Journal of the Operational Research Society (1995) 46, 1260–1268

A New Efficient Heuristic For the Minimum Set Covering Problem

MOHAMED AFIF[1], MHAND HIFI[2], VANGELIS TH. PASCHOS[1] and VASSILIS ZISSIMOPOULOS[3]

[1]LAMSADE, Université Paris-Dauphine, France; [2]CERMSEM, Université Paris I, France and [3]LRI, Université d'Orsay, 91405 Orsay, France

We solve approximately the minimum set covering problem by developing a new heuristic, which is essentially based on the flow algorithm originally developed by Ford and Fulkerson. We perform a comparative study of the performances (concerning solution qualities and execution times) of the flow algorithm as well as of the natural greedy heuristic for set covering originally studied by Johnson and Lovász.

Key words: combinatorial analysis, computational analysis, heuristics, integer programming, set covering

INTRODUCTION

Given a collection $\mathcal{S}$ ($|\mathcal{S}| = n$) of subsets of a finite set C ($|C| = m$), a set covering is a sub-collection $\mathcal{S}' \subseteq \mathcal{S}$ such that every element of C belongs to at least one member of $\mathcal{S}'$, and the minimum set covering problem SC is to find a set cover of minimum size. Recently, Lund and Yannakakis[1] proved a strong negative result for SC's approximability: SC cannot be approximated with the ratio $c \log m$ for any $c < 1/4$ unless $\text{NP} \subseteq \text{DTIME}[n^{\text{poly log } n}]$ (a conjecture weaker than P = NP but highly improbable).

In this context, and since SC is, from both the theoretical and practical points of view, a very interesting problem (for instance, problems on operating systems, databases, computer aided systems and satellite photograph systems can be represented and treated in terms of SC), we think that it is crucial (and challenging) to devise experimental algorithmic methods that find 'reasonable' solutions for instances of SC.

We present in this paper a new polynomial time approximation algorithm relying on the transformation of an instance of SC into an instance of a particular flow problem and, on using an adaptation of the flow-algorithm of Ford and Fulkerson[2,3] to solve this flow problem. We also implement the naturally greedy SC-algorithm (studied theoretically by Johnson[4] and Lovász[5]). For these two methods, we perform numerical experiments and comparative studies concerning solution qualities and execution times.

A NEW APPROXIMATION ALGORITHM FOR (UNWEIGHTED) SET COVERING

The transformation of SC into a flow problem relies on Definition 1.

Definition 1

The characteristic graph $B = (S, C, E)$ of an instance of SC is a bipartite graph with vertex set S corresponding to the family $\mathcal{S}$, vertex set C corresponding to the ground set C, and $E = \{s_i c_j : c_j \in S_i\}$. Given a bipartite $B = (S, C, E)$ representing an instance of SC, a layered network $N = (X, A, c, b)$ can be constructed with vertex set $X = S \cup C \cup s_0 \cup c_0$, where c_0 is the source of the network, s_0 its sink and $A = E \cup E_S \cup E_C$ is the arc set of the network.

Correspondence: V. Th. Paschos, LAMSADE, Université Paris-Dauphine, Place du Maréchal De Lattre de Tassigny, 75775 Paris Cedex 16, France

Here, $E_S = \{s_i s_0 : \forall s_j \in S\}$, $E_C = \{c_0 c_i : \forall c_i \in C\}$, and the arcs are oriented from c_0 to s_0. The vectors $c = (c_a)_{a \in A}$, $b = (b_a)_{a \in A}$, represent the capacities and the lower bounds, respectively, on the arcs of N and are defined as follows:

$$c_a = \begin{cases} 1 & a \in E_S \\ \dfrac{1}{|\Gamma^-(s_j)|} & a = c_i s_j, \quad i \neq 0 \vee j \neq 0 \\ \displaystyle\sum_{a' \in Ec_i} c_{a'} & a = c_0 c_i, \quad i = 1, \ldots, m \quad E_{c_i} = \{c_i s_j : s_j \in \Gamma^+(c_i)\}, \end{cases}$$

$$b_a = \begin{cases} 0 & a \in E_S \cup E \\ \min \{c_{a'} : a' \in \{c_i s_j : s_j \in \Gamma^+(c_i)\}\} & a = c_0 c_i, \quad i = 1, \ldots, m, \end{cases}$$

where $\Gamma(x)$ is the set of neighbours of x, $|\Gamma^+(x)|$ and $|\Gamma^-(x)|$ are the outer and inner degrees of vertex x.

Using the construction implied by Definition 1, SC reduces to a minimum flow problem Φ on N defined as follows (by $I^-(v_i)$ (respectively $I^+(v_i)$) we denote the set of incoming (respectively outgoing) arcs of vertex v_i):

$$\Phi = \begin{cases} \min \displaystyle\sum_{a \in E_S} \varphi_a & \\ \displaystyle\sum_{a \in I^-(v_i)} \varphi_a = \sum_{a \in I^+(v_i)} \varphi_a & v_i \in X \backslash \{s_0, c_0\} \\ b_a < \varphi_a \leq c_a, & a \in A \\ \varphi_a \in \{0, 1\}, & a \in E_S \\ \varphi_a \in \mathbb{Q}^+, & a \in E \cup E_C. \end{cases}$$

It is easy to see that SC can be polynomially reduced to Φ. In fact, given an instance I of SC, the construction of B takes $O(nm)$ steps; the construction of N also takes $O(|E|) = O(nm)$ steps, and the computation of the upper and lower capacities of the arcs of N can be performed in constant time. Consequently, the whole of the transformation of SC to Φ can be performed in $O(nm)$.

Example 1

Consider the following instance of SC: $\mathcal{S} = \{S_1, S_2, S_3\}$, $C = \{c_1, c_2, c_3, c_4\}$, $S_1 = \{c_1, c_2\}$, $S_2 = \{c_1, c_3, c_4\}$, $S_3 = \{c_3, c_4\}$. Figure 1(a) shows the characteristic bipartite graph B of the above instance, while Figure 1(b) represents the layered network N.

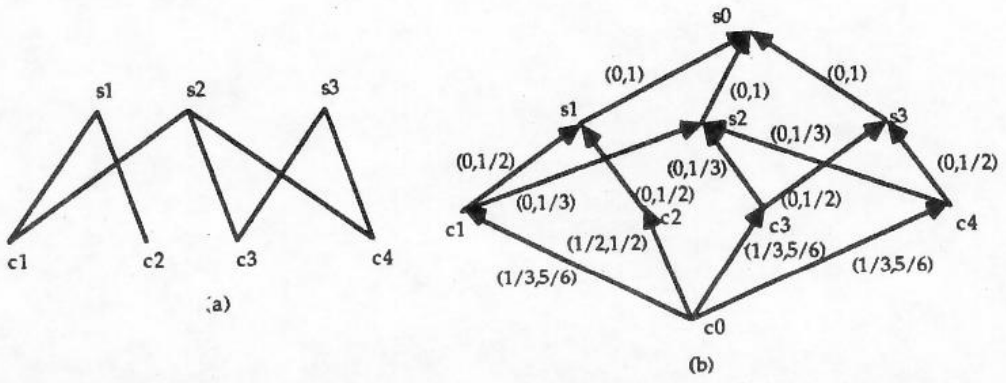

FIG. 1. (a) *Characteristic bipartite graph B of the set covering instance; (b) layered network N corresponding to B.*

Proposition 1 states the connection between the objective function values of the corresponding solutions for SC and Φ. In addition, in the proof of Proposition 1, the properties of solutions of Φ are stated.

Proposition 1

The (optimal) objective function values for the solutions of SC and Φ are equal.

Proof

Suppose that a feasible solution for Φ is given. Then, a feasible solution for SC can be obtained by choosing exactly those sets of $\mathcal{S}$ corresponding to the vertices s_i of N (Definition 1) for which $\varphi_{s_is_0} = 1$.

The fact that the so-obtained solution $\mathcal{S}'$ is feasible for SC is deduced from the following argument: suppose that there exists an element c_i such that $c_i \notin \bigcup_{s_j \in \mathcal{S}'} S_j$; then, in terms of Φ, this would mean that the flow from all the arcs in E out of c_i is equal to 0; and so, by flow preservation, the flow on the arc c_0c_i must also be equal to zero, violating the lower bound on such arcs.

Suppose now that a solution $\mathcal{S}'$ is given for SC. Then, a feasible solution for Φ on N can be obtained by taking the following flow values in $A(N)$:

- for the arcs $a = s_is_0$, $i = 1, \ldots, n$: $\varphi_a = 1_{\{S_i \in \mathcal{S}'\}}$;
- for the arcs $a = c_is_j$, $i = 1, \ldots, m$, $j = 1, \ldots, n$:

$$\varphi_a = \begin{cases} c_a & \text{if } S_j \in \mathcal{S}' \\ 0 & \text{otherwise;} \end{cases}$$

- for the arcs $a = c_0c_i$, $i = 1, \ldots, m$, we take: $\varphi_{c_0c_i} = \sum_{c_is_j \in E} \varphi_{c_is_j}$.

Obviously, for the arcs in $E_S \cup E$, the considered flow respects the capacities and the bounds on the arcs, and moreover, for the arcs in E_s, the integrity constraints. For the arcs c_0c_i, $i = 1, \ldots, m$, we have $0 < \varphi_{c_0c_i} = \sum_{c_is_j \in E} \varphi_{c_is_j} \leq \sum_{c_is_j \in E} c_{c_is_j} = c_{c_0c_i}$. On the other hand, for a vertex c_i of N, if $c_is_k = \text{argmin}_{c_is_l, s_l \in \Gamma^+(c_i)}\{c_{c_is_l}\}$, then $\varphi_{c_0c_i} = \sum_{c_is_j \in E} \varphi_{c_is_j} = \sum_{\{c_is_j : s_j \in \mathcal{S}'\}} \varphi_{c_is_j} \geq c_{c_is_j}$.

Concerning the flow preservation, it is easy to see that it holds for the S-vertices of N. By the way the flow values are chosen, if $S_i \in \mathcal{S}'$, then $\varphi_{s_is_0} = 1 = \sum_{c_js_i, c_js_i \in E} (1/\Gamma^-(s_i)) = \sum_{c_js_i, c_js_i \in E} \varphi_{c_js_i}$; on the other hand, if $S_i \notin \mathcal{S}'$, then $\varphi_{s_is_0} = 0$, the same holds for the flow values of arcs incoming in vertex s_i. For the C-vertices of N, the flow preservation is more evident, since $\varphi_{c_0c_i} = \sum_{c_is_i \in E} \varphi_{c_is_i}$. Consequently, given a feasible solution for SC, one can, in polynomial time, find a feasible solution for Φ.

Before discussing the relation between the optimal solutions of the two problems, let us first mention the following fact, which describes properties of solutions for the flow problem. For a feasible solution of Φ: (a) all C-vertices of N receive non-zero flow; (b) if an arc $a = s_is_0$ of E_S has a unit flow, then all arcs c_js_i, $c_j \in \Gamma^-(s_i)$, are saturated ($\varphi_{c_js_i} C_{c_js_i}$).

(a) In order to satisfy the constraints of Φ, the flow $\varphi_{c_0c_j}$ on an arc c_0c_j, $j = 1, \ldots, m$ has to respect $\varphi_{c_0c_j} > 0$.

(b) Clearly, the flow sum and the capacity over $\Gamma^-(s_i)$ equal 1. Let us suppose that an arc a' outgoing from a vertex in $\Gamma^{-1}(s_i)$ has $\varphi_{a'} < c_{a'}$; then, since $\sum_{c_js_i, c_j \in \Gamma^-(s_i)} \varphi_{c_js_i} = \sum_{c_js_i, c_j \in \Gamma^-(s_i)} c_{c_js_i}$ the quantity $c_{a'} - \varphi_{a'}$ would have to be sent via the other arcs, outgoing from vertices of $\Gamma^-(s_i)$; hence, there would be at least one arc $a'' = c_js_i$, $c_j \in \Gamma^-(s_i)$ such that $\varphi_{a''} > c_{a''}$, these violating the upper capacity constraint on a''.

We are now prepared to prove the following assertion.

Assertion

A minimum solution for Φ corresponds to a minimum solution of SC having the same value and vice versa.

In fact, whenever a minimum flow is given, the solution for SC, obtained as previously described, has cardinality equal to the minimum flow value; moreover, it is minimum because the hypothesis on the existence of a smaller solution Λ for SC would lead immediately, by using part (a) of the above fact as well as the described construction of a feasible flow from a SC solution, to a smaller solution for Φ of cardinality $|\Lambda|$.

For the converse, let us suppose that the minimum flow for Φ (let us denote by φ this solution and by $v(\varphi)$ its value) is smaller than the minimum solution of SC. Using the above-mentioned fact, φ sends flow throughout all the C-vertices of the network and, moreover, if a vertex s_i receives a unit flow, then all arcs $c_j s_i$, $c_j \in \Gamma^-(s_i)$, are saturated. The way a solution for SC is obtained from φ induces a new feasible solution for SC of cardinality equal to $v(\varphi)$. This completes the proof of the proposition.

THE MINIMUM FLOW ALGORITHM, ITS COMPLETENESS AND ITS COMPLEXITY

The minimum flow algorithm used as a heuristic for SC is Algorithm 1.

```
begin
    S* ← ∅;
    repeat
        CONSTRUCT(N);
        φ ← {c_a: ∀a ∈ A}
        repeat
            stop ← false; δ(c_0) ← ∞; X' ← {c_0}; A' ← ∅
            while (s_0 ∉ X') ∧ ¬stop do
                if ∃a = xy ∈ A, x ∈ X' ∧ y ∈ X\X' ∧ φ_a > b_a then
                  X' ← X' ∪ {y}; A'(y) ← a;
                  δ(y) ← min {δ(x), φ_a − b_a}; A' ← A' ∪ {a}
                  else stop ← true
                fi
            od
            if s_0 ∈ X' then δ ← δ(s_0) else δ ← 0 fi
            if δ ≠ 0 then x ← s_0;
              while x ≠ c_0 do a ← A'(x); φ_a ← φ_a − δ; x ← i(a) od
            fi
        until δ = 0
        S* ← S* ∪ {s_i: s_i s_0 = argmax {φ_a: a ∈ E_S}};
        Ā ← {s_i s_0} ∪ {a: t(a) = s_i} ∪ {c_0 c_j: c_j ∈ Γ^-(s_i)};
        C ← C\Γ^-(s_i); S ← S\{s_i}; A ← A\Ā
    until C = ∅
end.
```

ALGORITHM 1. *Minimum flow algorithm. By t(a), we denote the terminal extremity of arc a, while by i(a) we denote its initial extremity.*

Procedure CONSTRUCT comprises the application of Definition 1 either on the original SC-instance or its surviving instance after the deletion of an S-vertex, and its C-neighbours on the 14th and 16th lines of the outer **repeat** loop of Algorithm 1.

The inner **repeat** loop is the usual minimum flow algorithm[3] and computes a flow reduction δ and a path along which the flow will be reduced by δ ('reducing path'). We start with a feasible solution saturating all the arcs of N (it is easy to see that, from the way N is constructed, this solution is feasible), and we try to reduce the flow from c_0 to s_0. The reduction step δ along a path $[c_0 c_j s_i s_0]$ is defined recursively as follows: $\delta(c_0) = \infty$, for each arc $a = xy \in A$, $\delta(y) = \min\{\delta(x), \varphi_a - b_a\}$ and, finally, $\delta = \delta(s_0)$.

Once a minimum flow has been found, if the flows on the arcs of E_S are not integral, and given that flows are rational numbers, we choose the S-vertex through which the maximum (closest to 1) flow is sent. We then delete this vertex and its neighbours in C and we re-calculate the upper and lower capacities in the survived network. Thus, at the end of each iteration of the **repeat** loop of Algorithm 1, a number of C-vertices of N are deleted and, consequently, after a number of steps, all C-vertices will be removed: this fact meaning that they are covered by the S-neighbours that have entailed their deletion.

Procedure CONSTRUCT takes a time linear to the number of edges of B. On the other hand, Algorithm 1 treats every path of N only once because of the way the quantity δ is defined on the vertices of N and also to the way φ_a and b_a are computed for $a \in E$.

In fact, let us suppose that a path $A' = [c_0c_i, c_is_j, s_js_0]$, $(i, j) \neq (0, 0)$ has been selected for the flow reduction (inner **repeat** loop). Let us suppose also that the flows on the arcs and the quantities δ on the vertices of A' are $\varphi_{c_0c_j}$, $\varphi_{c_is_j}$, $\varphi_{s_js_0}$, $\delta(c_0)$, $\delta(c_i)$, $\delta(s_i)$ and $\delta(s_0)$, respectively. Then, the inner **repeat** loop determines $\delta = \min\{\delta(c_0), \delta(c_i), \delta(s_j), \delta(s_0)\}$. Let us note here that $\delta \neq \delta(c_0)$ because $\delta(c_0) = \infty$. We then have to examine the following cases concerning δ.

(i) $\delta = \delta(c_i)$; in this case, $\delta(c_i) = \varphi_{c_0c_i} - b_{c_0c_i}$, and the reduction of flow along the path considered gives $\varphi_{c_0c_i} = \varphi_{c_0c_i} - \delta(c_i) = b_{c_0c_i}$, $\varphi_{c_is_j} = \varphi_{c_is_j} - \delta(c_i) = \varphi_{c_is_j} - \varphi_{c_0c_i} + b_{c_0c_i}$, $\varphi_{s_js_0} = \varphi_{s_js_0} - \delta(c_i) = \varphi_{s_js_0} - \varphi_{c_0c_i} + b_{c_0c_i}$. It is clear that the bound $b_{c_0c_i}$ is attained for the arc c_0c_i and thus, Algorithm 1 cannot re-use this arc without violating the constraint on the capacity. Moreover, the neutralization of this arc also neutralizes a number of paths of N.

(ii) $\delta = \delta(s_j)$; then, with a similar reasoning, $\varphi_{c_is_j} = b_{c_is_j}$, and the lower bound is attained for the flow on arc c_is_j, this arc being neutralized, and consequently, a (maximal) path of N is also neutralized.

(iii) $\delta = \delta(s_0)$; the same reasoning as in case (i) implies that $\varphi_{s_js_0} = b_{s_js_0}$, and thus the neutralization of some paths containing arc s_js_0.

We conclude that each path of N is treated at most once, the number of paths being equal to the number of edges of B. So, the complexity of Algorithm 1 is of $O(mn)$.

Example 2

Let us consider Example 1. In Figure 2(a), the flow values on the arcs of N computed by the algorithm are shown. As one can see, Algorithm 1 fails to send integral flow values through the arcs of N; consequently, it selects vertex s_2, arc s_2s_0 realizing the maximum flow value on arcs of E_S. The 'reducing path' is the path $c_0c_3s_2s_0$. During the second (and last) iteration, the algorithm includes vertex s_1 in the solution and, thus, finally $S^* = \{s_1, s_2\}$. Figure 2(b) shows the network of the surviving instance of SC constructed by the Procedure

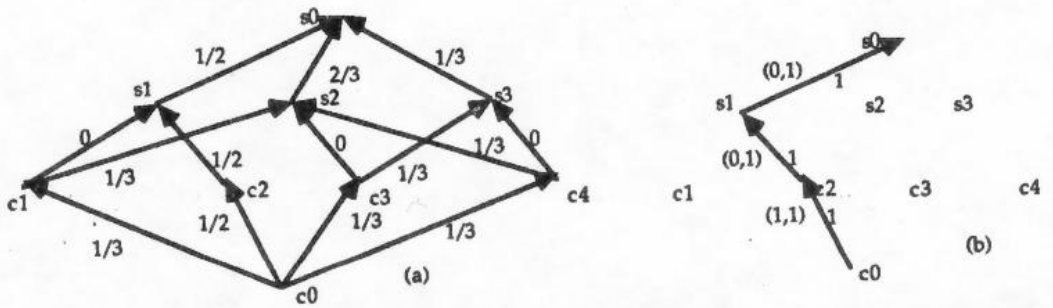

FIG. 2. (a) *The network N of Figure 1(b) with the flow values computed by Algorithm 1; (b) the surviving network (constructed by the Procedure CONSTRUCT), together with the upper and lower capacities and the flow passing through its arcs.*

CONSTRUCT, the upper and lower capacities on its arcs (the couple between the parentheses), and the flow passing through the arcs.

In fact, in the case where non-integral flow values are allowed the minimum flow problem is polynomial. The NP-completeness of Φ (that is, the discussion preceding Proposition 1 together with the proposition, constitute a proof for the NP-completeness of Φ) is due exactly to the 0–1 requirements of the E_S-arcs.

SOME REMARKS ON THE APPROXIMATION PERFORMANCE OF ALGORITHM 1

One can easily remark that, if every time we choose to reduce the flow along the path $[c_0c_js_is_0]$ such that the arc c_0c_j has the smaller lower capacity and the arc c_js_i the smaller upper one, then we produce Johnson's algorithm[4]. It is easy to see that such a path exists; moreover, the way the capacities are computed implies that choosing such a path introduces, in the solution of SC, the maximum cardinality set. If, on the other hand, we let Algorithm 1 choose arbitrarily the 'reducing path', then the solution for SC given by Algorithm 1 is not always similar to the one provided by Procedure 1.

```
begin
    S' ← ∅;
    repeat
        choose a set S_i of maximum cardinality;
        S' ← S' ∪ S_i;
        C ← C\S_i
    until C = ∅
end;
```

PROCEDURE 1. *Johnson's Heuristic. Its complexity is O(nm).*

The experimental results we present in the section entitled 'Discussion on the performance of the three heuristics' show that the two solutions are usually different. In Reference 4, Johnson presents a counter-example, proving that the expression for the approximation ratio of Procedure 1 is tight (the ratio of Procedure 1 is bounded above by $1 + \max_{s_i \in \mathcal{S}} \{|S_i|\}$. It is easy to see that Algorithm 1, when operating on this example, may perform a good sequence of set choices, leading to the optimal solution.

NUMERICAL RESULTS

We perform here two types of experiments. The first type consists of measuring the approximation ratio (this is a usual quality measure for an approximation algorithm[2]) of Algorithm 1, and the second of comparing Algorithm 1 with the greedy SC-algorithm (Procedure 1) from the points of view of execution time and solution quality efficiency.

We have produced three main groups of instances, the small, medium and large ones.

Concerning the small-size instances, we have produced ten groups of 20 instances per group (in total 200 instances), the size of each group (size, n, of the set family) being in {10, 20, . . ., 100}. The density of the instances, defined as the maximum cardinality of a set in the instance, was less than or equal to 7 for all the problem instances produced. The number of the sets, in which a given element of the ground set is contained, varied between 5 and 10 (again for all the problem instances produced). The cardinalities of the ground set varied from 5 to 45, for all instances.

For the exact solutions of the tested instances, we have developed a branch-and-bound method for SC; the basic methodology is given in Reference 2.

For every group of instance sizes, we have evaluated two kinds of approximation ratio, the worst case and the average case. For both methods the worst ratio equals 1.5. On the other hand, the corresponding average case ratios are 1.17 for Johnson's algorithm, and 1.09 for the flow algorithm. As can be seen from Figure 3—where the worst and the average ratios for the two heuristics per instance size are shown—by increasing instance sizes we have a good approximation to the behaviour of both methods tested. Moreover, Algorithm 1 always dominates Procedure 1 on both the worst and the average behaviours. Note also that the flow algorithm has optimally solved 51% of the tested instances while, for Johnson's algorithm, the percentage of the optimally solved instances was 21.5%. Consequently, a first conclusion that can be drawn is that Algorithm 1 seems to be very efficient and, therefore, a further theoretical study of it is quite challenging.

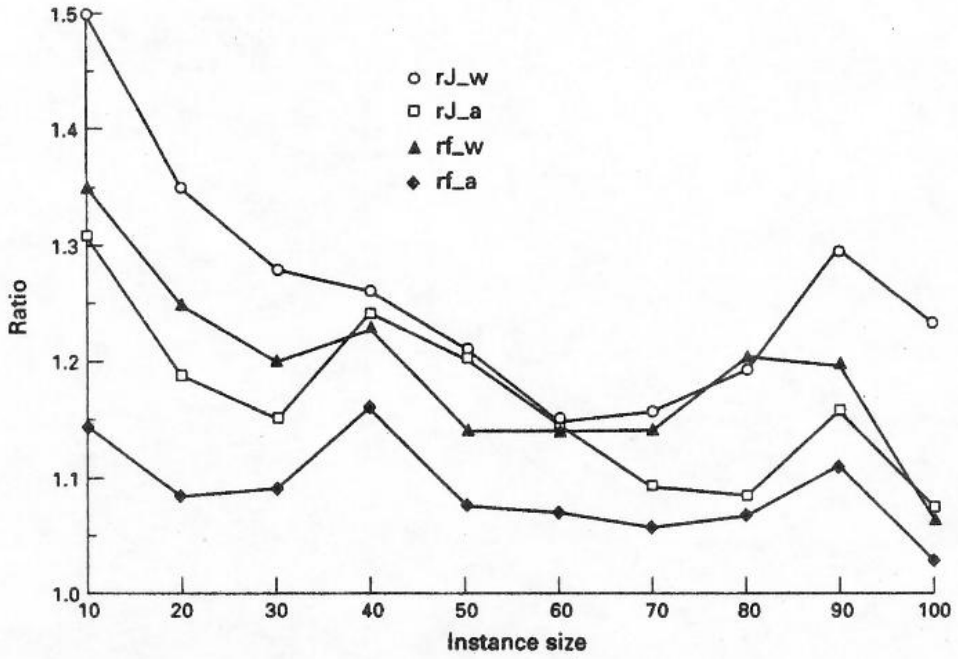

FIG. 3. *Average and worst approximation ratios as functions of instance sizes. By rJ_w, rJ_a, rf_w, rf_a, we denote the worst and the average case ratios of Procedure 1 and Algorithm 1, respectively.*

In Table 1, a study of the mean execution times (in milliseconds) for the two heuristics for each of the tested problem sizes is shown, while in Table 2 the average values of the main performance parameters of both heuristics are summarized (all the experiments have been performed on an IBM PS/2 286; the heuristics are implemented in PASCAL). Finally, in Figure 4, the execution times as functions of instance sizes are shown). The two algorithms

TABLE 1. *Mean execution times in milliseconds for Procedure 1 and Algorithm 1 when solving small-size instances*

Instance size	Procedure 1 average time	Algorithm 1 average time
10	44	43.1
20	65.6	57.7
30	105.8	98.9
40	123.4	124.3
50	132.6	137.4
60	175	177.8
70	212.3	209.2
80	252.2	243.7
90	298.4	310.1
99	348.6	372.3

TABLE 2. *Percentage of optimal solutions, average ratios and average execution times for Procedure 1 and Algorithm 1 on small-size instances*

	Procedure 1	Algorithm 1
% Optimal solutions	21.5	51
Average ratio	1.17	1.09
Average time	175.79	177.45

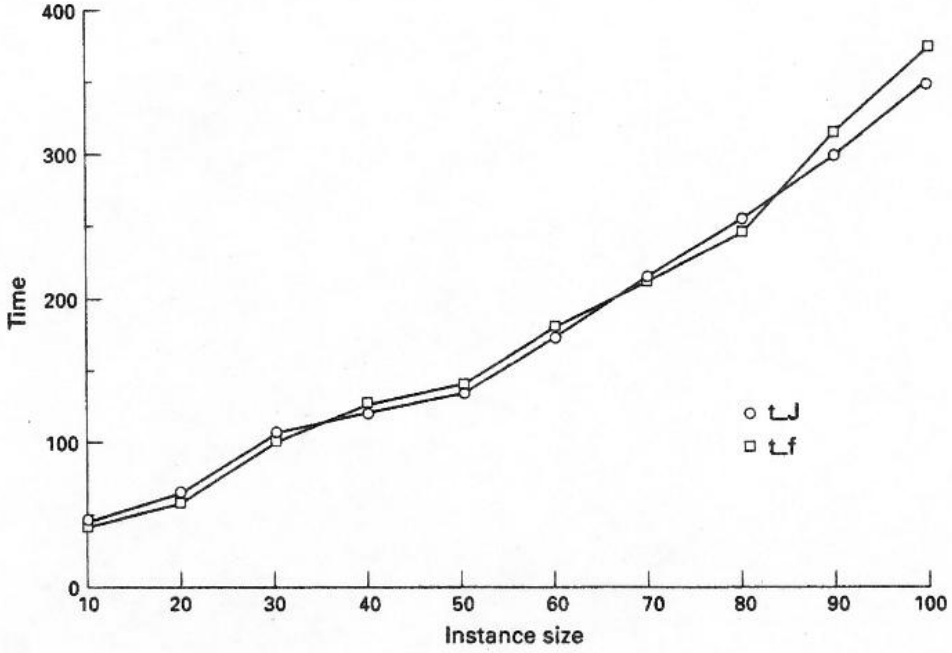

FIG. 4. *Execution times as functions of instance sizes. By t_J, t_f, we denote the execution times of Procedure 1 and Algorithm 1, respectively.*

have comparable mean execution times (Algorithm 1 being 0.9% slower than Procedure 1), but Algorithm 1 has substantially better approximation performance.

Next, in order to study the performances of the two algorithms on medium-size instances, we have randomly generated ten large instances including 800 subsets drawn from a ground set of 300 elements. The density of these instances is the same as the density of the small ones, and the number of the sets in which a given element of the ground set is contained, varies between 2 and 14. The cardinalities of the solutions, as well as the execution times, in ms, for the two sequential heuristics (second members of the pairs in the second and third columns), are shown in Table 3.

TABLE 3. *(Solution cardinalities, execution times) for Procedure 1 and Algorithm 1 on ten medium-size (n = 800, m = 300) instances*

Instance	Procedure 1	Algorithm 1
1	(35,2909)	(33,3368)
2	(25,2959)	(25,2828)
3	(28,5823)	(26,5556)
4	(16,3631)	(14,3914)
5	(26,2938)	(26,2769)
6	(39,2122)	(37,2080)
7	(30,1916)	(29,1513)
8	(19,1929)	(16,2144)
9	(26,4782)	(26,4836)
10	(26,3927)	(24,3798)

Algorithm 1 always dominates Procedure 1 from the point of view of the solutions' cardinality; moreover, on average, it is 0.4% faster than Procedure 1 (the average execution time of Procedure 1 is 3.293 s, while for Algorithm 1 the average execution time is 3.28 s).

Finally, we have generated ten large instances, including 2000 sets drawn from a ground set of 800 elements, of the same density as previously. As one can see in Table 4, where the solution cardinalities and the execution times for these instances are given, Algorithm 1 (being on average 0.465% faster) always dominates Procedure 1 from the points of view of both the solution quality and the execution time. For Procedure 1, the average execution time is 22.29 s, while for Algorithm 1 this average time is 22.18 s.

TABLE 4. *(Solution cardinalities, execution times) for Procedure 1 and Algorithm 1 on ten large ($n = 2000$, $m = 800$) instances*

Instance	Procedure 1	Algorithm 1
1	(658,19490)	(649,22566)
2	(661,20234)	(647,19338)
3	(656,23790)	(653,25645)
4	(648,13170)	(634,10384)
5	(604,32972)	(583,33344)
6	(585,39067)	(491,37275)
7	(688,20008)	(609,18857)
8	(633,14290)	(632,14007)
9	(592,13005)	(556,14455)
10	(467,26850)	(441,25968)

Concerning execution speeds, we think that for small-size instances, Procedure 1 is slightly faster than Algorithm 1 because the latter performs capacity and flow computations which, in the case of small sizes, have total computational time comparable to mn. But when mn increases, the time needed for capacity computations becomes negligible (in comparison with the quantity mn). This, in our opinion, informally explains why, for large mn values, the average execution times for the two heuristics become almost identical.

CONCLUSIONS

We have proposed a new heuristic treating SC as a restricted version of a flow problem. Our experiments show that this new heuristic is very efficient from the points of view of both the execution time and the approximation quality, and it dominates the naturally greedy algorithm which, however, is very efficient. An interesting theoretical problem arising from this paper seems to be the study of theoretical approximation bounds for Algorithm 1.

Acknowledgements—Many thanks to an anonymous referee for pertinent suggestions and useful remarks, which contributed to improving the presentation of this paper.

REFERENCES

1. C. LUND and M. YANNAKAKIS (1993) On the hardness of approximating minimization problems. In *Proc. Symposium on the Theory of Computing 1993*. pp. 286–293. Association for Computing Machinery, San Diego.
2. C. H. PAPADIMITRIOU and K. STEIGLITZ (1981) *Combinatorial Optimization: Algorithms and Complexity*. Prentice Hall, New Jersey.
3. M. SAKAROVITCH (1984) *Optimisation Combinatoire, Programmation Discrète*. Hermann, Paris.
4. D. S. JOHNSON (1974) Approximation algorithms for combinatorial problems. *J. Comp. Sys. Sci.* **9**, 256–278.
5. L. LOVÁSZ (1975) On the ratio of optimal integral and fractional covers. *Discr. Math.* **13**, 383–390.

Received September 1992; accepted February 1995 after four revisions