Informatics

Solution of Minimal Set Partition and Set Covering Problems

Natela Ananiashvili

Faculty of Exact and Natural Sciences, I. Javakhishvili Tbilisi State University, Tbilisi

(Presented by Academy Member Mindia Salukvadze)

ABSTRACT. The article considers solution of minimal set partition and set covering problems. As is known, the problems of minimal set partition and set covering belong to the class of complex NP problems. The efficient algorithm for precise solution of such problems does not exist nowadays (except in private cases). Solution time depends on a scale of the problem and it may significantly increase with it. A tree algorithm of simplified search is used. It became possible to decrease volume of occupied memory approximately 32·M-times with non-essential elaboration of programming techniques and computational time decreased approximately 32·M-times, where M is a number of covering subsets. For this purpose, partition matrix was compactly written and then basic operations were performed on columns of the matrix with logical operators. A complex of programs was developed in algorithmic language C++ and realized in Dev-C++ environment to solve these problems and check them on tests taken from OR-Library and real combinative problems that are known from the literature. The results are satisfactory and given in reasonable amount of time. The proposed complex of programs may be used for solution of the other problems of graph theory that can be deduced to problems of minimal set partition or set covering, or they are the sub-problems of such problems (for instance, problem of searching of dominant set of graph nodes). *© 2015 Bull. Georg. Natl. Acad. Sci.*

Key words: *set covering, set partition, precise solution.*

1. Introduction

In the problem of minimal set partition we have finite set $R = \{r_1, r_2, ..., r_N\}$ and collection (family) of subsets $L = \{S_1, S_2, ..., S_M\}$, $S_j \subset R$, $j = 1, 2, ..., M$, where each S_j has non-negative weight $c_j \geq 0$. Any subset (family) $L' = \{S_{j_1}, S_{j_2}, ..., S_{j_k}\}$ from L is called partition of set R, if the following two conditions are met:

$$\bigcup_{i=1}^{k} S_{j_i} = R \tag{1}$$

and

$$S_{j_h} \cap S_{j_l} = \otimes, \forall h, l \in \{1, 2, ..., k\}, h \neq l. \tag{2}$$

We must select the subset, where sum of weights $\sum_{j_i}^{j_k} c_{j_i}$ is minimal. The problem of minimal set partition is often written as a problem of linear programming. Assume $A = (a_{ij})$ is the matrix with $n \times m$ di-

Table 1. The results of experiments for set partition problem

Problem	Number of rows, N	Number of columns, M	Computational time (seconds)	Optimal value, f*	Note
sppnw06	50	6774	54.714	7810	
sppnw07	36	5172	0.24	5476	
sppnw08	24	434	0.063	35894	
sppnw09	40	3103	2981.71	67760	
sppnw10	24	853	0.195	68271	
sppnw12	27	626	0.188	14118	
sppnw16	139	148633	123.028	1181590	
sppnw20	22	685	0.06	16812	
sppnw21	25	577	0.044	7408	
sppnw24	19	1366	0.083	6314	
sppnw25	20	1217	0.228	5960	
sppnw31	26	2662	0.147	8038	
sppnw37	19	770	0.51	10068	
sppnw40	19	404	0.089	10809	
sppnw42	23	1079	0.037	7656	
sppkl01	55	7479	37.687	1113	Approximate value
sppnw03	59	43749	4347.16	25302	Approximate value
sppnw04	36	87482	568,216	17324	Approximate value

mensions, where $a_{ij} = 1$ if $i \in S_j$ and $a_{ij} = 0$ in other cases. Weight $c_j \geq 0$ corresponds to each S_j. Minimal covering implies the selection of columns of matrix A that will cover each row and their total weight will be minimal. We must minimize target function to formulate the problem

$$f(x) = \sum_{j=1}^{m} c_j \cdot x_j \tag{3}$$

with the following constraints:

$$\sum_{j=1}^{m} a_{ij} \cdot x_j, \; i = 1, \ldots, n, \; x_j \in \{0,1\}, j = 1, \ldots, m, \tag{4}$$

Here, variable x_j is equal to 1 if set S_j enters into covering. In other cases, x_j is equal to zero.

In the problem of minimal set covering, in difference from the problem of partition, non-intersection of covering sets is not necessary, i.e. it is not necessary to meet the above-mentioned condition (2). Other details are the same.

Because, the problems of minimal set partition and set covering belong to the class of complex NP problems [1]. The efficient algorithm for precise solution of such problems does not exist nowadays (except in private cases). Time of solution depends on the scale of problem and it may significantly increase. In practice, solution of these problems has applications in locating service facilities, transport scheduling [2], locating sources of power system [3] and data transfer [4]. Techniques of branches and edges are used for getting precise solution of these problems [2-6]. In recent days genetic algorithms [7-8], neuron nets [9] and other heuristic algorithms are often used to get approximate solutions of large-scale problems.

The current work offers precise solutions of problems of minimal set partition and set covering. Solution algorithm is based on the technique of branches and edges.

2. Solution of Problem of Minimal Set Partition

The article considers solution of the problem of minimal set partition [11] and then expands it to the problem of set covering. It is worthwhile to note that ap-

Table 2. The results of experiments for set covering problem

Problem	Number of Rows, N	Number of Columns, M	Computational Time (seconds)	Value of Coverage, f^*	Note
scpe1	50	500 (3972)	85.319	8	
scpe2	50	500 (4055)	23.483	10	
scpe3	50	500 (4066)	199.404	9	
scpe4	50	500 (3991)	2.067	11	
scpe5	50	500 (4063)	305.338	9	
scp41	200	1000 (3120)	0.222	573	Approximate value
scp53	200	2000 (6006)	0.062	353	Approximate value
scp63	200	1000 (8762)	0.634	183	Approximate value
scp410	200	1000 (3006)	0.217	752	Approximate value

probation of the program had good results for randomly developed matrices (by author) and for test problems taken from OR-Library[10]. The article shows the results of test problems of OR-Library solved by newly developed algorithm.

The logical design of problem solution program that is used for solving the problem of minimal set partition can be described in the following way: assume partition matrix is A, number of rows – N and number of columns – M. At the first stage the partition matrix A is divided into blocks, as we have in [2: 55-56]. Each block includes a column that corresponds to set Sj, if this set "covers" j_{th} node and may be some nodes from set $j+1, j+2,...,N$, but not from 1 to (j-1). Let us compactly write and "pack" each column (we'll describe the procedure of "packing" in details below). After this let us regroup the columns of each block according to the increment of weights (prices). It significantly decreases the number of selections and correspondingly, time necessary for realization. Then, let us use tree algorithm of simplified search presented in [2: 55-57]. Basic operations on columns of matrix are made by means of logical operators. It also gives economy in time of realization.

The offered procedure of packing allows to decrease computational time approximately 32•M-times with non-essential elaboration of programming techniques, because instead of N operation the program includes [N/32]+1 operations (here [N/32] denotes integer part of division of N by 32).

The concept of "packing" is the following: let us assume $A(N \times M)$ is the partition matrix, N – number of rows and M – number of columns. After dividing this matrix into blocks, it is reasonable to write it into Unsigned Long type dynamic matrix AP in "packed" form. Dimensions of matrix AP is ([N/32]+1) ×M, i.e. each column of matrix A will be packed in consecutive cells [N/32]+1 and written in the main memory. Because in tree algorithm of search each column is considered at least once and computational time for each column is decreased approximately 32-times, we can say that in the worst case computational time is decreased by 32•M-times.

The following algorithm is used to pack column A_j of matrix A:

Step 1: present N in the form $N = 32 \cdot l + k$, where $l \geq 0$ is integer, while k is integer and $0 \leq k < 32$;

Step 2: assume $Pck[ii] = 0, ii = 0,...,N$ (vector Pck should be Unsigned Long type); $i = 1$; $k_1 = 32$;

Step 3: assume $p = 1$;

Step 4: if $A_j\left((i-1)\cdot 32 + p - 1\right) = 1$, then assume c='0' ('0' is Unsigned Long type constant). Otherwise, assume c='1';

Step 5: assume $c = c << (32 - p)$ (where

$c = c << (32 - p)$ is the left shift in c with $(32 - p)$ binary digit; $Pck[i-1] = Pck[i-1] \mid c$, where $\mid$ is a logical sum.

Step 6: assume $p = p + 1$. If $p \le k$, then jump to the step 4;

Step 7: assume $i = i + 1$;

Step 8: if $i < l$, then jump to the step 4. If $i = l$, then assume $k_1 = k$ and jump to the step 3, otherwise $(i = l + 1)$ finish the algorithm.

The program also realizes reverse function of packing that unpacks (expands) a vector. This algorithm expands (unpacks) the Unsigned Long type array with [N/32]+1 dimensions into integer type array B with N components. Let us describe the algorithm of unpacking:

Step 1: present N in the form $N = 32 \cdot l + k$, where $l \ge 0$ is integer, while k is integer and $0 \le k < 32$;

Step 2: assume $B_j = 0, j = 1,...,N;\ i = 1;\ k_1 = 32;$

Step 3: assume $j = 1$, $c_0 =' 1'$ ('1' is the unsigned long type constant);

Step 4: assume $c = c_0 << (32 - j - 1)$ (where $c_0 << (32 - j - 1)$ is the left shift in c_0 $(32 - j - 1)$ with binary digit; assume $c_1 = Pck[i-1] \& c$, where & is a logical multiplication;

Step 5: if $c_1 \ne' 0'$, assume $B((i-1) \cdot 32 + j) = 1$;

Step 6: assume $j = j + 1$. If $j \le k_1$, then jump to the step 4;

Step 7: assume $i = i + 1$. If $i = l$, assume $k_1 = k$ and jump to the step 3, otherwise $(i = l + 1)$ finish the algorithm.

The program is checked on several randomly formed problems and on test files from OR-Library. The results of solutions of problems of OR-Library are shown below in the form of Table, where

$$f^* = min \sum_{j=1}^{M} c_j \cdot x_j . \tag{5}$$

We note that value is approximate for several problems. For instance, computing was stopped for the problem sppk101, when the value of function became 1113, because after over 1 hour of computation the result did not get any better. In every other case the problems are precisely solved.

If we compare these results with the results of solution of problems from [12] ([12] uses hybrid technique), the results of our article are better. For instance, in sppnw06 problem optimal value is 7810 and computational time is 54.714. In [12] the optimal value is 9788 and 8038; computational time is 78.7 and 30.3 seconds. Certain problems, for instance sppnw08, are not solved in [12]. In [13] the algorithm of linear programming is used that consumes the technique of minimal square. For sppnw06 problem optimal value is 7640 in [13]. CPU time for this problem is 0.14 and quantity of iterations is 924. Other solutions generally coincide with the above-mentioned results.

3. Solution of the Problem of Minimal Set Covering

The same idea [11] is used for solving the problem of minimal set covering, i.e. columns of cover matrix are written in packed form, as it was done for the problem of partition, then the program is modified and the problem of covering is solved. The important aspects of modification are the following: the columns of cover matrix were divided into blocks, exactly as it was done for the problem of partition, but as there was no requirement for non-intersection of cover columns, each column that corresponded to the set $S_j = \{r_{j_1}, r_{j_2}, ..., r_{j_k}\}$ was brought in every block: $r_{j_1}, r_{j_2}, ..., r_{j_k}$. Obviously, dimensions of cover matrix significantly increased. For example, in one case (problem scpe1) matrix with dimensions 50×500 took the dimensions 50×3972 and in other case (problem scpe2) dimensions increased to 50×4055. When the current problem is solved, it is not necessary to check non-intersection of every new column with current columns, if an initial scale of problem is not very large (50×500) or it did not increase dramatically. In the Table the column "Number of Columns" shows the number of matrix columns taken from the problem of

OR-Library. Number of columns indicated in parenthesis is based on our algorithm. Modified program used the matrix of the dimensions during computations.

The Table shows that cover algorithm is efficient for solution of relatively small-scale (approximately 50x500) problems and quickly gives precise solution. Besides, it quickly gives the approximate values for the solution of large-scale problems and this solution does not get better after over 1 hour of computation.

The algorithm was tested on computer with standard specifications: Intel(R) Pentium (R) Dual CPU E2220, 2.40 GHz, 2.00 GB of RAM.

The results show that the algorithm offered in the article is quite efficient for getting precise solutions of small-scale problems and approximate solutions of large-scale problems. Approximate solutions can be used as the initial approximations for the other algorithms.

ინფორმატიკა

უმცირესი დაყოფისა და დაფარვის ამოცანების ამოხსნა

ნ. ანანიაშვილი

ივ. ჯავახიშვილის თბილისის სახელმწიფო უნივერსიტეტი, ზუსტ და საბუნებისმეტყველო მეცნიერებათა ფაკულტეტი, თბილისი

(წარმოდგენილია აკადემიკოს მ. სალუქვაძის მიერ)

სტატიაში განხილულია უმცირესი დაყოფის და დაფარვის ამოცანების ამოხსნა. როგორც ცნობილია, უმცირესი დაყოფის და დაფარვის ამოცანები მიეკუთვნება NP რთული ამოცანების კლასს. დღეისათვის მათი ზუსტი ამოხსნისათვის გარდა მისი კერძო შემთხვევებისა, არ არსებობს ეფექტური ალგორითმი, ამოხსნის დრო დამოკიდებულია ამოცანის ზომებზე და ის შესაძლოა ძლიერ გაიზარდოს ამოცანის ზომის ზრდასთან ერთად. გამოყენებულია ძებნის ხის გამარტივებული ალგორითმი. პროგრამირების ტექნიკის არაარსებითი გართულების ხარჯზე შესაძლებელი გახდა გამოყენებული მეხსიერების 32·M-ჯერ შემცირება, ასევე გამოთვლისათვის საჭირო დროის მიახლოებით 32·M-ჯერ შემცირება, სადაც M დამფარავი ქვესიმრავლეების რაოდენობაა. ამ მიზნით ჯერ დაყოფის მატრიცა ჩაიწერა კომპაქტურად და შემდეგ მიღებული მატრიცის სვეტებზე ძირითადი ოპერაციები შესრულდა ლოგიკური ოპერატორებით. ამ ამოცანების ამოსახსნელად დაიწერა პროგრამების კომპლექსი ალგორითმულ ენა C++-ზე, რეალიზებულია Dev-C++ გარემოში. კომპლექსი აპრობირებულია ლიტერატურაში კარგად ცნობილ რეალურ კომბინატორული ტიპის ამოცანებზე – ცნობილი OR-Library-დან აღებულ სატესტო ამოცანებზე. მიღებულია საკმაოდ დამაკმაყოფილებელი შედეგები დროის გონივრულ ინტერვალებში. შემოთავაზებული მეთოდი შესაძლოა გამოყენებულ იქნეს გრაფთა თეორიის იმ პრაქტიკული ამოცანების ამოსახსნელად, რომლებიც უშუალოდ მიიყვანებიან უმცირესი დაყოფის ან დაფარვის ამოცანაზე, ან რომლებიც წარმოადგენენ მის ქვეამოცანას (მაგალითად გრაფში წვეროთა დომინირებადი სიმრავლის მოსაძებნად).

REFERENCES

1. *Garey M. R., Johnson D. S.* (1979) A guide to the theory of NP-completeness. San Francisco: W. H. Freeman and Co.
2. *Christofides N.* (1986) Computer science and applied mathematics, London. p. 55-57.
3. *Minieka E.* (1978) Optimization Algorithms For Networks And Graphs, New York.
4. *Berge C.* (1962) Theory of graphs and its applications, London.
5. *Balas E., Ho A.* (1980) Mathematical Programming 37-60.
6. *Fisher M. L., Kedia P.* (1990) Management Science 36*:* 674-688.
7. *Beasley J. E., Jonsten K.* (1992) European J. Oper. Res. **58**, 2: 293-300.
8. *Eremeev A. V.* (2000) Artificial Evolution. Proc. Berlin: Springer, 84-95.
9. *Grossman T., Wool A.* (1997) European J. Oper. Res. **101***,* 1: 81-92.
10. *Beasley J. E.* (1990) Journal of the Operational Research Society. **41**, 11: 1069-1072.
11. *Ananiashvili N.* (2014) V Annual international conference of the Georgian mathematical union, Batumi, p. 60.
12. *Crawford B*., *Soto R., Monfroy E., Castro C., Palma W. and Paredes F.* (2013) Mathematical Problems in Engineering, v. **2013**: 12 p.
13. *Seunghyun K.* (2007) Linear Programming Algorithms Using Least-squares Method, Georgia Institute of Technology. USA.

Received December, 2014