Efficient and Effective Practical Algorithms for the Set-Covering Problem

Qi Yang, Jamie McPeek, Adam Nofsinger
Department of Computer Science and Software Engineering
University of Wisconsin at Platteville
Platteville, WI 53818, USA

Abstract - *The set-covering problem is an interesting problem in computational complexity theory. In [1], the set-covering problem has been proved to be NP hard and a greedy heuristic algorithm is presented to solve the problem. In [2], the set-covering problem is found to be equivalent to the problem of identifying redundant search engines on the Web, and finding efficient and effective practical algorithms to the problem becomes a key issue in building a vary large-scale Web meta-search engine. A new algorithm Check-And-Remove (CAR) is proposed in [2] with a better time complexity than the greedy algorithm presented in [1]. However, in some cases the cover set produced by the new algorithm is too large to be acceptable. We propose some changes to the data structure that improve the performance of both algorithms. We also present a new greedy algorithm whose time complexity is the same as that of the CAR algorithm. The experimental results show that our final greedy algorithm runs faster than the CAR algorithm and produces better results in all test cases.*

Keywords: NP-Hard, Approximation solutions, Greedy algorithm, Set-covering problem.

1 Introduction

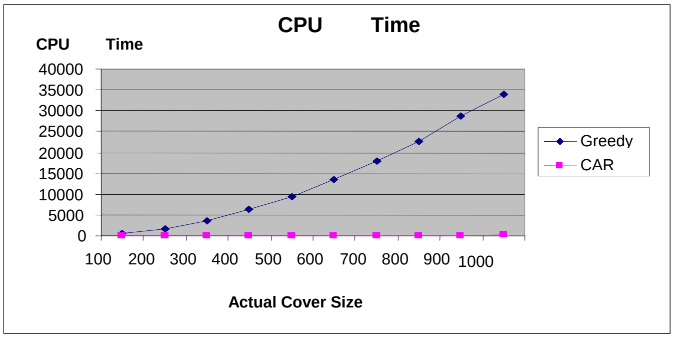
The set-covering problem is a well-defined mathematical problem and also an interesting problem in computational complexity theory. Given N sets, let X be the union of all the sets. An element is covered by a set if the element is in the set. A cover of X is a group of sets from the N sets such that every element of X is covered by at least one set in the group. The set-covering problem is to find a cover of X of the minimum size. In [1], the problem is discussed in detail and is proved to be NP hard.

Although the set-covering problem is an interesting problem in theory, it has not attracted much attention in research and industry communities, because no real applications were found to require a solution to the problem. In [2], the set-covering problem is found to be equivalent to the problem of identifying redundant search engines on the Web, and finding an effective and efficient practical algorithm to the problem becomes a key issue in building a very large-scale Web meta-search engine.

We need approximation solutions for this problem since the problem is NP hard. In [1], a greedy approximation algorithm is presented with a time complexity of O (M * N * min (M, N)), where N is the number of sets and M is the number of all elements of the union of the N sets. In [2], a new algorithm called Check-And-Remove (CAR) is proposed and its time complexity is O(N * M).

Some experimental results are reported in [2]. In all cases, the CAR algorithm runs much faster than the Greedy algorithm. The cover sizes from the two algorithms are very close to each other in most cases; but in one case where the actual minimum cover size is small with respect to the total number of sets, the cover size from the CAR algorithm is much larger than the actual minimum size, while that from the Greedy algorithm is very close to the actual minimum size. Some results from [2] are shown in the following.

Cover Sizes of the Two Algorithms					
Actual	100	300	500	700	900
Greedy	105.8	300.2	501	700.6	900.2
CAR	485.8	300	500	700	900



We have run the two algorithms with more data sets and observed the same results: The CAR algorithm is always much faster than the Greedy algorithm, but for some data sets the cover size from the CAR algorithm is larger than that from the Greedy algorithm.

In this paper, we propose some changes to the data structure to improve the performance of both algorithms. We also present a new greedy algorithm whose time complexity is the same as that of the CAR algorithm. The experimental results show that our final greedy algorithm runs faster than the CAR algorithm and produces better results in all test cases.

2 Data Generation

The implementation reported in [2] takes a special approach to generate data. The minimum cover size (CoverSize) is determined before hand and is passed to the data generator program, which first generates CoverSize non-overlapping sets, then generates other sets by randomly selecting elements from the union of the CoverSize sets. After generating all the sets, it shuffles them and outputs data to data files. The advantage of the approach is that the minimum cover size is known, but the produced data sets may not represent general cases.

We take a different approach to generate the test data. A range for the set size is decided before hand, and the size of each set is determined randomly according to the uniform or normal distribution. Similarly, a range for the elements (generated as integers but treated as strings by all algorithms) is given, and the elements are generated according the uniform or normal distribution. The minimum cover size is unknown, but by changing the ranges and the choice of distribution, more general data sets can be generated. All of our experiments use test data sets from the new approach, unless stated otherwise.

3 Algorithm Greedy and Algorithm CAR

The two algorithms are presented in the following, where ResultCover is the cover to be generated and Uncovered is the set of elements that are not covered by ResultCover. The greedy algorithm tries to find the best set (the one with the most uncovered elements) to add to the result cover; it should produce a better result (a smaller cover) and run slower, since it spends a lot of time to find the best set. The CAR algorithm takes the opposite approach: add any set to the result cover as long as it has at least one uncovered element. The algorithm should run faster, but the produced result cover may not be as good as that from the greedy algorithm. That is why the algorithm has a remove phase.

Notice that it is possible that a set is added to the result cover but could be removed from the result cover later after adding other sets to the result cover. For example, sets S1 = {1, 2, 4} and S2 = {1, 2, 5} are added to the result cover first. After adding sets S3 = {1, 2, 3} and S4 = {4, 5, 6}, S1 and S2 should be removed from the result set to get a better cover.

Algorithm Greedy

```
1. Set ResultCover to the empty set
2. Set Uncovered to the union of all sets
3. While Uncovered is not empty
   a. select a set S that is not in ResultCover and covers the
      most elements not covered by ResultCover
   b. add S to ResultCover
   c. remove all elements of S from Uncovered
```

Algorithm CAR (Check and Remove)

```
1. Set ResultCover to the empty set
2. For each set S
   a. determine if S has an element that is not covered by
      ResultCover
   b. add S to ResultCover if S has such an element
   c. exit the for loop if ResultCover is a cover of X
3. For each set S in ResultCover
   a. determine if S has an element that is not covered by any
      other set of ResultCover
   b. Remove S from ResultCover if S has no such an element
```

4 Row-wise vs. Column-wise

A matrix is used in [2] to represent the set-covering problem. Each row of the matrix represents a set and each column represents an element from the union of all the sets. The number of sets N is known and it is the number of rows of the matrix. The number of elements of each set is also known; but the number of elements of the union of the N sets (the number of columns) is unknown until all sets have been read in and sorted in some way, since an element could be covered by multiple sets. The following matrix represents the case with three sets and six elements.

	a	b	c	D	e	f
S1	0	1	1	0	1	0
S2	1	0	1	1	0	0
S3	1	1	0	1	0	1

The implementation in [2] uses a binary search tree to input data. Each node of the tree stores one element with a bitmap to indicate which sets cover the element. We can see that the bitmap represents the corresponding column of the matrix. After the tree is built, it is converted to an array of bitmap and both algorithms work with the array.

Both algorithms need to perform some operations on the rows of the matrix such as to find the number of elements in a set that are not covered by the result cover, to determine if a set contains an element that is not covered by the result cover, or to determine if a set in the result cover has an element that is not covered by any other sets in result cover.

Since a bitmap represents a column of the matrix, the cost of going through one row of the matrix is close to going through the entire matrix.

Our first improvement is to use bitmaps that represent rows of the matrix instead columns. The binary search tree remains the same; after the tree is built, it is converted to an array of row bitmaps instead of an array of column bitmaps. Both algorithms and their time complexity remain the same. The conversion takes some extra time, and the running time of the CAR algorithm increases a little bit when the total running time is very short. For example, the running time (seconds) increases from 0.01 to 0.09 and 0.31 to 0.39. But in all other cases, the running time is reduced a lot for both algorithms, especially for the Greedy algorithm. For example, the running time is reduced from 11.15 to 3.27 and 20.70 to 5.46 for the CAR algorithm, and from 0.63 to 0.28, 1220 to 161 and 5056 to 629 for the Greedy algorithm. The running time includes the time to convert the tree to the bitmap array and the time to find a cover, but excludes the time to read data to the tree since it's the same for both algorithms.

Running Times (seconds) of the Greedy Algorithm							
Col	0.63	53.9	300	1220	2130	3457	5056
Row	0.28	7.6	41	161	274	437	629

Running Times (seconds) of the CAR algorithm							
Col	0.01	0.31	1.63	6.36	11.15	16.92	20.70
Row	0.09	0.39	0.96	2.12	3.27	4.34	5.46

Running Times (seconds) of the Two algorithms							
Greed	0.28	7.6	41	161	274	437	629
CAR	0.09	0.39	0.96	2.12	3.27	4.34	5.46

The Greedy algorithm still runs much slower than the CAR algorithm, because it has a higher time complexity. But it produces smaller cover sets in most cases: 10 to 16, 191 to 235, and 625 to 648. Only when the cover size is close to the total number of sets, the cover size by the Greedy algorithm is slightly larger than that by the CAR algorithm: 849 to 824, and 984 to 975.

Cover Sizes of the Two algorithms							
Greed	10	87	191	424	625	849	984
CAR	16	120	235	467	648	824	975

The Greedy algorithm always tries to find the best set (the one with the most elements not covered by the result cover) to add to the result cover. It should produce better results in most cases, but it spends a lot of time to find the best set and pays a much higher cost for being greedy.

5 Algorithm Greedy Update

To improve the efficiency of algorithm Greedy, we modified it by keeping the count of elements of each set that have not been covered by the result cover and updating the counts when a new set is added to the result cover.

Algorithm Greedy Update

```
1. Set ResultCover to the empty set
2. Set Uncovered to the union of all sets
3. For each set, set the UncoveredCount to the size of the set
4. While Uncovered is not empty
   a. select a set that has the largest value of UncoveredCount
      among all sets not in ResultCover
   b. add the set to ResultCover
   c. remove all elements of the set from Uncovered
   d. update the value of UncoveredCount for each set not in
      ResultCover
```

The major issue here is to update the count of uncovered elements for each set. Before a set is added to the result cover, each element in the set is examined to see if the result cover already covers it. Nothing needs to be done if the element is covered already; otherwise, each set not in the result cover is examined and its uncovered count is decremented by one if the set contains the element.

In the following example, there are three sets and six elements. At beginning, no elements are covered by the result cover, and the uncovered count is 3, 3 and 4 for the three sets, respectively. Set S3 has the largest uncovered count and is added to the result cover first, indicated by (*) in the second table. The uncovered count for S1 is updated from 3 to 2, since it contains one element of S3 (b); the uncovered count of S2 is updated from 3 to 1, since it contains two elements of S3 (a and d). Now S1 has the largest uncovered count, and is added to the result cover, indicated by (*) in the last table ((#) indicates S3 was added to the result set before). S1 contains three elements, but element b is covered by the result cover before adding S1, and only elements c and e are examined. The uncovered count of S2 is updated from 1 to 0, since it contains element c.

	a	b	c	d	e	f
S1 (3)	0	1	1	0	1	0
S2 (3)	1	0	1	1	0	0
S3 (4)	1	1	0	1	0	1
ResultCover	0	0	0	0	0	0

	a	b	c	d	e	f
S1 (2)	0	1	1	0	1	0
S2 (1)	1	0	1	1	0	0
S3 (*)	1	1	0	1	0	1
ResultCover	1	1	0	1	0	1

	a	b	c	d	e	f
S1 (*)	0	1	1	0	1	0
S2 (0)	1	0	1	1	0	0
S3 (#)	1	1	0	1	0	1
ResultCover	1	1	1	1	1	1

For each element, the first time it is covered by the result cover, all sets not in the result cover will be examined to see if the set contains the element. This can be done easily as long as the index position is maintained. Thus the time complexity is the same as that of the CAR algorithm, O(N * M), where N is the number of all sets and M is the number of elements of the union of all sets. The experimental results show the running time of the Greedy Update algorithm is still larger than that of the CAR algorithm, but it is at the same magnitude as algorithm CAR.

Running Times (seconds) of the Two algorithms							
Update	0.15	0.92	2.26	5.13	7.31	10.1	13.1
CAR	0.09	0.39	0.96	2.12	3.27	4.34	5.46

6 Algorithm List and Remove (LAR)

Finally we implement the matrix using linked list. As we know, linked list works more efficiently for a sparse matrix. When the matrix is dense, the cover size will be small and the running time should be very short. We also add the remove phase to the greedy algorithm and call it Algorithm List and Remove (LAR).

Algorithm List and Remove (LAR)

```
1. Set ResultCover to the empty set
2. Set Uncovered to the union of all sets
3. For each set, set the UncoveredCount to the size of the set
4. While Uncovered is not empty
   a. select a set that has the largest value of UncoveredCount
      among all sets not in ResultCover
   b. add the set to ResultCover
   c. remove all elements of the set from Uncovered
   d. update the value of UncoveredCount for each set not in
      ResultCover
5. For each set S in ResultCover
   a. determine if S has an element that is not covered by any
      other set of ResultCover
   b. remove S from ResultCover if S has no such an element
```

Although the time complexity remains the same for both algorithms, Algorithm LAR runs faster than Algorithm CAR in all test cases. This is the advantage of combining linked list and updating the uncovered count for each set. Furthermore, the cover size from algorithm LAR is smaller than that from algorithm CAR in all cases. See the following tables for more details.

Running Times (seconds) of the Two algorithms							
LAR	0.21	0.35	0.51	0.86	1.11	1.40	1.66
CAR	0.26	0.49	0.65	1.01	1.24	1.46	1.67

Cover Sizes of the Two algorithms							
LAR	10	87	191	422	607	815	971
CAR	16	120	235	467	648	824	975

For the data sets generated in our experiments, we do not know the actual cover size, and it's not practical to find the actual size. We have run both algorithm CAR and algorithm LAR on the data sets generated by the approach from [2]. Recall that in this approach the actual cover size (CoverSize) is decided first, then CoverSize non-disjoint sets are generated, and other sets are generated by selecting elements from the union of the CoverSize sets. The total number of sets is fixed at 1000. The cover size from algorithm LAR is the same as the actual size in all cases. This is because the greedy algorithm is always trying to find the best sets to add to the result cover and will find the non-disjoint sets. For algorithm CAR, the cover size is the same as the actual cover size when the cover size is 200 and above; but when the actual cover size is smaller, the cover size is much larger than the actual size; this is because many other sets are selected before any of the non-disjoint sets gets a chance to be selected.

Cover Sizes of the Two algorithms							
Actual	50	70	90	110	200	500	900
LAR	50	70	90	110	200	500	900
CAR	291	391	496	528	200	500	900

7 Summary

We proposed a new version of greedy algorithm for the set-covering problem based on linked list presentation of a matrix and updating the uncovered count for each set. Our algorithm runs faster than the previously presented algorithm CAR and generates smaller result cover in all test cases.

8 References

[1] T. H. Cormen, C.E. Leiserson, R. L. Rivest. "Introduction to Algorithms". The MIT Press, 1991.

[2] R. Desai1, Q. Yang, Z. Wu, W. Meng, C. Yu. "Identifying Redundant Search Engines in a Very Large Scale Metasearch Engine Context". ACM WIDM'06 (8th ACM International Workshop on Web Information and Data Management, November 10, 2006, Arlington, Virginia, USA, pp. 51-58.