

Context

A compact and scalable representation of the immersive contents (which are acquired from the 360 degree camera or sensors like LIDAR) is done using point clouds. To have this compact and scalable representation, 3D vector quantization methods have been implemented using the approach of dividing the points cloud into hierarchy of truncated lattices. TSLVQ is an unbalanced tree structure using fast lattice vector quantization to sort data. The tree is a packing of embedded truncated lattice. A VQ is a function that map a vector X to a corresponding quantized vector.

For example consider the combination using two quantization methods:- $2 \times 2 \times 2$ and $3 \times 3 \times 3$. The splitting of a cell is based on the maximum distortion vs rate criteria compared from the two methods. Maximum rate limit is provided for stopping the splitting process. For compression after splitting, the points in a leaf are considered as a single point thus minimizing the data points. For reconstruction of the object multiple methods are studied and taking into comparison.

Lagrangian Method

To construct the tree structure a criteria is defined, λ , as being the ratio between delta distortion and delta rate for each node. Distortion is the sum of distance from the lattice point to all assigned objects inside the corresponding voronoi cell. Rate is the cost of coding all elements as one lattice element. When the rate increases, the precision decreases. The total distortion and rate of the tree is the sum of all distortion and rate values of leaves inside the tree. The λ representing the change in quality brought by one leaf quantization, as being a trade off between coding costs (rate) and closeness to the representation (distortion). $\lambda(n_i) = \Delta d(S_{n_i}) / \Delta r(S_{n_i})$.

Greedy approach: The root node, containing all the data, is quantized recursively.

Leaves are quantized by maximum λ order. In other words, leaves where the quantization offers the most decrease in distortion and least increase in rate. By using λ criteria in TSLVQ construction, distortion of the representation can be coupled with rate consideration. The ratio of the two metrics is the trade off between resolution and cost of coding the compressed points cloud.

Quantization Algorithm

For each quantization of the tree, the best leaf must be found first which is the lattice, where the λ generated from quantization is the highest. The evaluation of the λ value is done once per leaf but it needs to be updated under certain circumstances. The evaluation of newly added leaves from the previous step (initialization or quantization) are done before the start of the next quantization stage and we store the fresh leaves generated as pending leaves.

The evaluation of the quantization is done by applying the quantization process on the pending leaves. Computation of distortion and rate are then followed by the computation of change of distortion and rate, to obtain the λ value. This λ is used to estimate the benefit of the quantization step on the tree structure. For each new leaf, all the quantization method will be applied on a temporary node (copy of the leaf). The quantization with the best λ result will be chosen, and all the resulting quantization process will be saved inside the node. The modified node will still be seen as leaf by using the `isLeaf` variable but their splitting has already occurred. At quantization of best leaf the `isLeaf` state is changed to false to the leaf becomes a node, and all children are registered inside the pending leaves list since they are the fresh leaves generated.

Partition State

Partition state is a bitstream associated with each node. Each node is divided into several regions depending on the quantization scheme being tested and every region has a representant associated with them. Eg: 8 representants / ids for 2x2x2 and 27 for 3x3x3. All the children of a particular splitted node have a representant associated with them which corresponds to representant of the region. It is the id of the region where the child lies inside the node. So if one of the child of a node undergoes splitting, we update the partition state of the node. The partition state tells us using 0s and 1s the regions that are splitted. Eg: for 2x2x2 partition state = 01001101. Regions with representant = 1,4,5,7 are splitted and regions with rep = 0,2,3,6 are not splitted. Thus the region with representant equal to index of "1s" in partition state are splitted and others unsplitted.

Entropic Rate

The rate criteria is calculated using a probabilistic approach according to entropic algorithm. Every partition state has a probability occurency associated with it which tells how likely the state occurs in the tree. We calculate $\text{Rate}(\text{node}) = -\log_2(\text{Probability}(\text{partition_state}))$ which is the uncertainty of partition state. The idea is to have narrow and sharp probability distribution histogram for partition states. Less the spread in the distribution more efficient is the entropic algorithm. Entropic coding thus helps in achieving loss-less compression of the bitstream generated after traversing and coding the tree. Compression during quantization is lossy and compression using arithmetic code is loss-less.

Lambda Sorted

This has a list of all the leaves with there respective lambda scores since each leaf of the tree is a possible contender for splitting. The rate used to calculate lambda score uses the probability occurency of the partition state corresponding to parent of each leaf. When a new partition state is introduced which was not existing in probability table, the total number of states increase by 1. Hence we need to update completely all the values in lambda sorted. But when a new state is created and old one removed, the total number of states remain the same hence apart from the 2 states involved all other states do not undergo change in probability occurency. So we can use dynamic programming approach and keep the complete lambda sorted unchanged.

Hashing

To store the probabilities of all the partition states, we use a hash table. Advantage of this strategy is that partition state of any length can be stored in the table. Due to this, we are able to run several quantization methods in competition with each other. Lagrangian method works the same way as before but locating probability occurency of any state becomes easy due to hashing.

1) The probability table consists of :-

- At every index of hash table we have a chain.
- Every chain consist of a list of states.
- Every state in the chain has information about :-
 - name of partition state
 - number of times it occurs in the tree
 - probability occurency

2) The hashing function:- Consider a partition state [0,1,0,1,0,0,1,1].

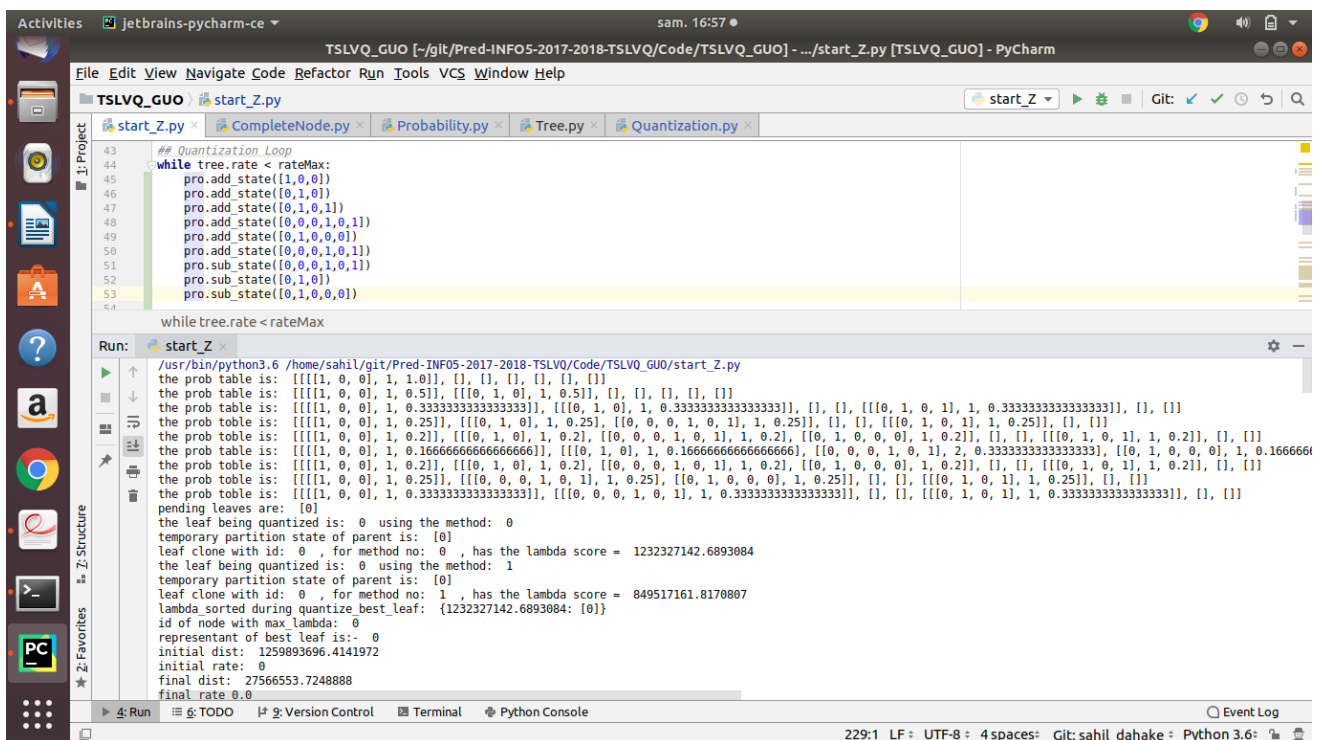
- Note that 1 occurs at index no : 1,3,6,7.
- Then we sum all these index where 1 occurs.
- Result is 17.
- Then finally take modulo 37 since the size of the hash table is 37.
- Partition state [0,1,0,1,0,0,1,1] will be stored in the chain at index 17.

3) After passing a state through the hash function, we get an index which determines the position of partition state in the table.

4) After locating the partition state in the respective chain, we increase or decrease the counter of the state.

5) After performing add_state or sub_state we have to update probabilities of all entities since total states have changed.

6) If the counter of the state goes to zero then we remove it. If the state is not found in the chain we create a new one and append it in the chain.



```
## Quantization Loop
while tree.rate < rateMax:
    pro.add_state([1,0,0])
    pro.add_state([0,1,0])
    pro.add_state([0,0,1])
    pro.add_state([0,0,0,1])
    pro.add_state([0,1,0,0])
    pro.add_state([0,0,1,0])
    pro.add_state([0,0,0,1,0])
    pro.add_state([0,0,0,1,0,1])
    pro.sub_state([0,1,0])
    pro.sub_state([0,1,0,0])

while tree.rate < rateMax
/usr/bin/python3.6 /home/sahil/git/Pred-INFO5-2017-2018-TSLVQ/Code/TSLVQ_GUO/start_Z.py
the prob table is: [[[[[1, 0, 0], 1, 1.0]], [[], [], [], [], [], []]], [[], [], [], [], [], []]]
the prob table is: [[[[[1, 0, 0], 1, 0.5]], [[0, 1, 0], 1, 0.5]], [[], [], [], [], [], []]], [[], [], [], [], [], []]]
the prob table is: [[[[[1, 0, 0], 1, 0.3333333333333333]], [[0, 1, 0], 1, 0.3333333333333333]], [[], [], [[0, 1, 0, 1], 1, 0.3333333333333333]], [[], []]]
the prob table is: [[[[[1, 0, 0], 1, 0.25]], [[0, 1, 0], 1, 0.25]], [[0, 0, 0, 1, 0, 1], 1, 0.25]], [[], [], [[0, 1, 0, 1], 1, 0.25]], [[], []]]
the prob table is: [[[[[1, 0, 0], 1, 0.2]], [[0, 1, 0], 1, 0.2]], [[0, 0, 0, 1, 0, 1], 1, 0.2]], [[0, 1, 0, 0, 0], 1, 0.2]], [[], [], [[0, 1, 0, 1], 1, 0.2]], [[], []]]
the prob table is: [[[[[1, 0, 0], 1, 0.16666666666666666]], [[0, 1, 0], 1, 0.16666666666666666]], [[0, 0, 0, 1, 0, 1], 2, 0.3333333333333333]], [[0, 1, 0, 0, 0], 1, 0.16666666666666666]]
the prob table is: [[[[[1, 0, 0], 1, 0.2]], [[0, 1, 0], 1, 0.2]], [[0, 0, 0, 1, 0, 1], 1, 0.2]], [[0, 1, 0, 0, 0], 1, 0.2]], [[], [], [[0, 1, 0, 1], 1, 0.2]], [[], []]]
the prob table is: [[[[[1, 0, 0], 1, 0.25]], [[0, 0, 0, 1, 0, 1], 1, 0.25]], [[0, 1, 0, 0, 0], 1, 0.25]], [[], [], [[0, 1, 0, 1], 1, 0.25]], [[], []]]
the prob table is: [[[[[1, 0, 0], 1, 0.3333333333333333]], [[0, 0, 0, 1, 0, 1], 1, 0.3333333333333333]], [[], [], [[0, 1, 0, 1], 1, 0.3333333333333333]], [[], []]]
pending leaves are: [0]
the leaf being quantized is: 0 using the method: 0
temporary partition state of parent is: [0]
leaf clone with id: 0 , for method no: 0 , has the lambda score = 1232327142.6893084
the leaf being quantized is: 0 using the method: 1
temporary partition state of parent is: [0]
leaf clone with id: 0 , for method no: 1 , has the lambda score = 849517161.8170807
lambda_sorted during quantize_best_leaf: {1232327142.6893084: [0]}
id of node with max_lambda: 0
representant of best leaf is:- 0
initial dist: 1259893696.4141972
initial rate: 0
final dist: 27566553.7248888
final rate 0.0
```