Algorithm 1 Cost Evaluation.

Input: Addition fee *additionFee*, time step fee *timeFee*, error propagation fee *propagationFee*

```
procedure COST_EVALUATION(closeNode, equation)
    $costs \leftarrow \emptyset$
    $costs.replacement \leftarrow$ evaluateReplacement(closeNode, equation)
    $costs.addition \leftarrow$ evaluateAddition(closeNode, equation)
    $costs.merge \leftarrow$ evaluateMerge(closeNode, equation)
    $costs.preMerge \leftarrow$ evaluate_preMerge(closeNode, equation)
    $costs.postMerge \leftarrow$ evaluate_postMerge(closeNode, equation)
    $lowestCost \leftarrow costs.getLowestCost()$
    return $lowestCost$
end procedure
```

Algorithm 2 Evaluate Change.
Input: *changedEquation* represents the new fitted function from the combination of the previous fitted function values and its timeSteps.

```
procedure EVALUATE_CHANGE(closeNode, changedEquation)
    $nodes \leftarrow$ getRelatedNodesFromHistory(closeNode)
    $timeSteps \leftarrow changedEquation.timeSteps$
    $improvements \leftarrow \emptyset$
    for each $node \in nodes$ do
        $nodeEquation \leftarrow node.getEquation()$
        $nodePoints \leftarrow nodeEquation.evaluatePoints(timeSteps)$
        $currentSimilarNode \leftarrow node.getSimilarNode()$
        $currentDistance \leftarrow node.getDistance()$
        $changedPoints \leftarrow changedEquation.evaluatePoints(timeSteps)$
        $newDistance \leftarrow$ getEuclideanDistance(observedPoints, changedPoints)
        $improvement \leftarrow newDistance - currentDistance$
        $changes \leftarrow$ storeChanges(currentSimilarNode,changedEquation, newDistance)
        $improvements \leftarrow improvements + improvement$
    end for
     return {changes : changes, improvements : improvements}
end procedure
```