Finding Optimal Solution for the Mutually Exclusive Set Cover Problem

Jingyi/Joy Xu

October 8, 2018

Abstract

In this paper, I am going to introduce a detailed algorithm for the Mutually Exclusive Set Cover problem. This is an NP-hard problem motivated by a practical ride-sharing needs in real-life. Currently, companies in ride-sharing business generally use approximation method. We want to search for optimal solution and learn the limits/constraints about exact algorithm.

1 Introduction

In recent years, a wide variety of Internet companies build upon the general idea of utilizing spare resources of people. For example, AirBnB utilize spare room space of house owners to host travelers. Uber and Lyft make use of extra car space of personal car owners to pick up riders. How to optimally utilize the free space is the main focus of my project.

I am particularly interested in ride-sharing problem. Suppose there are several riders R_1, R_2, R_3 wanting to travel at the same time. Suppose $\{(R_1, R_2), (R_3)\}$, $\{(R_1), (R_2, R_3)\}$, $\{(R_1, R_2, R_3)\}$ are the allowed assignments, subjecting to some constraints. Which assignment should we choose to give the minimum number of subsets? In the above case, $\{(R_1, R_2, R_3)\}$ gives the minimum number of subsets. This can be modeled as an optimization problem of Mutually Exclusive Set Cover Problem.

Before introducing Mutually Exclusive Set Cover problem, let me talk about what a Set Cover problem is. Given a set U of elements {1,2,...,n} (called the universe) and a collection S of m sets whose union equals the universe, the set cover problem is to identify the smallest sub-collection of S whose union equals the universe. For example, consider the universe U={1,2,3,4,5} and the collection of sets S={{1,2,3},{2,4},{3,4},{2,4,5}, {4,5} } Clearly the union of S is U. However, we can cover all of the elements with the following, smaller number of sets:{{1,2,3},{2,4,5}} or {{1,2,3}, {4,5}}.

The Mutually Exclusive Set Cover problem is the same as Set Cover problem, except with one added condition: the sets in the solution sub-collection of S cannot contain overlapping elements. In the above example of Set Cover problem, the sub-collection {{1,2,3}, {4,5}} is a solution to the Mutually Exclusive Set Cover problem, while the sub-collection {{1,2,3},{2,4,5}} is not. Obviously, there is always a solution to Set Cover problem, while it is very possible that a Mutually Exclusive Set Cover problem does not have a solution.

The focus of the paper is finding the sub-collection $\mathcal{F}'$ with minimum number of non-overlapping sets to cover all the elements in the universe U. Below is the detailed formulation of ride-sharing problem into my modified Mutually Exclusive Set Cover. You can skip the next paragraph if you want to, as it does not affect the rest of the paper.

The ride-sharing problem can be modeled this way: Treat each rider as the element in the set. Each valid grouping of riders is a set in the collection S. A group of riders is valid only if it fulfills some constraints. There are several possible candidates of constraints: it can be that they share the same destination; or it can be that the time taken for each rider to reach its destination using ride-sharing will not exceed twice the time taken by rider to go to its destination directly; and so on. Deciding the constraints is not the focus of this paper. In this paper, I assume some algorithm

has already placed valid groups of elements as sets in the collection S. The optimal solution will be a sub-collection with minimum number of non-overlapping sets that can cover all the elements.

The research on Mutually Exclusive Set Cover problem is little. The research on Set Cover problem is extensive, though most of the research on Set Cover have been focused on approximation methods. Papers like (Fei98; Sla97) aim to bring a tighter bound on approximation methods of set cover problem. (BHK09) gives an algorithm of $O(2^n)$ of finding k subsets in S with maximum weight sum that cover all elements in U. (The solution may not exist). There has been some variations of Set Cover problems, but none of them is really studying what I am interested in. For example, Disjoint Set Cover problem is interested in partitioning S into as many set covers as possible, where a set cover is defined as a collection of subsets whose union is U.(CD05). So far as I know, the closest research I believe, (LLC+15) introduces an exact algorithm for weighted mutually exclusive maximum set cover problem, in which the solution is a set of minimum number of non-overlapping subsets which covers the maximum number of elements. In that paper, the author also introduces the constraint of mutual-exclusiveness. However, the research is not about finding a set to cover all the elements. I believe (LLC+15) is currently the state-of-art algorithm on Mutually Exclusive Maximum Set Cover problem. I will covert their method onto Mutually Exclusive Set Cover problem and compare my algorithm with theirs.

In Section 2, I will present the problem space representation. In Section 3, I will give a detailed explanation of my algorithm. In Section 4, I will compare my results with (LLC+15) method with experimental results.

Should I include the proof for NP-completeness?

2 Observations and Representation Space

There are three essential questions in this research problem.
1. What is our branching factor? Should we branch on each set? For example, the input collection S is {{1,3},{2,4},{3,4} }. Choosing {1,3},{2,4} in this order is the same as choosing {2,4}, {1,3}. In order to reduce the duplicate states and branch factor, we choose inclusion-exclusion algorithm, instead of branching on number of sets.

Problem space graph: initially all the sets in the space will be drawn as dotted line.

Figure 1: Inclusion Operation

Actions:

1. Inclusion Action: Choose a set and draw the set as solid line. In the mean time, we remove all the dotted set intersecting with the solid set. (This means we include the set as part of the solution.) See Figure 1

2. Exclusion Action: Choose a set and remove the dotted line of the set from the graph. In other words, we exclude the set from the solution set. See Figure 2

Goal State: Every element is in a solid set.

End State: There exists at least one element not covered by any set in the graph.

2. Is there an easy way to determine whether Mutually Exclusive Set Cover problem has a solution? If the problem given does not have a solution, this will become the worst-case scenario as we need to exhaust the search space to find out that if there is no solution. In addition, it will not give us an tighter bound in solution size.

If an element is only covered by one set, then this set must be included in the solution. Let us call this set the Sole Set. Hence at the start of the algorithm, we can perform inclusion action for all the sole sets. More sole sets might be created during this process. We perform the inclusion action for all the sole sets until either the end stat happens, or all the elements remaining are covered by more than 1 set. This way, we would be able to determine whether the problem has a solution or not early.

Figure 2: Exclusion Operation

Since we are interested in finding the cover with minimum number of sets, we inevitably need to exhaust the search space to prove that our solution is the minimum set cover. The earlier we find a tighter bound, the more nodes we can prune. Thus I choose the branch-and-bound depth-first algorithm. Hence it results in the next question to be asked.

3. Is there an admissible heuristic that we can effectively prune the nodes? If not, is there a heuristic to see which node should be searched first so that we reach a tighter bound earlier? Observation: For example, given a set $\mathcal{F}$: $\{\{3,5\},\{1,2,4\},\{3,6\},\{4,7\},\{5,8\}\}$, which has 8 elements in total. Define S_1={3,5}, S_2={1,2,4}, S_3={3,6}, S_4={4,7}, S_5={5,8}

See figure 3, we can represent the problem as a graph. Let each set in $\mathcal{S}$ be a circle covering the elements in the set. Let us define field: a set of sets where each element in it can reach any other element by passing through intersected element. In figure 3, we can see that element 6 can link to 8 by passing through the first intersected element 3 and then second intersected element 5. In this case, $\mathcal{S}$ has two fields. Once a problem is broken into smaller fields, we just need to solve locally optimal sub problem. Hence one possible heuristic is to choose to branch on the set, which is intersected with more sets. When using depth-first branch and bound, this method might enable us to find a tight bound fast.

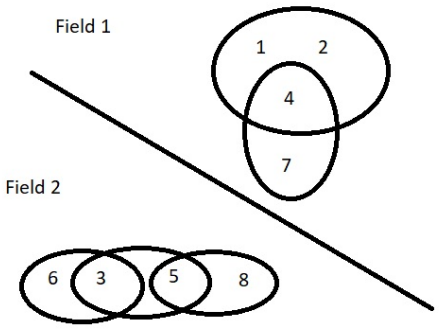

Figure 3: Graph with 2 fields

3 Algorithm

My algorithm is basically a branch and bound depth-first search. Initially set the bound as the number of elements in the set, as each set contains at least one element. Hence the |solution| $\not>$ |X|.

BBDF (X, F, bound) //X is the element set, F is the graph
Input:a collection S of sets
Output:a sub-collection with minimum number of sets that cover all the element
or an empty set if no solution
check if the graph is goal state
if goal state:
return Solution={remaining sets}
check if the graph is at end-state
if End-state:
return $\emptyset$
check if the graph has sole set
if Sole Set exists:
Perform inclusion action for all the Sole Sets
if encounters end state: return $\emptyset$
Else: Solution={Sole Sets}
bound = bound - |Solution|

union-find the solution into different fields $F_i \in F$
if the number of fields is larger than 1
for each field F_i
if $\cup BBDF(X_i, F_i, bound)$ returns $\emptyset$ or |solution|>bound, returns $\emptyset$
Else: solution= solution $\cup BBDF(X_i, F_i, cur - sol)$
return solution
else:
choose a set s with largest degree of intersection

//here is the inclusion step
add s to solution set and remove neighbor(s) from the original graph to form new F_1
if BBDF(X-element(s), F-neighbor(s), bound) does not return $\emptyset$,
$Solution_1 = \{x\} \cup BBDF(X - element(s), F - neighbor(s), bound)$
update bound=min(bound, $|Solution_1|$
else:
$Solution_1 = \emptyset$

//here is the exclusion step
remove x from the original graph to form new F_2

if BBDF(X,F-s, bound) does not returns $\emptyset$:
$Solution_2 = BBDF(X, F - x, bound)$
Else
$Solution_2 = \emptyset$

if both solutions are $\emptyset$
return $\emptyset$
Else:
return the best solution among $Solution_1$ and $Solution_2$ //best solution means the sub-collection with the minimum number of sets

The algorithm uses a top-down approach to recursively calls BBDF function. At each recursive call, BBDF checks whether the sub-colleciton F it gets from upper level is in goal state or end-state. Then it checks if Sole Sets exist for the current collection. If Sole Sets exist, include all the Sole Sets in Solution. During the process, if end state occurs, return $\emptyset$.

After the above actions, it checks whether the sub-colleciton F consists of more than 1 non-connected fields. If so, the solution would be a union of the solution set of each non-connected field. Hence it breaks down the sub-collection F into fields and pass each field to the next recursive cal of BBDF.

If the sub-collection F passed to the BBDF consists of only 1 field, BBDF chooses a set s with highest connectivity to branch. We either perform an inclusion action of set s (include s in the final solution set) or we perform an exclusion action of set s (exclude s in the final solution set). Then we compare the solution of the two actions and choose the best one to pass upward.

During the process, I also add the local bound to pass down, in an effort to prune more nodes.

4 Experimental Results

The above algorithm without bound and early detection of no-solution is the state-of-art algorithm (Inclusion-Exclusion Algorithm) on finding minimum number of mutually exclusive sets that cover maximum number of elements. So I converted the inclusion-exclusion algorithm to fit my problem. In addition to the inclusion-exclusion algorithm I added local bound components in an effort to prune more nodes, and early detection of no-solution case. So I compare the results between inclusion-exclusion algorithm and BBDF algorithm. Tables below show the total number of nodes expanded on average.

There are two cases I want to explore: First case is when number of sets in the input set is around the same magnitude as the number of elements, refer to Table 1. I have performed 10 runs of 100 test cases, where each test case has 100 input sets with varying set size between 1 to 5. The elements in the input sets are random numbers from 1 to 10. It can be shown from the table that it reduces the number of steps by 13%.

	10 Elements Case
Inclusion-Exclusion Algorithm	15045
Bounded Inclusion-Exclusion Algorithm	13037

Table 1: Total Node Expanded on Average when |input|=100, |element|=10

The Second case I want to explore is when the input size is much larger than the number of elements. I tried to run 100 elements with 1000 sets, but it takes too long to run. So I used 50 elements with 100 sets as the comparison. Since in this case, my algorithm should run better as there will be more cases that the algorithm cannot find a solution. The earlier we can determine that there is no solution possible, the less steps we need to run. Table 2 shows the experimental results of 10 runs of 100 test cases, where each test case has 100 input sets with varying set size between 1 to 5. The elements in the input sets are random numbers from 1 to 50.

	50 Element Case
Inclusion-Exclusion Algorithm	1939
Bounded Inclusion-Exclusion Algorithm	103

Table 2: Total Nodes Expanded on Average when |input|=100, |element|=50

It can be seen from the table 2 that BBDF performs much better than inclusion-exclusion algorithm especially in determining no solution cases. It reduces the number of steps by almost 95%.

5 Conclusion

I formulate this problem myself. I have searched literature and think there is no exact previous work on this problem before, as I am interested in an exact solution. Most of the work are interested in approximation methods. They either does not require covering all the elements or does not require solution sets to be pairwise disjoint. So I formulate the whole problem myself. I am most proud of finding a good graph formulation of the problem, as I have tried various ways of representing set as node, or representing element as node and various ways of representing different things as edges. I found the above representation the easiest to help people to understand the problem and spot end state or goal state right away on graph.

I tried different algorithms but in the end still find the depth-first algorithm from (LLC+15) better than mine. It was a depth-first algorithm to find the minimum number of sets that cover maximum number of elements in the set. I added a branch-and-bound in addition to its depth-first algorithm. However, we use the same heuristic.

Some future work that can be done for this problem: the heuristic I used is the largest degree of intersection of the chosen set, but ideally I want to choose a set that breaks the problem to largest number of fields. I think there might be some "connectivity" literature out there, which I think would be interesting to apply on this problem. Another thing is to relax the constraint from "covering every element" to "covering maximum number of element". The branch-and-bound might be more useful in this case as the problem will always has a solution initially. Bound will be more useful in that case. Another thing is to derive a tighter complexity analysis than the current one.

I thank Luyao Yuan, Tal Friedman and Professor Richard Korf for helpful suggestion and discussion.

References

[BHK09] Andreas Björklund, Thore Husfeldt, and Mikko Koivisto. Set partitioning via inclusion-exclusion. *SIAM Journal on Computing*, 39(2):546–563, 2009.

[CD05] Mihaela Cardei and Ding-Zhu Du. Improving wireless sensor network lifetime through power aware organization. *Wireless Networks*, 11(3):333–340, 2005.

[Fei98] Uriel Feige. A threshold of ln n for approximating set cover. *Journal of the ACM (JACM)*, 45(4):634–652, 1998.

[LLC+15] Songjian Lu, Kevin N Lu, Shi-Yuan Cheng, Bo Hu, Xiaojun Ma, Nicholas Nystrom, and Xinghua Lu. Identifying driver genomic alterations in cancers by searching minimum-weight, mutually exclusive sets. *PLoS computational biology*, 11(8):e1004257, 2015.

[Sla97] Petr Slavık. A tight analysis of the greedy algorithm for set cover. *Journal of Algorithms*, 25(2):237–254, 1997.