Technical Report
TIME-SPACE TRADEOFF IN MATRIX UPDATE

Despite the performance benefits of fully leveraging matrix broadcasting, our matrix-based route priority update process[1] could introduce innegligible size inflation during intermediate steps. Specifically, Line 16 of MATRIXUPDATE involves broadcasting operations that result in an intermediate matrix of size $n \times n \times n$. This broadcasting step leads to a memory footprint that can be prohibitively large, especially for large values of n. To address this size inflation issue, two strategies can be employed: partial-loop and fully-loop versions of the matrix update function. These strategies balance the time-space tradeoff differently and are outlined as follows.

Algorithm 1 Partial-Loop Matrix-Based Route Priority Update

1: **function** MATRIXUPDATEPARTIALLOOP(L, P^T)
2: **external** BYTEUPDATE
3: **external** BYTESELECTION
4: **external** $f1$, $f2$
5: $n = \text{SIZE}(L,\ \text{axis} = 0)$
6: **for** $j = 0$ to $n-1$ **do**
7: $\hat{P}^{T+1} = f1(L,\ P^{T+1}[\ldots,\ j])$
8: $P^{T+1}[\ldots,\ j] = f2(\hat{P}^{T+1},\ \text{reduce\_along\_axis} = 1)$
9: **return** P^{T+1}

Algorithm 2 Fully-Loop Matrix-Based Route Priority Update

1: **function** MATRIXUPDATEFULLYLOOP(L, P^T)
2: **external** BYTEUPDATE
3: **external** BYTESELECTION
4: **external** $f1$, $f2$
5: $n = \text{SIZE}(L,\ \text{axis} = 0)$
6: **for** $j = 0$ to $n-1$ **do**
7: **for** $i = 0$ to $n-1$ **do**
8: $\hat{P}^{T+1} = f1(L[i,\ \ldots],\ P^T[\ldots,\ j])$
9: $P^{T+1}[i,\ j] = f2(\hat{P}^{T+1})$
10: **return** P^{T+1}

Partial-Loop Version. The partial-loop version mitigates the size inflation by processing the matrix in smaller chunks, thus reducing the memory required for intermediate steps. As shown in Algorithm 1, this version reduces the memory footprint by computing intermediate results for one column at a time. As a result, the intermediate size inflation incurred by $\hat{P}^{T+1}$ (Line 7) is only $n \times n$, instead of $n \times n \times n$ as in MATRIXUPDATE. While the memory overhead is significantly lowered, the overall computational time may slightly increase due to the explicit iterative processing of columns.

Fully-Loop Version. The fully-loop version completely avoids intermediate broadcasting by computing each cell of the result matrix individually. As shown in Algorithm 2, this version eliminates intermediate size inflation as the size of $\hat{P}^{T+1}$ (Line 8) remains n. Thus, the memory overhead is minimized, but the computational time is increased due to the fully nested loops. In practice, this method can be further optimized using parallel processing techniques or libraries like Numba to accelerate the cell-wise computations.

In general, the partial-loop version offers a balance by reducing memory overhead at the cost of increased computational time for column-wise processing. The fully-loop version minimizes memory overhead completely but increases computational overhead significantly due to the nested loops. Both strategies provide viable solutions to the size inflation issue inherent in the pure matrix-based version.

[1] See report on *Matrix-Based Route Priority Update*, which details MATRIXUPDATE, BYTEUPDATE, BYTESELECTION, $f1$, and $f2$, specifically.