CSC 742
Team k – Prashant Gupta, Divya Jain

Problem

Set Cover Problem: Given a universe X of n items and a collection S of 'm' subsets of X:
S = {S1, S2, . . . , Sm}. We assume that the union of all of the sets in S is X, with |X| = n.
The aim is to find the smallest sub-collection of sets in S that covers all of X.
Finding an optimal solution is NP Hard, but we present here an efficiency comparison of a baseline and a proposed effective algorithm which gives close to optimal solutions.

Motivation

The primary reason we chose this particular problem was because this problem has a lot of applications in different areas of life. And though finding an optimal solution is a NP Hard problem, we have tried to implement an algorithm which improves upon the drawbacks of a baseline inefficient algorithm or a greedy algorithm. This algorithm performs well with a huge amount of data, which enables it to be scalable.

Research Paper

"Set Cover Algorithms for very large datasets"
Authors : Graham Cormode (AT&T Labs - Research, Florham Park, NJ, USA); Howard Karloff((AT&T Labs - Research, Florham Park, NJ, USA); Anthony Wirth(The University of Melbourne, Parkville, Australia)
Published In :
Proceeding
CIKM '10 Proceedings of the 19th ACM international conference on Information and knowledge management
Pages 479-488
ACM New York, NY, USA ©2010

Relation Structure of database in Oracle

Table Name: setCover (create table setCover (input varchar2 (100)))
Each tuple of the table is a set of space separated integers representing the elements of the universe, hence each tuple represents a set and the number of tuples represents the number of sets.

Goal of the Algorithm : To find minimum number of sets which cover the universe of elements existing in all the sets.

Baseline Algorithm
Pseudo code

Connect to the oracle database where input is stored
Read each row of the database to get the collection of sets.
Max_Number_of_sets = count(setCoverList)
Initialize SetCover_Array: It stores the sets which cover all the distinct elements of the Collection. This collection of sets is setCoverList.

For number_of_sets = 1 to number_of_sets = Max_Number_of_sets
    Generate combinations of number_of_sets
    Check if combination covers all distinct elements of the collection
    If(true)
        Add to SetCover_Array
Sort the SetCover_Array
Output the first set of SetCover_Array. This contains the minimum number of sets which covers the universe of the elements of database.

Proposed Algorithm
Pseudo code

Connect to the oracle database where input is stored
Read each row of the database to get the collection of sets.
This collection of sets is setCoverList.
Select a real valued parameter p > 1
For each set of the setCoverList, a 'k' value is generated based on the size of the set and the parameter p and placed in a key value Tree Map with k as the key.
k value for each set S_i is decided based on the following criteria
    a set belongs to a value k when it satisfies $p^k \leq |S_i| \leq p^{k+1}$
(The tree map orders the values of 'k' in descending order)

For each k in descending order, get the corresponding sets S^k

    For each set S_i in S^k

- If $|Si \setminus C| \geq p^k$: add i to Σ and update C. where Σ consists of the solution sets and C consists of the covered elements. Also increment the answer variable 'Ans' count.

  |Si\c| denotes the elements of the set which are uncovered after removing the elements which are covered.
- Else: let set $S_i \dashleftarrow Si \setminus C$ and add the updated set to sub collection $S^{(k')}$, where the new set size satisfies $p^{k'} < |Si| < p^{k'+1}$ (and therefore k' < k).

    For each set Si in S(0):

- If $|Si \setminus C| = 1$: add i to Σ and update C.

Output the variable 'Ans', it gives the minimum number of sets that cover all the elements of the dataset.

Example to follow the above given proposed algorithm

Let the initial set collection given be as follows:

S1 {1,2,3,4,5} S2 {1,2,4,6,7} S3 {1,6,7} S4 {2,3,7} S5 {7,8} S6 {5,8} S7 {3,9} S8 {1} S9 {5} S10 {9}

We take parameter p = 2 in this example.

Then based on the equation $p^k \leq |S_i| \leq p^{k+1}$ k from 1 to 3 (since maximum size of a set is 5 here), the sets are divided into the following key value pairs and stored into tree map as follows

K	Sets
2	{1,2,3,4,5} , {1,2,4,6,7} (Since size of the set is between 4(pow(2,2) and 7(pow(2,3))
1	{1,6,7}, {2,3,7}, {7,8}, {5,8}, {3,9} (Since size of set is between 2(pow(2,1) and 3(pow(2,2))
0	{1}, {5} (Since size of set is 1(pow(2,0))

We first process the first set of k = 2 {1,2,3,4,5} and since all the elements are right now uncovered, we add all five elements to C and S1 to Σ and increment Ans to 1.
Next we process S2 {1,2,4,6,7}. Now S\C in this case becomes 2 because 3 out of 5 elements are covered already by the previous set. Now since the uncovered elements are only two, this set is appended to the set list of k = 1 since it now satisfies between 2(pow(2,1)) and 3(pow(2,2))
And this is how we process every set, keep updating Σ and C and incrementing Ans till we reach the point where C consists of every unique element of the universe of dataset. This is when we output value of Ans as the minimum set cover count.

Improvements made over the course of the project

- Tried to explain the pseudo code of the proposed algorithm in a more structured way, explaining the meaning of each symbol used.
- Also have given an example which exactly proceeds on the same structure as the pseudo code , hence anyone trying to understand the working of the algorithm can easily do so with the help of the example
- Initially, we took standard greedy algorithm to be our baseline algorithm. The greedy algorithm chose the sets with the highest number of elements first and then consequently, tried to find the minimum set cover. But then we realized that if we generate all the combinations of set one by one, it would take the maximum time to obtain the result as it will need to process each and every combination to check whether it covers all the elements of the universe. Hence we decided to pursue this particular method as our baseline algorithm as this seemed more inefficient compared to even standard greedy algorithm, which enables us to come up with better efficiency comparison results between baseline and proposed algorithm.
- Introducing the parameter affects both the efficiency and the running time of the algorithm. It shrinks the set size once the elements are added to the covered list C and hence ensures that when the set is again processed, it doesn't have atleast more than p^{k+1} elements uncovered. This improves the running time and the efficiency of the algorithm by multiple folds. Please refer to the following section which evaluates the efficiency of the baseline algorithm and the proposed algorithm and compares the performance to corroborate why using the proposed algorithm is suggested.

Experimental Evaluation of efficiency
Baseline vs. Proposed Algorithm

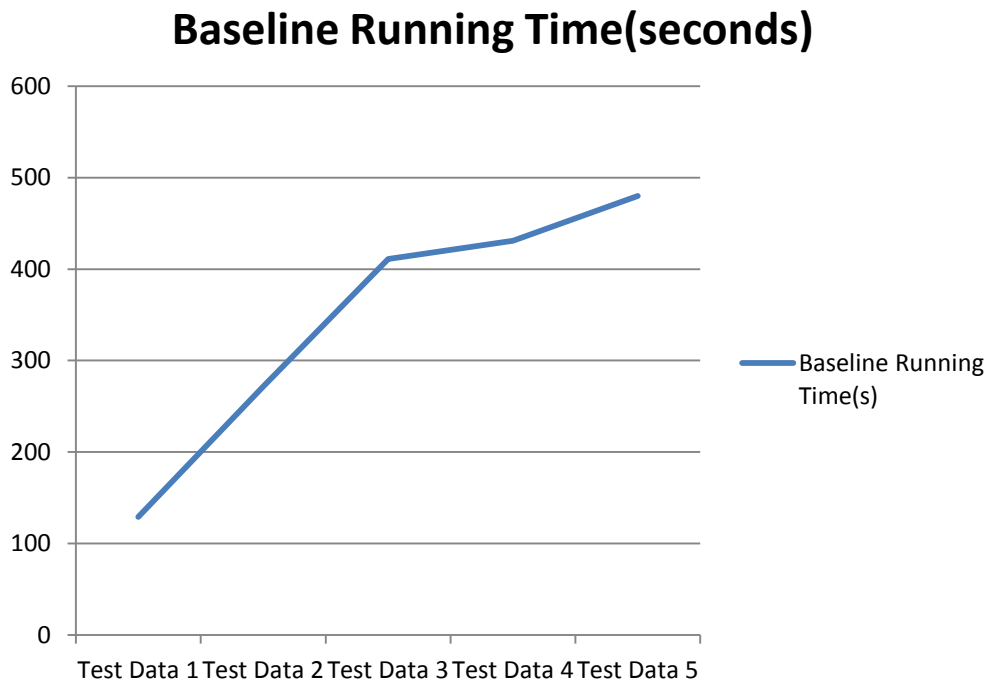
Baseline Algorithm searches for every combination from size 1 to the size of the set cover list and then finds the minimum combination which covers all the elements. This makes the runtime of the baseline algorithm exponential and inefficient.

The proposed algorithm introduces an approximation parameter, which reduces the runtime of the algorithm to a great extent. It does so, by reducing the size of the sets on every traversal according to the number of elements left uncovered. This improves the efficiency and the running time of the process of finding the minimum set cover. The tradeoff is only of choosing an appropriate approximation parameter value (p), as that determines the initial distribution of the sets according to their sizes.

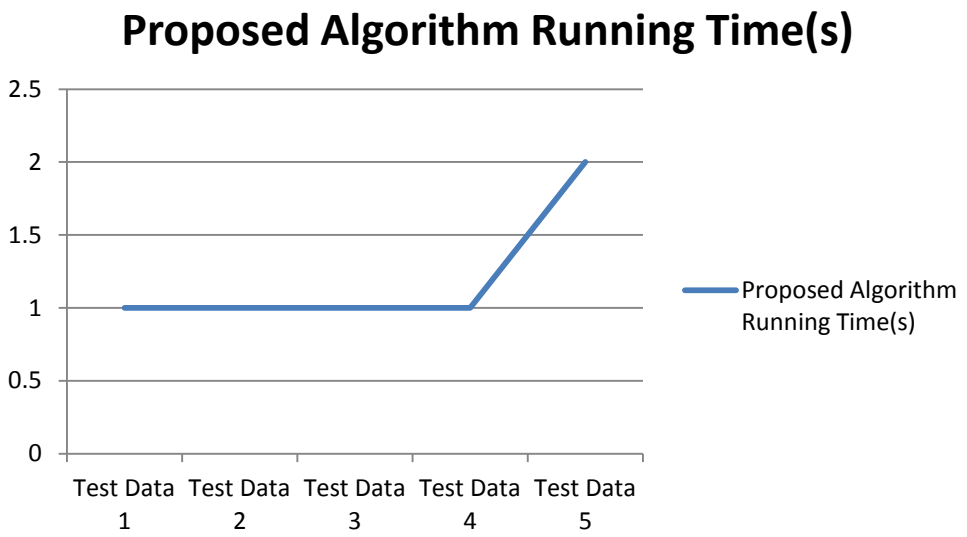
Given below are five test cases and the graphical representation of the performance evaluation of the baseline algorithm and the proposed algorithm.

Test Data	Number of Sets	Number of Elements per set	Baseline Running Time(s)	Proposed Algorithm Running Time(s)
Test Data 1	23	15	129	1
Test Data 2	24	15	272	1
Test Data 3	25	10	411	1
Test Data 4	25	11	431	1
Test Data 5	25	13	480	2

Five Test Cases



Graph of Baseline Running time for the five test cases



Graph of Efficient Algorithm Running time for the five test cases

Conclusion

Through this project, we managed to implement an efficient algorithm which solves the problem of finding the minimum set cover if not completely(NP Hard) but in optimal time. This algorithm definitely gives a much improved running time than the naïve set search or even the greedy algorithm. It also helped us to understand the practical applications of finding the set cover in day to day life and we believe that this algorithm can help in these areas to a great extent.