University of British Columbia

Optimization within AWS Cloud Computing

MATH 441

Mathematical Modelling: Discrete Optimization Problems

Kyle Tu - 30341572

Lise Meidahl Münsberg - 17795634

Shrey Thapar - 45688439

Shruti Vijaykumar Seetharam - 55121776

July 25, 2023

Contents

Introduction

For every company, it is essential to store files on a large scale. Some files are important to be able to always retrieve, whereas other files are less crucial. The company might have some files which are used multiple times a day for a period, whereas other files are only opened very seldom, but they might still be important to keep for many years. As companies expand more people need to retrieve the companies' files, both in the same office, but also across different offices, cities, and even countries. More and more companies, therefore, decide to use cloud solutions for storing their files. This solution comes with many benefits, but it can also be rather expensive, or inefficient if the company does not choose the right way of storing its files in the cloud. A possible cloud-storing solution is the Amazon AWS S3 service. S3, which stands for Simple Storage Service, has multiple tiers. The tiers have different specifications for how often the data is accessed, how fast the data can be accessed, how many availability zones each tier offers, and much more [2]. As the storage grows from Gigabytes to Terabytes and Petabytes, the costs of storage will grow to millions. Thus when taking egress and retrieval into consideration, means that the costs a company spends on storage only keep growing. To account for this, it is common for companies to try to utilize different storage buckets to their needs. For example, some storage buckets offer cheap egress and retrievals, whereas others offer cheaper long-term storage. [6]

1.1 Amazon AWS S3

As mentioned Amazon AWS S3 is a cloud-storing solution, where companies can pay to have their data stored in the cloud. The service has different tiers, meaning different storage possibilities. Each tier then has some probabilities, which makes it good for storing specific types of data.

In this project, we look at six tiers. The first tier, called *Tier 1* in the report is for data that is frequently accessed. This tier has a high storage cost, but a low retrieval cost. The retrieval time and the minimum storage duration are also low for this tier.

The other tiers have lower storage costs, but higher values in the other properties. In this report, we will mostly look at *Retrieval Time*, *Minimum Storage Duration*, *Storage Cost*, and *Retrieval Charge*, but there are many more properties for these tiers which could be taking into consideration.

Amazon AWS S3 also has different storage possibilities in different regions and there is a cost of moving files from one region to another.

1.2 Problem description

There are multiple ways to optimize the storage to minimize certain fields. In this project, we will use linear programming and mixed integer programming to optimize the cost of storage in S3 depending on a company's requirements about how much data they want to store and how often they want to retrieve the specific data.

We will first start with a very simple problem linear problem, where we just look at a few properties of the tiers. Afterward, we will extend the problem to a binary problem to make it more realistic and usable for companies. Hereafter we will try to formulate the problem from Amazon's point of view, as they also wish to decrease the cost of running the cloud servers, to both maximize their profit, as well as keeping their prices low. Furthermore, we will look into storing in different regions, and how a company can cheapest move and store their data in different regions.

The report also discusses the possibilities and limitations of the different models implemented in the project, and how they can be improved.

The simple linear problem

2.1 Formulation of the problem

We are starting out by making a very simple problem. We are therefore not looking at the individual files, but just at the amount of space, the company needs to have stored. We will here use the decision variable: $Q = [q_1, q_2, q_3, q_4, q_5, q_6]^T$, where each q_i is the amount of storage in tier i in GB.
Our goal is then to minimize the storage cost:

$$\min c^T Q$$

where c is a vector containing the cost per GB in each tier, the data for the vector can be seen in Table 2.1.
The company also has some constraints that need to be satisfied. Some of the attributes in Table 2.1 will be added to the constraints. And the constraints could e.g. look like the following, where the attributes are vectors:

$$\frac{\text{Retrieval time} \cdot Q}{S} \leq R_{limit}$$
$$\frac{\text{Min Storage duration} \cdot Q}{S} \leq P_{limit}$$
$$\sum_{i=1}^{6} q_i \geq S$$
$$q_i \geq 0, \text{for } i \in [1, ..., 6]$$

The reason that we define the constraints by the total amount of space, S, is so the limits are average values, it is therefore easier for the company to choose these values, and they do not have to change the values if they change the amount of space needed.

Here S is the amount of storage needed, and R_{limit} and P_{limit} are some limits that the company can set for their case. These constraints can be combined using an A matrix and a b vector. The A matrix will look like the following, whereas the b vector will change for each company, as it is the b vector that defines the limits for the specific company.

$$A = \begin{bmatrix} \text{Retrieval time} \\ \text{Min Storage duration} \\ -1 & -1 & \cdots & -1 & -1 \end{bmatrix}$$

Where the first 2 rows are vectors containing the values for each tier, which can be found in Table 2.1.

The b vector will be the following:

$$b = \begin{bmatrix} R_{limit} \\ P_{limit} \\ -S \end{bmatrix}$$

Which are all values that depend on the company.
The problem can then be written more compactly, which can be solved as the following problem:

$$\begin{aligned} \min\ & c^T Q \\ & s.t \\ & Ax \leq b \\ & x \geq 0 \end{aligned}$$

2.2 Data used for the problem

We think it could be interesting to look at some of the data seen in Table 2.1 to optimize the storage for a company. The table shows different values for the different tiers where the company can store its files. The table is not filled in yet, but it gives an idea of the various constraints we will look into.

For our first iteration of the problem, we looked at just Retrieval Time, Minimum Storage Duration, and Storage Costs with the following information as in Table 2.1:

	Retrieval Time (hours)	Min Storage Duration (days)	Storage Cost ($/GB)
S3 Standard	0.001	0.5	0.023
S3 Standard-IA	0.01	30	0.0125
S3 One Zone IA	0.1	30	0.01
S3 Glacier Instant Retrieval	6	90	0.04
S3 Glacier Flexible Retrieval	24	90	0.0036
S3 Glacier Deep Archive	30	180	0.00099

Table 2.1: Data used for initial calculations

2.3 Solve a problem

We now both have a problem formulation and some data, and we can solve a problem. The only values we need for solving the problem are a value for P_{limit}, one for R_{limit}, and the amount of storage needed. The company sets these values. The original problem we solved using PuLP had $P_{limit} = 30$, $R_{limit} = 3$, and $S = 5000GB$. The result we got was to put all 5000 GB of data in Tier 1, and the overall cost was $50. This is not ideal and realistic, so we looked at a different problem.

And we here choose $P_{limit} = 15$, $R_{limit} = 0.01$, and $S = 100,000GB$. Using PuLP, we solve this problem and get an objective value of $2229. In Figure 2.1 we can see that the company should store most data in Tier 1, second most in Tier 2, and a little bit in Tier 3. It is also seen that the company should not store anything in the last three tiers.

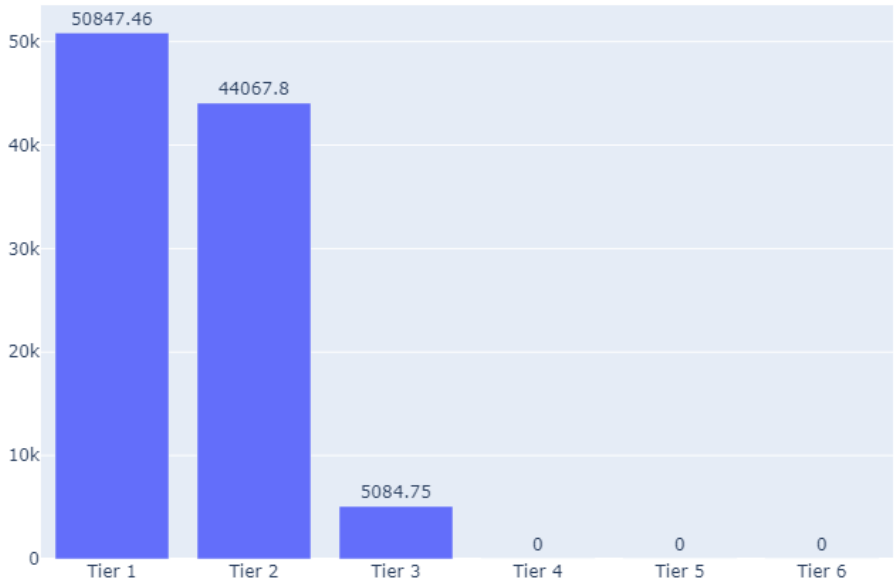


Figure 2.1: Visualization of solution

When solving this problem we did not look at the company's individual files, this means that the model does not tell which files to store in each tier. The model also did not take the retrieval costs into account when finding the optimal solution. We, therefore, want to improve the model by adding these things. This is done in chapter 3.

Expanding the model

In chapter 2 we created a very simple linear model for minimizing the cost of cloud storage for a company, given some criteria that the company set. But the model did not take the number of retrievals into account, some tiers have a higher cost for retrieving files than others, this fact we ignored in the first part. Another disadvantage of the simple linear model is the fact that the company does not get told which files should be where. This means that the model just says how much space they should have in each tier, but this is not too much help if they do not know which files they should have in each tier.

To solve these problems we can extend our model from chapter 2. This is done by adding the companies' files. For each file, we need two values, one that tells the size of the file and one that tells the expected number of retrievals. Since the cloud solution keeps the number of retrievals, the company could for instance automatically calculate the expected number of retrievals by looking at past rates. The data for all the files are then defined as two vectors, one with all the sizes and one with all the expected number of retrievals.

3.1 Formulation of problem

We define the number of files as F. We define the vector with the sizes as $size$ and the vector with the number of retrievals as $retrieval$.

When solving this problem we have $(6 \cdot F)$ binary decision variables. We define the decision variables are defined as x_{ij}, where i is the tier and j is the file. If $x_{ij} = 1$ it means that file j should be located in tier i and if $x_{ij} = 0$ it means that the file j is not located in tier i.

For each tier, we have the cost pr. GB c as in the previous model, but now we also have a vector $rcost_i$ which is the cost for 1000 retrievals for tier i.

We can now define the objective function as the following:

$$\min \sum_{i=1}^{6} \sum_{j=1}^{F} (x_{ij}(size_j \cdot c_i + retrieval_j \cdot rcost_i)) \tag{3.1}$$

In the objective function in Equation 3.1 if file j is in tier i we multiply the size of the file in GB by the cost of storage in that tier, which is the first part of the sum. To this, we add the cost of retrievals in that tier times the number of expected retrievals for the file as well. Since x_{ij} is only 1 when file j is in tier i the cost is only added if this is true, and else the cost will be equal to 0.

We now need some constraints to the problem as well. For the constraints, we need to define a few constants. First, we define the total amount of space as $S = \sum_{j=1}^{F} size_j$ and the total

number of retrievals as $R = \sum_{j=1} retrieval_j$.

We first add a constraint for the retrieval rate, as in the LP. But here we decide to make the adjusted average, as we take the expected number of retrievals into account. This constraint is defined in Equation 3.2.

$$\sum_{i=1}^{6}\sum_{j=1}^{F}\left(\frac{x_{i,j}\cdot retrieval_j \cdot \text{retrieval time}_i}{R}\right) \leq R_{limit} \tag{3.2}$$

We also add the constraint for the storage duration, as before, this is seen as Equation 3.3.

$$\sum_{i=1}^{6}\sum_{j=1}^{F}\left(\frac{x_{i,j}\cdot \text{min storage duration}_i}{F}\right) \leq P_{limit} \tag{3.3}$$

Lastly, we add a constraint that each file should be in exactly one of the tiers, this is defined in Equation 3.4.

$$\sum_{i=1}^{6} x_{i,j} = 1, \qquad \forall j \in [1, ..., F] \tag{3.4}$$

This problem can then be solved using an optimization solver. We have chosen to use Gurubi, which is a commercial solver built on the branch and bound algorithm. We have implemented the model in Julia, and the code can be found in the Julia file *Math441_Model2.jl*.

3.2 Data for the problem

Most of the data for the tiers are the same as in chapter 2, but we now add the cost for 1000 retrievals for each tier, as this is a part of the objective function. These values are therefore added to the data, which gives the Tier-data seen in Table 3.1.

	Retrieval Time (hours)	Min Storage Duration (days)	Storage Cost ($/GB)	Retrieval Charge (*rcost*)
S3 Standard	0.001	0.5	0.023	0.005
S3 Standard-IA	0.01	30	0.0125	0.01
S3 One Zone IA	0.1	30	0.01	0.01
S3 Glacier Instant Retrieval	6	90	0.04	0.02
S3 Glacier Flexible Retrieval	24	90	0.0036	0.03
S3 Glacier Deep Archive	30	180	0.00099	0.05

Table 3.1: Data used for initial calculations

3.3 Solve a problem

To solve this problem we first create some values for a fictive company. This is done by first choosing the number of files the company has, in this example we set this to $NFiles = 10000$. We also define the maximum size of their files in MB $MaxFilesSize = 10000$ and their maximum retrieval times, $MaxRetrivalTimes = 10000$. We can now create two random vectors, one for the file sizes and one for the retrieval times, the length of the vectors will be $NFiles$. We decide to divide both vectors with 1000, to get the values in GB and 1000 retrievals as our costs are in these units. The code for creating the data can be found in the Julia file *Data2.jl*

For a company, these vectors should be able to get from their files, but we decided to create random ones.

The company then needs to decide on its limits for the retrieval rate and the storage duration. We here chose the retrieval rate as, $R_{limit} = 0.1$ and the storage duration as $P_{limit} = 45$.

We now solve the problem, and we get 60000 binary variables that tell the company what files should be in each tier. As this is a lot of numbers, we have decided to visualize the amount of space the company should have in each tier, as this is just 6 numbers compared to the 60000. The company can also decide on a file and then check what tier it is supposed to be in, which we will also visualize.

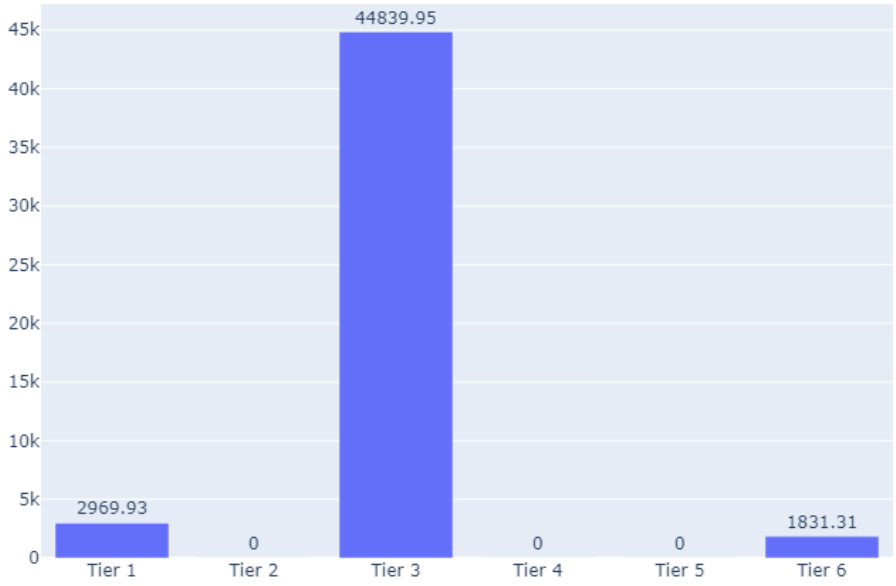


Figure 3.1: Visualization of solution

In Figure 3.1 we can see that the company should store 2970 GB in Tier 1, 44840 GB in Tier 3, and 1831 GB in Tier 6. The objective value of the solution is \$950.

The difference between this model and the one in chapter 2 is that we now know where each file should be stored. We can therefore look at a specific file, we here for instance look at file number 1000. This file has a size of 7.321 GB, the expected number of monthly retrievals is 3624 and the file should be in Tier 3. We can find this data for all tiers, which means that the company now knows which files to locate in which tier.

Simulation of 100 companies

Using the same setup as the original problem in chapter 2, we also simulated the same problem for 100 fictional companies and proceeded to solve it.

In order to create the data for these 100 companies, we generated values uniformly for all of the constraints in the possible ranges for them. We then passed these values to the function and optimized each company's storage. The result we got was a 6×100 matrix, with the rows being a tier and the columns being a company each.

Such a matrix is hard to read, so we used a heatmap to picture it. We've used four separate heatmaps as seen in Figure 4.1 to Figure 4.4, one for each quarter to demonstrate the optimal storage in order to make it easier to compare it across all 100 companies. Note that the companies in these figures go from 0 - 99 as the data was generated using Python and this uses Python indexing. The exact numbers are stored in the Group Project Data file attached.

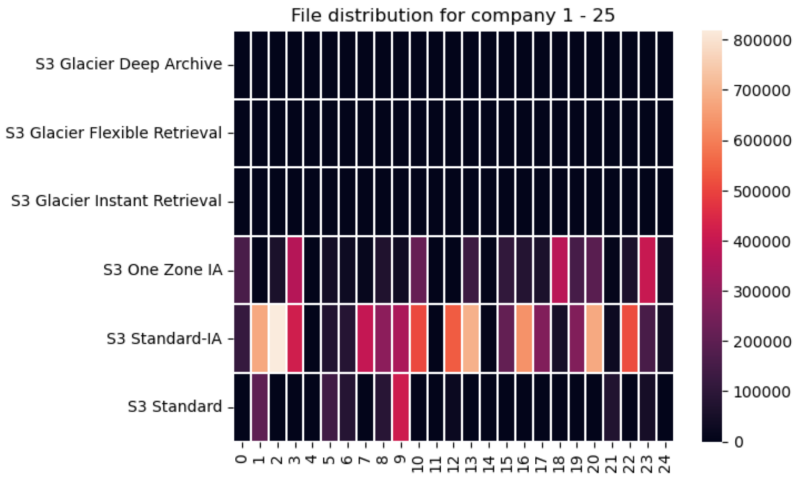


Figure 4.1: Distribution of Company 1-25

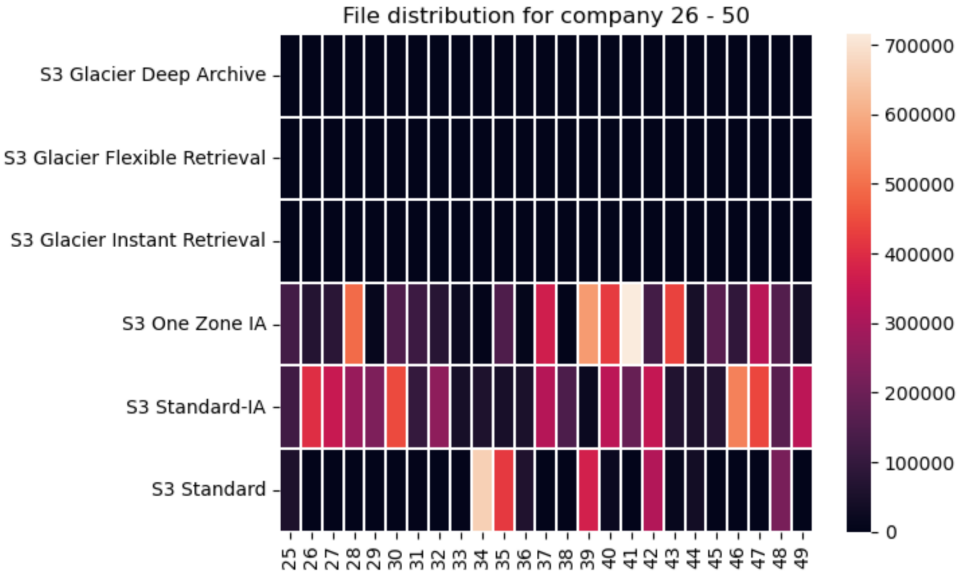


Figure 4.2: Distribution of Company 26-50

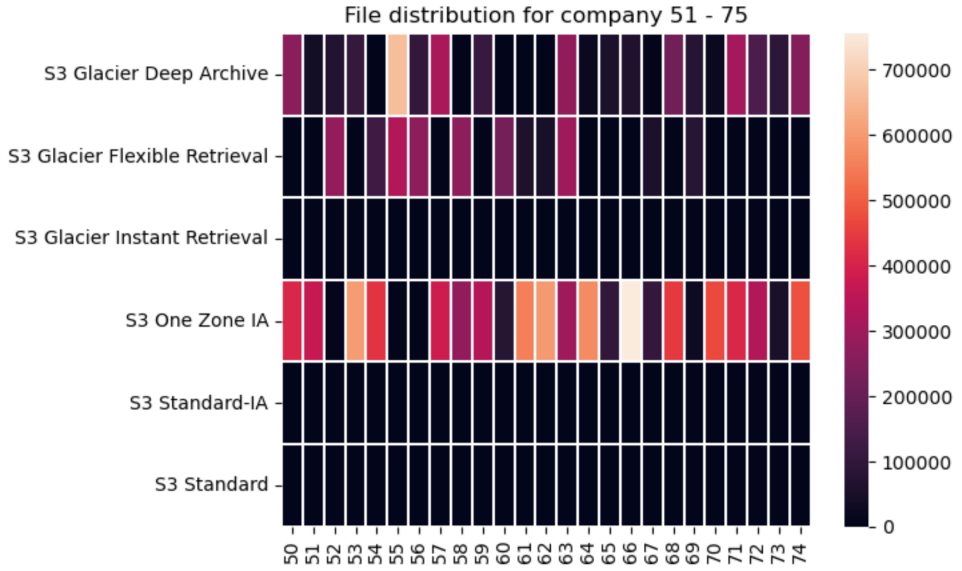


Figure 4.3: Distribution of Company 51-75

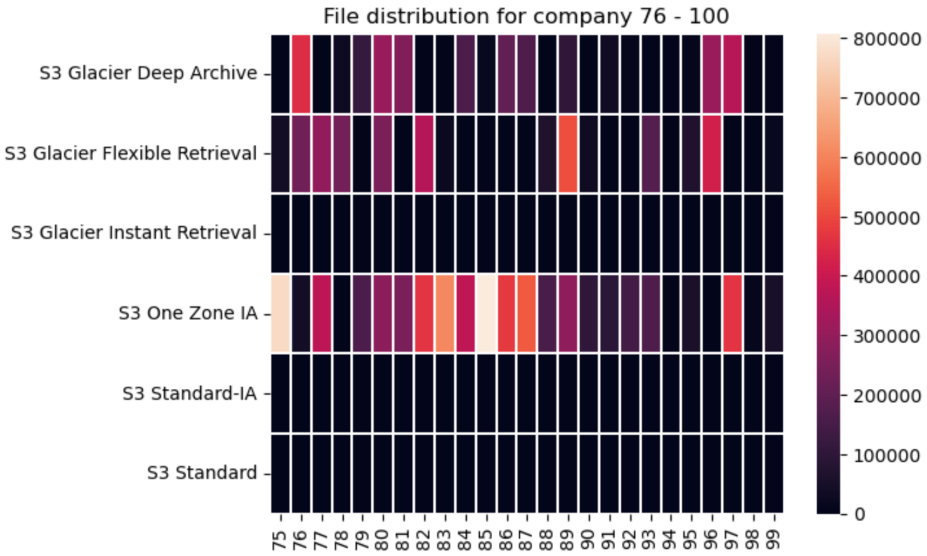


Figure 4.4: Distribution of Company 76-100

What's interesting to note is that across all companies, the Glacier Instant Retrieval Tier has no data. We can only assume why this is, but a possible reason is that the cost is quite similar to the tiers below and above, and the retrieval time and storage duration do not justify the cost.

Using The Bin Packing Algorithm

5.1 Defining The Problem

Since one of our objectives is to formulate the problem from Amazon's point of view, we now look at the most optimal way Amazon would store data for all 100 companies utilizing the 6×100 matrix generated from the previous linear program.

Suppose we have n, $c \in \mathbb{Z}^+$. Bin Packing looks at how to store n items of varying capacities in the minimum number of bins possible. Each bin is of fixed capacity c. Let A denote our 6×100 matrix. We shall split it into six vectors such that B_j is a vector containing the storage needed in tier j for each of the 100 companies. We will fix the storage of the bins to $c = 100$GB and we will attempt to fit n_i in the fewest possible bins, where $0 \leq i < 100$.

We look at two separate algorithms that can help us minimize the number of bins. One is Next Fit and the other is First Fit. Since we are mimicking real world storage, we assume that the size in advance is not known thus the usage of these online algorithms. Otherwise, we could sort B_j in decreasing order and get a tighter upper bound.

5.1.1 Next Fit

- checks item, if it fits in the bin it places it, otherwise uses a new bin if there is no space.
- If the optimal number of bins is given by M, the total number of bins will not exceed $2M$
- The pseudo-code for this algorithm is given as follows:

```
define nextFitAlgorithm(array, capacity):
    bin_num = 0
    bin_curr_capacity = capacity
    for i in range(0, length(array)):
        if array[i] cannot fit in bin:
                increment bin_num by 1
            bin_curr_capacity = 100 - curr_item_weight
        else:
                bin_curr_capacity = bin_curr_capaciy - array[i]
    return bin_num;
```

5.1.2 First Fit

- checks item weight scans all the previous bins to see if the item can fit and places it accordingly. Only starts a new bin if the item does not fit in any of the previous bins.

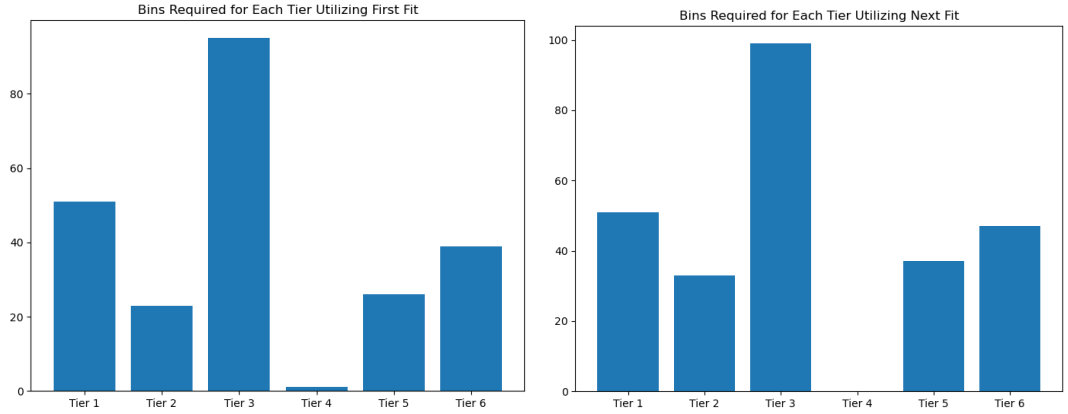
- If the optimal number of bins is given by M, the total number of bins will not exceed $1.7M$
- The pseudo-code for this algorithm is given as follows:

```
define firstFitAlgorithm(array, length_of_array, capacity):
    bin_num = 0
    bin_remaining_space = [0, 0, ... length_of_array]

    for i in range(0, length_of_array):
        curr = 0
        while (curr < bin_num):
            if (bin_remaining_space[curr] >= array[i]):
                bin_remaining_space[curr] -= array[i]
                break
            curr += 1

        if (curr == bin_num):
            bin_remaining_space[bin_num] = capacity - array[i]
    return bin_num
```

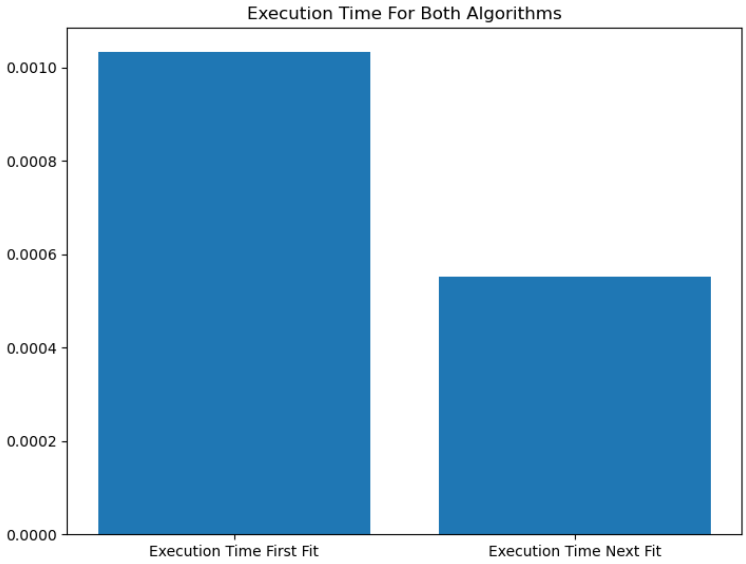
The pseudo-code as well as the implementation for both algorithms have been inspired and adapted from Shinjanpatra who wrote the following article.

5.1.3 Results

Utilizing the First Fit algorithm shows us that only 196 bins are required to host the data for every tier. Utilizing the Next Fit algorithm, 220 bins are required. That is 2400 GB more being utilized by the Next Fit algorithm.



On the other hand, we see that the Next Fit algorithm runs almost 3 times faster than First Fit. First Fit's execution time is approximately 0.00103 seconds whereas Next Fit's execution time is 0.00055 seconds. Thus, we see a trade-off between space and latency. As reducing storage costs and low latency cloud services are both some of Amazon's primary goals, the choice of algorithm to utilize needs to be evaluated on a case-by-case basis for individual customers.


Execution Time For Both Algorithms
0.0010
0.0008
0.0006
0.0004
0.0002
0.0000
Execution Time First Fit
Execution Time Next Fit

Applications to Network Flow

6.1 Introduction

6.1.1 Defining the Problem

Now let us imagine that a user has some amount of data that they'd like to move to new potential regions. This means that the regions that the data end up in must be ones where there aren't any, to begin with.

6.1.2 Defining Variables

There will be a cost of storage per GB of data for each region, as well as the cost of transfer between any two regions per GB. The data can be separated into frequently used, and infrequently used. Thus there will be different storage costs for each region by frequency type. The user can define their start amount of frequently used and infrequently used data in each region. A max limit for the amount stored in each region is also introduced.

The regions, the transfer cost, and the storage cost are all compiled from AWS S3.

6.2 Formulating the linear programming problem

Where $k = 0$ if the data is frequently used and $k = 1$ if the data is infrequently used. Where $storage_{j,k}$ represents the cost to store a GB of data of type k at location j, and $transferi, j$ represents the cost to transfer a gigabyte of data from location i to j. $x_{i,j,k}$ represents the number of GB of data of type k going from location i to location j. Then our objective function will be

$$\text{minimize} \sum_k \sum_j \sum_i (\text{storage}_{j,k} + \text{transfer}_{i,j}) * x_{i,j,k} \tag{6.1}$$

Let $start_{i,k}$ represent the amount of data originally in location i of type k.
The constraints will then be:

- The amount of data leaving location i should be the same as the data originally in i. This also checks that all of the data originally in i has been transferred.

$$\sum_j x_{i,j,k} = \text{start}_{i,k}, \text{ such that } i \neq j, \text{ for all } i, k \tag{6.2}$$

- Limit for storage

$$\sum_i x_{i,j,k} \leq \text{LIMIT}, \text{ for all } j,k \tag{6.3}$$

- No incoming data for any storage that originally has data.

$$\sum_j x_{i,j,k} = 0, \text{ such that start}_{j,k} = 0, \text{ for all } i,k \tag{6.4}$$

- Variables are at least 0

$$x_{i,j,k} \geq 0, \text{for all } i,j,k \tag{6.5}$$

6.3 Results

Below is a possible diagram of our starting network flow problem. The weighted edges represent the cost per GB of transferring data over. We can see in the example that we currently have data in the US West, South America, and Europe regions.

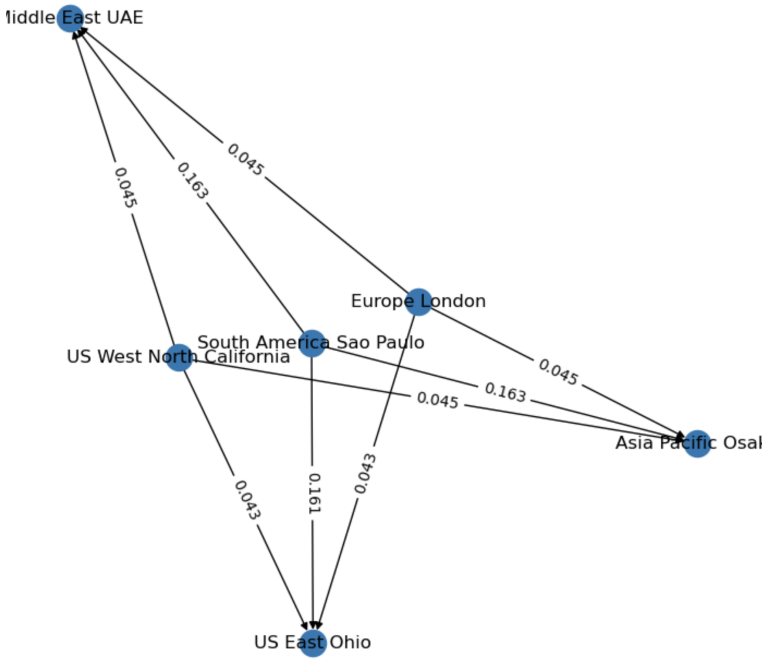


Figure 6.1: Network flow problem

The Jupyter Notebook file attached will also generate another graph that represents the solution to the problem stated.

We have visualized the solution for the frequently used data in Figure 6.2. We see that of the 17 GB of data originally from the US West region, 2 GB should move to Asia Pacific, and the rest to US East.
These graphs only represent the problem and solution for either the frequently used or infrequently used data. Another pair of graphs will be generated to represent the other type.

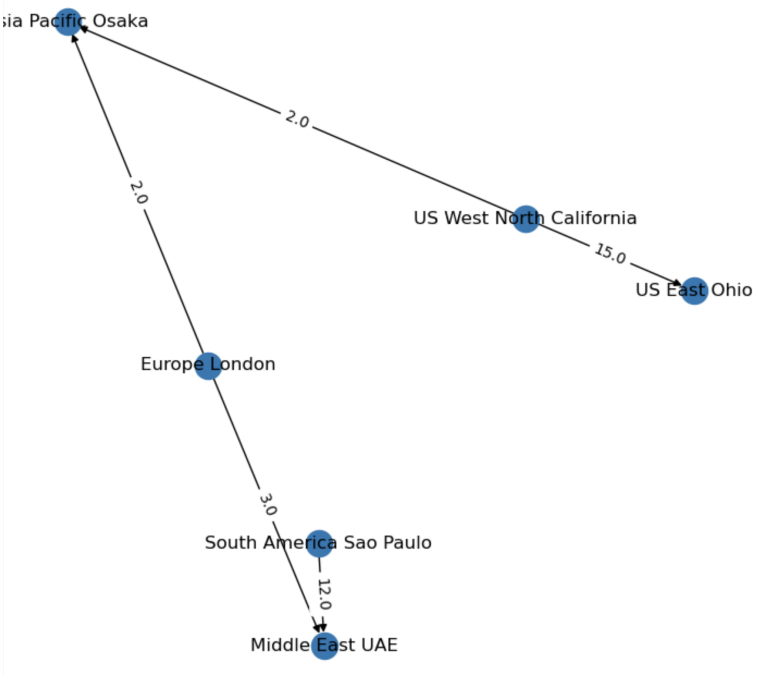


Figure 6.2: Solution to network flow problem

Discussion

The models implemented in the report are quite simple, and can accordingly help a company figure out how to divide its files into different tiers. But an interesting notice is that our models never assign any storage to Tier 4. This does not mean that no data in the world should be stored in Tier 4, because then it would probably not exist. It, therefore, shows one of the limitations of our models, we chose only a few properties for the tiers, even though there are many more. But as we have created the framework for the model more properties can easily be added as new constraints.

If we look at the simulated data, the first 50 companies store all their data in Tiers 1-3, and the next 50 store their data in Tiers 3, 5, and 6. This uniformity comes from the data we generated, and it is unlikely that in reality, all companies ONLY store their data in either Tiers 1-3 or Tiers 3, 5, and 6. Our model could also be improved in this regard.

Furthermore, the first model is definitely the fastest to run, as it is just an LP, but the second model is more realistic as it also takes the retrieval rate into account, and it can take data for all the company's files into account as well.

The second model could also be improved by taking more properties into account. As all companies have some files which are more important than others, there could be added importance factors to the files. The important files could then have a higher weight in the constraints, as well as a penalty for retrieval times in the objective function, as the company might lose money if it takes a long time to retrieve an important urgent file.

In regards to the Bin Packing algorithms, we see that Next Fit has a faster code execution time than First Fit. However, this is because the current implementation of First Fit is done in $O(n^2)$ time. Further research shows that First Fit can be implemented in $O(n \log n)$ time if the inherent data structure utilized is a balanced search tree, as opposed to a simple array structure [3].

This project is, therefore, a good starting point in optimizing the distribution of files in different tiers, and the models can then be improved for the specific company, as every company has different criteria and different priorities. This also means that it is impossible to create a model that is perfect for all companies, even though most of the frameworks can be the same for different companies.

Conclusion

Through this project, we have looked at ways to optimize a company's usage of AWS S3 as well as how Amazon can optimize their different servers with the simulated data.

While the models we have created are not entirely accurate, we can build upon them and customize them for each company. We discovered that it is hard to create a uniform model across all companies and depending on each company's requirements, the model will likely have to be adjusted.

We also discovered that a bin-packing algorithm could be used to figure out how many servers are needed to store a specific amount of data when each company's data should be stored on the same server. We discovered that the first fit bin-packing algorithm gave better results compared to the Next fit algorithm.

Furthermore, we implemented a graph problem that a company can use to figure out which regions to move files to if they wish to store files in multiple regions.

Bibliography

[1] *Amazon S3 Intelligent-Tiering storage class*. URL: https://aws.amazon.com/s3/storage-classes/intelligent-tiering/.

[2] *Amazon S3 Storage Classes*. URL: https://aws.amazon.com/s3/storage-classes/.

[3] *Bin Packing*. URL: https://www.ics.uci.edu/~goodrich/teach/cs165/notes/BinPacking.pdf.

[4] *Bin packing problem*. URL: https://en.wikipedia.org/wiki/Bin_packing_problem (visited on 11/07/2022).

[5] *Bin Packing Problem (Minimize number of used Bins)*. URL: https://www.geeksforgeeks.org/bin-packing-problem-minimize-number-of-used-bins/ (visited on 11/07/2022).

[6] *How Amagi uses Amazon S3 Glacier Instant Retrieval to optimize media storage costs*. URL: https://aws.amazon.com/blogs/storage/how-amagi-uses-amazon-s3-glacier-instant-retrieval-to-optimize-media-storage-costs/ (visited on 10/07/2022).

[7] *Transitioning objects using Amazon S3 lifecycle. (n.d.)*. URL: https://docs.aws.amazon.com/AmazonS3/latest/userguide/lifecycle-transition-general-considerations.html (visited on 10/08/2022).