A MORE DETAILS OF OUR KD-TREE-BASED INDEX SOLUTION

• `KDT-tree` **construction.** Given a list of BC-points L, we can construct the `KDT-Index` level by level in a top-down manner: we first choose the dimension D and select the BC-point with medium coordinate in the dimension for the root, and then go through L to get the A, S for the root. Afterward, the unselected BC-points in L will be sent to the children of the root, and we continue to construct the `KDT-index` in the children nodes.

• `KDT-tree`**-based binary δ-temporal triangle counting.** Given a time window $[t_s, t_e]$ and a duration δ, the counting result equals the summarized R of nodes in the `KDT-index` whose A is contained by the cube $[t_s, t_e] \times [t_s, t_e] \times [0, \delta]$. We apply a recursive method starting from the root to find the counting result, as shown in Algorithm 6. Initially, we let *node* be the root of `KDT-Index` (line 1). Then, we use a function `KDsum` to sum the R values of nodes whose A is contained by the cube (line 2). If the cube A of the current node is contained by the query cube, we return the R of the node (line 3). If the intersection of the cube and A is empty, we return 0 (line 5); otherwise, we continue the recursion in the children nodes (line 6).

Algorithm 6: Sum the R values of nodes in a cube

Input: `KDT-Index`, a query cube $[t_s, t_e] \times [t_s, t_e] \times [0, \delta]$
Output: Total R value of nodes whose A is contained by the cube

1 *node* ← root of `KDT-Index`;
2 **Function** `KDsum`(*node*):
3   **if** $node.A \subseteq [t_s, t_e] \times [t_s, t_e] \times [0, \delta]$ **then return** $node.S$;
4   **else**
5   **if** $node.A \cap [t_s, t_e] \times [t_s, t_e] \times [0, \delta] = \emptyset$ **then return** 0;
6   **else return** `KDsum`($node.lChild$) + `KDsum`($node.rChild$);

Based on analysis of [34], we can conclude that Algorithm 6 completes in $O(\Delta^{\frac{2}{3}})$ time.

• `KDT-tree` **maintenance.** For the dynamic temporal graph, when a new BC-point is generated, we update the `KDT-Index` starting from the root node in a top-down manner. Specifically, we first update the cube A and sum R in the current node, and then compare the coordinates of the new BC-point and the BC-point of the current node in the selected dimension. If the new BC-point has a larger value in the selected dimension, it will be in the right child to continue the update; otherwise, it will be in the left child.