Brief Announcement: Self-adjusting Networks based on SkipList

Anonymous author
Anonymous affiliation

1 Algorithms

In this Section, we give the description of networks based on `SkipList` and `SplayList`.

`SkipList` is a search data structure where each node has some height and they are linked with the neighbours on each level.

`SplayList` is the self-adjusting version of the `SkipList`: we go from the head to the target node and increment the counters; if the counters satisfy some condition we either increase or decrease the height of the corresponding node.

▶ **Algorithm 1** (SkipListNet). *Suppose we are asked the routing request between u and v (for simplicity, $u < v$). Our algorithm performs two phases: 1) go to the left and up until we get the next key bigger than v; 2) do the search request for v there. The example of the request is shown on Figure 1.*

This is the static algorithm, i.e., the network does not change.

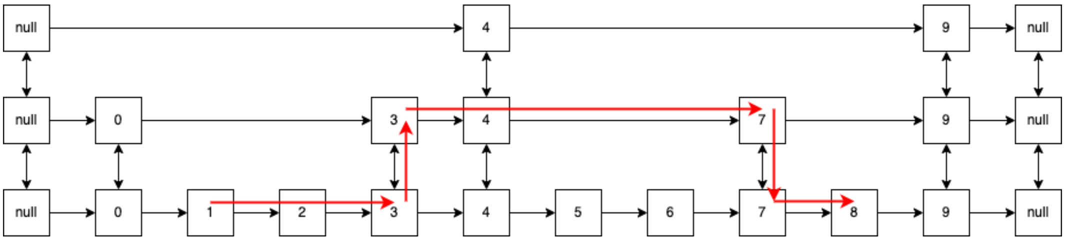

Figure 1 The route between the nodes 1 and 8.

Now, we consider the algorithms based on the `SplayList`.

▶ **Algorithm 2** (SimpleSplayListNet). *Suppose we are asked the routing request between u and v (for simplicity, $u < v$). In our algorithm, we traverse from u to the root node, increment counters, check all adjusting conditions, and update the network if necessary. Then, we traverse from the root to v, increment counters, check the conditions, and update the network if necessary.*

▶ **Algorithm 3** (TreeSplayListNet). *Suppose we are asked the routing request between u and v (for simplicity, $u < v$). We traverse from u to the common ancestor to the left (in the terms of `SkipList`) of u and v and then to v. On these paths, we update the counters and check the adjusting conditions. Finally, we go from the common ancestor to the root and update the counters by* 2.

Both, `SimpleSplayListNet` and `TreeSplayListNet` serve requests with the static-optimal cost.

Now, we present a structure that has more links that before. In the worst case it has $O(n \log n)$ links instead of $O(n)$ in algorithms before.

▶ **Algorithm 4** (SkipParentChildNet). *The structure of this network is a little bit more involved than before. At first, we introduce `LeftRightSplayNet`. For that we split the keys* *into two equal halves and build the* `SplayList` *on both parts: the right one is the usual* `SplayList`*, while the left one is the* `SplayList` *but from right to the left, i.e., the root is the rightmost node. Then, suppose we need to make a routing request from the left half to the right half — we use the search request in the left* `SplayList` *and, then, the search request in the right* `SplayList`*.*

Now, we are ready to present the general structure. Unfortunately, `LeftRightSplayNet` *works only for the requests from the left part to the right part, but we could have requests on the two nodes in one part. For that, we go recursively on both parts and build* `LeftRightSplayNet` *on them, and so on. Thus, if we need to pass a routing request between u and v, we go to the node in the structure tree that contains both u and v but in different halves and serve the request.*

This algorithm works a little better than the previous ones.

Now, we introduce a random data structure: we perform adjustments only once in c requests.

▶ **Algorithm 5** (ProbabilityTreeSplayListNet)**.** *This algorithm is based on* `TreeSplayListNet` *(Algorithm 3). Suppose we are asked the routing request between u and v (for simplicity, $u < v$). With probability $\frac{1}{c}$ we perform the algorithm from* `TreeSplayListNet` *(Algorithm 3) with updates of the counters and adjustments. In all other cases, we perform the simple up-down algorithm from* `SkipListNet` *(Algorithm 1).*

If calculated and proven properly, this algorithm is also static-optimal.

2 Experiments

Experiments were performed on: 1) the real-life workloads Facebook, HPC and ProjectTor; 2) the synthetic ones: the uniform and the workload with temporal locality 0.5.

As one can see on the table, the best average length of the paths is achieved by the randomized algorithm.

Table 1 The comparison of the average length of requests.

–	**Facebook**	**HPC**	**ProjectToR**	**Uniform**	**Locality** 0.5
SkipListNet	*19.62*	*10.13*	*8.81*	*7.94*	*14.35*
SimpleSplayListNet	*18.52*	*19.05*	*5.45*	*13.38*	*19.84*
TreeSplayListNet	*17.46*	*16.18*	*4.68*	*11.86*	*18.14*
SkipParentChildNet	*15.92*	*12.33*	*4.15*	*9.07*	*14.19*
ProbabilityFrontTreeSplayListNet	*12.36*	*8.0*	*2.65*	*5.76*	*10.23*