithms. Brief descriptions of the algorithms follow.

NoLP This is a generalization of Gavril's algorithm [14]. Each inequality is inspected in turn, and if it is not satisfied yet then all its variables are picked.

Thresh This is Hochbaum's algorithm [19]. Each variable for which the fractional solution value $x_j^* \geq 1/k$ is picked.

SortLP This algorithm is from [24]. The variables are checked in a decreasing order of their fractional x_j^* values, and each variable that appears in an unsatisfied inequality is picked.

Gr This is the simple greedy algorithm of [21,8]. In each iteration the variable that appears in the largest number of unsatisfied inequalities is picked. Ties are broken lexicographically (i.e., the variable with the smallest index is picked first).

R-Gr This is the randomized greedy algorithm. It is identical to the simple greedy algorithm Gr, except that ties are broken at random. The basic algorithm is iterated N times (we used $N = 100$) and the minimal solution is returned.

T-Gr This algorithm only considers variables with $x_j^* \geq 1/k$. Among these, the algorithm picks variables according to the greedy rule, with ties broken according to $\boldsymbol{x}^*$ (the variable with the largest x_j^* value is picked first).

RR This is a randomized rounding algorithm, similar to that of [25]. Each fractional value x_j^* is multiplied by a scaling factor $\delta > 1$. Then for each variable a biased coin with probability δx_j^* is tossed, and if the coin is "1" then the variable is picked (if $\delta x_j^* > 1$ then the variable is picked with probability 1). If the resultant solution is not feasible then it is discarded. This procedure is iterated N times (we used $N = 100$), and the lowest cost solution is returned. We used a factor of $\delta = 2.5 \times C_{\mathrm{Gr}}/C^*$ where C_{Gr} is the cost found by the Gr algorithm. The value of δ was always in the range 3.12–6.31 on the test problems. Using this factor, feasible solutions were always found by the algorithm, and their cost was relatively low.

Alt-Gr This is an alternating greedy algorithm. The algorithm works in iterations, and in each iteration two steps are performed. First, as in algorithm Gr, the variable that appears in the largest number of unsatisfied inequalities is added to the solution. Let Δ denote the increase in the number of satisfied inequalities due to this addition. Then, variables are repeatedly discarded from the solution, as long as the total decrease in the number of satisfied inequalities is at most $\Delta - 1$. In each repetition the variable that causes the minimal decrease is discarded. In particular, any variable that becomes redundant after the addition in the first step is discarded. Ties in

both steps are broken lexicographically. Note that a variable may be discarded from the solution and then picked again in a later iteration. However the discarding rule ensures that the number of satisfied inequalities strictly increases with each iteration so the algorithm always terminates.

NN This is a neural network algorithm. The network has two types of units (neurons), one representing the inequalities (rows) and the other representing the variables (columns). They are connected as a bipartite graph which represents the incidence relations between rows and columns in the A matrix (see Fig. 1).

In its basic mode of operation, using steepest descent deterministic dynamics, the network is set in an arbitrary initial state and converges to a final state which represents a valid minimal cover. On top of the basic network dynamics, the algorithm applies a gradual change of the threshold parameter of the variable (column) units' activation function, thus implementing a deterministic t-annealing procedure. At the top-most level, the algorithm has an adaptive restart procedure to search for initial states. We used $N = 100$ such restarts. The network and the algorithm are described in detail in [16].

The first seven algorithms (all except Alt-Gr and NN) can terminate with a solution which is not minimal. Therefore a redundancy elimination procedure was performed on their solutions. For the RR algorithm, this procedure was performed on each feasible solution. The procedure discards variables one by one until a minimal solution is attained. In each iteration the procedure calculates the redundancy r_i of each inequality (the number of extra variables covering it). The minimal redundancy mr_j of a variable j is the minimal r_i of any inequality i that j appears in. The variable to be discarded is the one with the maximal mr_j value.

3. Methods

3.1. Random generated problems

The algorithms were tested on 60 randomly generated problems, (see Table 1 for details), which are available electronically [5]. Problem sets 4–6 are from [2], problem sets A–E are from [3], and problem sets NRE–NRF are from [4]. All the problems were randomly produced according to the specified densities, using the scheme of [2], namely every column covers at least one row and every row is covered by at least two columns.

Table 1
Details of the random problems

Problem set	Rows (m)	Columns (n)	Density [1] (%)	Max number of ones per row [2]	Number of (k) problems
4	200	1000	2	36	10
5	"	2000	2	60	"
6	"	1000	5	71	5
A	300	3000	2	81	"
B	"	"	5	192	"
C	400	4000	2	105	"
D	"	"	5	244	"
E	50	500	20	124	"
NRE	500	5000	10	561	"
NRF	"	"	20	1086	"

[1] The density is the percentage of ones in the A matrix.
[2] The maximum is taken over all the problems in each set and over the rows in each problem.

All these problems were originally generated to test *weighted* SCP algorithms. In order to use them for our unicost SCP algorithms, we discarded the column costs. This modification implies that the costs obtained by our algorithms are incomparable with the optimal costs reported in the above-mentioned papers. One exception is problem set E, which was originally produced as a unicost problem set.

3.2. Combinatorial problems

The algorithms were also tested on 10 problems that arise out of two combinatorial questions. See Table 2 for details.

The first set of problems, the CYC set, is derived from an old question of Erdős, cf. [11]: "what is the minimal number of edges of a hypercube that can be chosen so that every cycle of 4 edges contains at least one chosen edge ?". This question translates to an SCP, with hypercube edges corresponding to the variables and an inequality per 4-cycle. If the dimension of the hypercube is d, then the SCP has $n = d2^{d-1}$ variables and $m = 2^{d-2}\binom{d}{2}$ inequalities with $k = 4$ variables each. The optimal fractional solution of this SCP has

a cost of $C^* = n/4$, and a possible (non-unique) fractional solution vector is $x^* = (1/4, 1/4, \ldots, 1/4)$.

The optimal solution for this problem is known for all $d \leq 6$. For larger d it is an open question. For the currently best explicit solutions see [7,18] and the references therein. We used problems generated from hypercubes of dimension $d = 6, 7, 8, 9, 10, 11$.

The second set of problems, the CLR set, is derived from another question of Erdős [10]: "what is $m_d(r)$, the minimal number of r-tuples that can be chosen so that every coloring of d elements with 2 colors contains at least one monochromatic r-tuple ?" (Note: in combinatorics texts this function is usually denoted $m(n)$, where n replaces our r as the size of the tuples, and minimizing over all possible universe sizes d). The question is interesting only when $r < \lceil d/2 \rceil$, otherwise there is no solution (except for the case $r = \lceil d/2 \rceil$ with d odd; but then only a trivial solution exists, namely all r-tuples must be chosen).

This question translates to an SCP with r-tuples corresponding to variables and an inequality per coloring, with the A matrix defined by having $a_{ij} = 1$ iff r-tuple j is monochromatic under coloring i. The SCP has $n = \binom{d}{r}$ variables and $m = 2^{d-1} - 1$ inequalities, discarding the all-white/all-black colorings and ignoring symmetry. The maximal number of variables per inequality is $k = \binom{d-1}{r}$, and the minimal number is $k_{\min} = \binom{\lceil d/2 \rceil}{r} + \binom{\lfloor d/2 \rfloor}{r}$. The optimal fractional solution of this SCP has a cost of $C^* = n/k_{\min}$, and a possible solution vector is $x^* = (1/k_{\min}, 1/k_{\min}, \ldots, 1/k_{\min})$.

This is an open combinatorial question for $r \geq 4$. For the currently best lower bounds for $m_d(4)$, and best explicit solution constructions, see [15] and the references therein. We used problems generated with $r = 4$ and $d = 10, 11, 12, 13$.

We have contributed the CYC and CLR problem sets to the OR-library, thus they are now available electronically [5].

3.3. The tests

The algorithms were programmed in C, using the IBM AIX XL C compiler, with optimization turned on. The tests were conducted on an IBM RS6000 model 370 workstation with 128MB memory. The linear programming solver used was R.J. Vanderbei's interior-point solver LOQO [28].

Table 2
Details of the combinatorial problems

Problem	Rows (m)	Columns (n)	Density (%)	Number of ones per row, Min–Max
CYC.6	240	192	2.1	4–4
CYC.7	672	448	0.9	4–4
CYC.8	1792	1024	0.4	4–4
CYC.9	4608	2304	0.2	4–4
CYC.10	11520	5120	0.08	4–4
CYC.11	28160	11264	0.04	4–4
CLR.10-4	511	210	12.3	10–126
CLR.11-4	1023	330	12.4	20–210
CLR.12-4	2047	495	12.5	30–330
CLR.13-4	4095	715	12.5	50–495

Table 3 contains the costs our algorithms obtained on the random problems, along with the fractional solution costs. Table 4 contains the results obtained on the combinatorial problems. The minimal costs that were found for each problem are displayed in boxes on the appropriate columns.

Table 5 contains a summary of the quality of the algorithms' solutions for each class of problems. Since the optimal solution is unknown for all the problems we used (except set E and CYC.6), Table 5 contains statistics of the deviation from the best known solution. For the random problems the comparison is against the lowest cost found by any of the algorithms, and on the combinatorial problems the comparison is against the best explicit solutions known. Table 7 contains the CPU times used by the algorithms and by the LOQO solver. Note that implementation details can be significant in the time consumption of the algorithms. We have attempted to make the comparison as fair as possible by using the same internal data structures and same compilation flags on all algorithms.

As pointed out in Section 3.2, the optimal fractional costs of the combinatorial problems are known. Moreover, LOQO consistently found optimal solution vectors with uniform x_j^* values on all problems of the CYC and CLR sets, e.g., $x^* = (1/4, \ldots, 1/4)$. Therefore in order to save time, LOQO was not used on the largest problems (CYC.10 and CYC.11). Instead, the uniform optimal fractional solution was used as input to the algorithms.

Table 3
Computational results of the random problems

Problem	LP	NoLP	Thresh	SortLP	RR	T-Gr	Gr	Alt-Gr	NN	R-Gr
4.1	32.80	65	52	45	44	43	41	42	43	41
4.2	31.71	63	49	44	42	41	41	39	39	38
4.3	32.45	64	53	45	45	42	43	43	40	41
4.4	33.27	64	54	47	46	46	44	44	43	41
4.5	32.79	70	51	45	44	43	44	44	41	40
4.6	32.24	66	52	45	44	43	43	44	41	40
4.7	33.52	68	50	47	45	43	43	43	43	41
4.8	31.77	70	49	45	43	42	42	42	40	40
4.9	32.89	71	51	48	44	43	42	42	42	40
4.10	33.31	67	52	47	45	44	43	43	42	41
5.1	28.73	69	47	42	42	40	37	37	37	35
5.2	28.58	66	47	40	40	39	38	38	37	35
5.3	28.62	62	45	42	41	38	37	37	37	36
5.4	28.61	65	47	43	42	38	39	37	37	36
5.5	28.25	64	46	41	41	39	37	37	37	36
5.6	28.86	62	48	44	43	40	40	38	37	36
5.7	28.09	65	45	41	41	37	38	38	35	36
5.8	28.93	69	45	41	41	39	39	39	37	37
5.9	28.98	72	47	41	41	38	38	38	38	36
5.10	28.79	67	43	42	40	39	39	38	37	36
6.1	14.78	35	30	27	26	24	23	23	22	21
6.2	14.29	35	30	24	26	22	22	22	21	21
6.3	14.87	38	30	27	27	23	23	23	21	22
6.4	14.68	37	29	27	25	24	22	22	22	22
6.5	14.92	40	30	28	25	24	23	23	23	22

Table 3
Computational results of the random problem—continued

Problem	LP	NoLP	Thresh	SortLP	RR	T-Gr	Gr	Alt-Gr	NN	R-Gr
A.1	29.55	76	56	48	50	43	42	42	41	40
A.2	29.82	81	55	48	48	44	42	44	42	41
A.3	29.76	83	57	52	48	44	43	42	41	40
A.4	29.09	73	54	50	47	43	41	41	40	40
A.5	29.57	79	55	51	47	44	43	43	41	40
B.1	14.09	43	34	29	30	25	24	24	24	23
B.2	13.99	41	32	31	30	25	23	23	23	22
B.3	14.06	44	33	30	29	24	23	23	23	22
B.4	14.01	45	32	32	30	26	24	24	24	23
B.5	13.98	43	32	31	30	25	25	25	23	23
C.1	31.10	86	64	58	57	49	47	47	46	45
C.2	31.42	90	64	56	55	48	47	46	45	45
C.3	31.05	83	61	55	56	47	47	48	47	45
C.4	31.04	80	66	54	57	51	46	46	46	46
C.5	31.28	85	66	60	56	49	47	47	46	45
D.1	14.58	47	36	32	34	29	27	27	26	26
D.2	14.74	48	39	35	34	28	26	25	26	26
D.3	14.65	41	39	34	33	29	27	27	26	25
D.4	14.63	49	36	36	33	28	26	27	26	26
D.5	14.62	49	38	34	34	29	27	27	26	26
E.1	3.48	11	7	6	6	6	5	5	5	5
E.2	3.38	8	8	6	5	5	5	5	5	5
E.3	3.30	12	8	7	5	5	5	5	5	5
E.4	3.45	10	8	7	5	5	6	5	5	5
E.5	3.39	10	9	6	5	6	5	5	5	5

Table 3
Computational results of the random problem—continued

Problem	LP	NoLP	Thresh	SortLP	RR	T-Gr	Gr	Alt-Gr	NN	R-Gr
NRE.1	8.18	30	28	23	23	19	18	18	17	17
NRE.2	8.18	30	26	25	23	18	18	18	17	17
NRE.3	8.19	30	24	23	22	19	18	18	17	17
NRE.4	8.19	30	25	23	22	20	18	18	17	17
NRE.5	8.19	29	25	24	23	19	18	18	17	17
NRF.1	4.36	16	18	16	13	11	11	11	12	10
NRF.2	4.36	16	18	15	14	11	11	11	12	11
NRF.3	4.36	17	17	15	14	11	11	11	12	11
NRF.4	4.36	16	16	14	13	11	11	11	12	11
NRF.5	4.36	16	18	15	13	11	11	11	12	11

Table 4
Computational results of the combinatorial problems

Problem	LP	Best[1]	NoLP	Thresh	SortLP	RR	T-Gr	Gr	Alt-Gr	NN	R-Gr
CYC.6	48.0	60	76	76	85	69	60	60	60	62	64
CYC.7	112.0	144	192	182	217	171	144	144	144	161	160
CYC.8	256.0	344	428	428	503	406	352	352	352	387	385
CYC.9	576.0	780	1024	984	1174	961	816	816	816	901	907
CYC.10	1280.0	1792	2220	2220	2633	2193	1916	1916	1916	2060	2081
CYC.11	2816.0	4103	5120	4952	5893	4934	4280	4280	4268	4661	4710
CLR.10-4	21.00	25	35	35	42	32	32	32	32	29	28
CLR.11-4	16.50	23	35	37	40	34	29	30	30	28	27
CLR.12-4	16.50	26	37	35	46	33	31	31	28	29	27
CLR.13-4	14.30	26	35	35	46	38	32	32	32	29	31

[1]The currently best explicit solutions, from [18] (CYC problem) and [15] (CLR problem).

4. Discussion

4.1. The random problems

On the 60 random problems, the best solution was found 56 times by the R-Gr algorithm, 24 times by the NN algorithm, 12 times by the Alt-Gr and Gr algorithms, and 7 times by algorithm T-Gr. On average the R-Gr algorithm deviated by 0.24% from the best value, NN deviated by 2.85%, Alt-Gr deviated by 4.53%, Gr deviated by 5.06%, and T-Gr deviated by 8.34% (see Table 5). Algorithm RR performed worse, and found the best value only on 4 problems of set E. SortLP, Thresh and NoLP performed much worse, and found strictly larger solutions than the previous algorithms on all the problems.

It is clear from Table 3 that algorithm R-Gr performed extremely well on all the random problem sets. Algorithm NN did nearly as well on most problems, except set NRF. The Gr and Alt-Gr algorithms gave reasonable results in general but were slightly inferior to the previous algorithms. The other algorithms did significantly worse.

Note that on all the problems in set E, the R-Gr, NN and Alt-Gr algorithms found the best cost. The found cost, 5, is known to be optimal for all the problems in this set [3].

As these problems were generated randomly, one could assume that the results obtained would fit the predictions of [29]. Denote the optimal cost of a random SCP with m rows by C_m. Assuming that the matrix entries are chosen to be $a_{ij} = 1$ independently with probability ϱ, and that the number of columns $n \gg \ln m/(\ln 1/(1-\varrho))$, then Theorem 3.1 of [29] says that

$$\lim_{m\to\infty} \frac{C_m}{\ln m} = \frac{1}{\ln 1/(1-\varrho)} \quad \text{a.e.}$$

Somewhat surprisingly, the costs found by our algorithms are significantly less than predicted (see Table 6). For densities $\varrho = 0.02, 0.05, 0.1, 0.2$ the found costs are roughly 6.8, 4.8, 3.5, 2.6 times *smaller* than the prediction, respectively. Note that the costs of the unknown optimal solutions are no larger than the best costs our algorithms found, so for the optimal costs the distance from the prediction can only be larger than shown. However, for $\varrho = 0.02$ and 0.05 the found costs are almost proportional to $\ln m$, which agrees with the predicted logarithmic dependence on m (see the $C_{\text{avg}}/\ln m$ data in Table 6).

The most likely cause for the discrepancy between the predictions and the empirical data is the fact that the matrix entries were *not* chosen with full independence. For instance, all-zero columns are not allowed; but when $200 \le m \le 400$ and the density is 2%, the expected number of ones per column is between 4 and 8, so discarding all-zero columns causes significant dependency. The effect of this dependency decreases when ϱ increases, which fits the shown behavior (for larger values of ϱ the discrepancy is smaller). Another possible cause for this discrepancy may be that m is not sufficiently large for the asymptotic behavior to be visible.

4.2. The combinatorial problems

On the CYC set of problems, the Alt-Gr, Gr and T-Gr algorithms found the best solutions, with equal costs, on the 5 smaller problems, but only Alt-Gr found the best solution for the largest problem, CYC.11 (see Table 4). Their average deviations from the best explicit solution are 2.98%, 3.03% and 3.03% respectively (see Table 5). Note that on problem CYC.6 our best solution (60) is known to be optimal [18], and on CYC.7 our best solution (144) matches the currently best explicit solution [7]. Algorithms NN and R-Gr did slightly worse, and never found the best solution. Algorithms RR, SortLP, Thresh and NoLP performed much worse on all CYC problems.

On the CLR set of problems, the R-Gr algorithm found the best solution four times (average deviation 13.12%) and the NN algorithm found the best solution once (average deviation 15.20%). Algorithms Alt-Gr, Gr, T-Gr were slightly worse, and as before, RR, SortLP, Thresh and NoLP performed much worse.

Recall that the optimal fractional solutions used for the problems in both the CYC and CLR sets were uniform (e.g., $x_j^* = 1/4$ for all j on the CYC problems). This holds both for manually created solutions, and for solutions found by the interior-point solver, LOQO. Such a uniform fractional solution causes SortLP and Thresh to find huge solutions (in fact, the Thresh algorithm picks all n variables). These solutions are much improved by the redundancy-elimination step, but not by enough to compete with the better algorithms.

Table 5
Average and maximum deviation percentages of the algorithms' solution costs from the best found values (problem sets 4–NRF) or from the best explicit solution (CYC and CLR), [100 × (solution_val − best)/best]

Set		NoLP	Thresh	SortLP	RR	T-Gr	Gr	Alt-Gr	NN	R-Gr
4–NRF	Avg	80.06%	41.24%	25.80%	19.91%	8.34%	5.06%	4.53%	2.85%	0.24%
	Max	140.00%	80.00%	60.00%	36.36%	20.00%	20.00%	10.00%	20.00%	4.76%
CYC	Avg	27.40%	24.70%	46.61%	19.60%	3.03%	3.03%	2.98%	11.95%	12.82%
	Max	33.33%	26.67%	50.69%	23.21%	6.92%	6.92%	6.92%	15.51%	16.28%
CLR	Avg	42.27%	42.53%	73.94%	37.23%	24.10%	25.19%	22.30%	15.20%	13.12%
	Max	52.17%	60.87%	76.92%	47.83%	28.00%	30.43%	30.43%	21.74%	19.23%

Table 6
Comparison between the predictions of [29] and the empirical data

Set	ϱ	m	C_{pred} [1]	C_{avg} [2]	$C_{\text{avg}}/\ln m$
4	0.02	200	261.86	40.2	7.58
5	0.02	200	261.86	35.8	6.75
A	0.02	300	282.15	40.2	7.04
C	0.02	400	296.51	45.2	7.54
6	0.05	200	103.16	21.4	4.03
B	0.05	300	111.15	22.6	3.96
D	0.05	400	116.81	25.6	4.27
NRE	0.10	500	58.98	17.0	2.73
NRF	0.20	500	27.85	10.8	1.73

[1]The predicted value, $C_{\text{pred}} = \ln m/(\ln 1/(1-\varrho))$.
[2]The average of the costs of the best solutions found on the problems of the set.

We have also tested the algorithms on the two smallest CYC problems (CYC.6 and CYC.7) using a non-uniform fractional solution generated by a revised-simplex solver. This improved the solutions found by the low quality algorithms SortLP and Thresh, however the solutions found by the better T-Gr became worse, so we did not pursue this direction further.

5. Conclusions

The overall best algorithm we tested is the randomized greedy algorithm R-Gr, and the neural network algorithm NN is a close second. However their good performance comes at a price of relatively high execution time (see Table 7). This is mostly due to the fact that both algorithms run their basic computation, which is roughly equivalent in time complexity to the greedy algorithm Gr, for many initial states (we used $N = 100$ such repetitions). When the execution time is crucial, the Alt-Gr algorithm is a viable option, giving comparable results at a fraction of the time, or alternatively one can decrease the value of N, say to $N = 1$.

Note also that on the CYC set both R-Gr and NN performed quite poorly, and the NN algorithm was slightly inferior to the greedy algorithms on all problems of the NRF set. We conclude that the best choice of algorithm depends on the particular class of input problems it is intended to solve, and that none of the algorithms we tested is universally best.

Another conclusion we can draw from our tests is that the effort invested in solving the fractional linear program does not seem to pay off. The execution time of the LP solver is more than double that of our slowest algorithms on the larger problems (Table 7). Moreover, the algorithms using the fractional solution are inferior to others; the Thresh and SortLP algorithms found very costly solutions on all the input problems, RR performed well only on set E despite its high CPU cost, and only T-Gr performed moderately well (but not as well as R-Gr, NN, or Alt-Gr).

Finally, we remark that the worst-case approximation ratios did not predict the algorithms' behavior

Table 7
CPU time in seconds for some of the algorithms. The values for problems 4–NRF represent the average CPU time used on the problems in each set. The CPU time consumed by the RR algorithm, which uses the fractional solution, is actually the sum of the values in the LP and RR columns

Problem	LP	RR	Gr	Alt-Gr	NN	R-Gr
4	2.5	1.0	0.0	0.0	6.6	1.6
5	4.2	2.4	0.0	0.0	12.0	3.0
6	4.0	1.6	0.0	0.0	4.4	2.0
A	13.4	4.2	1.0	1.0	21.0	6.0
B	21.0	7.0	4.0	4.0	17.2	8.2
C	31.0	6.4	2.6	2.6	30.8	10.0
D	45.8	11.6	7.0	7.0	26.8	14.2
E	0.0	0.6	0.0	0.0	1.0	0.0
NRE	135.6	37.0	23.0	23.0	48.8	38.0
NRF	274.8	87.8	48.0	46.4	90.2	71.2
CYC.6	0.0	0.0	0.0	0.0	1.0	0.0
CYC.7	5.0	1.0	0.0	0.0	7.0	2.0
CYC.8	55.0	6.0	0.0	0.0	39.0	12.0
CYC.9	693.0	13.0	2.0	2.0	192.0	58.0
CYC.10	-	137.0	7.0	7.0	941.0	288.0
CYC.11	-	852.0	26.0	28.0	4907.0	1455.0
CLR.10-4	3.0	3.0	1.0	1.0	3.0	2.0
CLR.11-4	13.0	7.0	4.0	4.0	9.0	7.0
CLR.12-4	49.0	20.0	15.0	13.0	25.0	21.0
CLR.13-4	183.0	52.0	42.0	38.0	68.0	58.0

well. Saying that an algorithm's solution is no more than $\ln m$ times the optimal cost (the bound for the greedy algorithm, Gr) is meaningless on all the inputs we tested. Even saying that a solution is no more than 4 times the optimal *fractional* cost, (the bound for the Thresh, SortLP and T-Gr algorithms on the CYC problem set) turns out to mean that no more than n variables are picked, again a triviality. Moreover, our tests show that the approximation ratio did not predict the algorithms' *relative* quality either. When the number k of ones per row is small (e.g., $k = 4$), algorithms with an approximation ratio of k, such as NoLP, Thresh, SortLP or T-Gr, would seem to have an advantage over the logarithmic ratio of the Gr algorithm. On all the inputs we tested we consistently found that the *opposite* is true.

Acknowledgements

An early version of some of the algorithms was programmed by Leo Reyzin. We would like to thank Uri Feige and David Peleg for some stimulating discussions, and Manny Knill for suggesting the combinatorial problems.

References

[1] Arora, S., Lund, C., Motwani, R., Sudan, M., and Szegedy, M. (1992), "Proof verification and hardness of approximating problems", in: *Proceedings 33rd IEEE Symp. Found. of Comp. Science* 14-23.

[2] Balas, E., and Ho, A. (1980), "Set covering algorithms using cutting planes, heuristics, and subgradient optimization: a computational study", *Math. Programming* 12, 37-60.

[3] Beasley, J.E. (1987), "An algorithm for set covering problem", *Euro. J. Operational Research* 31, 85-93.

[4] Beasley, J.E. (1990), "A lagrangian heuristic for set covering problems", *Naval Research Logistics* 37, 151-164.

[5] Beasley, J.E. (1990), "OR-library: Distributing test problems by electronic mail", *J. Oper. Res. Soc.* 41/11, 1069-1072. See also in WWW site http://mscmga.ms.ic.ac.uk/info.html.

[6] Beasley, J.E., and Chu, P.C. (1996), "A genetic algorithm for the set covering problem", *Euro. J. Operational Research* 94, 392-404.

[7] Brass, P., Harborth, H., and Nienborg, H. (1995), "On the maximum number of edges in a C_4-free subgraph of Q_n", *J. Graph Theory* 19/1, 17-23.

[8] Chvátal, V. (1979), "A greedy heuristic for the set-covering problem", *Math. of Oper. Res.* 4/3, 233-235.

[9] Croall, I.F., and Mason, J.P., eds. (1991), *Industrial Applications of Neural Networks*, Springer-Verlag, Berlin.

[10] Erdős, P. (1963), "On a combinatorial problem I", *Nordisk Mat. Tidskrift* 11, 5-10.

[11] Erdős, P. (1990), "On some of my favorite problems in graph theory and block designs", *Le Matematische* 45, 61–74.
[12] Feige, U. (1993), Personal communication.
[13] Garey, M.R., and Johnson, D.S. (1979), *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco.
[14] Gavril, F. (1974), appears in [13], p. 134.
[15] Goldberg, M.K., and Russell, H.C. (1995), "Toward computing $m(4)$", *Ars Combinatoria* 3/9, 139–148.
[16] Grossman, T. (1994), "A neural network algorithm for the minimum cover problem", unpublished manuscript.
[17] Grossman, T., and Wool, A. (1994), "Computational experience with approximation algorithms for the set covering problem", Technical Report CS94-25, The Weizmann Institute of Science, Rehovot, Israel.
[18] Harborth, H., and Nienborg, H. (1995), "Maximum number of edges in a six-cube without four-cycles" (to appear in *Bull. Inst. Combin. Appl.*).
[19] Hochbaum, D.S. (1982), "Approximation algorithms for the set covering and vertex cover problems", *SIAM J. Computing* 11/3, 555–556.
[20] Jefries, C. (1991), *Code Recognition and Set Selection with Neural Networks*, Birkhauser, Boston.
[21] Johnson, D.S. (1974), "Approximation algorithms for combinatorial problems", *J. Computer System Sci.* 9, 256–278.
[22] Lovász, L. (1975), "On the ratio of optimal integral and fractional covers", *Disc. Math.* 13, 383–390.
[23] Papadimitriou, C.H., and Yannakakis, M. (1991), "Optimization, approximation, and complexity classes", *J. Computer System Sci.* 43, 425–440.
[24] Peleg, D., Schechtman, G., and Wool, A. (1993), "Approximating bounded 0-1 integer linear programs", in: *Proceedings 2nd Israel Symp. Theory of Computing Sys.*, Netanya, Israel, 69–77.
[25] Peleg, D., Schechtman, G., and Wool, A. (1996), "Randomized approximation of bounded multicovering problems", *Algorithmica* (to appear).
[26] Peng, Y., and Reggia, J.A. (1989), "A connectionist model for diagnostic problem solving", *IEEE Transactions on Systems, Man, and Cybernetics* 19, 285.
[27] Sen, S. (1993), "Minimal cost set covering using probabilistic methods", in: *Proceedings ACM Symp. Applied Computing*, Indianapolis, 157–164.
[28] Vanderbei, R.J. (1992), "LOQO user's manual", *Program in Statistics & Operations Research*, Princeton University, NJ.
[29] Vercellis, C. (1984), "A probabilistic analysis of the set covering problem", *Annals of Oper. Res.* 1, 255–271.
[30] Wedelin, D. (1994), "An algorithm for 0-1 programming with application to airline crew scheduling", in: *Euro. Symp. Algorithms.*