NP-Hard problems using Map-Reduce

A Project Report Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science in Computer Science

Supervised by
Dr. Ivona Bezáková
Dr. Edith Hemaspaandra

Department of Computer Science
B. Thomas Golisano College of Computing and Information Sciences
Rochester Institute of Technology
Rochester, New York

November 20, 2013
The project “NP-Hard Problems using Map-Reduce” has been approved by the following Examination Committee:

____________________________
Dr. Ivona Bezáková
Project Committee Advisor

____________________________
Dr. Edith Hemaspaandra
Project Committee Co-Advisor

____________________________
Dr. Xumin Liu
Project Committee Reader

____________________________
Dr. James Heliotis
Project Committee Observer
Acknowledgments

I would like to thank all my friends and professors for helping me out in every step towards the completion of the Masters degree.

The Masters project would have not been made possible without the knowledge and help from the professors. I would like to thank the committee chair, Prof. Ivona Bezáková, committee co-advisor, Prof. Edith Hemaspaandra for guiding me through the entire course of the project and study towards my Masters degree. I also would like to thank Prof. Xumin Liu and Prof. James Heliotis for reviewing and providing suggestions for improvements.

This project is also dedicated to my mother and father for their continuous support and encouragement.
Table of Contents

1. Abstract .................................................................................................................. 5
2. Introduction ............................................................................................................. 6
3. Background ............................................................................................................. 7
   3.1. Map-Reduce: .................................................................................................... 7
   3.2. Hadoop: .......................................................................................................... 8
   3.3. Max-K Cover problem: .................................................................................. 10
   3.4. Maximum Clique problem: .......................................................................... 11
   3.5. Subset Sum problem: ................................................................................... 13
4. Algorithm Design ..................................................................................................... 14
   4.1. Non-Approximation Algorithms .................................................................. 14
   4.1.1. Max K-Cover .............................................................................................. 14
   4.1.2. Maximum Clique ....................................................................................... 15
   4.1.3. Subset Sum ................................................................................................ 16
   4.2. Map-Reduce Algorithms .................................................................................. 16
   4.2.1. Max-Cover ................................................................................................ 16
   4.2.2. Maximum Clique ....................................................................................... 19
   4.2.3. Subset Sum ................................................................................................ 22
5. Implementation Details ........................................................................................... 25
   5.1. Hadoop environment setup .......................................................................... 25
   5.1.1. Single node setup ..................................................................................... 25
   5.1.2. Multi node setup ....................................................................................... 25
   5.2. Generation of the inputs .............................................................................. 25
   5.3. Output Collection ........................................................................................... 26
6. Experimental Results .............................................................................................. 27
   6.1. Preparation of the testing environment ....................................................... 27
   6.2. Output Collected ........................................................................................... 28
7. Conclusion and Future work .................................................................................. 32
8. Bibliography ............................................................................................................ 33
List of Figures

Figure 1: Map-Reduce Configuration................................................................. 8
Figure 2: Second Possible Configuration for Map-Reduce................................. 8
Figure 3: Hadoop Architecture........................................................................ 9
Figure 4: Max K-Cover Example.................................................................... 10
Figure 5: Maximum Clique Example................................................................. 12
Figure 6: Details of the Computations for Max Cover Map-Reduce.................... 18
Figure 7: Max Cover of the Computations for Maximum Clique Map-Reduce .... 21
Figure 8: Details of the Computations for Subset Sum Map-Reduce.................... 23
Figure 9: Graphs of the Maximum Clique Output Data .................................... 28
Figure 10: Graphs of the Max K-Cover Output Data ........................................ 29
Figure 11: Graphs of the Subset Sum Output Data ........................................... 30
1. Abstract

NP-hard (non-deterministic polynomial-time hard) problems is a class of problems that are at least as hard as the hardest problems in NP.

This project focuses on three NP-hard problems, Max-K Cover problem, Maximum Clique problem and Subset Sum problem. The Max-K Cover problem is to select K sets from a collection of sets in order to maximize the size of the union. Currently, a favorite approach to solve Max-K Cover problem is a greedy algorithm, which selects the largest set and removes its elements from the other sets. It then selects the next largest set and continues the same K times to get the maximum union of K sets. The Max-Clique is the problem of selecting the largest sub graph of a given graph, wherein the sub-graph every vertex is adjacent to every other vertex. The more common algorithm to solve the Max-Clique problem is the brute-force serial algorithm. The Subset Sum problem states that one needs to find the subset of a given set of numbers, where the sum of the numbers in the subset is equal to K. One of the common algorithms to solve the Subset Sum problem, is the brute-force serial algorithm.

Map-Reduce framework is a novel idea by Google to solve problems that involve large computations. Map-Reduce consists of two distinct parts: mappers and reducers. The main problem is divided into mapper tasks and reducer tasks and then given to the Map-Reduce framework for the computation. The parallel execution of the tasks is performed by the mappers and the intermediate outputs from the mappers are fed to the reducers. The reducers merge the result and subsequently, give out the final output.

In this project, I investigated the above mentioned NP-Hard problems and the applicability of the Map-Reduce framework for those problems. I have coded the Map-Reduce and the sequential algorithms for all the NP-Hard problems. I experimented with the algorithms for different input parameters and evaluated the results. The comparison between the Map-Reduce and the sequential algorithms has been attempted to determine if the Map-Reduce framework makes the solvability of the NP-Hard problems better than the conventional method.
2. Introduction

The project consists of two main parts; first we need to study the Map-Reduce framework and its architecture, and second, we need to study the NP-Hard problems and how we can improve the solvability of the NP-Hard problems using the Map-Reduce framework.

NP-Hard problems need lots of computation time and power for execution. Map-Reduce framework [2] has the capability to perform heavy tasks by dividing the execution tasks into smaller sections called map and reduce. The map and reduce both perform their computations in parallel. With Map-Reduce, the overall operations will not become polynomial (at least not when solving the problems exactly) but hope comes from some speedup due to parallelism.

We plan to implement an Iterative task as a Map-Reduce task since Map-Reduce generally involves two steps: Map and Reduce. The inputs are passed to the map function and the intermediate output from maps are passed to the reduce function. The reduce function outputs the final result. The Map-Reduce technique and Hadoop are explained in more detail in the subsequent sections.

The project focuses on the three NP-Hard problems and study the applicability of the Map-Reduce framework. The three problems are Max-K Cover, Maximum Clique and the Subset Sum problem. A Map-Reduce Max-K Cover algorithm has been discussed in the paper by Chierichetti et al. [1] and a Map-Reduce Maximum Clique algorithm is being proposed in a paper by Peng et al. [4]. The Map-Reduce algorithm for the Subset Sum problem has been coded using the information gathered from the two mentioned papers. The other papers, which discusses the ideas related to Map-Reduce algorithm has also been studied. An interesting paper by Cohen [9], discusses different approaches to solve a variety of graph problems in the Map-Reduce pattern.

I have coded the brute-force algorithms for the above NP-Hard problems and implemented them on the Hadoop cluster. Experiments have been conducted to analyze the performance and evaluate whether Map-Reduce improves the time taken to compute the final results for the NP-Hard problems.
3. Background

3.1. Map-Reduce:

Map-Reduce framework is one of the most powerful computation models that has been very helpful in performing a large-scale operation on the huge data. It provides parallelism over the map nodes and helps to improve the performance of the algorithm used. It enables to use multiple processors efficiently by performing the divided tasks on each of them.

Many organizations use the Map-Reduce framework to perform either data-intensive operation or processor-intensive operation. For example: Google uses Map-Reduce to perform the web crawling operation in order to index the results in the database [2].

Map-Reduce works in the master-slave fashion, where the main master decides the number of maps needed according to the inputs and allocates them to the individual map accordingly. Map-Reduce is very efficient in handling failures since the master has the ability to restart a slave operation if it fails.

The parallelism of the map-reduce task is done by splitting the input in the form key-value pairs, among the map nodes. The map nodes perform the operations and gives out the set of intermediate key-value tuples. The reduce nodes consume all the intermediate outputs of the map nodes and merges them to perform the final operation. The reduce node then gives out the final result.

Various combinations of map and reduce nodes can realize the Map-Reduce job. We can have multiple map nodes and multiple reduce nodes, or multiple maps and single reduce, or only map nodes for the computations [5].

We are interested in the two different combination of the Map-Reduce nodes as shown in the figure 1 and 2, since they are applicable to the three NP-Hard problems under investigation as part of the project.

One of the combinations of the Map-Reduce framework is explained in the figure 1 [5]:

---

7
In the figure 1, we can see that the inputs to the Map-Reduce framework, are divided into fragments and they are being fed into multiple map nodes. Map nodes perform the map function on the inputs and the results are sorted and merged to make new inputs for the Reduce node. The Reduce nodes perform the final computation and give out the final output.

In the figure 2, we have the one map and one reduce operation, which are connected as a loop. This arrangement is for computing the partial results and using multiple partial results to compute the final result. There can be multiple iterations of Map-Reduce operations to get the final result.

### 3.2. Hadoop:

Apache Hadoop is an open-source software framework that is used for developing distributed applications involving a lot of computations. It can run the applications on large clusters. Hadoop is developed in Java and has libraries for Map-
Reduce. One can perform the map and reduce operations using Hadoop by implementing the built in programming interfaces provided.

Hadoop [5] provides these Java classes: mapper, reducer and the main program. One can extend the map and reduce interfaces to develop the custom map and reduce jobs according to the requirements of the solution. We have the ability to specify the custom input data types and the custom intermediate data types, which will be used for data transfer between map nodes and the reduce nodes. There is also a facility to modify the default input reader and output writer.

![Hadoop Architecture](image)

**Figure 3: Hadoop Architecture**

Hadoop provides a distributed file system, known as HDFS, that stores data on the nodes in the cluster. The map, reduce and the distributed file system are designed by Hadoop such that failures are automatically handled by the framework. The Hadoop Architecture is shown in the figure 3 [5].

We can have a Hadoop cluster with a single master and multiple worker nodes. The master node will have a JobTracker, TaskTracker, NameNode and DataNode. A worker node has both a DataNode and TaskTracker. JobTracker’s main job is to track
the progress of individual map or reduce jobs using the data obtain from the TaskTracker. TaskTracker tracks the process of the tasks within the current node.

For the HDFS layer, we have the DataNode which will consist of the data needed for the computation in that particular node. The data management is done by the NameNode. NameNode always stores the initial input file and the final result. The standard communication between the nodes are done using ssh protocol and Java RMI.

3.3. Max-K Cover problem:

The formal definition of the Max-K Cover problem is:

“Given an integer \( k \geq 0 \) & a collection of sets \( S=\{S_1, S_2, .. S_n\} \), \( S^* \subseteq S \) is the Max-K Cover if \( |S^*| = k \) & the union of sets in \( S^* \) is maximized over all subsets of \( S \) of size \( k \).”

The Max-K Cover problem can be explained by the following example. Suppose we have multiple sets (in this case 5 sets) in the problem under consideration as shown in the figure 4:

![Figure 1: Max K-Cover Example](image)

In the figure 4, the dots represent the elements and the selected regions represent the sets. We can define the Max-K Cover problem as, if \( k=3 \), find a collection of \( k \) sets, which will have the largest union. From Figure 4, we can see that sets marked as S1, S2 and S3 are the best selected sets.

As an application of the Max-K Cover problem, consider the following scenario: A company wants to advertise a particular product and they have possible
placements for the ad. The i-th placement gets viewed by a set $S_i$ of people. The company can afford to buy $K$ advertisements. Which locations should the company choose to maximize the number of the people who view it? This particular problem can be solved by Max-K Cover algorithm.

The most common algorithms for Max-K Cover is the greedy approximation algorithm and the exact brute force algorithm. The greedy algorithm involves the simple steps of selecting the largest set and subsequently selecting the second largest set from the list of sets with the largest set and its elements removed. The greedy algorithm does not always give the best results. For example, in the figure 4, for $k=3$, if we take the largest set first we are left with three sets of size 3 each. Hence, with the greedy algorithm, we select the 4 sets, i.e. $S_4$, $S_5$, $S_6$ and $S_7$, we don’t get the optimal solution, which is the three sets $S_1$, $S_2$ and $S_3$.

The brute force algorithm takes all possible combinations of $K$ sets from the list of sets and checks which combination has the largest union. The brute force algorithm is an exact algorithm. The running time of the algorithm is $O(2^n)$. I implemented the brute force Max-K Cover algorithm.

3.4. Maximum Clique problem:

For a given graph, a Clique is a special sub graph, where every vertex is adjacent to every other vertex. The Maximum Clique problem is, given a (possibly huge) graph, we need to find the maximum sub graph possible such that the sub graph is a clique. The problem is known to be NP-Hard [7].

Lin Peng, Wang Zebing and Guo Ming [4] have shown that for any sequential problem, it is always possible to use Map-Reduce by not utilizing the parallelism. They have designed the Max-Clique Map-Reduce based algorithm and compared the performance with the single machine serial algorithm with pruning. They explained that the Map-Reduce framework does not provide any mechanism for having a global variable and it is needed to pass the data between map and reduce nodes with the help of files. This reduces the performance of the Map-Reduce based algorithm.
A clear example, where the Maximum Clique can be applied, is finding a closed group of friends in a social network. The closed group of friends is the group, where everyone knows each other.

I have explained the common algorithm for the Maximum Clique problem using the figure 5:

Suppose we have the graph with 5 vertices as shown in the figure 5. The Maximum Clique algorithm involves a number of steps. First, with every edge we check the possibility of clique size 3 with all the other vertices of the graph. If we choose edge (1, 2), we check if (1, 2, 3), (1, 2, 4), (1, 2, 5) can form a clique. From the list we can see that with edge (1, 2), we get two cliques (1, 2, 3) and (1, 2, 4). We continue this step till we get all the cliques of size 3. In the next step, the algorithm tries to include a vertex from the graph to the clique of size 3 to form the clique of size 4. We continue with these steps till we have reached the Maximum Clique. In the figure 5, we have the Maximum Clique as (1, 2, 3, 4).

Pseudocode:
MaxClique (List vertices[]):

    currentCliqueList = List of vertices E in G
    while(currentCliqueList.size() != 0):
        newCliqueList = null
        for (i from 0 to currentCliqueList.size()):
            for(vertex V in graph G where V ∉ currentCliqueList[i]) :
                if(isClique(V U currentCliqueList[i])):
newCliqueList.add(V U currentCliqueList[i]) (no duplicates)
currentCliqueList = newCliqueList

Solution: currentCliqueList.

Peng et al. [4] designed and implemented the brute force pruning algorithm for Max-Clique. I have implemented the brute force Maximum Clique algorithm, different than Peng et al. [4], where I did not use the pruning since we cannot share the same global graph over multiple maps. I also coded the Map-Reduce based algorithm and made sure that they work with Hadoop.

3.5. Subset Sum problem:

The Subset Sum problem is defined as a pair (S, L), where \( S = \{x_1, x_2, ..., x_n\} \) is a set of positive integers and \( K \) is a positive integer. The decision problem asks whether there is a subset of \( S \) with its elements summing to \( K \). This problem is NP-Hard.[7]

For the given set of integers, is there a non-empty subset whose sum is 12? For example, given the set \{7, 3, 2, 5, 8\}, the answer is yes because the subset \{7, 5\} sums to 12. We can also have the subset \{7, 3, 2\} where the sum is also equal to 12.

The naive approach of finding the subsets is by computing the sum of the elements of every possible subset of \( S \) of size \( K \) and then selecting the sets whose sum is equal to \( K \). This algorithm is a brute force type and it requires exponential time.

The better algorithm for the Subset Sum problem is the dynamic programming algorithm [6], but there is a need for the same data structure has to be shared by multiple maps. Due to Map-Reduce design, we cannot have the data shared among the maps. Therefore, I chose not to use the dynamic algorithm for my evaluation. The brute force algorithm has been designed with Map-Reduce pattern and implemented on the cluster for evaluation.
4. Algorithm Design

Currently, we have non-approximation algorithms available for the Max-K Cover, the Maximum Clique and the Subset Sum NP-Hard problems. All of these algorithms are exponential in nature. We need to convert these algorithms into two main parts: map and reduce. This Map-Reduce algorithm can be executed with the help of Hadoop.

We have implemented the algorithms for the three above NP-Hard problems. For uniformity reasons, we have chosen the non-approximate exact algorithms that always produce the correct results. The algorithms are explained in detail as follows:

4.1. Non-Approximation Algorithms
   4.1.1. Max K-Cover

We have designed the brute force algorithm, where we first create the list of all possible combinations of K sets from the list of sets. We find the largest cover for K sets by comparing the size of unions of the combinations in the list.

Algorithm:
1. Inputs the list of sets \(S_1, S_2, \ldots, S_n\) and K
2. \(\text{maxUnionCombination} = \text{null}\)
3. for \((x \text{ from } 1 \text{ to } 2^n)\):
   a. \(\text{binaryX} = x \text{ in binary number format}\)
   b. \(\text{setCombination} = \text{null}\)
   c. if \((\text{number of ‘1’ in binaryX} == K)\):
      i. for \((i \text{ from } 0 \text{ to } \text{binary.length}-1)\):
         1. if \((\text{binary}[i] == ‘1’)\)
            a. \(\text{setCombination.add(setList}[i]\))
         ii. if \((\text{union(maxUnionCombination)} < \text{union(setCombination)})\) :
            1. \(\text{maxUnionCombination} = \text{setCombination}\)
4. Solution: maxUnionCombination
4.1.2. Maximum Clique

We have designed the brute force algorithm for Maximum Clique, where we check all the possible sub-graphs of the input graph and determine the Maximum Clique. First, we find out all the cliques of size 3 in the graph. Then, we proceed to find clique of size 4 using the previous cliques found and the graph. The algorithm repeats the above steps repeated till we reach the maximum possible clique.

Algorithm:

Method clique(List vertices):
    for (every vertex V1 in the vertices):
        for (every vertex V2 in vertices where V1 != V2):
            if (not exist edge(V1,V2) in vertices):
                return false;
        return true;

1. Inputs the list of vertices of graph G
2. currentCliqueList = List of vertices E in G
3. while (currentCliqueList.size() != 0):
    a. newCliqueList = null
    b. for (i from 0 to currentCliqueList.size()):
        i. for (vertex V in graph G where V $\notin$ currentCliqueList[i]):
            1. if (clique(V U currentCliqueList[i])):
                a. newCliqueList.add(V U currentCliqueList[i]) (no duplicates)
        c. currentCliqueList = newCliqueList
4.1.3. **Subset Sum**

We have designed the brute force Subset Sum algorithm. In this algorithm, we create all possible combinations of elements from the list of elements in the set. We then check the sum of combinations and output the combinations where the sum of the elements is equal to K.

**Algorithm:**
1. Inputs the list of sets S and K
2. possibleCombination = null
3. n = S.size()
4. for (x from 1 to 2^n):
   a. binaryX = x in binary number format
   b. sumSet = 0
   c. currentCombination = null
   d. for (i from 0 to binary.length-1):
      i. if(binary[i] == ‘1’)
         1. sumSet +=S[i]
         2. curentCombination.add(S[i])
   e. if(sumSet == K)
      i. possibleCombination.add(currentCombination)
5. Solution: possibleCombination.

4.2. **Map-Reduce Algorithms**

4.2.1. **Max-Cover**

The Map-Reduce algorithm for the Max-K Cover consists of the two distinct parts: map method and reduce method. The map method takes the input set from the file and finds the combinations of the elements of the sets. We have parallelized the operations to find all the possible combinations by breaking the $2^n$ loop into the smaller loops according to the number of maps.
The input and output data types for the Map-Reduce algorithms are Key: LongWritable and Value: Text

The value $2^n$ is broken down into parts by the following steps: Suppose the current map is the $2^{nd}$ map from the list of the 8 maps. Therefore, $i = 2, N_m = 8$. The range for the current map will be: $(((2^n)/N_m) * (i-1)) + 1$ to $((2^n)/N_m) * (i))$

**Map Algorithm:**
1. Inputs the list of sets $S_1, S_2, \ldots S_n$ and $K$ via File input
2. $maxUnionCombination = null$
3. $N_m = \text{num of maps}$
4. $i = \text{map’s number}$
5. for ($x$ from $(((2^n)/N_m) * (i-1)) + 1$) to $((2^n)/N_m) * (i))$:
   a. $\text{binaryX} = x$ in binary number format
   b. $\text{setCombination} = null$
   c. if(number of ‘1’ in binaryX == $K$):
      i. for ($i$ from 0 to binary.length-1):
         1. if(binary[i] == ‘1’)
            a. $\text{setCombination.add(setList[i])}$
      ii. if(union($maxUnionCombination$) < union($setCombination$)):
         1. $maxUnionCombination = setCombination$
6. Solution: $maxUnionCombination$

The Reduce method takes the $maxUnionCombination$ for each map and checks which of the $maxUnionCombination$ is having the largest union of elements, thereby finding the Max-K Cover result.

**Reduce Algorithm:**
1. Inputs the list of results $R$ from maps via File input
2. $maxUnionCombination = null$
3. for (every r in R):
   
   a. if (\text{union} (\text{maxUnionCombination}) < \text{union}(r)) :
      
      i. maxUnionCombination = r

4. Solution: maxUnionCombination

![Figure 3: Details of the Computations for MaxCover Map-Reduce](image)

In the figure 6, we see the simple example of 4 sets and K=2 as an input to the Map-Reduce framework. As per the Map-Reduce Max-K Cover algorithm, we split $2^n$ operations among the number of maps. The parallelism has been realized here since the range of numbers is split among the map nodes and the operation is done on them simultaneously.

Here since we have 4 maps, we can see the range of numbers in binary form for the respective maps. First map gets value 0 to 3 which is 0000 to 0011. With these numbers, we have the largest possible union of sets to be (S3, S4), where 1 in a binary number is a set selected. Similarly, all other maps performs the operations and gives out the largest cover (i.e. union) of the sets for their range of numbers.

Reduce operation gets all the intermediate outputs of the maps and merge them. It checks the largest possible union with the given set combinations and gives out the final output. Here, we have the final output as (S2, S4), which has a largest union of (1, 2, 3, 4).
4.2.2. Maximum Clique

We have designed the brute force algorithm, where we check all the possible subgraphs to find the Maximum Clique. The algorithm has multiple iterations of Map-Reduce till the Maximum Clique is achieved. In the first iteration we have First Map method and the Reduce method which reads from the input files and output to the next iteration. For the rest of the iteration after first, we use Split Map and the Reduce method till we reach the Maximum Clique.

The input and output data types for all the iteration is Key: LongWriteable and Value: Text. We have the custom data type of VerticesSet for the intermediate data between maps and reduce.

Method clique(List vertices):

for(every vertex V1 in the vertices) :
    for(every vertex V2 in vertices where V1 != V2) :
        if(not exist edge(V1,V2) in vertices):
            return false;

return true;

The First Map algorithm takes the inputs and performs the clique search on the input file. It gives out the intermediate output which consists of the original graph and the cliques found so far.

First Map Algorithm:

1. Inputs the list of edges of graph G
2. currentCliqueList = List of vertices E in G
3. newCliqueList = currentCliqueList
4. while(newCliqueList.size() != 0):
   a. newCliqueList = null
   b. for (i from 0 to currentCliqueList.size()):
i. for(vertex V in graph G where currentCliqueList[i]) :
   1. if(clique(V U currentCliqueList[i])):
      a. newCliqueList.add(V U currentCliqueList[i])
      c. currentCliqueList = newCliqueList

5. Solution: currentCliqueList.

Split Map algorithms perform the clique search algorithms on intermediate outputs and it is executed again and again till the max-clique is found. For example, it first finds cliques of size 3 and then the next iteration it finds clique of size 4 and so on.

**Split Map Algorithm:**
1. Inputs the list of edges of graph G, Cliques found so far
2. currentCliqueList = cliques found so far
3. newCliqueList = currentCliqueList
4. while(newCliqueList.size() != 0):
   a. newCliqueList = null
   b. for (i from 0 to currentCliqueList.size()):
      i. for(vertex V in graph G where currentCliqueList[i]) :
         1. if(clique(V U currentCliqueList[i])):
            a. newCliqueList.add(V U currentCliqueList[i])
      c. currentCliqueList = newCliqueList
5. Solution: currentCliqueList.

**Reduce Algorithm:**
1. Inputs the list of cliques found in different maps.
2. Find the clique of max size:
   a. If one max-clique:
      i. Solution: max-clique
b. Else If two or more max-cliques:
   i. Solution: One of the max-cliques

![Figure 4: Details of the Computations for Maximum Clique Map-Reduce](image)

In the figure 7, the input to the Map-Reduce framework is the adjacency list of the edges of the graph. The logic of the map is divided among the First map and the Split maps. First maps read the input and gives out the same input along with the next possible clique size. Here, we have 6 edges of the graph, First map computes the clique of size 3. The output of the First map is the clique of size 3 and the 6 edges of the graph. This output is read by the Split maps. Split maps perform the operation of finding the next largest clique. The output of split maps is fed to the next split map in the form of a loop. The loop terminates once the largest clique is found. In the figure 7, the Split map found clique of size 4 and it outputs a clique of size 4 and cliques of size 3 (i.e. input).

Here, we have used the idea explained in figure 2, where we have the partial results given back to the map nodes. This design does not give any parallelism. In the graph algorithms we need to have the original graph shared between the map nodes. Due to the architecture of Map-Reduce, we cannot have a global memory. In theory, this makes the performance more or less the same as sequential algorithm.

Reduce operation reads all the intermediate inputs and determines the largest possible clique. In case, there are two largest possible cliques, the reduce operation outputs one of the largest clique.
4.2.3. Subset Sum

The Map-Reduce algorithm for the Subset Sum problem consists of the two distinct parts: map method and reduce method. The map method takes the input set from the file and finds the combinations of the elements of the sets. We have parallelized the operations to find all the possible combinations by breaking the $2^n$ loop into the smaller loops according to the number of maps.

The input and output data types for the Map-Reduce algorithms are Key: LongWriteable and Value: Text

The value $2^n$ is broken down into parts by the following steps: Suppose the current map is the 2nd map from the list of the 8 maps. Therefore, $i = 2, N_m = 8$. The range for the current map will be: $(((2^n)/N_m) * (i-1)) + 1$ to $((2^n)/N_m) * (i))$

**Map Algorithm:**
1. Inputs the list of sets $S$ and $K$
2. $possibleCombination = null$
3. $n = S.size()$
4. for ($x$ from $(((2^n)/N_m) * (i-1)) + 1$ to $((2^n)/N_m) * i)$:
   a. $binaryX = x$ in binary number format
   b. $sumSet = 0$
   c. $currentCombination = null$
   d. for ($i$ from 0 to $binary.length-1$):
      i. if($binary[i] == ‘1’$)
         1. $sumSet += S[i]$
         2. $currentCombination.add(S[i])$
   e. if($sumSet == K$)
      i. $possibleCombination.add(currentCombination)$
5. Solution: $possibleCombination$. 
The reduce method takes the possibleCombination for each map and outputs the list of the possibleCombination, thus finding the resulting subsets with a sum equal to K.

**Reduce Algorithm:**

1. Inputs the list of results R from maps via File input
2. finalListOfSubsets = null
3. for (every r in R):
   a. finalListOfSubsets.add(r)
4. Solution: finalListOfSubsets

![Diagram](image)

**Figure 5: Details of the Computations for Subset Sum Map-Reduce**

In the figure 8, we see that the logic of Map-Reduce operation is similar to what we have seen in the Map-Reduce Max-K Cover algorithm. Here, the input is the set and K. The \(2^n\) operations are split among the maps. The parallelism has been realized, here in the similar way as in Max-K Cover algorithm, since the range of numbers is split among the map nodes and the operation is done on them simultaneously.

The maps outputs the possible subsets, where the sum is equal to K. We can see that only two maps have given the result in the example shown in the figure 8. The Reduce operation merges all the intermediate results of the maps and gives out the possible subsets as the final result.
Here, the algorithm is analogous to the Max K-Cover algorithm. Here the parallelism is straightforward and it expects to have the speedup of a factor of N. When coding, I felt that Map-Reduce is not necessarily needed here since we can perform with same speedup with a simple parallel program.
5. Implementation Details

5.1. Hadoop environment setup

5.1.1. Single node setup

The guided documentation from Apache Hadoop website [11] have been used for the setup of the single Map-Reduce node. We first modified the default configuration files named mapred-site.xml, core-site.xml and hdfs-site.xml in order to enable the creation of the namenode and the task tracker. Once the Hadoop namenode and the job tracker is successfully started, we have the two web pages, where we can see the status of the jobs and the data in the namenode on the web browser.

HDFS (Hadoop Distributed File System) was configured using the built in commands and the input files were pushed into it. We had to set up the passwordless ssh so that the communication between namenode and the job trackers is not blocked by the password prompts.

5.1.2. Multi node setup

For the multi-node Hadoop, we have used AWS (Amazon Web Services) Cloud. We have used EMR (Elastic Map-Reduce) jobs as the Hadoop cluster setup jobs. For the experimental purposes, I have taken 3 machines. One of the machine will act as the master and the other two will be slaves.

As per the documentation on the Apache Hadoop website [11], I have configured the namenode and the job tracker. I have set up the passwordless ssh between machines for job tracker to communicate with the task trackers.

5.2. Generation of the inputs

I generated the inputs of different sizes for the Max-K Cover, Maximum Clique problem and the Subset Sum problem. The inputs to the Maximum Clique algorithm was collected from the website created by Dharwadker [10].
I implemented a bash and python scripts for generation of the inputs and modify them to be compatible for Map-Reduce and the sequential accordingly. The inputs will be different sets and K values for Max-K Cover and different graphs for Maximum Clique. For the Subset Sum problem, I have taken different sets and different values of K. I also used Pseudo random number generator from Computer Science Course Library by Prof. Alan Kaminsky[12], to make the random input data for the algorithms.

5.3. Output Collection

I have collected output result in the form of files from all the algorithms and calculated the time taken for every execution. The time was compared between Map-Reduce and sequential algorithm in the form of the graph in order to analyze the results. I have made sequential algorithm, Map-Reduce Single node and multi-node Map-Reduce graphs. I combined all graphs together to make comparison easier. multi-node Map-Reduce data were obtained from Amazon Elastic Map-Reduce output directory in the Amazon Simple Storage Service (S3).
6. Experimental Results

The purpose of this experiment was primarily to evaluate the performance of the serial brute force and the Map-Reduce based algorithms of the three NP-Hard problems. Therefore, my inputs were different graphs for Max Clique algorithms, different sets of for Subset Sum algorithms and different sets for Max K-Cover algorithms. Experiments were carried out on both technique implementations with respect to the size and type of inputs.

6.1. Preparation of the testing environment

I have used the following machine for sequential brute force algorithms and the single node Hadoop run.

Processor: Intel core™ i3-2350M CPU @ 2.30GHz 2.30GHz
Memory: 4GB RAM
Hard Drive: 500 GB
Operating system: Linux Ubuntu 3.11.0.12 “Saucy Salamander”
Java Version: 1.6.0_45
Hadoop Version: 1.0.3

I did the brute force sequential algorithm runs on the machine making sure that there is no other memory consuming or processor consuming process running on it. I calculate the optimal running time by calculating the start and end time. I did a similar thing for a single node Hadoop run on the same machine.

For multi-node Hadoop on Amazon Cloud. Amazon Elastic Map-Reduce instance details are: (3 instances were used)

AMI Version: 2.4.2 (Amazon Linux Image for Amazon Cloud)
Instances: 3 m1.small
(Processor: Intel Xeon 1vCPU 1 ECU, Memory: 1.7 GB, HDD: 160 GB)
Hadoop Version: 1.0.3
Java Version: 1.6.0_45
For Elastic Map-Reduce of Amazon Cloud, I created the job flow with the three instances (i.e. hosts) and used the job run time provided by the Amazon console. I verified the outputs and made sure they are correct.

6.2. Output Collected

I collected the time for all three NP-Hard problems and made the graphs as shown below: (I have used Computer Science Course Library by Prof. Alan Kaminsky)[12]

**Maximum Clique:**

![Figure 9: Graphs of the Maximum Clique Output Data](image)
Looking at the graphs in the Figure 9, I can see that the sequential algorithm gives the exponential type of curve. We can see that the Map-Reduce algorithms for single node takes more time than the sequential one for the small inputs but as the input increases the performance becomes better for single node Map-Reduce algorithm. The similar behavior is seen in the multi-node Map-Reduce Data.

**Max K-Cover:**

![Graphs of the Max K-Cover Output Data](image)

From the graphs in the Figure 10, I can see that the sequential algorithm gives the exponential type of curve. The Map-Reduce algorithm for single node takes more
time than the sequential one for the small inputs but as the input increases the performance improves for single node Map-Reduce. The similar behavior is seen in the multi-node Map-Reduce.

**Subset Sum:**

![Subset Sum Graphs](image)

**Figure 11: Graphs of the Subset Sum Output Data**

From the graphs in Figure 11, All the graphs give the exponential type of curve. We can see that the Map-Reduce algorithms for single node takes more time than the sequential one for the small inputs but as the input increases the performance
becomes better for Map-Reduce algorithms. The similar behavior is seen in the multi-node Map-Reduce output data.

Hadoop takes some fraction of time to initiate the maps and reduce and also takes some time to transfer the data between maps and reduce nodes. Most of the data transfer happens using files. This causes lots of I/O operations and hence it takes more time than the sequential algorithms. But, if we increase the inputs, due to parallelism we can get a good speed up.

For multiple node Hadoop (i.e. multiple hosts, where each has one node of Hadoop), we have noticed a loss of time in the network communication between master and slave nodes. This makes the performance degrade further as compared to single node Map-Reduce. But, if we increase the size of the inputs, I am assuming that we would be able to see better performance.

For Maximum Clique algorithm, due to the Map-Reduce structure of no global variable, we were forced to transfer the graph data between map and reduce nodes. This makes parallelism not possible and hence the performance does not come out to be better for Map-Reduce algorithms. This is the case where the Map-Reduce approach is not suitable and the performance can be increased using other parallelism frameworks like Parallel Java[13], where the global variables can be shared.
7. Conclusion and Future work.

The results show that the Map-Reduce algorithms gives better performance than the brute force algorithms of Max K-Cover, Subset Sum problems and the Max-Clique problem. Since there is no facility to have the data shared globally among the nodes of the Map-Reduce framework, we can see the performance of Max-Clique Map-Reduce algorithm and Max K-Cover Map-Reduce algorithm less than the theoretical expected.

For multi-host Map-Reduce, we did not get the expected speedup of factor N since there has been time spent in the network data transfer and the initiation of the nodes in the run. Also, I have noticed that due to Master-Slave architecture, the Hadoop spawns another map node if one node fails or does not provide results due to timeout of the network.

I also felt that Map-Reduce is not a good framework for Max-Clique algorithm since we could achieve good parallelism using the other parallel framework where we can have the data shared among the threads or processes. In future, we should investigate other parallel framework and compare them with Map-Reduce framework.

In future, if Google or Apache comes up with the way to share the data globally, we should run the algorithms again and I am sure that we would be able to see a better performance for all the three NP-Hard problems. We would also be able to come up with a better algorithm since we would be able to read data from other nodes and perform backtracking. We need to test for other NP-Hard problems whether Map-Reduce can be a good parallel framework for better performance. It's not possible to generalize the result with these three algorithms.
8. Bibliography


[9] Nathan Bell. 2012. Graph problems and Map-Reduce, Carleton University, Ottawa, Canada. Masters project. https://sites.google.com/site/nathan5704project/


