Algorithm **MinDist**
Input: Node t
Output: Pair list P of size k;
Object list B of size $O(\sqrt{k})$
1. TB := list of objects in t of size $O(\sqrt{k})$
2. c := number of children of t
3. **for** i = 1 to c
4. $CB[i], CP[i]$:= MinDist($t.child[i]$)
5. Add $t.edge[i]$ cost to each object in $CB[i]$
6. **end for**
7. B := Merge($CB[1], \ldots, CB[c], TB$)
8. TP := GenPairs(B)
9. P := Merge($CP[1], \ldots, CP[c], TP$)

Figure 5: The MinDist algorithm.

second kind, we need a list of objects that are close to the subtree of the children nodes. The lists can then be joined to generate the necessary object pairs. Thus, the MinDist algorithm computed at the root of the ontology returns the top-k pairs.

As shown in Figure 5, each node t maintains two lists: (i) a list of pairs of objects P ordered by their d_{min} distances; and (ii) a list of objects B ordered by their minimum distances d_i to the node t. The length of P is at most k. The length of B should be enough to ensure that k distinct pairs of objects can be generated from B. The number of terms required to do that is $k' = O(\lceil\sqrt{k}\rceil)$.[8]

When MinDist is called on a node t, it selects k' objects from its list L into TB. L is the list of objects associated with that term. MinDist is then called on each of its c children. The cost of the edge from t to its child is added to the objects in the corresponding child's object list (line 5). This is done to ensure that the distances are maintained correctly. The c sorted object lists and the list of objects in t are then merged to produce the sorted list B.

The merging (line 7) is done using a heap data structure [4]. The heap is initialized with $c + 1$ elements at position 1 of each of the child lists and the list TB. The minimum element is then extracted into B. Since all the individual lists are sorted, the properties of heap guarantee that the object extracted has the least d_{min} distance from this node. The object at the next position of the list from where this minimum object came is then inserted into the heap. This is repeated k' times.

All the possible k pairs are then generated from the k' objects in B (method GenPairs in line 8). This list TP computes the k best distances of the object pairs which are not in any of the subtrees.

TP is finally merged with the pair lists $CP[i], i = 1 \ldots c$ from the children to produce the final pair list P using a heap in the same manner as above (line 9).

5.1 Analysis of Time Complexity

In this section, we analyze the time and space complexities of the MinDist algorithm.

We first analyze the time required to compute the inverted index. The object descriptions are read once, and for each term in an object, the corresponding list is accessed in $O(1)$ time using hashing, and the object is inserted at the top of list in another $O(1)$ time. The total time required for this phase is, therefore, $O(D)$.

We next analyze the running time for the main phase of the algorithm. Selecting k' objects in TB requires $O(k')$ time. Adding the child edge costs to each object in CB lists takes $O(k'c)$ time.

At every step of the merging operation, the object with the minimum distance is extracted from the heap and another object is inserted. The size of the heap is, therefore, never more than $O(c)$. Extracting the minimum element and inserting another object into the heap takes $O(\log c)$ time. Since the operation is repeated k' times, the total running time of the merging procedure is $O(k' \log c)$.

If, however, the objects in the child lists are not unique, k' operations may not be enough to select k' different objects. Thus, a hashtable is used to ensure that an object is inserted into the heap only once. First, all the lists are scanned in $O(k'c)$ time. If an object appears for the first time, it is inserted into the hashtable with the object identifier as the key and the distance as the value. If an object appears twice, the one with the minimum distance is maintained in the hashtable. Before any object is inserted into the heap, the hashtable is checked. If this object is different from the one maintained in the hashtable, then there exists another copy of this object with a smaller distance. Hence, this object does not need to be considered. This limits the number of heap operations to $O(k')$. Assuming that the hashtable operations take constant time, the running time then is $O(k' \log c)$.

Sorting k local object pairs requires $O(k \log k)$ time.

Finally, the sorted pair lists at the node and the children are merged in $O(kc + k \log c)$ time using a heap and a hashtable in a similar manner as before.

Thus, the total running time of the MinDist algorithm at a node with c children is $O(kc + k \log k)$.

The algorithm is run once at each node of the ontology. Assuming that there are T terms in the ontology, the total number of children for *all* the nodes is $O(T)$. Hence, the

[8] Since $k'(k'-1)/2 \geq k$, the actual number of terms required is $k' = \lceil 1/2 + \sqrt{1/4 + 2k}\rceil$.