CSC 742 Project Proposal
Team K – pgupta7, djain2

Research paper Bibliography

Title: 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

Problem Description

Given a collection of sets, each of which consists of each of which is drawn from a common universe of possible items. The problem of Set Cover is to find the smallest subcollection of these sets so that their union consists of every item from the universe of items. Though finding an optimal solution is NP hard and greedy algorithm finds only a solution which is somewhat close to the optimal, the problem arises when the input data set is very large and we need to store it in disk and require disk efficiency.

Why is this algorithm not part of any standard database-management system

Efficient Processing of large datasets is challenging using standard database systems. The problem of Set Cover can be solved using a standard greedy algorithm resulting in a somewhat optimal solution but it is very difficult and costly when the input is very large and we need disks to store data. It requires random accesses to disks which turn out to be expensive as compared to linear search of the disk and produces heavy overhead and accessing costs. The efficient algorithm that we are implementing results in a solution similar to the standard greedy algorithm but improves the speed and the cost of accessing by almost ten fold using the modern disk technology.

Important real-life applications

- In the field of operations research, the problem of choosing where to locate a number of facilities so that all sites are within a certain distance of the closest facility can be modeled as an instance of Set Cover. Here, each set corresponds to the sites that are covered by each possible facility location.

- In machine learning, a classifier may be based on picking examples to label. Each item is classified based on the example(s) that cover it; the goal is to ensure that all items are covered by some example, leading to an instance of Set Cover.
- In planning, it is necessary to choose how to allocate resources well even when demands are predicted to vary over time. Variations of Set Cover have been used to capture these requirements.
- In data mining, it is often necessary to find a“minimal explanation” for patterns in data. Given data that correspond to a number of positive examples, each with a number of binary features (such as patients with various genetic sequences), the goal is to find a set of features so that every example has a positive example of one of these features. This implies an instance of Set Cover in which each set corresponds to a feature, and contains the examples which have that feature.
- In data quality, a set of simple rules which describes the observed data helps users understand the structure in their data. Given a set of rules that are consistent with the observed data, the tableau generation problem is to find a subset of rules which explains the data without redundancy. This is captured by applying Set Cover to the collection of rules.
- In information retrieval, each document covers a set of topics. In response to a query, we wish to retrieve the smallest set of documents that covers the topics in the query, i.e., to find a set cover.

Input and Output of the Algorithm

Input : We assume 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 objective of the algorithm is to find a subcollection of sets in S, of minimum size, that covers all of X.

Example: input of m = 10 sets over the universe {A,B,C,D,E,F,G,H, I} of size 9.

S1 ABCDE
S2 ABDFG
S3 AFG
S4 BCG
S5 GH
S6 EH
S7 CI
S8 A
S9 E
S10 I

Output : Though finding a minimal set cover is a NP hard problem, and hence the output of our algorithm in terms of the number of sets is similar to that given by a standard greedy approach, i.e. the number of sets output are close to an optimal solution, but our efficient algorithm scales well to larger datasets and provides output ten times faster than the standard baseline algorithm, especially when the problem instance needs to be stored on a disk.
Therefore the output is a sub collection of the input datasets of size close to the minimal which covers all of X.

<u>Scope of input/output in paper</u>

In the scope of the paper, the authors have experimented on millions of real datasets spanning several orders of magnitude and against variations of the standard greedy algorithms. We would be performing the testing of the algorithm on a limited size of dataset , not as large as the million size. We would compare the efficiency and speed and cost of computation of results by the efficient algorithm versus the standard greedy algorithm.

<u>Working of the algorithm</u>

The algorithm partitions the sets into sub-collections based on the sizes of the sets. we assume a real valued parameter $P \geq 1$ which governs the running time of the algorithm.

Initially, we assign set Si to subcollection S(k) if pk <· |Si| < pk+1; let K be the largest k with non-empty S(k). The algorithm then proceeds in two loops:

* For k -> K down to 1:

- 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
- Else: let set S_i ⇠ Si \ C and add the updated set to subcollection S(k'), where the new set size satisfies pk' < |Si| < pk'+1 (and therefore k' < k).

* For each set Si in S(0):

– If |Si \ C| = 1: add i to Σ and update C.

Through this algorithm implementation, aim is to find a set whose uncovered element count is close to maximal, in a way that the interaction with the disk or secondary memory is in a friendly manner with few passes through the data and a good approximation factor.

<u>Baseline Algorithm</u>

The baseline algorithm for the set cover problem is the standard greedy algorithm.

Let Σ be the set of indices of sets in the and let C be the elements covered so far. Initially there are no sets in the solution and every element is uncovered, so that Σ = Φ; and C = Φ;. We repeat the following steps until all elements are covered, that is, C = X:

Choose (one of) the set(s) with the maximum value of |Si \ C|; let the index of this set be i.*

Add i to Σ and update C to C ∪ Si**

<u>Execution of the greedy algorithm on example input</u>

After step 1:

S2 abdFG

S3 aFG

S4 bcG

S5 GH

S6 eH

S7 cI

S8 a

S9 e

S10 I

After step 2:

S3 afg

S4 bcg

S5 gH

S6 eH

S7 cI

S8 a

S9 e

S10 I

After step 3:

S3 afg

S4 bcg

S6 eh

S7 cI

S8 a

S9 e

S10 I

where the alphabets in small case represents elements covered and in big case uncovered elements.

The greedy algorithm results in three sets S2, S6 and S7 giving close to optimal solution.

The chief problem with efficiently implementing the greedy algorithm is that it demands picking the set with the largest number of uncovered items. As each new set is chosen, because it covers items that might be present in other sets, it is necessary to find all sets which contain the covered items, and adjust the count of uncovered items in each set accordingly. Thus, algorithms for Set Cover must retrieve information from disk or main memory (depending on the implementation) in order to calculate the sizes of sets and to determine which items are in which set and vice versa. On large problem instances, these frequent calls for data have a significant effect on the running time of the algorithm.

Correctness Test

We will describe how the efficient algorithm works on the sample input given in the input/output section. We assume the parameter p = 2. Thus, we break the sets initially into those of size 1, 2–3 and 4–7. The algorithm first considers the sets of size 4–7, and selects the first set found, ABCDE. The next set now has only 2 uncovered items (F and G), so the new set is appended to list 2, and the first step finishes. At the start of the next step, many items are now covered (indicated in lower case); however, the “uncovered sizes” of the sets are not known to the algorithm until these sets are inspected. The set FG has been written to the end of list 2: the original items ABD have been removed. In step 2, AFG has two uncovered elements and so is added to the solution. Consequently, when the algorithm passes through the list of sets supposed to be of size 2–3, it finds that there are no sets of “uncovered size” remaining in this range, due to items’ being covered. In step 3, each of the sets I and GH has one uncovered item when inspected, and so is selected. The remaining sets in list 1 have no uncovered items. Thus, the chosen cover is ABCDE, AFG, GH, I.

The new algorithm executed on the sample input

At start of step 1:

4–7 ABCDE, ABDFG

2–3 AFG, BCG, GH, EH, CI, AFG

1 A, E, I

At start of step 2:

2–3 aFG, bcG, GH, eH, cI, (abd)FG

1 a, e, I

After the third set has been selected:

2–3 bcg, gH, eH, cI, (abd)fg

1 a, e, I

At start of step 3:

1 a, e, I, (g)H, (e)H, (c)I

Final Cover: ABCDE, AFG, GH, I