Tripartite Integer Partitioning Problem

Matthew Borkowski, Nadeem Ahmad, Yousef EL-Qawasmi, Hatim Khan, Dr. Ilias Kotsireas

Department of Physics and Computer Science, Wilfrid Laurier University, Waterloo ON

PROBLEM

Given three large text files consisting of millions of rows of integers and 50 columns, find a set of rows in each text file so that they add up to a given sum λ

INPUT:

- A positive integer λ(sample value: λ = 100)
- 3 text files A, B, C, with k columns and 10 million rows each(sample value: k = 40)

OUTPUT:

- 3 rows LA, LB , LC in files A, B, C respectively, s.t.

$$\begin{cases} LA(1) + LB(1) + LC(1) = \lambda \\ \cdots\cdots\cdots \\ LA(k) + LB(k) + LC(k) = \lambda \end{cases}$$

SOLUTION

1. For each column of A, B, and C:
   a. Create a hash table for each column with the values as keys and empty lists as values
   b. Loop through the column and add the index of each value to its corresponding list in the hash table
   c. Loop through the hash tables for A and C and use an equation to find the potential value of B
   d. Check if the potential value of B is in the hash table for B. If it is, add the tuple (index of A value, index of B value, index of C value) to a set of potential solutions.
2. Repeat steps 1a-1d for the second and subsequent columns of A, B, and C, but only hash the values that were found in the intersection of the previous columns.
3. Return the set of potential solutions.

The idea behind the algorithm is to use hashing to make it efficient to check if a value exists in a column of a file, and to use the intersection of the previous columns to narrow down the range of values to hash in the current column. By doing this, the algorithm avoids having to loop through all the values in a column for every combination of A, B, and C

RESULTS

- Cut down the potential rows to on average, 1% of the original size
  - This method can be applied to future data matching algorithms
- Provides valid solutions to the 3 way matching problem

TECHNOLOGIES

PREVIOUS ALGORITHMS

Various algorithms were tried:

- **Column-driven Algorithm**
  - work on each column separately, in order to break up the problem is several sub-problems, using the fact that the equation a + b + c = λ
- **The mod 10 hash function approach**
  - As a pre-processing step, one can apply the mod 10 hash function to every element of the input files, i.e. retained the last digit only
- **Hash-Threading Algorithm**
  - uses threading and hashing to create a 2D array to store the first column indexes of File B, allowing for O(1) lookup. Potential combinations of A, B, and C are checked using threads to find elements that equal to λ

code snippet for the algorithm to find potential paths

```
def check_potential_paths(a_hash,b_hash,c_hash):
    potential_array=[]
    faila=[]
    failc=set()

    for i in range(len(a_hash)):
        if a_hash[i]=='':
            faila.append(i)
            continue

        for j in range(len(c_hash)):
            if c_hash[j]=='':
                failc.add(j)
                continue

            b_val=lam-j-i
            if b_hash[b_val]!='':
                potential_array.append((i,b_val,j))
            else:
                continue


    return potential_array
```

algorithm for hashing the first column of file A

```
for i in range(len(A_col)):
    value=A_col[i]

    if a_hash[value]== "":
        a_hash[value]=a_hash[value]+str(counterA)
        counterA+=1
    else:
        a_hash[value]=a_hash[value]+","+str(counterA)
        counterA+=1
```

```
for i in A_col:
    if counterA in a_constraint:
        value=A_col[counterA]

        if a_hash[value]== "":
            a_hash[value]=a_hash[value]+str(counterA)
            counterA+=1
        else:
            a_hash[value]=a_hash[value]+","+str(counterA)
            counterA+=1
    else:
        counterA+=1
```

Second hashing algorithm using constraints to narrow down data

REFERENCES

Kotsireas, Ilias. “Efficient Algorithms for Matching Problems.” Cargo Lab, Wilfrid Laurier University

Nabizadeh, Hadi, and Derek Eager. "An Efficient Approach to Finding Triangles in Large-Scale Graphs." Journal of Experimental Algorithmics 18 (2013): 1-16. https://doi.org/10.1145/2498435.2498439.

Gonnet, Gaston H., and Timothy G. Griffin. "A New Three-Pivot Quicksort, with Applications to Computing Convex Hulls in 3D." Proceedings of the 26th Annual Symposium on Computational Geometry, ACM Press, 2010, pp. 346-353. https://people.csail.mit.edu/virgi/6.s078/papers/real3sum.pdf.

GITHUB