Portfolio Project: Analysis of Algorithms and Data Structures

Donghwan Kim

Colorado State University Global

CSC 506: Design and Analysis of Algorithms

Dr. Lori Farr

June 12, 2022

Portfolio Project: Analysis of Algorithms and Data Structures

This Portfolio Project is concerned with data structures and operations for multidimensional lists which become popular in the emerging fields of modern machine learning and data science. Fortran 90/95 provides built-in data structure of multi-dimensional arrays both in static and dynamic array but has some limitations to be used. Python programming language provides only one-dimensional dynamic list which can contain multiple data types in a list. As will be shown, however, Python built-in data structures can be used to design multi-dimensional lists in various ways. This project discusses possible data structures of multi-dimensional lists in Python, defines operations to the data structure candidates, and compares the performance of the different data structures and their operations. Specifically, it implements five different data structures of multi-dimensional lists based on Python built-in data structures like list, nested list, dictionary, and so on and tests performance on how fast they can insert data. A simulation-based performance test shows that a python list with nested dictionary performs best among the five different data structures. A data structure with separate lists, which is commonly used in Fortran 90/95, performs well in both data insertion and group-by search-and-calculation operation.

Data Structures for Multi-dimensional Lists

Multi-dimensional lists become one of the core data structures in the emerging fields of artificial intelligence and data analytics. To properly work with data in the fields, a data structure should contain various data types of numbers, strings, booleans, and so on. For example, it should be possible to contain both image data and text data in a multi-dimensional list, which is essential in many applications of artificial intelligence and machine learning.

In general, a 2-dimensional list is denoted X with $m \times n$ elements where m is the number of rows and n is the number of columns. It is usually expressed as X[i,j] for $i = 1, 2, \cdots, m$ and $j = 1, 2, \cdots, n$. The elements can be accessed with the index [i, j] for the i'th row and j'th column. The index of 3-dimensional arrays has the form of [i, j, k]. Fortran 90/95

programming language provides multi-dimensional array data structure up to 7 dimensions but has some limitations (Wagener 1998). Fortran 90/95 declares a two-dimensional array with $DIMENSION(m, n)$. The 1-dimensional array is considered as a sequence of elements like Python list. Once the 1-dimensional array like the Python list is extended to higher-dimensional lists, there are lots of considerations in designing data structure and operations. The order of storage also affects time complexity. The update(X) operation is possible, for example, with X[i, j] for each elements or X[:,j] for all the elements in the j'th column. In this way, more high-dimensional lists are possible.

However, Fortran 90/95 does not allow different data types in a multi-dimensional array (Van Rossum and Drake Jr 2009; Python Documentation 2022). All the columns should be declared as same data type. Numbers can not be with strings, for example. One way to incorporate multiple data types is to use separate arrays by data type. It will add complexity in designing operations. To serve as a data container, a multi-dimensional list has to contain various types of data including numbers and strings. However, all the elements in a column should be in the same type. Both numbers and strings can not be in the same column. In the sense, Excel spreadsheets have drawbacks. Python does not provide multi-dimensional list (Van Rossum and Drake Jr 2009). However, as will be shown, the multi-dimensional list can be implemented in various ways with Python built-in data structures.

Implementing Five Different Data Structures

The five different data structures are designed and implemented with Python. The multi-dimensional list can be constructed with n separate Python lists each of which has m elements. The first data structure named DS1 is as follows:

$$\textbf{DS1}.\ \ X[:,0], X[:,1], X[:,2], \cdots, X[:,n-1]$$

where $X[:,j] = [X[0,j], X[1,j], \cdots, X[m-1,j]]$ for $j = 0, 1, \cdots, n-1$. It is commonly used in Fortran 90/95 and as will be proven, it performs well in terms of time efficiency. The data

structure named DS1 can be implemented with five Python list where NObs is the number of rows. Data is created with the Python random module.

```
Algorithm (DS1).
1.  XOne = [None] * NObs
2.  XTwo = [None] * NObs
3.  XThree = [None] * NObs
4.  XFour = [None] * NObs
5.  YData = [None] * NObs
6.
7.  #Insert data
8.  for i in range(NObs):
9.  XOne[i] = float(random.randint(0,4))
10. XTwo[i] = random.random()*100
11. XThree[i] = float(random.randint(0,1))
12. XFour[i] = random.random()*50
13. YData[i] = 1 + 3 * XOne[i] + 5 * XTwo[i] - 6 * XThree[i] +
14.            9 * XFour[i] + random.random()
```

Algorithm (DS2) shows how a multi-dimensional list can be implemented in a list. The data structure DS2 has the length of NObs * NVar in column-major order where NVar is the number of columns. The first column X[:,0] is stored in the first DS[0:NObs-1]. The second column X[:,1] is stored in the DS[NObs:(NObs*2) - 1]. An alternative is to contain all the

elements in a list

DS2.

$$[X(0,0), X(1,0), X(2,0), \cdots, X(m-1,0),$$
$$X(0,1), X(1,1), X(2,1), \cdots, X(m-1,1),$$
$$X(0,2), X(1,2), X(2,2), \cdots, X(m-1,2),$$
$$\vdots,$$
$$X(0,n-1), X(1,n-1), \cdots, X(m-1,n-1)]$$

```
Algorithm (DS2).
1. DS2 = [None] * NObs * NVar
2. for i in range(NObs):
3.     DS2[i] = float(random.randint(0,4))
4.     DS2[i + NObs] = random.random()*100
5.     DS2[i + NObs*2] = float(random.randint(0,1))
6.     DS2[i + NObs*3] = random.random()*50
7.     DS2[i + NObs*4] = 1 + 3 * DS2[i] + 5 * DS2[i + NObs] -
8.                       6 * DS2[i + NObs*2] +
9.                       9 * DS2[i + NObs*3] + random.random()
```

Python nested list can also be used to implement the data structure in two different ways:

DS3. $[X[:,0], X[:,1], X[:,2], \cdots, X[:,n-1]]$

DS4. $[X[0,:], X[1,:], X[2,:], \cdots, X[m-1,:]]$

where $X[i,:] = X[i,0], X[i,1], \cdots, X[i,n-1]$ for $i = 0, 1, \cdots, m-1$. The data structure DS3 is implemented in Python using Algorithm (DS3) which uses a Python nested list. Data can be accessed with DS3[j][i]. That is, the first nested list in DS3 contains the first

column of X. Unlike the DS3, the data structure DS4 stores data row-by-row (Lutz 2013). Simulation-based performance tests show that it does not perform well.

```
Algorithm (DS3).
1. DS3 = [[None]*NObs for _ in range(NVar)]
2. for i in range(NObs):
3.     DS3[0][i] = float(random.randint(0,4))
4.     DS3[1][i] = random.random()*100
5.     DS3[2][i] = float(random.randint(0,1))
6.     DS3[3][i] = random.random()*50
7.     DS3[4][i] = 1 + 3 * DS3[0][i] + 5 * DS3[1][i] -
8.                 6 * DS3[2][i] +
9.                 9 * DS3[3][i] + random.random()
```

```
Algorithm (DS4).
1. DS4 = [[None]*NVar for _ in range(NObs)]
2. for i in range(NObs):
3.     DS4[i][0] = float(random.randint(0,4))
4.     DS4[i][1] = random.random()*100
5.     DS4[i][2] = float(random.randint(0,1))
6.     DS4[i][3] = random.random()
7.     DS4[i][4] = 1 + 3 * DS4[i][0] + 5 * DS4[i][1] - 6 * DS4[i][2] +
8.                 9 * DS4[i][3] + random.random()
```

The data structure DS5 uses dictionaries in a list. Python dictionary is a good data

structure for data lookup (Lysecky and Vahid 2019).

DS5.

$$[\{`col0' : X[0,0], `col1' : X[0,1], \cdots, `colLast' : X[0,n-1]\},$$
$$\{`col0' : X[1,0], `col1' : X[1,1], \cdots, `colLast' : X[1,n-1]\},$$
$$\{`col0' : X[2,0], `col1' : X[2,1], \cdots, `colLast' : X[2,n-1]\},$$
$$\vdots$$
$$\{`col0' : X[m-1,0], `col1' : X[m-1,1], \cdots, `colLast' : X[m-1,n-1]\}$$

```
Algorithm (DS5).
1. DS5 = [None] * NObs
2. for i in range(NObs):
3.     v0 = float(random.randint(0,4))
4.     v1 = random.random()*100
5.     v2 = float(random.randint(0,1))
6.     v3 = random.random()
7.     v4 = 1 + 3 * v0 + 5 * v1 - 6 * v2 + 9 * v3 + random.random()
8.
9.     DS5[i] = {'C1': v0, 'C2': v1, 'C3': v2, 'C4': v3, 'C5': v4}
```

Simulation-Based Performance Tests

This project conducts simulation-based performance test for the five different data structures of multi-dimensional lists and their operations. The first simulation is done to test how fast data can be inserted into the five data structures. The number of columns tested are 100,000, 250,000, and 500,000. Below shows sample mean, sample median, and sample standard deviation for 500 samples (Casella and Berger 2002: for more details on simulation). The findings show that data structure 5 and 1 perform better but data structure 4 does not perform well.

```
-----------------------------------------
Data structure  Mean   Median   Sta. dev.
-----------------------------------------
Length of data 100000
DS 1       0.313   0.312   0.041
DS 2       0.367   0.359   0.045
DS 3       0.334   0.328   0.044
DS 4       0.413   0.406   0.044
DS 5       0.318   0.312   0.045
Length of data 250000
DS 1       0.821   0.781   0.125
DS 2       0.963   0.906   0.144
DS 3       0.884   0.828   0.165
DS 4       1.211   1.187   0.189
DS 5       0.735   0.703   0.089
Length of data 500000
DS 1       1.605   1.562   0.178
DS 2       1.873   1.828   0.176
DS 3       1.672   1.656   0.158
DS 4       2.433   2.406   0.197
DS 5       1.433   1.405   0.131
-----------------------------------------
```

One of the powerful operations with a multiple-type multi-dimensional list is group-by operations. For example, it is used to calculate country-level city populations by year. Here group becomes country and year. The underlying mechanism in this case is that (1) sort city-population data by country and year, (2) aggregate all the city pop data by the country-year group, and then (3) print out the country-level urban population by the group. This

project tests this kind of operation with the five data structures. An algorithm below shows the operation with the above data structure 1, that is, Algorithm (DS1). This operation can be designed for other data structures as shown in the attached source code. The performance test shows that data structure 1 performs better but data structure 4 and 5 does not perform well.

Algorithm.

```
1.  XOne_unique = list(set(XOne))
2.  XThree_unique = list(set(XThree))
3.  YByGroup = []
4.  for XValOne in XOne_unique:
5.      for XValThree in XThree_unique:
6.          YVal = []
7.          for i in range(NObs):
8.              if XOne[i] == XValOne and XThree[i] == XValThree:
9.                  YVal.append(YData[i])
10.         YVal_mean = sum(YVal)/len(YVal)
11.         YByGroup.append([XValOne, XValThree, YVal_mean])
```

Conclusions

Multi-dimensional lists are essential in data-centered Artificial Intelligence and Machine Learning where time efficiency matters. In this Portfolio Project, five different data structures for the multi-dimensional list are implemented with Python built-in data structures including list, nest list, and a mix of list and dictionary. Simulation-based performance tests show that (1) for the data insertion operation, the two data structures which perform better in terms of time efficiency are one with a mix of list and dictionary and one with separate lists and (2) for the group-by operation, the well-performing data structure are one with separate lists. The separate lists for columns of multi-dimensional lists, commonly used in Fortran 90/95, is shown to perform well in Python.

References

Burden, A. M., Burden, R. L., and Faires, J. D. (2016). *Numerical analysis, 10th ed.* Cengage Learning.

Casella, G. and Berger, R. L. (2002). *Statistical inference*. Belmont, CA: Duxbury.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to algorithms*. The MIT Press.

Lysecky, R. and Vahid, F. (2019). Design and analysis of algorithms. In Lysecky, R. and Vahid, F., editors, *Data structures essential: Pseudocode with python examples*. Zybooks.

Lutz, M. (2013). *Learning Python*. O'Reilly Media.

Van Rossum, G. and Drake Jr, F. L. (2009). *Python tutorial*. Scotts Valley, CA: CreateSpace.

Wagener, J. (1998). *Fortran 90 concise reference*. Absoft Corporation.

Python Documentation (2022). Python 3.10.4 documentation: The tutorial. `https://docs.python.org/3/`.