Examine Technology Gaps in Algorithm Structures

Yiqiao Yin
Ph.D. Student

Abstract

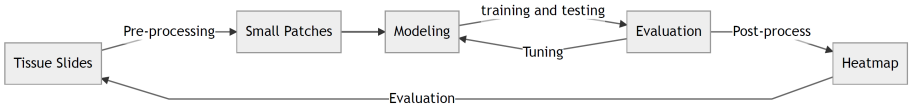
This report explores the concepts of novel data ordering techniques. Particularly, the paper investigates the new alternatives in data structure, data ordering, data processing, and data base management. The paper provides some technical discussion and present some sample code to demonstrate the argument.

1 Introduction of Algorithm Structure

Database management system and data platform play important roles for today's development in software engineering and database ordering and management. A famous example is the Enterprise Resources Planning System (ERP system) which is originally developed from the Manufacturing Resources Planning (MRP system) (Moon, 2007; Al-Mashari, 2003; O'Leary, 2004). These generations of development focus on providing methodologies to all resources in every different component of a company. It is not surprising that the letter "E" from the name means "enterprise". This is because, in theory, the platform is designed to serve all levels in organization, financial planning, and business operations. In practice, ERP systems are capable of integrating inventory data which is designed to operate on the cloud and also via a variety of different locations. Just like each business practice has its own unique optimized environment, the ERP system needs to be able to adapt every building block of an enterprise or a corporation.

Another famous example is the Electronic health record or (EHR) system. It is an systemic collection of patients data designed to store and manage all available health information in a digital approach (Majeed et al., 2008; Safran and Goldberg, 2000; Garets and Davis, 2005). These information serve as the basis of health care setting where physicians and doctors can review information of patients throughout a secured network program. At enterprise-wide setup, the information is also able to be shared across different practice therapeutic areas as well as different medical practices. This is a famous example because data forms are stored completely different in every single branch of the hospitals and there are insurance companies involved to interfere the data collection process. The EHR system is capable of storing data and capture the state of the patient across time. Due to this setup, the tedious label of tracking down the patients' health data is largely omitted which makes the diagnosis procedure much more efficient than before. However, the premise is that the EHR system is indeed capable of the flexibility of editing and modifying patients' health data without leaking important information to an unrelated third party. This also calls for the needs of point-

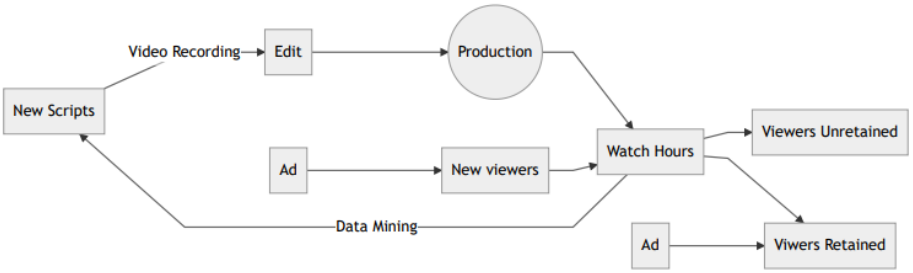
Figure 1: **Executive Diagram for Digital Pathology**.



to-care if high quality of data and healthcare services are provided. In other words, the implementation of such EHR system is a key objective for many hospitals and healthcare related organizations. An executive diagram of digital pathology is presented in Figure 1. The tissue samples are collected and exported as images. These images are SVS files that are large-scale, i.e. 4-8 GB in sizes. The pre-processing is required to generate small patches of pixel values from the original image. The patches are used for training set and the models are built. The model has different tuning parameters of which need to be tuned throughout a rigorous evaluation procedure. Data structures are converted from images to arrays in the modeling process. The output, masks or annotations, is then converted back into image format to create heatmap such that images can be generated to deploy to machine for doctors to evaluate. Not all image slides are evaluated correctly. Hence, this requires more evaluation procedure which can require us to start from the pre-prepossessing step.

A third example is the Customer Relationship Management (CRM). This type of technology is crucial at all operational levels. It is the key component of managing the organization's relationships and interactions between managers and customers. Allow me to use my own story as an example. The YouTube channel YinsCapital was founded 2 years ago. Initially the channel was created for fun and it is designed for leisure purpose. However, it has come to the founders' attention that some videos are doing well in terms of watch hours. Hence, this calls an interest to start a data mining project on the old videos posted. YouTube provides tabular data that consist of features such as "likes", "viewable rates", "cards clicked" and so on. These features have some contributing power to push for high watch hours which can become Ad sales. Once realized the root of the key objectives, a CRM system was immediately developed to ensure high customers retention rate and consistent watch hours. A diagram of this executive platform is presented in Figure 2. In other words, a CRM platform is required to set up for to develop more interesting videos.

Figure 2: **Executive Diagram of YouTube CRM**.



In the beginning stages, the scripts of the new stories are created on the spot without any motivation to retain any customers or viewers. This is risky business practice, however,

it is the only viable approach without data mining. A series of videos have been posted and made into the production. Hence, these videos are generating views. The watch hours can be collected and the attributes of the videos along with watch hours can be recorded as well. This forms the initial version of the customer database that can be helpful for R&D team to conduct the experiments using machine learning techniques. Regression models can be built to predict the watch hours given the attributes of a video. Inferences can also be made using statistical approach to explain what are the important features to trigger high watch hours. These results are collected and send to the Director of film production to push more scripts following the same topic which is supported by empirical evidence resulting from pattern recognition team using machine learning algorithms. The key objective is to increase watch hours so that the business can produce revenue and income. The essence to increase watch hours is to attract new viewers and also retain old viewers.

Now that the business operation and activities are cleared defined, the role of CRM system needs to follow. The data structure is a key because the video along with related attributes are essential for data mining. The data from scripts are text format. When the story goes through recording and editing to production, it will become video format. The final objective is the watch hours which can simply be a numerical form. Hence, to keep things simple, denote the script to be $\mathcal{S}$, the video to be $\mathcal{V}$, and the attributes to be $\mathcal{X}$. The watch hours $Y \in \mathbb{R}$ can be modeled using the following equation

$$Y = f(\mathcal{S}, \mathcal{V}, \mathcal{X}) \tag{1}$$

where $f(\cdot)$ is a function that can be sought using machine learning.

2 Motivation to Explore Novel Alternatives for Data structuring and Algorithmic Design

Disregard the areas of practice (healthcare, technology, finance, or else), the data structure and data ordering are crucial components in a development pipeline. As discussed in the above portion of this report, the ERP system, the EHR system, and the CRM platform all require different structural design to handle different forms of data. In addition, it is important to review some famous algorithmic design to understand the motivation of why it is desirable to come up with novel techniques.

2.1 Problem Statement

In algorithmic design and data ordering and structuring, the first key concept is to understand the key-value pairs. This is referring to the one-one map between the key and its corresponding contents also commonly known as Hash Table. From practical perspective, the important property for hash table is that there should not be any collision. When handling a collision issue, it is either directly resolve via a different style of coding or the algorithmic can allow multiple items in the same bucket to form the content of that key.

The second types of problem is linked lists. In data structure and algorithmic design, a linked list is usually a list of nodes. Each node carries attributes such as "value" and "next" where the "next" stores the address of the next node where "value" stores the content of what is inside of this node. The problem with linked lists is that they are tricky to handle and cannot always be easily manipulated. Sometimes the chained lists or linked lists are even hard coded to avoid code crash. The third type of problem that can raise potential issues is "heap". In data structure and algorithmic design, heaps are

the type of data that rarely exists and is definitely more esoteric data structure. Usually, scholars do not do sufficient research in this type of data structures and they can barely be broadly implemented. However, there is one type of practical area that uses heap structure is finding the top K elements in some unsorted array.

2.2 Potential Impact of the Problem

The first problem to address is the collision issue in the algorithmic design. In Figure 1 and Figure 2, both diagrams presented operational and business activities without the strict premise stating all data structure must be one-one map. In other words, the issue of collision problem in hash tables exist because not all real world data contain the required assumption for hash tables to operate properly.

For the second problem described above, i.e. the chained lists or the linked lists, it is a common problem in model building. In both Figure 1 and Figure 2, modeling component posts great importance in the entire business operation. This is unavoidable and it is a "must" for the system to learn from the past data. However, a model such as an Artificial Neural Network (ANN) can sometimes be soft coded using functional API instead of sequential API. This would be a real world scenario that linked lists play into effect. If the data structure is not properly managed, the neural network or ANN model could fall apart. The last problem described above is the "heap" structure. The "heap" structure is designed to search for the top elements in some unsorted array according to certain threshold. Most likely, programmers prefer to work with sorted lists. In this case, "heap" produces the same ideal items without sorting the array. In regards to the consumption time, it is not as ideal, because it is $\mathcal{O}(k\log(n))$. In addition, for unlucky situation, a complete sort must have $\mathcal{O}(n\log(n))$ which is not trivial and can be costly to run.

2.3 Algorithmic Design

This section presents some potential algorithmic design addressing each and every problems stated above.

First, it is discussed above that the hash tables have potential collision problems which is almost always inevitable because the previous section of the table listed out popular real world scenarios that do not necessarily possess the premises required for hash tables to exist. One way to work around this concept is the following. Some pseudo code can be presented below. A hash table can simply be designed using dictionary, i.e. this is the curly bracket "{}" in python syntax. The for loop dictates the index every step of the way so that the item can properly be extracted one by one.

```
hash_table = {} # this is a dictionary
for i in hash_table:
print(i)
```

Each item in the dictionary does not have to have the same length or even the same object. For example, the same dictionary defined above can be expanded using a list of integers or a combination of integers and letters. The for loop dictates the output to be each item from the list.

```
hash_table = {
    'a': [1,2],
    'b': [1,2,3,'c']
} # this is a dictionary
for i in hash_table:
```

```
print(i)

# output
[1,2] \\
[1,2,3,'c'] \\
```

For the second issue regarding linked or chained list, a potential solution with some basic algebraic operation can be presented below. The example presents a simply defined node that is written in a class object. The "ThisNode" class object has "value" and "next" defined in the initiation. Then there is a while loop inside and the while loop iterative adds value to an output. A function, "list_to_linkedlist" is created to execute an operation using "ThisNode" object. The function checks each item in the entered number and checks that whether it is equivalent with the "head" or not. It returns the updated value "head" in the end.

```
class ThisNode(object):
  def __init__(self, value, next=None):
    self.value = value
    self.next = next

  def __str__(self):
    curr = self
    output = ''
    while curr:
      output += str(curr.value)
      curr = curr.next
    return output

def list_to_linkedlist(some_list):
  head = None
  curr = None
  for n in some_list:
    if not head:
      head = ThisNode(n)
      curr = head
    else:
      curr.next = ThisNode(n)
      curr = curr.next
  return head

n = ThisNode(1, ThisNode(2, ThisNode(3)))
print(n)

print(list([10, 20, 30]))
```

In machine learning when using ANN, suggested by the previous section, functional API can sometimes be used to take care of repetitive coding. Some sample code is presented below. The algorithm starts with "inputs" of which the dimension should be defined. The hidden layers are tricky to be defined if the user desires to make it soft coded. In other words, user can enter any list of integers in the beginning and the algorithm automatically updates the hidden layers of the neural network model.

```
# parameter
params = [128, 64, 32, 10]

```

```
# initiate the model input
inputs = tf.keras.Input(...)

# the first layer
dense = layers.Dense(params[0], activation="relu")
x = dense(inputs)

# hidden layers
for i in params[1::]:
    x = layers.Dense(params[i], activation="relu")(x)

# the last layer
outputs = layers.Dense(params[10])(x)

# finalized the model
model = keras.Model(inputs=inputs, outputs=outputs, name="
    ↪ mnist_model")
```

For "heap" data structure, one approach is to work with queues and it is a good start because most languages contain built-in support functions. Take python programming as an example, the heap structure can be imported using "heapq" library. For example, as the sample code shown below, a list of unsorted numbers can be defined using the following code. Then the "heapq" library has the "heapify" function that can directly produce a heap structure.

```
import heapq

some_list = [5, 40, 3, 1, 2, 70, 8]
heapq.heapify(some_list)
print(some_list)
print(heapq.heappop(some_list))
print(heapq.heappop(some_list))
print(heapq.heappop(some_list))
print(heapq.heappop(some_list))
print(heapq.heappop(some_list))
print(heapq.heappop(some_list))
print(heapq.heappop(some_list))
```

To sort a heap structure, the code below is a good example, along with the hand-written implementation of a heap. This is considered one of the most complicated algorithm, but it is fairly straightforward upon reading it.

```
def heapify(arr, n, i):
    largest = i # Initialize largest as root
    left = 2 * i + 1 # left = 2*i + 1
    right = 2 * i + 2 # right = 2*i + 2

    # See if left child of root exists and is
    # greater than root
    if left < n and arr[largest] < arr[left]:
        largest = left

    # See if right child of root exists and is
    # greater than root
    if right < n and arr[largest] < arr[right]:
```

```
        largest = right

    # Change root, if needed
    if largest != i:
        arr[i],arr[largest] = arr[largest],arr[i] # swap

        # Heapify the root.
        heapify(arr, n, largest)

# The main function to sort an array of given size
def heapSort(arr):
    n = len(arr)

    # Build a maxheap.
    for i in range(n, -1, -1):
        heapify(arr, n, i)

    # One by one extract elements
    for i in range(n-1, 0, -1):
        arr[i], arr[0] = arr[0], arr[i] # swap
        heapify(arr, i, 0)

# Driver code to test above
arr = [ 12, 11, 13, 5, 6, 7]
heapSort(arr)
n = len(arr)
for i in range(n):
    print ("%d" %arr[i], end=' ')
# 5 6 7 11 12 13
```

3 Conclusion

This report examined famous algorithm structures to identify some of the famous research problems in data structure and algorithmic design. Famous structures include (1) hash tables, (2) linked lists, and (3) heaps. The paper started with the introductory section and explained with live examples in the ERP, MRP,and CRM system the issues and social impact of data structures. Then the report stated the problems from technical prospective. The report listed out the potential impact of these data structures based on the technical component and produced some code samples to illustrate the root of the problems.

References

Al-Mashari, M. (2003). Enterprise resource planning (erp) systems: a research agenda. *Industrial Management & Data Systems*, 103(1):22–27.

Garets, D. and Davis, M. (2005). Electronic patient records. *Healthcare Informatics Online*, 4.

Majeed, A., Car, J., and Sheikh, A. (2008). Accuracy and completeness of electronic patient records in primary care.

Moon, Y. B. (2007). Enterprise resource planning (erp): a review of the literature. *International journal of management and enterprise development*, 4(3):235–264.

O'Leary, D. E. (2004). Enterprise resource planning (erp) systems: an empirical analysis of benefits. *Journal of emerging Technologies in Accounting*, 1(1):63–72.

Safran, C. and Goldberg, H. (2000). Electronic patient records and the impact of the internet. *International journal of medical informatics*, 60(2):77–83.