Attachments

MATDISC

Optimal Partitioning of Stores in Network Management

1.1 Introduction

As a key feature of the system created, the network manager can effortlessly perform an optimal partitioning of a list containing any number of stores. By simply providing a file with the necessary store details, the program empowers the manager to analyze and process the data, automatically determining the most balanced and efficient distribution of stores into two sub lists. This algorithm calculates and identifies the partitioning scheme that minimizes the difference between the two sub lists, ensuring a nearly equal distribution of resources.

With this feature, network managers can now achieve optimal resource allocation without the need for manual calculations or guesswork. By leveraging the system's advanced capabilities, they can effectively streamline operations, enhance network performance, and promote equitable resource utilization across the store network. This automated partitioning process not only saves valuable time and effort but also guarantees the best possible

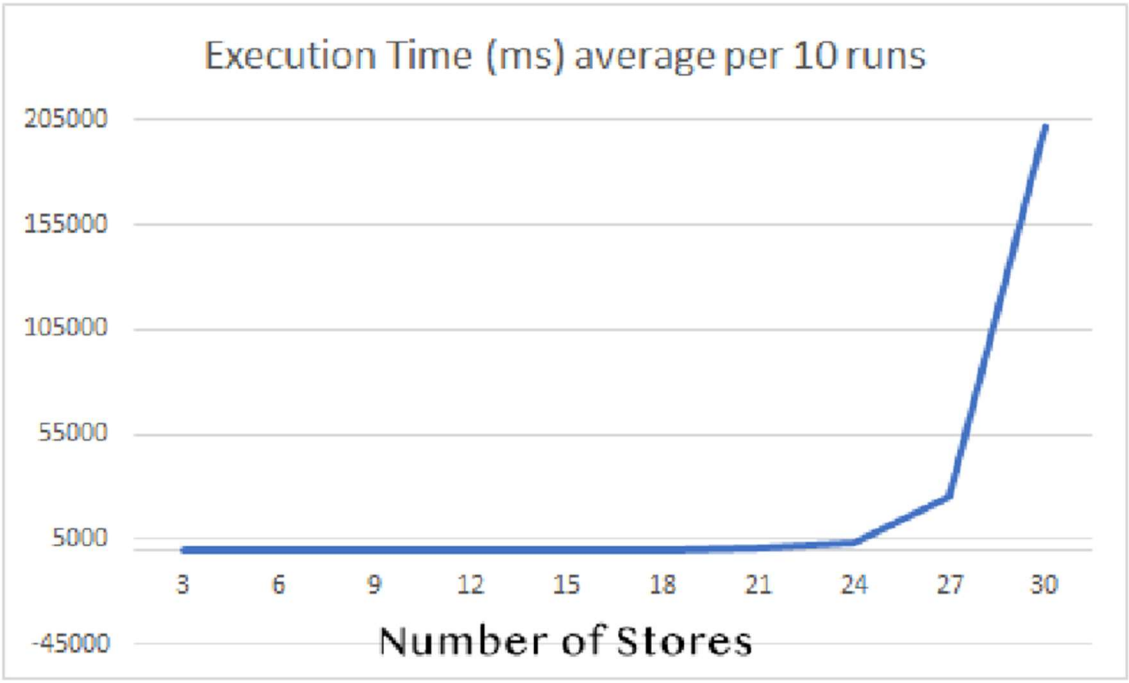
allocation strategy, maximizing efficiency and minimizing disparities between store subsets.

Pseudocode of the implemented Brute Force Algorithm:

```
Procedure minPartition(storeCount,numStores)
        for i := 1 to totalCombinations do
                if count of set bits in i is equal to numPartitions then
                        partition := empty list
                        count := 0
                end if

                for j := 0 to numStores do
                        if bit at position j in i is 1 then
                                add j to partition
                                Increment count by 1
                        end if
                end for

          if count is equal to numPartitions then
                sum1 := 0
                 sum2 := 0
                sublist1 := empty list
                sublist2 := empty list

                for storeId := 0 to numStores do
                        propertiesCount := get storeCount at index storeId
                                if storeId is in partition then
                                        add propertiesCount to sublist1
                                        increment sum1 by propertiesCount
                                else
                                         add propertiesCount to sublist2
                                        increment sum2 by propertiesCount
                                end if
                    end for

        difference := absolute value of (sum1 - sum2)

        if difference is less than minDifference then
           minDifference := difference
           minPartition := copy of partition
        end if
end for
```

1.2 Runtime tests for inputs of varying sizes

In this test the algorithm was tested with n lists with different store sizes, to evaluate how the run time evolves overtime.
The runtimes generated are totally dependent on the processing power of the CPU and so in other machines runtime may vary.
The inputs (numStores) in this test were 3,6,9,12,15,18,21,24,27,30.
This will show how much the size of the input is relevant to the execution time of the algorithm.



The graph indicates that there is a point where the execution time increases exponentially as the number of stores surpasses a certain threshold. This suggests that there may be an inefficiency in the system when dealing with many stores. It could be an indication of limitations in the underlying infrastructure or in the algorithm used for processing tasks with increasing store counts. Given that the implemented algorithm

was a brute force approach, it is expected that the execution time increases significantly as the number of stores grows. Brute force algorithms typically involve exhaustive search or computation of all possible combinations, which can result in exponential time complexity.

1.3 Worst-case time complexity analysis

The worst-case complexity of an algorithm refers to the maximum number of resources such as execution time or memory that the algorithm requires when given an input of a certain size most denoted as n in asymptotic notation. It provides an upper bound on the resources needed by the algorithm ensuring that the algorithm will complete within the specified time frame even for the largest possible input.

Steps	**Time Complexity**
for i := 1 to totalCombinations do	O(2^numStores)
if count of set bits in i is equal to numPartitions then	O(numStores * 2^numStores)
partition := empty list, count := 0	O(1)
for j := 0 to numStores do	O(numStores)
if bit at position j in i is 1 then	O(1)
add j to partition, Increment count by 1	O(1)
if count is equal to numPartitions then	O(1)
sum1 := 0 ; sum2 := 0 ; sublist1 := empty list ; sublist2 := empty list	O(1)
for storeId := 0 to numStores do	O(numStores)
propertiesCount := get storeCount at index storeId	O(1)
if storeId is in partition then	O(numPartitions)

The maximum time complexity theory states that the total time complexity of the algorithm is determined by the step with the most time complexity among all steps of the algorithm. This theory requires us to focus on time-consuming tasks and provides an upper bound on the efficiency of the algorithm used.

In this algorithm, the first loop "for totalCombinations is from i := 1" has a time complexity of O(2^numStores). This step overcomes the overall time complexity of the algorithm. Other steps with lower time complexity are also covered by this step size.

Thus, we conclude that the maximum time complexity of the entire algorithm is O(2 ^ numStores).

MATCP

1 Simple Linear Regression

1.1 Overview of Simple Linear Regression (Brief theoretical description.)

Simple Linear Regression is a statistical technique used to understand and model the linear relationship between an independent variable and a dependent variable. In this case, we are interested in predicting the sale price of properties based on a single independent variable, such as the property area or distance from the center or number of bedrooms or number of bathrooms or number of parking spaces in the property.

1.2 Simple Linear Regression Model

1.2.1 Model significance (Brief explanation of the results obtained by the Anova table, including the information of correlation coefficient.)

After performing Simple Linear Regression and obtaining the regression coefficients, it is important to assess the significance of the model. We can refer to the Anova table and the correlation coefficient to do so.

The Anova table, also known as the analysis of variance table, provides information about the statistical significance of the regression model. It helps us determine whether the relationship between the independent variable and the dependent variable is statistically significant.