APPENDIX A
COMPLETE PROOF OF THEOREM 1

Theorem 1 (Efficiency)**.** *Suppose the edges of E^ϕ are stored continuously (e.g., to an array or a file system), and an operation to find the pointer of $E^\phi[i]$ by using i is $\mathcal{O}(1)$ (e.g., RAM or standard file systems). Then, there exists an $\mathcal{O}(1)$ algorithm to compute the chunk-based edge partitioning excluding the graph data movement, and it does not depend on the graph size, such as $|V|$ and $|E|$.*

Proof. From the definition, in order to compute the chunk-based edge partitioning in $\mathcal{O}(1)$, the algorithm needs to calculate $\sum_{x=0}^{p-1}\left\lfloor\frac{|E|+x}{k}\right\rfloor$ in $\mathcal{O}(1)$. $\sum_{x=0}^{p-1}\left\lfloor\frac{|E|+x}{k}\right\rfloor$ requires $\mathcal{O}(p)$ computational time in a naive way.

The summation can be modified as follows:

$$\sum_{x=0}^{p-1}\left\lfloor\frac{|E|+x}{k}\right\rfloor=\sum_{x=0}^{p-1}\left\{\left\lfloor\frac{|E|}{k}\right\rfloor+\left\lfloor\frac{(|E| \bmod k)+x}{k}\right\rfloor\right\}$$

Here, $\left\lfloor\frac{(|E| \bmod k)+x}{k}\right\rfloor$ is 0 or 1 in $x=0,1,...,p-1$, as follows:

$$\left\lfloor\frac{(|E| \bmod k)+x}{k}\right\rfloor=\begin{cases}0 & \text{if } (|E| \bmod k)+x<k\\ 1 & \text{otherwise}\end{cases}$$

Therefore,

$$\begin{aligned}\sum_{x=0}^{p-1}\left\lfloor\frac{(|E| \bmod k)+x}{k}\right\rfloor &= \begin{cases}0 & \text{if } k+(|E| \bmod k)\geq p\\ p-k+(|E| \bmod k) & \text{otherwise}\end{cases}\\ &= \max\left(0,p-k+(|E| \bmod k)\right)\end{aligned}$$

We define $\theta_k(p):=\max\left(0,p-k+(|E| \bmod k)\right)$. Then, the following formula is established:

$$\sum_{x=0}^{p-1}\left\lfloor\frac{|E|+x}{k}\right\rfloor=p\left\lfloor\frac{|E|}{k}\right\rfloor+\theta_k(p).$$

This can be computed in $\mathcal{O}(1)$. ■

APPENDIX B
COMPLETE PROOF OF THEOREM 2

Theorem 2 (Migration Cost)**.** *Suppose a set of ordered edges is initially split into k partitions via the chunk-based edge partitioning, and the edges are repartitioned into $k+x$ parts by adding x new processes (i.e., scale out). We assume that $|E|$ is much larger than k and x such that $(|E| \bmod k+x)/|E|<(k+x)/|E|\approx 0$ and that the ids of new partitions are $k,k+1,...,k+x-1$.*

Then, the number of migrated edges when applying repartitioning is approximately $\frac{x|E|}{2k(k+x)}\left\lceil\frac{k}{x}\right\rceil\left(\left\lceil\frac{k}{x}\right\rceil+1\right)+\frac{|E|}{k}\left(k-\left\lceil\frac{k}{x}\right\rceil\right)$. The cost for scaling in is the same (i.e., from $k+x$ to k partitions) since it is a reverse operation of scaling out.

Proof. We consider a simple case where $|E| \bmod k=0, |E| \bmod (k+1)=0, |E| \bmod (k+2)=0, ..., |E| \bmod (k+x)=0$. Then, there are two cases in the edge migration for partition $i(i\in[0,k))$: (i) some of the edges in partition i are migrated to other partitions, or (ii) all of the edges in partition i are migrated to other partitions.

Case (i): In this case, for partition i, the edges from $i\frac{|E|}{k}$-th edge to $(i+1)\frac{|E|}{k+x}$-th are kept in partition i, while from $(i+1)\frac{|E|}{k+x}$-th to $(i+1)\frac{|E|}{k}$-th edges are migrated to other partitions.

Thus the number of migrated edges for partition i is represented as follows:

$$(i+1)\frac{|E|}{k}-(i+1)\frac{|E|}{k+x}=(i+1)\frac{|E|n}{(k+x)k}$$

Case (i) happens when $(i+1)\frac{|E|n}{(k+x)k}>\frac{|E|}{k}$.

$$\begin{aligned}&(i+1)\frac{|E|n}{k(k+x)}>\frac{|E|}{k}\\ \Leftrightarrow\quad &(i+1)>\frac{k+x}{x}\\ \Leftrightarrow\quad &i>\frac{k}{x}\end{aligned}$$

Therefore, Case (i) happens when $i>\frac{k}{x}$.

Case (ii): In the other case (i.e., $i\leq\frac{k}{x}$), all of the edges in partition i are migrated to other partitions. Thus, the number of migrated edges for partition i is $\frac{|E|}{k}$.

Therefore, to summarize Cases (i) and (ii), the total number of migrated edges from $i=0$ to $i=k-1$ is formalized as follows:

$$\begin{aligned}&\sum_{0\leq i<\frac{k}{x}}(i+1)\frac{|E|x}{(k+x)k}+\sum_{\frac{k}{x}\leq i<k}\frac{|E|}{k}\\ =\;&\frac{|E|x}{(k+x)k}\sum_{0\leq i<\frac{k}{x}}(i+1)+\frac{|E|}{k}\sum_{\frac{k}{x}\leq i<k}1\\ =\;&\frac{x|E|}{2k(k+x)}\left\lceil\frac{k}{x}\right\rceil\left(\left\lceil\frac{k}{x}\right\rceil+1\right)+\frac{|E|}{k}\left(k-\left\lceil\frac{k}{x}\right\rceil\right)\end{aligned}$$

The aforementioned simplified proof can be straightforwardly generalized for the case of $|E| \bmod k\neq 0, |E| \bmod k+1\neq 0, ..., |E| \bmod k+x\neq 0$, based on the assumption $(|E| \bmod k+x)/|E|\approx 0$. ■

APPENDIX C
COMPLETE PROOF OF THEOREM 3

Theorem 3 (NP-hardness)**.** *The graph edge ordering problem is NP-hard if $|E|$ is much larger than k_{max} so that less than k_{max} edges do not affect the optimized result.*

Proof. We first show that the graph edge ordering problem is NP-hard for single k, i.e., $k_{min}=k_{max}$. We then prove the general case of multiple k, i.e., $k_{min}<k_{max}$.

Case of Single k: Suppose $k_{min}=k_{max}=k$. The objective of the graph edge ordering problem is represented as follows:

$$\min_{\phi\in\Phi}\frac{1}{|V|}\sum_{p=0}^{k-1}\left|V\left(E^\phi_{ch}\left(\sum_{x=0}^{p-1}\left\lfloor\frac{|E|+x}{k}\right\rfloor,\left\lfloor\frac{|E|+p}{k}\right\rfloor\right)\right)\right|. \quad (6)$$

Now, we define a function to convert the edge order into the partition, $\texttt{ID2P}_k$: $i \mapsto p$, as Algorithm 2. By using $\texttt{ID2P}_k$, we can generate new edge partitions from the edge orders in linear time.

Algorithm 2: Conversion from Edge ID to Partition

Input : i – Ordered Edge ID
Output: p – Partition ID

```
ID2P_k(i)
    p ← 0; cur ← ⌊(|E|+p)/k⌋
    while i < cur do
        p ← p + 1; cur ← cur + ⌊(|E|+p)/k⌋
    return p
```

Suppose the order ϕ_{opt} is the optimal solution for the graph edge ordering problem. Then, the edge partitions converted from ϕ_{opt} via $\texttt{ID2P}_k$ is also the optimal solution for the edge partitioning problem in a case when $\epsilon \approx 0$ in Definition 1.

The reason is as follows. If the edge partitions converted from ϕ_{opt} via $\texttt{ID2P}_k$ is *not* the optimal solution (more specifically, more than k_{max} edges are in the different partitions from the optimal partitions), then there exist another optimal edge partitions, $\mathcal{E}_k^{opt} := \{\mathcal{E}_k^{opt}[p] \mid 0 \le p < k\}$, which provides a better solution for the edge partitioning problem than ϕ_{opt}. Based on $\mathcal{E}_k^{opt}$, we can generate new edge ordering ϕ' in such a way that for p

$$\mathcal{E}_k^{opt}[p] = \left\{ E^{\phi'}[b], E^{\phi'}[b+1], ..., E^{\phi'}\left[b + \lfloor \tfrac{|E|+p}{k} \rfloor - 1\right] \right\},$$

where $b := \sum_{x=0}^{p-1} \left\lfloor \frac{|E|+x}{k} \right\rfloor$. Since $\mathcal{E}_k^{opt}$ provides the optimal solution,

$$RF(\mathcal{E}_k^{opt}) := \frac{1}{|V|} \sum_{p=0}^{k-1} |V(\mathcal{E}_k^{opt}[p])|$$
$$= \frac{1}{|V|} \sum_{p=0}^{k-1} \left| V\left(E_{ch}^{\phi'} \left(\sum_{x=0}^{p-1} \left\lfloor \frac{|E|+x}{k} \right\rfloor, \left\lfloor \frac{|E|+p}{k} \right\rfloor \right) \right) \right|$$

is the optimal value. On the other hand, ϕ_{opt} provides the optimal value of Eq. (6) as follows:

$$\frac{1}{|V|} \sum_{p=0}^{k-1} \left| V\left(E_{ch}^{\phi_{opt}} \left(\sum_{x=0}^{p-1} \left\lfloor \frac{|E|+x}{k} \right\rfloor, \left\lfloor \frac{|E|+p}{k} \right\rfloor \right) \right) \right|.$$

This is a contradiction to the assumption that $\mathcal{E}_k^{opt}$ provides the better solution than ϕ_{opt}. Thus, ϕ_{opt} can provide the optimal solution for the edge partitioning problem as well.

Therefore, the problem (6) is reducible to the balanced k-way edge partitioning problem, which is an NP-hard problem as proved in [9].

Case of $k_{min} < k_{max}$: We explain the case when $k_{min} = 2$ and $k_{max} = 3$. The following discussion can be straightforwardly generalized to any k_{min} and k_{max}.

According to Definition 4, we define a function, $Num(k,p)$, for the normalized number of vertices involved in the chunk of edges as follows:

$$Num(k,p) := \frac{1}{|V|} \left| V\left(E_{ch}^{\phi} \left(\sum_{x=0}^{p-1} \left\lfloor \frac{|E|+x}{k} \right\rfloor, \left\lfloor \frac{|E|+p}{k} \right\rfloor \right) \right) \right|.$$

Suppose $k_{min} = 2$ and $k_{max} = 3$, we will show the NP-hardness of the optimization problem as follows:

$$\min_{\phi \in \Phi} \sum_{k=2}^{3} \sum_{p=0}^{k-1} N(k,p) = \min_{\phi \in \Phi} \{ Num(2,0) + Num(2,1) + Num(3,0) + Num(3,1) + Num(3,2) \}. \quad (7)$$

Here, based on the above discussion of the single k, the following optimization problems are already proved to be NP-hard:

$$\min_{\phi \in \Phi} \{ Num(2,0) + Num(2,1) \} \quad (8)$$
$$\min_{\phi \in \Phi} \{ Num(3,0) + Num(3,1) + Num(3,2) \}. \quad (9)$$

Suppose ϕ_{opt} is the optimal order for (7), then the order can be also the optimal for (8) and (9). Thus, if (7) is not NP-hard, it is a contradiction to the NP-hardness of (8) and (9). Therefore, (7) is also NP-hard. To summarize, the graph edge ordering problem is NP-hard. ■

Appendix D
Complete Proof of Lemma 1

Lemma 1. *Definition 4 and Definition 5 are equivalent.*

Proof. In Eq. (2), according to the definition of $\beta_k(i)$, $f_k\left(i, \left\lfloor \frac{|E|+\texttt{ID2P}_k(i)}{k} \right\rfloor\right)$ is non-zero only if

$$i = \left\lfloor \frac{|E|}{k} \right\rfloor - 1, \left\lfloor \frac{|E|}{k} \right\rfloor + \left\lfloor \frac{|E|+1}{k} \right\rfloor - 1, ..., \sum_{x=0}^{k-1} \left\lfloor \frac{|E|+x}{k} \right\rfloor - 1$$

Therefore,

$$\sum_{k=k_{min}}^{k_{max}} \sum_{i=0}^{|E|-1} f_k\left(i, \left\lfloor \frac{|E|+\texttt{ID2P}_k(i)}{k} \right\rfloor\right)$$
$$= \sum_{k=k_{min}}^{k_{max}} \sum_{p=0}^{k-1} f_k\left(\sum_{x=0}^{p} \left\lfloor \frac{|E|+x}{k} \right\rfloor - 1, \left\lfloor \frac{|E|+p}{k} \right\rfloor \right)$$
$$= \sum_{k=k_{min}}^{k_{max}} \sum_{p=0}^{k-1} \left| V\left(E_{ch}^{\phi} \left(\sum_{x=0}^{p-1} \left\lfloor \frac{|E|+x}{k} \right\rfloor, \left\lfloor \frac{|E|+p}{k} \right\rfloor \right) \right) \right|,$$

which is equal to Eq. (1). Therefore, Def. 4 and Def. 5 are equivalent. ■

Appendix E
Complete Proof of Theorem 4

Theorem 4 (Efficiency of Baseline Algorithm)**.** *Time complexity of Algorithm 2 is $O\left(\frac{k_{max}^2 |E|^2 \cdot |V|^2}{k_{min}}\right)$, where k_{max} is much larger than k_{min} s.t. $k_{max} - k_{min} \approx k_{max}$.*

Proof. The outermost loop (Lines 2-18) computes $O(|V|)$ iterations. In each iteration, the inner loop from Line 8 to 11 also computes $O(|V|)$ iterations. Moreover, at Line 10, the summation requires $O(|E| \cdot (k_{max} - k_{min}))$ computational time while $f_k\left(X^\phi, i, \frac{|E|}{k}\right)$ requires $O\left(\frac{k_{max}|E|}{k_{min}}\right)$. Therefore, in total, it requires $O\left(\frac{|V|^2 \cdot |E| (k_{max}-k_{min}) \cdot k_{max}|E|}{k_{min}}\right) = O\left(\frac{k_{max}^2 |E|^2 \cdot |V|^2}{k_{min}}\right)$ under the assumption that $k_{max} - k_{min} \approx k_{max}$. ■

APPENDIX F
COMPLETE PROOF OF LEMMA 2

Lemma 2. *Suppose $|E|$ is much larger than k_{max} such that $w := \left\lfloor \frac{|E|}{k} \right\rfloor = \left\lfloor \frac{|E| + ID2P_k(\cdot)}{k} \right\rfloor$ and $D[v] < \frac{|E|}{k_{max}}$ for $\forall v \in V$. Then,*

$$\forall v, u \in V_{rest} \cap V(X^\phi), \quad p(v) > p(u) \Rightarrow F_v > F_u,$$

where F_v and F_u are the value of Eq. (3) for $X^\phi + N(v)$ and $X^\phi + N(u)$ respectively, as shown in Line 10 of Algorithm 2.

Proof. Suppose $Xv^\phi := X^\phi + N(v)$, $Xu^\phi := X^\phi + N(u)$.

$$\begin{aligned}
& F_v > F_u \quad \Leftrightarrow \quad F_v - F_u > 0 \\
\Leftrightarrow \quad & \sum_{k=k_{min}}^{k_{max}} \sum_{i=0}^{|E|-1} \left\{ f\left(Xv^\phi, i, w\right) - f\left(Xu^\phi, i, w\right) \right\} > 0 \\
\Leftarrow \quad & \sum_{k=k_{min}}^{k_{max}} \sum_{i=0}^{|E|-1} \left\{ \left|V\left(Xv^\phi_{ch}(i-w+1,w)\right)\right| - \left|V\left(Xu^\phi_{ch}(i-w+1,w)\right)\right| \right\} \\
& > 0 \\
\Leftrightarrow \quad & \sum_{k=k_{min}}^{k_{max}} \sum_{i=|X|^\phi}^{|E|-1} \{\Delta V(v,i) - \Delta V(u,i)\} > 0, \qquad (10)
\end{aligned}$$

where $\Delta V(v,i) := |V(Xv^\phi_{ch}(i-w+1,w))| - |V(X^\phi_{ch}(i-w+1,w))|$.

Next, we will calculate $\Delta V(v,i)$ for $i \geq |X^\phi|$. Intuitively, $\Delta V(v,i)$ means the number of additional replicated vertices in a chunk when we select v to expand the ordered edges. For each chunk determined by i, each additional replicated vertex comes from v or $N(v)$. Thus, $\Delta V(v,i)$ can be represented by the sum of two functions:

$$\Delta V(v,i) = \chi(i) + n(i),$$

where $\chi(i)$ is the number of replicated vertices caused by v; $n(i)$ is caused by $N(v)$.

First, $\chi(i)$ is the indicator function. If $X^\phi_{ch}(i-w+1,w)$ already involves v, then the number of replicated vertices does not increase due to the additional v. Therefore, $\chi(i)$ is 0. Specifically, this case appears if $i > M[v] + w$, because $X^\phi_{ch}(i-w+1,w)$ involves an edge e whose order is $M[v]$ (i.e., $\phi(e) = M[v]$). Otherwise, v's replication is newly added to the chunk $Xv^\phi_{ch}(i-w+1,w)$, and thus $\chi(i)$ is 1. Therefore, $\chi(i)$ is represented as follows:

$$\chi(i) = \begin{cases} 1 & \text{if } i \in [M[v]+w, |X^\phi| + D[v] + w) \\ 0 & \text{if } i \notin [M[v]+w, |X^\phi| + D[v] + w) \end{cases}$$

where we also consider a case that i is larger so that $Xv^\phi_{ch}(i-w+1,w)$ is empty. In this case, $\chi(i)$ is obviously 0.

Second, $n(i)$ is the number of the additional vertices derived from $N(v)$. Its value can be represented as follows:

$$n(i) = \begin{cases} i - |X^\phi| & (|X^\phi| \leq i < |X^\phi| + D[v]) \\ D[v] & (|X^\phi| + D[v] \leq i < |X^\phi| + w) \\ D[v] - i + |X^\phi| + w & (|X^\phi| + w \leq i < |X^\phi| + D[v] + w) \\ 0 & (|X^\phi| + D[v] + w < i) \end{cases}$$

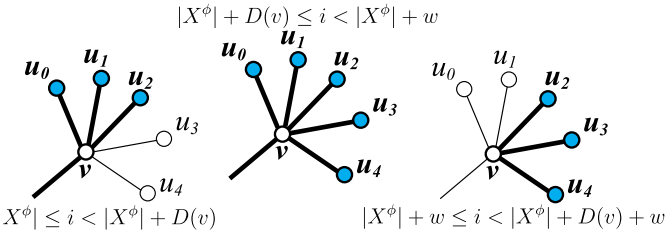

Fig. 11. The value of $n(i)$: # of the additional vertices derived from $N(v) = \{u_0, u_1, ..., u_4\}$. Blue vertices are the additional when v is selected for the expansion.

Figure 11 shows an example of these cases. Suppose v is selected in the greedy algorithm and new edge orders are assigned to v's neighbor edges, e_{v,u_0}, e_{v,u_1}, e_{v,u_2}, e_{v,u_3}, and e_{v,u_4}. Then, if $|X^\phi| \leq i < |X^\phi| + D[v]$, a part of $N(v)$ are added (e.g., u_0, u_1, u_2 in Figure 11). If $|X^\phi| + D[v] \leq i < |X^\phi| + w$, all vertices in $N(v)$ are added (e.g., u_0, u_1, u_2, u_3, u_4 in Figure 11). If $|X^\phi| + w \leq i < |X^\phi| + D[v] + w$, also a part of $N(v)$ are added (e.g., u_2, u_3, u_4 in Figure 11). If $|X^\phi| + D[v] + w \leq i$, then $Xv^\phi_{ch}(i-w+1,w)$ involves no vertices.

Therefore,

$$\begin{aligned}
\sum_{i=|X^\phi|}^{|E|-1} \Delta V(v,i) = & \sum_{i=|X^\phi|}^{|E|-1} \chi(i) + \sum_{i=|X^\phi|}^{|X^\phi|+D[v]-1} \{i - |X^\phi|\} + \sum_{i=|X^\phi|+D[v]}^{|X^\phi|+w-1} D[v] \\
& + \sum_{i=|X^\phi|+w}^{|X^\phi|+w+D[v]-1} \{D[v] - i + |X^\phi| + w\} \\
= \quad & wD[v] + |X^\phi| + D[v] - M[v]
\end{aligned}$$

Let $\Delta D := D[v] - D[u]$ and $\Delta M := M[v] - M[u]$.

$$\begin{aligned}
& \sum_{i=|X^\phi|}^{|E|-1} \{\Delta V(v,i) - \Delta V(u,i)\} = w\Delta D + \Delta D - \Delta M \\
& \sim w\Delta D - \Delta M \quad (\because w > \tfrac{|E|}{k_{max}} \gg 1)
\end{aligned}$$

Therefore,

$$\begin{aligned}
& p(v) > p(u) \\
\Leftrightarrow \quad & \alpha \cdot D[v] - \beta \cdot M[v] > \alpha \cdot D[u] - \beta \cdot M[u] \\
\Leftrightarrow \quad & \sum_{k=k_{min}}^{k_{max}} (w \cdot D[v] - M[v]) > \sum_{k=k_{min}}^{k_{max}} (w \cdot D[u] - M[u])
\end{aligned}$$

$$\Leftrightarrow \sum_{k=k_{min}}^{k_{max}} (w\Delta D - \Delta M) > 0$$

$$\Leftrightarrow \sum_{k=k_{min}}^{k_{max}} \sum_{i=|X^{\phi}|}^{|E|-1} \{\Delta V(v,i) - \Delta V(u,i)\} > 0$$

$$\Rightarrow F_v > F_u. \quad (\because (10))$$

Thus, the lemma is proved. ■

Appendix G
Complete Proof of Theorem 5

Theorem 5 (Efficiency of Fast Algorithm)**.** *Suppose PQ be a standard priority-queue implementation, where dequeue, update or enqueue can be operated in $O(\log n)$. Then, time complexity of Algorithm 3 is $O(d_{max}^2|V|\log|V|)$, where d_{max} is the maximum degree.*

Proof. The outermost loop (Lines 3 – 18) requires $O(|V|)$ for each vertex. Then, the inner loop (Lines 7 – 17) needs $O(d_{max})$ for each neighbors of v_{min}. Finally, the innermost loop (Lines 10 – 15) requires $O(d_{max})$ for each neighbors and $O(\log|V|)$ for updating PQ at Line 15. To sum up, the total time complexity of Algorithm 3 is $O(d_{max}^2|V|\log|V|)$. ■

Appendix H
Complete Proof of Theorem 6

Theorem 6 (Upper Bound of Partitioning Quality)**.** *Consider that in each iteration of Algorithm 3 (Lines 3-18) the number of new ordered edges is smaller than the smallest partition size and that $\delta = \left\lfloor \frac{|E|}{k_{max}} \right\rfloor - 1$.*

Let E be ordered by Algorithm 3 and then partitioned into k parts, $\mathcal{E}_k$, via chunk-based partitioning ($k_{min} \le k \le k_{max}$). Then, the replication factor, RF_k, for the k edge partitions has an upper bound as follows:

$$RF_k := \sum_{p=0}^{k-1} \frac{|V(\mathcal{E}_k[p])|}{|V|} \le \frac{|V|+|E|+k}{|V|}$$

Proof. Assume that k is given in advance. We consider a new partitioning algorithm based on the ordering algorithm (Algorithm 3) and conversion function $\texttt{ID2P}_k(i)$ (Algorithm 1) as follows:

- Initially, the k edge partitions, $\mathcal{E}_k$, are empty.
- Run Algorithm 3 and insert i-th ordered edge to $\mathcal{E}_k[\texttt{ID2P}_k(i)]$.

In the new partitioning algorithm, the edge partitions are incrementally determined from $\mathcal{E}_k[0]$, $\mathcal{E}_k[1]$, ..., to $\mathcal{E}_k[k-1]$. Obviously, the partitioning results obtained by the new partitioning algorithm is the same as the ones by our proposed method (i.e., completing Algorithm 3 *before* the chunk-based edge partitioning for k.)

Let t be an iteration counter for Lines 3-18 of Algorithm 3, and $\Phi(t)$ be a potential function over t defined as follows:

$$\Phi(t) := |V_{rest}(t)| + |E_{rest}(t)| + p_{rest}(t) + \sum_{p=0}^{k-1} |V(\mathcal{E}_k(t)[p])|,$$

where $V_{rest}(t)$ is a set of vertices adjacent to at least one non-ordered edge; $E_{rest}(t)$ is a set of non-ordered edges at t; $p_{rest}(t)$ is the number of partitions which still have spaces to insert edges; $\mathcal{E}_k(t)$ is a set of edge partitions at t.

Suppose the ordering algorithm terminates at T. We will show that (a) $\Phi(0) = |V| + |E| + k$, (b) $\Phi(T) = \sum_{p=0}^{k-1} V(\mathcal{E}_k[p])$, and (c) $\Phi(0) \ge \Phi(T)$. The first two equations (a) and (b) are obvious, i.e.,

$$\Phi(0) := |V_{rest}(0)| + |E_{rest}(0)| + p_{rest}(0) + \sum_{p=0}^{k-1} |V(\mathcal{E}_k(0)[p])|$$
$$= |V| + |E| + k$$

$$\Phi(T) := |V_{rest}(T)| + |E_{rest}(T)| + p_{rest}(T) + \sum_{p=0}^{k-1} |V(\mathcal{E}_k(T)[p])|$$
$$= \sum_{p=0}^{k-1} |V(\mathcal{E}_k[p])|$$

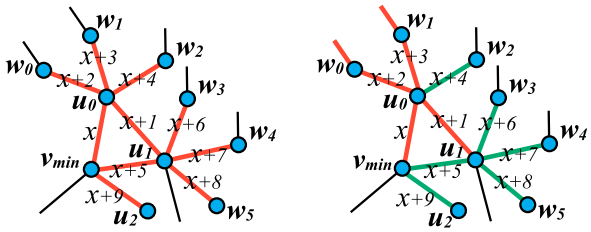

Fig. 12. One Iteration for Ordering.

For (c), we will show $\Phi(t) - \Phi(t-1) \ge 0$ for $0 < t \le T$. Let $\Delta\Phi := \Phi(t) - \Phi(t-1)$, $\Delta V_{rest} := |V_{rest}(t)| - |V_{rest}(t-1)|$, $\Delta E_{rest} := |E_{rest}(t)| - |E_{rest}(t-1)|$, $\Delta p_{rest} := p_{rest}(t) - p_{rest}(t-1)$, and $\Delta V(\mathcal{E}) := \sum_{p=0}^{k-1} |V(\mathcal{E}_k(t)[p])| - \sum_{p=0}^{k-1} |V(\mathcal{E}_k(t-1)[p])|$.

For t-th iteration where v_{min} is selected for expansion, we define the number of v_{min}'s one-hop neighbor edges which are ordered at t as n_{one} and the number of v_{min}'s two-hop neighbor edges which are ordered at t as n_{two}. For example in Figure 12, v_{min} is selected and the edges are ordered from x to $x+9$. Here, $n_{one} = 3$ (Edges x, $x+5$, $x+9$) and $n_{two} = 7$ (Edges $x+1, x+2, x+3, x+4, x+6, x+7, x+8$).

Then, $\Delta V_{rest} \le -1$ because all of v_{min}'s neighbor edges are assigned at the iteration. $\Delta E_{rest} = -(n_{one} + n_{two})$ from the definition.

$\Delta p_{rest} = 0$ if all the ordered edges during the iteration are inserted to the same partition as the previous iteration and the partition has still have free space (Case A). Otherwise (Case B), i.e., if the partitioning set becomes full during the current iteration, then $\Delta p_{rest} = -1$. Note that during an iteration of the algorithm, there cannot be more than one partitioning set that becomes full due to the assumption that the number of new assigned edges in each iteration is smaller than the smallest partition size.

For $\Delta V(\mathcal{E})$, we consider Case A and B as well. In Case A, $\Delta V(\mathcal{E}) \le 1 + n_{one} + n_{two}$ because v_{min} may be newly

inserted; n_{one} vertices and up to n_{two} vertices may be to the current partition. For example in Figure 12, $u_i (i = 0, 1, 2)$ are newly inserted as n_{one} and $w_i (i = 0, .., 5)$ are as $n_{two} - 1 (\leq n_{two})$.

In Case B, $\Delta V(\mathcal{E})$ satisfies the following equation:

$$\Delta V(\mathcal{E}) \leq 2 + (n_{one} + 2) + (n_{two} - 2). \tag{11}$$

As explained above in Case B, there cannot be more than one partitioning set that becomes full. This is translated into having up to two partitions at an iteration. Thus, let the two partitions be $\mathcal{E}_k[p_0]$ and $\mathcal{E}_k[p_1]$, where each edge is inserted to $\mathcal{E}_k[p_0]$ at first and then $\mathcal{E}_k[p_1]$ after splitting.

The first "2" in Eq.(11) means that v_{min} may be inserted to up to two partitions ($\mathcal{E}_k[p_0]$ and $\mathcal{E}_k[p_1]$). The second "$(n_{one} + 2)$" in Eq.(11) means that n_{one} vertices are inserted to either $\mathcal{E}_k[p_0]$ or $\mathcal{E}_k[p_1]$. In addition, up to two of n_{one} vertices may be inserted to both $\mathcal{E}_k[p_0]$ and $\mathcal{E}_k[p_1]$. This is because there may be up to two partitions at an iteration.

Let the splitting point be between i and $i + 1$. The last "$(n_{two} - 2)$" in Eq.(11) means that up to n_{two} vertices are inserted to either $\mathcal{E}_k[p_0]$ or $\mathcal{E}_k[p_1]$. But at least the last two edges for $\mathcal{E}_k[p_0]$ (i.e., $i - 1$-th and i-th edges) never increase the number of duplicated vertices. This is because the two-hop vertices adjacent to $i - 1$-th and i-th edges must belong to $\mathcal{E}_k[p_0]$. The above is proved as follows.

Let the two-hop vertices adjacent to $i - 1$-th and i-th edges be w and w'; X^{ϕ} be the ordered edges up to i-th edge. Note that w and w' must be in $V(X^{\phi}_{ch}(i - \delta, \delta))$ and $V(X^{\phi}_{ch}(i + 1 - \delta, \delta))$ respectively, due to the condition in Line 11 of Algorithm 3. Then, according to the assumption that $\delta = \left\lfloor \frac{|E|}{k_{max}} \right\rfloor - 1$, w is in

$$\begin{aligned}
&V\left(X^{\phi}_{ch}\left(i - \left(\left\lfloor \frac{|E|}{k_{max}} \right\rfloor - 1\right), \left\lfloor \frac{|E|}{k_{max}} \right\rfloor\right)\right) \\
&= V\left(X^{\phi}_{ch}\left(i + 1 - \left\lfloor \frac{|E|}{k_{max}} \right\rfloor, \left\lfloor \frac{|E|}{k_{max}} \right\rfloor\right)\right) \\
&\subseteq V\left(X^{\phi}_{ch}\left(i + 1 - \left\lfloor \frac{|E| + p_0}{k_{max}} \right\rfloor, \left\lfloor \frac{|E| + p_0}{k_{max}} \right\rfloor\right)\right) \\
&\subseteq V\left(X^{\phi}_{ch}\left(i + 1 - \left\lfloor \frac{|E| + p_0}{k} \right\rfloor, \left\lfloor \frac{|E| + p_0}{k} \right\rfloor\right)\right) \\
&= V(\mathcal{E}_k[p_0])
\end{aligned}$$

Similarly, w' is also in $V(\mathcal{E}_k[p_0])$.

For example in Figure 12, the ordered edges are split between $x + 3$ and $x + 4$ ($\mathcal{E}_k[p_0]$ is red. $\mathcal{E}_k[p_1]$ is green). In this case, v_{min} is inserted to both $\mathcal{E}_k[p_0]$ and $\mathcal{E}_k[p_1]$. u_0 and u_1 are also inserted to both, but u_2 is inserted only to $\mathcal{E}_k[p_1]$. w_0–w_5 are inserted to either $\mathcal{E}_k[p_0]$ or $\mathcal{E}_k[p_1]$, but w_0 and w_1 are not duplicated because they include edges which have already been assigned to $\mathcal{E}_k[p_0]$. Thus, in total, $\Delta V(\mathcal{E}) = 2 + 2 + 2 + 1 + 4 = 11$, which is smaller than $2 + (n_{one} + 2) + (n_{two} - 2) = 12$.

Then, to summarize the above discussion, $\Delta\Phi$ can be calculated as follows:

$$\begin{aligned}
&\Delta\Phi = \Delta V_{rest} + \Delta E_{rest} + \Delta p_{rest} + \Delta V(\mathcal{E}) \\
&\leq \begin{cases} -1 - (n_{one} + n_{two}) + 0 + 1 + n_{one} + n_{two} & \text{(Case A)} \\ -1 - (n_{one} + n_{two}) - 1 + 2 + (n_{one} + 2) + (n_{two} - 2) & \text{(Case B)} \end{cases} \\
&= 0
\end{aligned}$$

Therefore, $\Delta\Phi \leq 0$ and thus (c) hold.

Based on (a), (b), (c), we establish the following equation:

$$RF_k := \sum_{p=0}^{k-1} \frac{|V(\mathcal{E}_k[p])|}{|V|} = \frac{\Phi(T)}{|V|} \leq \frac{\Phi(0)}{|V|} = \frac{|V| + |E| + k}{|V|}$$

Finally, the theorem is proved. ■

APPENDIX I
ADDITIONAL EXPERIMENT

A. Migration Cost

We evaluate the migration cost in dynamic scaling (*ScaleOut* and *ScaleIn* in the previous section). We use three methods for the comparison: *BVC*, *1D*, and *CEP*. *BVC* is designed for the efficient migration as its objective is defined as the minimization of the migration cost (Def. 2). *1D* is a representative of the other partitioning methods that do not take the migration cost into account. Each partitioned edge may basically move to any of the other partitions.

Figure 13 shows the number of migrated edges in the two scenarios. *BVC* and *CEP* are almost the same number, outperforming *1D*. This is due to the fact that the migration methods in *BVC* and *CEP* are very similar, where their difference is to align the edges to the ordering id space (*CEP*) or to the hash ring in consistent hashing (*BVC*). In both methods, edges are split into the continuous chunks, and thus, the number of migrated edges is almost the same.

Figure 14 shows the actual elapsed time to migrate the edges and their values under the different network performances and sizes of each edge value. We emulate the different network bandwidth from 1Gbps to 32Gbps according to the instance specifications in Amazon EC2 [54]. The size of value per edge is changed from 0 to 32 bytes.

In contrast to the number of migrated edges, *CEP* and *1D* outperform *BVC*. This is because, in *BVC*, the edges are communicated in two phases: the initial migration and refinement for balancing edges. The refinement includes a lot of barrier synchronizations to share the edge balanceness among the distributed processes, especially in small ϵ and k. *BVC* is considered to be more appropriate for larger ϵ and k as evaluated in [20] (where ϵ is around 100 times bigger than our case and k is over 100). On the other hand, in *CEP* and *1D*, the graph data are communicated in the single data shuffling and do not include the multiple global synchronizations.

An interesting insight from the evaluation is that the performance difference/improvement in data migration time is relatively small even though the number of migrated edges

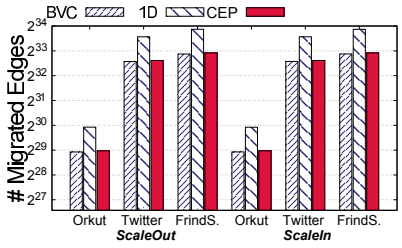

Fig. 13. Total # of Migrated Edges in *ScaleOut* and *ScaleIn*.

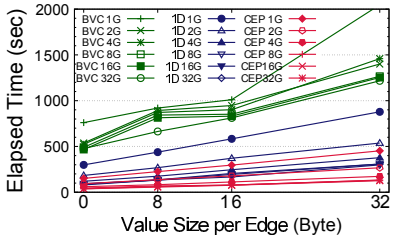

Fig. 14. Migration Time for *ScaleOut* with `FriendS`..

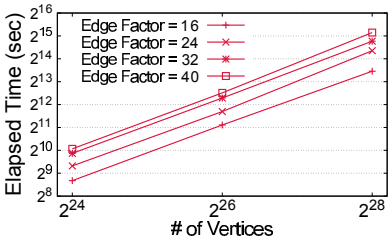

Fig. 15. Scalability of *GEO* with RMAT Graphs.

is largely different and the data migration itself is time-consuming (in some cases, it is slower than the partitioning time). In contrast, the partitioning time as shown in Figure 7 exhibits a lot of variation in each of the methods examined, and thus its performance improvement may substantially influence the overall workload. This is the reason why we focus on the performance of the partitioning time instead of the migration cost as discussed in Def. 2.

B. Scalability

Figure 15 shows the scalability of *GEO*. We use RMAT [55], a common synthetic model for social networks. According to the real-world social networks in Table II, we change Edge Factor of RMAT (i.e., average degree) from 16 to 40 and the graph size up to 10 billion-edge scale. Overall, the performance changes linearly as the increase of the graph size. However, *GEO* as well as its other counterparts (i.e., high-quality graph partitioning and graph ordering methods) have a scalability limitation. That is, if the preprocessing time is very large (e.g., due to the large graph size), whereas the actual analysis time is relatively small (e.g., due to the high parallelization), then the benefit by the preprocessing cannot be amortized. Such a limitation gives us the motivation to devise parallel and distributed algorithms to speed up *GEO*. This is listed as our future work in Section VII.