Summer of Bitcoin - Challenge Report

Sahil Jitesh Gangurde

Indian Institute of Information Technology, Gwalior, India

Abstract. The challenge consisted of a mempool file which had some transactions along with their fee, weight and their parent transaction ids. The challenge was to create a block of these transactions such that the block would favour the bitcoin miners such that for each block created miner would get maximum total fee. The constraint for this challenge was the weight, i.e. for each block created the total weight of the transactions contributing to the block should not be more than 4,000,000. The solution involved using sorting with DSU data structure. The algorithms first creates the graphs of parent-child transactions and then sorts their parents according to the fee/weight ratio. This optimization technique provided and overall fee profit of 5,547,206 and a total weight of 3,998,644 in the block.

1 Naive Approach

Sorting the non-parent transactions based on fee The most naive and sensible optimization technique is to sort the non-parent transactions with respect to the fee and then pick the transactions having greater fee until we hit the maximum weight upper bound.

This method gives a miner's profit of 2,884,784 which is less as the total fee in the whole data is 7,485,591. Hence a better approach was necessary.

2 Optimized Naive Approach

2.1 Drawbacks of above mentioned methods

The biggest drawback of the above mentioned method was that if the fee is high but also the weight is also high then it fills up the block quickly. In fact on the given data it fills only <50 transaction entries. To overcome this drawback weight factor is needed to be involved as it accounts for the decision making process of selecting a transaction to create a block.

2.2 Inspiration from Knapsack

The main objective of this challenge is to select the transactions such that the ratio fee/weight is as greater as possible. to achieve this a technique similar to knapsack was used.

Instead of sorting the transactions based on fee I created a custom comparator which takes two entries of transactions and sorts them on the basis of the fee/weight ratio. Below is the code snippet of the comparator:

```
    double A = fee_a/weight_a;
    double B = fee_b/weight_b;
    return (A-B) > ((fabs(A)<fabs(B)?fabs(B):fabs(A))*EPS);
```

This resulting in entries of 2,673 transactions. This also resulted in a total fee collection of 4,803,206 which is 26% improvement over the naive approach.

3 Drawbacks of the above approaches

I approached the whole problem keeping aside the parent-child relation and neglecting the transactions having those relationships. but considering the fact that not whole mempool entries will have these relationships so there will always be at least 1 transaction which would be independent and will be counted by the algorithm. But this does gives a loss of fee profit as maximum transaction are not processed and hence may lead to loss.

4 Improving the drawbacks

I tried to apply the above technique on the connected components of transactions. Here transactions having parent-child relation can be accounted for a single forest (Disjoint Set terminology) and hence they can be then sorted using the above sort function. The image (fig 1) shown below clearly depicts what the forest would look like along with non-parent transactions.

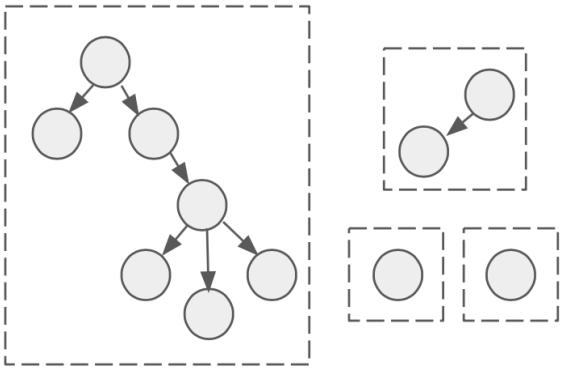

Fig. 1. Each rectangular box represent one connected component and in each connected component there is a parent child transaction node (parent pointing towards the child)

The parents of the connected components would be sorted according to the method mentioned above and then one with the greatest value of fee/weight would be chosen. Once this process is done then is the connected component is no empty then there would be rearrangement in the parent node and the

connected component would spit into 'n' components where 'n' is the number of children which the picked parent has. Then again sort the parents of connected components according to the above method and repeat the same process until the weight¡max allowed block weight.

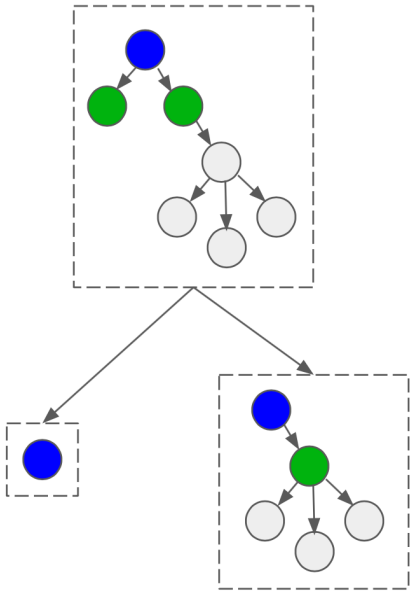

Fig. 2. Blue node represents the main parent transaction in each connected component and the green nodes represent the immediate child transactions of the the parent. Once the parent is picked the connected component splits into 'n' components where the new parents are the previous children of the ancestor connected component.

5 Conclusion

The optimization methods discussed above are well tested and can be technically used to increase miner's profit. But we also have to keep in mind the time complexity of each of these algorithms such that they do not under perform asymptotically under very large mempool transaction entries.