1. The Multi-sequence algorithm

In this appendix, we give additional details on the multi-sequence algorithm.

The pseudocode

Algorithm 1.1: MULTI-SEQUENCE ALGORITHM()

input \( r(1:1), s(1:1), T \)
traversed = zeros(M, M);
pqueue.push ((1, 1), r(1)+s(1))
repeat
  \((i, j, d) \leftarrow \text{pqueue.pop}()\)
  traversed(i, j) \leftarrow 1
output (i, j)
  if \( j < L \) and \((i==1 \text{ or traversed}(i-1, j+1))\)
    then \text{pqueue.push} ((i, j+1), r(i)+s(j+1))
  if \( i < L \) and \((j==1 \text{ or traversed}(i+1, j-1))\)
    then \text{pqueue.push} ((i+1, j), r(i+1)+s(j))
until enough traversed

Proof of corollary 1

We prove the corollary 1 in two steps. First, we prove that the algorithm will visit all cells in the L-by-L table (completeness – given that long enough candidate list is requested) and then we prove that it will visit the cells in the right order (monotonicity).

Completeness

We prove the completeness by induction on the value of sum \((i + j)\) for a fixed value of \(L\). The base of induction, i.e. the case \(i + j = 2\) is trivial as the cell \((1, 1)\) is always traversed at the first step. Assume that all cells \((i, j), i+j \leq k\) will be traversed. Consider a cell \((i, j)\) with \(i + j = k + 1\). Both its predecessors \((i - 1, j)\) and \((i, j - 1)\) will be traversed by the assumption. Then, after the traversal of the second predecessor the cell \((i, j)\) will be pushed into the queue and eventually popped from the queue (given that long enough candidate list is requested). Thus the induction step is proved and the completeness is verified.

Monotonicity

Let us now show that for any two cells \((i_1, j_1), (i_2, j_2)\) such that \((i_2, j_2)\) was traversed immediately after \((i_1, j_1)\), the monotonicity holds, i.e. \(r(i_1) + s(j_1) \leq r(i_2) + s(j_2)\).

Assume the contrary, i.e. \(r(i_1) + s(j_1) > r(i_2) + s(j_2)\). This would mean that \((i_2, j_2)\) was pushed into the priority queue after \((i_1, j_1)\) was traversed (otherwise the algorithm would have popped \((i_2, j_2)\) from the priority queue first). However, after the traverse of \((i_1, j_1)\) algorithm can push into the queue only either \((i_1 + 1), j_1\) or \((i_1, j_1 + 1)\) (or both). As \(r(i + 1) \geq r(i)\) and \(s(i + 1) \geq s(i)\), the initial assumption \(r(i_1) + s(j_1) > r(i_2) + s(j_2)\) cannot hold for any of those two cases.

Proof of corollary 2

Let us estimate the minimum number of cells that the multi-List algorithm has to traverse to get a priority queue of size \(s\). Consider the number of cells traversed in each row (denote them \(n_i\)). It is easy to see that a) \(n_i\) is monotonically non-increasing; b) each row has at most one cell in the priority queue (for the same reason), c) if \(n_i = n_i + 1\) than the row \(i + 1\) has no cells in the priority queue. All three statements follows from the fact that each cell can be added to the queue (or traversed) only after all of its predecessors, i.e. all cells with both coordinates smaller or equal to a given one, have been traversed.

Therefore, to get \(s\) cells in the priority queue, there should be at least \(s - 1\) non-empty rows with total number of traversed cells equals \(1 + 2 + \cdots + (s - 1) = \frac{s(s - 1)}{2}\). Therefore, \(\frac{s(s - 1)}{2} \leq t\), where \(t\) is the number of steps (=number of traversed cells). Solving this quadratic inequality gives the bound.

2. Occupancy statistics

We have measured the number of non-zero entries for the multi-indices constructed for the BIGANN dataset. We have observed that the number of occupied entries in the multi-index was 31% of all entries for the second-order inverted index with \(K = 2^{14}\), 62% of all entries for the second-order inverted index with \(K = 2^{12}\), and 18% of all entries for the fourth-order inverted multi-index with \(K = 2^7\).