Electronic supplement for paper:

Tomas Skopal, David Hoksza:
Improving the Performance of M-tree Family by Nearest-Neighbor Graphs, ADBIS 2007, Varna, Bulgaria, LNCS XXXX, Springer, 2007

Listing 1 *(k-NN query algorithm)*

```
Node ChooseNode(PRQueue PR) {
    let d_min(T(O_i^*)) = min{d_min(T(O_i))}, considering all entries in PR
    remove entry [ptr(T(O_i^*)), d_min(T(O_i^*))] from PR
    return ptr(T(O_i^*))
}

QueryResult kNNQuery(kNNQuery (Q, k), ordering heuristic H) {
    PR = {[ptr(root), ∞]}
    for i = 1 to k do
        NN[i] = [−, ∞]      /* r_Q = NN[k].d_max = ∞ */
    while PR is not empty do {
        NextNode = ChooseNode(PR)
        NodeSearch(NextNode, (Q, k), H)
    }
    return NN
}
```

The adaptation for M*-tree is presented in Listing 2, where an M*-tree node is processed. In addition to the original processing of M-tree node, the NN-graph filtering is inserted between the parent and basic filtering steps, however, we additionally store the already used sacrifices and use them repeatedly for filtering the same way as a new sacrifice is used (let us call this *recycled NN-graph filtering*). The rationale for this is specific for kNN processing – a sacrifice which was not successful in its first attempt could succeed after a sufficiently large decrease of the dynamically improving query radius r_Q.

Listing 2 *(k-NN query algorithm, the NodeSearch)*

```
NodeSearch(Node N, kNNQuery (Q, k), ordering heuristic H) {
  let P be the parent routing object of N
        /* if N is root then δ(O_i, P)=δ(P, Q)=0 */
  let filtered be an array of boolean flags, size of filtered is |N|
  set filtered[entry(O_i)]=false, ∀entry(O_i) ∈ N
  let usedSacrifices = ∅
  let SQ be a queue filled with all entries of N, ordered by H(N)
    if N is not a leaf then {
    while SQ not empty
      fetch rout(S_i) from the beginning of SQ
      /* parent filtering */
      if |δ(P, Q) − δ(S_i, P)| > r_Q + r_{S_i} then
        filtered[rout(S_i)] = true;
      if not filtered[rout(S_i)] then {
        compute δ(S_i, Q)
        insert ⟨S_i, δ(S_i, Q)⟩ into usedSacrifices
        if d_min(T(S_i)) ≤ r_Q then {
          insert [ptr(T(S_i)), d_min(T(S_i))] to PR
          /* basic filtering */
          if d_max(T(S_i)) < r_Q then {
            r_Q = NNUpdate([−, d_max(T(S_i))])
            remove PR requests for which d_min(T(S_i)) > r_Q
          }
        }
        /* NN-graph filtering */
        NF = ∅
        for each S_j in usedSacrifices do
          NF = NF ∪ FilterByNNGraph(N, ⟨S_j, δ(S_j, Q)⟩, (Q, r_Q))
        move all entries in QS ∩ NF to the beginning of QS
      }
    }
  } else { /* N is a leaf */
    while SQ not empty
      fetch grnd(S_i) from the beginning of SQ
      /* parent filtering */
      if |δ(P, Q) − δ(S_i, P)| > r_Q then
        filtered[grnd(S_i)] = true;
      if not filtered[grnd(S_i)] then {
        compute δ(S_i, Q)
        insert ⟨S_i, δ(S_i, Q)⟩ into usedSacrifices
        /* basic filtering */
        if δ(S_i, Q)) ≤ r_Q then {
          r_Q = NNUpdate([−, d_max(T(S_i))])
          remove PR requests for which d_min(T(S_i)) > r_Q
        }
        /* NN-graph filtering */
        NF = ∅
        for each (S_j,δ(S_j, Q)) in usedSacrifices do
          NF = NF ∪ FilterByNNGraph(N, ⟨S_j, δ(S_j, Q)⟩, (Q, r_Q))
        move all entries in QS ∩ NF to the beginning of QS
      }
    }
  }
}
```