Word-Tree Filtering Heuristic

by Ali Mohammad Pur Fard

Generating the filter map

Let c_{rows} be the number of rows in a given keyboard layout, and c_{cols} be the number of columns in the aforementioned keyboard layout.

Define x_A and y_A as $x_A = (\frac{c_{rows}+1}{2})$ and $y_A = (\frac{c_{cols}+1}{2})$.

As such, for all keyboard layouts, the probability of correctness function P_c can be defined

as $P_c(i_0, i_1, j_0, j_1) = 2 \cdot thresh_1 - C_t \cdot (abs(i_1 - i_0 + offset_{i_0} - offset_{i_1}) + abs(j_1 - j_0))$

where $C_t = \frac{thresh_1 \cdot m_{max}}{m_{min} \cdot (x_A + y_A - 2)}$

and $offset_r = \frac{cRows_{max} - cRows_r}{2}$.

Where $thresh_1$ is a parameter that gives the half-way threshold and its scale, m_{max} and m_{min} are the maximum and minimum number of mistakes allowed in a single word.

(see my *implementation* for more info)

Filtering the word-tree

If the word does not exist in the tree, per char in input word and active char in current subtree, $thresh_{node} = thresh_{parent(node)} \cdot thresh_{value}$

If $thresh_{node} < thresh_1$, disable the entire subtree.

At the end, a single pass is performed over the tree to copy the remaining nodes into a given array, which can later be compared with, through the use of Levenstein edit distance.

This heuristic tends to punish char removals more than char additions, however.