A Window Calculation

Algorithm 4 illustrates the window calculation procedure used in Lines 6 and 15 of Algorithm 1, where a window W_a^l with length $2o^l$ halved by a is requested.

The algorithm first calculates the left window boundary $w_{\min}$. For clarity, trees at the children of leaves and the parent of the root are defined as empty. Two variables are defined in Line 1: c records the currently under-processed node during tree traversal, and r works as a *budget* to record the number of values that should be checked and is greater than the left boundary. After locating at the node with value a (Line 1), it has to decide which branch to traverse using the size of its left subtree, $|\mathcal{T}_{left}|$. Remember that the tree size rooted at each WBT node is recorded by the node itself. If $r \le |\mathcal{T}_{left}|$ (Line 2), the answer is in the left subtree, and we set $w_{\min}$ to the r-th closest value to a. For example, in an ordered array $[1, 2, 3, 4, 5]$, the second closest value to 5 is 3 and the forth is 1. Otherwise, the boundary is less than the smallest value in $\mathcal{T}_{left}$, and we need "climb up" to find the potential subtree that may contain the left boundary value in Lines 5–15. In Lines 9–13, we have "climbed" from the right child, which means the left subtree of the parent may contain the answer. If the budget is only one (Line 11), the value of the parent is what we want, otherwise we get the $(r-1)$-th closest value to a in the left subtree (Line 12). If the left subtree cannot cover the budget (Line 13, $|p.\mathcal{T}_{left}| + 1 < r$), we reduce the budget by $|p.\mathcal{T}_{left}| + 1$ and keep looking for the potential tree node. In Line 14, we have "climbed" from the left child, and it means that the value of the current node is smaller than the expected left window boundary, thus we keep climbing. If we have checked all ancestors including the tree root but the window boundary cannot be found (Line 8), $w_{\min}$ is set to the dataset boundary, i.e. the minimum value.

After the determination of $w_{\min}$, $w_{\max}$ can be found with a dual version of the above procedure, which can be generated by replacing $\mathcal{T}_{left} \leftrightarrow \mathcal{T}_{right}$, min $\rightarrow$ max, $w_{\min} \rightarrow w_{\max}$. For example, if we have checked all ancestors in Line 8, $w_{\max}$ should be set to the *maximum* (replacing minimum) value in $p.\mathcal{T}$. The single-branch tree traversal (i.e. only one node is visited on each level of the tree on the search path) is conducted at most three times for one boundary: locating value a in Line 1; "climbing" towards root for the potential ancestor; and finding the closest value to a in the potential subtree.

The time complexity of Algorithm 4 scales to $O(\log n)$, where n denotes the number of tree nodes.

Algorithm 4: GetWindow

Input: a: attribute value; l: window graph layer
Output: W_a^l: window of attribute value a in layer l
Hyperparameter: o: window boosting base

1 $c \leftarrow$ WBT node with value a, $r \leftarrow o^l$;
2 **if** $r \le |c.\mathcal{T}_{left}|$ **then**
3 $w_{\min} \leftarrow$ the r-th closest value to a in $c.\mathcal{T}_{left}$;
4 **else**
5 $r \leftarrow r - |c.\mathcal{T}_{left}|$;
6 **while** $c.\mathcal{T} \neq \emptyset$ **do**
7 $p \leftarrow$ parent of c;
8 **if** $p.\mathcal{T} = \emptyset$ **then** $w_{\min} \leftarrow$ min value in $p.\mathcal{T}$;
9 **else if** *c is the root of $p.\mathcal{T}_{right}$* **then**
10 **if** $|p.\mathcal{T}_{left}| + 1 \ge r$ **then**
11 **if** $r = 1$ **then** $w_{\min} \leftarrow$ value of p;
12 **else** $w_{\min} \leftarrow$ the $(r-1)$-th closest value to a in $p.\mathcal{T}_{left}$;
13 **else** $r \leftarrow r - (|p.\mathcal{T}_{left}| + 1)$;
14 **else** $c \leftarrow p$;
15 **if** $w_{\min}$ *is determined* **then break**;
16 $w_{\max}$ is calculated by a dual version of the above procedure: replacing $\mathcal{T}_{left} \leftrightarrow \mathcal{T}_{right}$, min $\rightarrow$ max, $w_{\min} \rightarrow w_{\max}$;
17 **return** $W_a^l = [w_{\min}, w_{\max}]$;

B Range Filter Selectivity Calculation

Algorithm 5 shows how to count the size of the filtered dataset n' used in Line 1 of Algorithm 3, which reflects the selectivity of range filter R in Definition 3. The algorithm first calculates the true ranges $[x_u, y_l]$, where the boundaries exist in the dataset and are covered by R. Then, it calculates their ordered ranks and uses the subtraction as the final cardinality.

The rank calculation procedure starts from the root of WBT and traverses to the node with the target value a. If the target value is less than the value of the visited node, we jump to the left child. If the target is greater, which means that the rank is also greater, we increase *rank* by $|c.\mathcal{T}_{left}| + 1$ and jump to the right child. Otherwise, the node of the target value is found. After increasing *rank* by the size of its left subtree where the values are all less than a, we return the accumulated *rank* as the result. The algorithm requires four rounds of single-branch tree traversals. Thus, the time complexity is also logarithmic. In the real-world implementation, the complexity can be further improved by combining the boundary determination in Lines 1–2 and the rank calculation in Line 3 together to obtain two round of traversals.

Algorithm 5: FilteredSetCardinality

Input: $R = [x, y]$: range filter
Output: n': cardinality of the filtered dataset

1 $x_u \leftarrow$ upper bound of x;
2 $y_l \leftarrow$ lower bound of y;
3 $i \leftarrow$ GetRank(x_u), $j \leftarrow$ GetRank(y_l);
4 **return** $n' = j - i + 1$;
5 **procedure** GetRank(a)
6 $c \leftarrow$ root of WBT, $rank \leftarrow 0$;
7 **while** $c.\mathcal{T} \neq \emptyset$ **do**
8 **if** $a <$ *value of* c **then** $c \leftarrow$ root of $c.\mathcal{T}_{left}$;
9 **else if** $a >$ *value of* c **then**
10 $rank \leftarrow rank + |c.\mathcal{T}_{left}| + 1$;
11 $c \leftarrow$ root of $c.\mathcal{T}_{right}$;
12 **else** ▷ a = value of c, target found.
13 $rank \leftarrow rank + |c.\mathcal{T}_{left}|$;
14 **break**;
15 **return** *rank*;

C Proof of Theorem 3.2

Theorem 3.2. *In the landing layer l_d, the expected fraction f_R of in-range neighbors at a single hop on the path is bounded by*

$$f_R = \begin{cases} \left(\frac{1}{\sqrt{o}}, \frac{1}{2}\right) & (a)\, l \in (l'-1, l'-\frac{1}{2}), o > 4, \\ \left[\frac{\sqrt{2}}{2} - \frac{1}{4o^{l+1}}, \frac{3}{4} - \frac{1}{4o^{l+1}}\right) & (b)\, l \in (l'-1, l'-\frac{1}{2}), o \le 4, \\ \left[\frac{3}{4} - \frac{1}{4o^{l}}, 1 - \frac{o^l+1}{4o^{l+\frac{1}{2}}}\right] & (c)\, l \in [l'-\frac{1}{2}, l'], \end{cases} \quad (1)$$

where n' is the number of in-range vectors for filter $R = [x, y]$, $l' = \log_o \frac{n'}{2}$, and $l = l_h = \lfloor \log_o \frac{n'}{2} \rfloor$, which is the highest layer defined in Line 2 of Algorithm 3, whose window size is less than R.

Proof. The landing layer l_d is determined by comparing $\frac{2o^l}{n'}$ and $\frac{n'}{2o^{l+1}}$. Range filter $[x, y]$ can be rewritten to $[x, x+n'-1]$ by assuming all attribute values are sequential. Moreover, vectors have an equal probability to be selected as neighbors in a certain range. There are two situations:

• **Situation 1.** $2o^{l+\frac{1}{2}} < n' < 2o^{l+1}$, then $\frac{2o^l}{n'} < \frac{n'}{2o^{l+1}}$ and $l_d = l+1$, the half window size is o^{l+1}. For a vector with attribute value $x+i$ in range $[x, x+n'-1]$ where $i \in [0, n'-1]$, the window of a vector with attribute value $x+i$ is $W^{l+1}_{x+i} = [x+i-o^{l+1}+1, x+i+o^{l+1}-1]$.

Case (a). When $n' < o^{l+1}$, the range filter is always covered by the windows. In this case, $2o^{l+\frac{1}{2}} < o^{l+1}$ and $o > 4$. The fraction can be calculated by $f_R = \frac{n'}{2o^{l+1}} \in (\frac{1}{\sqrt{o}}, \frac{1}{2})$.

Case (b). When $n' \ge o^{l+1}$, some windows fail to cover the whole range. If the window left boundary is less than the filter left boundary, i.e. $x+i-o^{l+1} < x \Rightarrow i \le o^{l+1}-1$ The fraction of in-range vectors is $f_1 = \frac{(x+i+o^{l+1}-1)-x+1}{2o^{l+1}}$, and the expectation is

$$\bar{f_1} = \mathbb{E}_{i \in [0, o^{l+1}-1]} \left[\frac{i+o^{l+1}}{2o^{l+1}} \right], \quad (2)$$

which is $\bar{f_1} = \frac{1}{4}(3 - o^{l+1})$. On the other hand, if $i \in [o^{l+1}, n'-1]$, the window right boundary is greater than the filter right boundary. The fraction is $f_2 = \frac{(x+n'-1)-(x+i-o^{l+1}+1)+1}{2o^{l+1}}$, and

$$\bar{f_2} = \mathbb{E}_{i \in [o^{l+1}, n'-1]} \left[\frac{n'+o^{l+1}-1-i}{2o^{l+1}} \right], \quad (3)$$

which is $\bar{f_2} = \frac{n'+o^{l+1}-1}{4o^{l+1}}$. Combining two parts, $f_R = w_1\bar{f_1} + w_2\bar{f_2}$, where $w_1 = \frac{o^{l+1}}{n'}$ and $w_2 = \frac{n'-o^{l+1}}{n'}$, the total fraction is

$$\begin{aligned} f_R &= \frac{3o^{l+1}-1}{4o^{l+1}} \cdot \frac{o^{l+1}}{n'} + \frac{n'+o^{l+1}-1}{4o^{l+1}} \cdot \frac{n'-o^{l+1}}{n'} \\ &= \frac{o^{l+1}}{2n'} + \frac{n'-1}{4o^{l+1}} \in \left[\frac{1}{4}\left(2\sqrt{2} - o^{-(l+1)}\right), \frac{1}{4}\left(3 - o^{-(l+1)}\right) \right), \end{aligned} \quad (4)$$

and the lower bound is at $n' = \sqrt{2}o^{l+1} \in [2o^{l+\frac{1}{2}}, 2o^{l+1})$, while the upper bound is determined by $n' = 2o^{l+1}$.

• **Situation 2.** Case (c). $2o^l \le n' \le 2o^{l+\frac{1}{2}}$, then $\frac{2o^l}{n'} \ge \frac{n'}{2o^{l+1}}$ and $l_d = l$, the half window size is o^l. For a vector with attribute value $x+i$ in range $[x, x+n'-1]$ where $i \in [0, n'-1]$, its window in Layer l is $W^l_{x+i} = [x+i-o^l+1, x+i+o^l-1]$.

When $i \le o^l - 1$, we have $x+i-o^l+1 < x$ that the left boundary of the window is less than that of the filter. The fraction of in-range vectors over the window size is $f_1 = \frac{(x+i+o^l-1)-x+1}{2o^l}$. The expected fraction of this situation is

$$\bar{f_1} = \mathbb{E}_{i \in [0, o^l-1]} \left[\frac{i+o^l}{2o^l} \right], \quad (5)$$

which is $\bar{f_1} = \frac{1}{4}(3 - o^{-l})$. When $i \ge n' - o^l$, we have $x+n'-1 < x+i+o^l-1$ that the right boundary of the filter is less than that of the window. The fraction $f_2 = \frac{(x+n'-1)-(x+i-o^l+1)+1}{2o^l}$. We can calculate $\bar{f_2} = \frac{1}{4}(3-o^{-l})$ in the same way as $\bar{f_1}$. When $o^l - 1 < i < n' - o^l$, the expected fraction is $\bar{f_3} = 1$ because all windows of these elements are covered by the filter.

Finally, the total fraction of in-range neighbors is $w_1\bar{f_1} + w_2\bar{f_2} + w_3\bar{f_3}$ where $w_1 = w_2 = \frac{o^l}{n'}$ and $w_3 = \frac{n'-2o^l}{n'}$.

$$\begin{aligned} f_R &= \frac{2o^l}{n'}\left(\frac{1}{4}\left(3 - o^{-l}\right)\right) + \frac{n'-2o^l}{n'} \cdot 1 \\ &= 1 - \frac{o^l+1}{2n'} \in \left[\frac{1}{4}\left(3 - o^{-l}\right), 1 - \frac{o^l+1}{4o^{l+\frac{1}{2}}} \right]. \end{aligned} \quad (6)$$

□