**Proof of Invariants (Reviewer 2 Question 1):** We mildly modify the 1-feasibility condition (3) and enforce it only on any edge $(a, b)$ with a flow $\sigma(a, b) < \min\{s_b, d_a\}$, i.e., only on forward edges in the residual network. While this change does not impact the algorithm or its correctness, it significantly simplifies the proof of invariants. In the submitted version, only Lemma 2.1 uses (3) in (L193). We adapt this proof for the new definition of (3). Initially transform $\sigma$ and $\sigma'$ as follows. For any edge $(a, b)$ that does not satisfy (3), its flow $\sigma(a, b)$ is $\min\{s_b, d_a\}$. We reduce $d_a$ and $s_b$ by $\sigma'(a, b)$, reduce the flow on the edge $(a, b)$ in $\sigma'$ to 0, and reduce $\sigma(a, b)$ to $\min\{s_b, d_a\} - \sigma'(a, b)$. Both $\sigma$ and $\sigma'$ continue to be maximum transport plans and this transformation does not change the difference in their costs. Additionally, every edge with a positive flow in $\sigma'$ now satisfies (3) and (L193) continues to hold.

**Proof of (I1):** We show that after the dual updates of a Hungarian search, every forward edge satisfies (3) and every backward edge satisfies (4). From the shortest path property, we have **(E1):** $\ell_u + s(u, v) \geq \ell_v$. There are four possibilities: (i) $\ell_u < \ell_t$ and $\ell_v < \ell_t$, (ii) $\ell_u \geq \ell_t$ and $\ell_v < \ell_t$, (iii) $\ell_u < \ell_t$ and $\ell_v \geq \ell_t$ or (iv) $\ell_u \geq \ell_t$ and $\ell_v \geq \ell_t$. We present the proof for case (i); the other three cases are similar. For case (i), if $(u, v)$ is a forward (resp. backward) edge then $u \in B$, $v \in A$ (resp. $u \in A$, $v \in B$). The updated dual weights $\tilde{y}(u) = y(u) + \ell_t - \ell_u$ (resp. $\tilde{y}(u) = y(u) - \ell_t + \ell_u$) and $\tilde{y}(v) = y(v) - \ell_t + \ell_v$ (resp. $\tilde{y}(v) = y(v) + \ell_t - \ell_v$), and the updated dual weight sum is $\tilde{y}(u) + \tilde{y}(v) = y(u) + y(v) + \ell_v - \ell_u$ (resp. $\tilde{y}(u) + \tilde{y}(v) = y(u) + y(v) - \ell_v + \ell_u$). From (E1), $\tilde{y}(u)+\tilde{y}(v) \leq y(u)+y(v)+s(u, v) = \lfloor 2c(u, v)/\delta' \rfloor + 1$ (resp. $\tilde{y}(u)+\tilde{y}(v) \geq y(u)+y(v)-s(u, v) = \lfloor 2c(u, v)/\delta' \rfloor$). The last equality follows from the definition of slack for a forward (resp. backward) edge. Note that the augment procedure may introduce new edges into the residual network. Any such new forward (resp. backward) edge will have a slack of 1 because the corresponding backward (resp. forward) edge is admissible implying they satisfy (3) (resp. (4)).

**Proof of (I2):** We show that, after the dual updates are conducted by Hungarian search, the shortest path $P$ from $s$ to $t$ (ignoring vertices $s$ and $t$) is an admissible augmenting path between free vertices $b$ and $a$. For any edge $(u, v)$ on $P$, by construction $\ell_u \leq \ell_t$ and $\ell_v \leq \ell_t$. Repeating the calculations of case (i) of the proof of (I1), the updated dual weight sum is $\tilde{y}(u) + \tilde{y}(v) = y(u) + y(v) + \ell_v - \ell_u$ (resp. $\tilde{y}(u) + \tilde{y}(v) = y(u) + y(v) - \ell_v + \ell_u$). Note that for edges on the shortest path $P$, (E1) holds with equality and so, $\tilde{y}(u) + \tilde{y}(v) = y(u) + y(v) + s(u, v) = \lfloor 2c(u, v)/\delta' \rfloor + 1$ (resp. $\tilde{y}(u) + \tilde{y}(v) = y(v) + y(v) + s(u, v) = \lfloor 2c(u, v)/\delta' \rfloor$), i.e., the path $P$ is an admissible augmenting path. The partial DFS step conducts a search from every free supply vertex including $b$. This will lead to the discovery of at least one augmenting path in the admissible graph. We show that at the end of partial DFS step, there is no augmenting path in the admissible graph. Note that any vertex $v$ removed from $\mathcal{A}$ is a vertex from which the DFS search backtracked. Due to the fact that Hungarian search does not create a cycle in the admissible graph [Lemma 2.3, Gabow Tarjan SICOMP'89], we can conclude that there is no admissible path from $v$ to any free demand node. So the deleted vertex and edge could not have participated in an augmenting path of admissible edges. At the end of partial DFS step, every free supply vertex is deleted from $\mathcal{A}$, so there are no admissible augmenting paths from any free supply vertex.

**Scalability and Parallel Implementations (R2 Q2 & Q3c):** As shown in [Sharathkumar, Agarwal SODA 2012 Section 3.1, 3.2], Hungarian search and partial DFS can be implemented in $\mathcal{O}(n\Phi(n))$ time, where $\Phi(n)$ is the query/update time of a dynamic weighted nearest-neighbor (DNN) data structure. Many distances including squared Euclidean distance admits DNN with poly-logarithmic time search/update operations. We achieve $\tilde{\mathcal{O}}(n)$ execution time for these distances. See also [Agarwal Sharathkumar STOC 14, Section 4 (i)–(iii)] for a relative $\varepsilon$-approximation algorithm using the approximate nearest neighbor (ANN) data structure. Design of ANN-based near-linear time additive approximations and their practical implementation will be stated as potential open questions for the future.

An alternate implementation of Gabow-Tarjan's algorithm [Lahn, Raghvendra SODA 2019, Section 2.1] may be easier to parallelize. This implementation does not require Hungarian Search but only an iterative execution of partial DFS from each free vertex in the admissible graph. Paths in an admissible graph are of length $\mathcal{O}(C/\delta)$ (similar to Lemma 2.4 proof) which may aid in the execution of DFS from each free vertex in parallel. We will pose parallel implementation of our algorithm as an important open question. In our paper and experiments, we focused on sequential execution only.

**Experiments (R2 Q3a, Q3b):** Note that the value of $C$ for all Sinkhorn comparisons (Figure 2) in our paper was fixed at 7.112 (squared Euclidean costs scaled by the median cost), so $\delta/C$ was in the range of $[0.0035, 0.028]$ (includes moderate values). When comparing with other algorithms, we understand that the actual execution time may vary based on the choice of programming language, implementation details and the available resources. Therefore, we give a comparison (right) of the number of iterations taken by our algorithm with those in Table 1 for moderate values of $\delta/C$ (with a setup similar to Figure 2). For Sinkhorn, APDAGD, and our algorithm, each iteration takes $\Theta(n^2)$ time. For the Greenkhorn algorithm, the number of iterations is given by the total row/column updates divided by $n$. In the cases we tested, our algorithm performs fewer iterations on average. We saw similar results for $n \in [500, 2500]$, $\delta/C = 0.05$ on synthetic data. Note that the time taken for augmentations in our algorithm was negligible for our tests.