Supplementary Material of Optimal Gradient Checkpoint Search for Arbitrary Computation Graphs

Jianwei Feng, Dong Huang
Robotics Institute, Carnegie Mellon University
Pittsburgh, PA 15213
jfeng1@andrew.cmu.edu, donghuang@cmu.edu

1. Extra Examples

1.1. Extra Examples for Linear Computation Graph

Fig. 1: Example linear network for gradient checkpoint training. Letter denotes index of vertex and green vertices denote gradient checkpoints.

Fig. 1 is an example linear network for gradient checkpoint training. Vertices $v_i$, $v_j$, $v_k$ are stored in the first forward. During backward from $v_k$ to $v_j$, a local forward is conducted starting from $v_j$ to recompute $v_p$ and $v_q$. During backward from $v_j$ to $v_i$, $v_t$ is recomputed from $v_i$. The re-computed vertices are used to compute gradients for backward.

Fig. 2: Example of memory cost function for Linear Computation Graph. Number in the vertex denotes its memory cost, and green vertices denotes gradient checkpoints.

Fig. 2 explains how we compute memory loss on the linear computation graph with gradient checkpoint training. Recall that the loss function is as follows:

$$\min_{V^R} \left( \sum_i l(v_i^R) + \max_i l(v_i^R, v_{i+1}^R) \right), \quad (1)$$

In Fig. 2, $V^R$ is the GCs set (green vertices). $\sum_i l(v_i^R)$ is simply the sum of all the vertices in between. Therefore, maximal Re-forward loss is $\max \{ 13, 8 \} = 13$, and total Loss is $29 + 13 = 42$.

Fig. 3: Denote $C = \max_i l(v_i^R, v_{i+1}^R)$. Given $C = 8$, this is an example of Accessibility Graph. Number in the vertex denotes its loss. The red edges are newly added accessibility edges in the accessibility graph.

Fig. 3 shows an example of accessibility graph. Given a certain $C = \max_i l(v_i^R, v_{i+1}^R)$, in the accessibility graph, two vertex have accessibility edge as long as their Re-forwarding loss is not greater than $C$. For example, the first vertex (loss 10) and the third vertex (loss 9) has accessibility edge because the re-forwarding loss (8) is not greater than $C$.

1.2. Extra Examples for Arbitrary Computation Graph

Fig. 4: Example of gradient checkpointing for non-linear computation graph. Number in the vertex denotes its memory cost, and green vertices denote gradient checkpoints.

Fig. 4 shows an example of gradient checkpointing for non-linear computation graph. Letter denotes vertex index, number denotes vertex memory cost and green vertices denote gradient checkpoints. The edges in the right graph are folded Independent Segment in the left graph.

Fig. 4 shows an example of gradient checkpoint training.
ing for a non-linear computation graph. Suppose we select vertices \{v_i, v_k, v_i, v_j\} as gradient checkpoints. Based on the gradient checkpoints, the computation graph has IS \{s_{i\bar{k}}, s_{k\bar{i}}, s_{k\bar{j}}, s_{j\bar{i}}\}. If we view these IS as edges and fold the details within IS, we will get another ACG shown in the right of Fig. 4, where the vertices are GCs and the edges are IS in the original graph. During the first forward, only GCs \{v_i, v_k, v_i, v_j\} are stored. Re-forwarding and backward are conducted independently in each IS.

2. Proof

2.1. Definitions and Lemmas

Definition 1. Independent Segment(IS): An IS \(s_{ij} = (V^{ij}, E^{ij})\) satisfies the following three properties: 1. All the vertices in \(V^{ij}\) have a common ancestor \(v_i\) and a common descendant \(v_j\); 2. Denote the set of edges between two arbitrary vertices in \(V^{ij}\) is \(E^i\), the edge from \(v_i\) to \(v_j\) (if exists) as \(e_{ij}\). \(E^i\) must be either \(E^i\) or \(E^j\); 3. An arbitrary \(v_k \in \left(V^{ij} - \{v_i, v_j\}\right)\) doesn’t have edge with another arbitrary \(v_l \notin V^{ij}\). For multiple valid IS between \(v_i\) and \(v_j\), we denote the largest one as \(s_{ij}\).

Definition 1 is to clarify properties of IS in maths for the following proof.

Definition 2. \(s_{ij} = (V^{ij} - \{v_i\}, E^{ij})\). \(s_{ij} = (V^{ij} - \{v_i\}, E^{ij})\). \(s_{ij} = (V^{ij} - \{v_i, v_j\}, E^{ij})\).

For convenience, we define notations for IS \(s_{ij}\) excluding \(v_i\), or \(v_j\) or both.

Lemma 1. If \(s_{ij} \cap s_{kt} \neq \emptyset \) and \(s_{ij} \notin s_{kt}\) and \(s_{kt} \notin s_{ij}\), then \(s_{ij} \cap s_{kt} = (s_{kj})\) or \(s_{ij} \cap s_{kt} = (s_{tk})\).

Proof. Let \(s_{ij} \cap s_{kt} = s_{pq} = (V^{pq}, E^{pq})\). If \(v_p \neq v_i\) and \(v_p \neq v_k\) and \(v_q \neq v_j\) and \(v_q \neq v_j\), then \(v_i, v_k\) has path to \(v_p\) and \(v_j, v_q\) has path from \(v_p\). Therefore, \(v_p\) has at least 2 immediate parents \(v_a, v_b\) with \(v_a \in s_{ij}, v_b \in s_{kt}, v_b \notin s_{ij}\). If so, the independence of \(s_{ij}\) and \(s_{kt}\) is violated. Therefore, \(v_p\) must be \(v_i\) or \(v_k\).

Same on \(v_q, v_q\) must be \(v_i\) or \(v_j\).

If \(v_p = v_i, v_q = v_j\), then \(s_{ij} \subset s_{kt}\). If \(v_p = v_k, v_q = v_t\), then \(s_{kt} \subset s_{ij}\). Therefore, \(v_p = v_i, v_q = v_j\) or \(v_p = v_k, v_q = v_t\). Suppose \(v_p = v_k, v_q = v_t\), let’s prove \(s_{pq}\) is an IS.

With \(s_{pq} \subset s_{kt}, \forall v_l \in (s_{pq}), v_l\) has no edge with \(v_2 \notin (s_{ij})\). With \(s_{pq} \subset s_{ij}, \forall v_l \in (s_{pq}), v_l\) has no edge with \(v_2 \notin (s_{ij})\). Therefore, \(\forall v_l \in (s_{pq}), v_l\) has no edge with \(v_2 \notin (s_{pq})\). The independence of \(s_{pq}\) is guaranteed.

In the discussion before, we can see the source vertex \(v_p\) of \(s_{pq}\) must be either \(v_i\) or \(v_k\). If \(v_i\) and \(v_k\) are both the source vertices of \(s_{pq}\), then \(v_i \in s_{kt}\) and \(v_k \in s_{ij}\), \(v_i\) has path to \(v_k\) and \(v_k\) has path to \(v_i\), which will force \(v_i = v_k\) because the \(s_{ij}, s_{kt}\) is acyclic. Same on \(v_q, s_{pq}\) can only have 1 source vertex and 1 target vertex. Therefore, \(s_{pq}\) is an IS.

Therefore, \((s_{ij}) \cap (s_{kt}) = (s_{kj})\) or \((s_{ij}) \cap (s_{kt}) = (s_{ij})\).

Lemma 2. The intersection of IS \(s_{pq} = s_{ij} \cap s_{kt} \neq \emptyset\) is also an IS.

Proof. Given the independence of \(s_{ij}\) and \(s_{kt}\), the independence of \(s_{pq}\) is obvious. The remaining thing is whether \(s_{pq}\) only has 1 source vertex and 1 target vertex. In the proof of Lemma 1, we can see any source or target vertex of \(s_{pq}\) will eventually become source or target vertex of \(s_{ij}\) and \(s_{kt}\). With simple discussion, we can have this lemma.

Lemma 3. If \(s_{ij} \cap s_{kt} = s_{kj} \neq \emptyset\), then \(v_k\) is the linear splitting vertex of \(s_{ij}\) and \(v_j\) is the linear splitting vertex of \(s_{kt}\).

Proof. Let’s first prove that \(v_k\) is the linear splitting vertex of \(s_{ij}\).

Let \(s_{ik} = s_{ij} - s_{kj} + \{v_k\}\). Obviously, \(s_{ij} = s_{ik} \cup s_{kj}\) and \(s_{ik} \cap s_{kj} = \{v_k\}\). We only need to prove that \(s_{ik}\) is IS.

\(v_i\) is obviously the only source vertex of \(s_{ik}\) because \(v_i\) is source vertex of \(s_{ij}\). We discuss the target vertex here. If \(v_k\) is not the target vertex of \(s_{ik}\), as \(v_k \in s\), \(v_k\) must have path to the target vertex \(v\) of \(s_{ik}\) and \(v\) also has path to \(v_j\) as \(v \in s_{ij}\). Because \(v \notin s_{kj}\), in the path from \(v\) to \(v_j\), there exists an edge that connects a vertex \(v_1 \in s_{ik}\) with a vertex \(v_2 \in s_{kt}\) which violates the independence of \(s_{kt}\). Therefore, the target vertex of \(s_{ik}\) can only be \(v_k\).

As \(s_{ik} \subset s_{ij}\), \(v_1 \in s_{ik}, v_1\) has no edge with \(v_2 \notin s_{ij}\). As \(s_{kj}\) is IS, \(v_1 \in (s_{kj}), v_1\) has no edge with \(v_2 \in (s_{kj})\). \(v_1 \in (s_{kj}), v_1\) can only have edge with \(v_k \in s_{ik}\). Thus the independence of \(s_{ik}\) is guaranteed. Therefore, \(s_{ik}\) IS, \(v_k\) is the linear splitting vertex of \(s_{ij}\).

Similarly, \(v_j\) is the splitting vertex of \(s_{kt}\).

Lemma 4. If \(s_{ij}\) has \(n\) splitting vertices \(\{v_1, v_2, ..., v_n\}\), then \(s_{ij} = s_{i1} \cup s_{i2} \cup ... \cup s_{in}\).

Proof. If \(n = 2\), the splitting vertices are \(v_1, v_2, s_{ij} = s_{i1} \cup s_{i2}\). \(s_{i1} = s_{i2} \cup s_{j}\). Let \(v_1 \in s_{i2}, v_1 \neq v_2\), then \(s_{ij} \cap s_{i2} = \emptyset\). According to Lemma 3, \(v_1\) is linear splitting vertex of \(s_{i2}\) and \(v_2\) is linear splitting vertex of \(s_{1j}\). Therefore, \(s_{ij} = s_{i1} \cup s_{i2} \cup s_{j}\).

For \(n > 2\), the lemma can be proved by repetitively using the conclusion in \(n = 2\).

Lemma 5. If the Complicate IS \(s_{ij}\) has division \(\{s_{pq}\}\), and \(|\{s_{pq}\}| > 2\), denote \(\{v\}\) as all the connecting vertices of \(s \in \{s_{pq}\}\). Then \(\forall v \in \{v\}, v \neq v_1, v_2, v_3\) is the connecting vertex of at least 3 member IS of the division.
Proof. If $v_b$ is the connecting vertex of only 2 member IS, suppose the 2 members are $s_{ab}$ and $s_{bc}$. If so, $s_{ab}$ and $s_{bc}$ can be merged into $s_{ac}$. If $s_{ac} \neq s_{ij}$, this violates the definition of division of Complicate IS. Otherwise, it violates the condition that $s_{ij}$ is not a Branch IS and $|\{s_{pq}\}| > 2$.

It is impossible that the 2 members are $s_{ab}$ and $s_{bc}$ because in this way $v_b$ has no path to $v_a$ and violates the definition of IS. If $v_b$ is the connecting vertex of only 1 member IS of the division, then $v_b$ must be either $v_i$ or $v_j$. Therefore, this lemma is proved.

\[ \square \]

Lemma 6. For Complicate IS $s_{ij}$, any member IS $s_{pq}$ of its division can not be the subset of another IS $s_{kt}$, i.e. $s_{pq} \subsetneqq s_{kt}$.

Proof. The source vertex of $s_{kt}$ is $v_k$ and target vertex is $v_t$. Suppose a member IS $s_{pq} \subsetneqq s_{kt}$.

Suppose another member IS $s_{ab}$ of the division has its source vertex $v_a$ inside $s_{kt}$ and target vertex $v_b$ outside $s_{kt}$. Then the boundary vertex (the vertex that has edges to the non-overlapping parts of both sets) must be $v_t$, otherwise the independence of $s_{kt}$ will be violated. Notice that $v_t$ is inside $s_{ab}$ and the independence of $s_{ab}$ needs to be guaranteed, for $\forall v_x \in s_{kt}, v_x \notin s \cap s_{ab}, v_y \in s_{kt} \cap s_{ab}, v_y$ has no edge with $v_y$. Therefore, $v_a$ is a splitting vertex of $s_{kt}$.

Similarly, if $s_{ba}$ has its target vertex $v_a$ inside $s_{kt}$ and source vertex $v_b$ outside $s_{kt}$, the boundary vertex must be $v_b$ and $v_b$ is a splitting vertex of $s_{kt}$.

For the member IS $s_{kt}$, from the discussion above, we know that there are at most 2 members in the division that can overlap with $s_{kt}$. Other members must be either completely inside $s_{kt}$ or completely outside $s_{kt}$. Let’s discuss the number of members that overlaps with $s_{kt}$.

If there are 0 member that overlaps with $s_{kt}$, $s_{kt}$ is the union of a subset of members of the division, which violates the definition of division.

If there is 1 member that overlaps with $s_{kt}$, suppose the corresponding splitting vertex is $v_b$, and the boundary vertex is actually $v_t$. Then $s_{hb}$ is an IS containing $s_{pq}$ and corresponds to the situation of 0 member overlapping. $s_{hb}$ is the union of a subset of members of the division, and violates the definition of maximal split.

If there are 2 members that overlaps with $s_{kt}$, suppose they generate two different splitting vertex $v_{b1}$ and $v_{b2}$. Then $s_{ab}$ is an IS containing $s_{pq}$ and corresponds to the situation of 0 member overlapping. $s_{ab}$ is the union of a subset of members of the division, and violates the definition of division.

If they generate the same splitting vertex $v_{b1}$, from lemma 5, $v_b$ is also the connecting vertex of at least 1 other member $s_{ab}$ which has to be inside $s_{kt}$. Suppose the two overlapping members are $s_{bc}$ that contains $v_k$, and $s_{bd}$ that contains $v_i$. As the source vertex of $s_{kt}$, $v_k$ has path to $v_b$ and $v_k$ has path to $v_a$, which implies $v_b$ has path to $v_a$. As the target vertex of $s_{kt}$, $v_t$ has path from $v_b$ and $v_t$ has path from $v_a$, which implies $v_b$ has path from $v_a$. This conflicts with the fact that $s_{kt}$ is acyclic. Therefore, this case is not possible.

Therefore, this lemma is proved.

\[ \square \]

Lemma 7. If Complicate IS $s_{ij}$ has at least 1 vertex excluding $v_i$ and $v_j$, then its division has length $> 2$.

Proof. As $s_{ij}$ is Complicate, the members IS of its division cannot have the source vertex as $v_i$ and target vertex as $v_j$ at the same time. If $s_{ij}$ has at least 1 vertex excluding $v_i$ and $v_j$, and its division has length 2, then its division must be $\{s_{ik}, s_{kj}\}$, and $v_k$ will be the linear splitting vertex of $s_{ij}$, which violates that $s_{ij}$ has no splitting vertex.

If $s_{ij}$ has no splitting vertex, it has at least 2 edges and cannot have a trivial length 1 maximal split. Therefore, its maximal split has length $> 2$.

\[ \square \]

2.2. Uniqueness of Division Tree

To prove this uniqueness, we simply discuss the uniqueness of division of Linear IS, Branch IS and Complicate IS.

2.2.1 Uniqueness of Division of Linear IS

Proof. By the definition of Linear IS division and Lemma 4, the uniqueness of the division is equivalent to the uniqueness of the linear splitting vertex set of a Linear IS. The linear splitting vertex set is obviously unique.

\[ \square \]

2.2.2 Uniqueness of Division of Branch IS

Proof. If there exists another division, there must be a branch member $s_{ij}^1$ in division 1 and a branch member $s_{ij}^2$ in division 2, where $s_{ij}^1 \cap s_{ij}^2 \neq \{v_i, v_j\}$ and $s_{ij}^1 \neq s_{ij}^2$.

Denote $s_{ij}^3 = s_{ij}^1 \cap s_{ij}^2$. By Lemma 1 and 2, $s_{ij}^3$ is also an IS. From the definition of the division, we know $s_{ij}^1$ and $s_{ij}^2$ cannot be divided into more branches, $s_{ij}^1 = s_{ij}^3 = s_{ij}^2$. Therefore, the division of Branch IS is unique.

\[ \square \]

2.2.3 Uniqueness of Division of Complicate IS

Proof. As the IS in the division tree has at least 1 vertex excluding source and target vertex, with Lemma 7, we know that the division of Complicate IS $s_{ij}$ will have length $> 2$. Denote this division as $\{s_{pq}\}$, we only need to prove this division is unique.

Suppose there is another different division $\{s'_{pq}\}$, let us only check the difference between $\{s_{pq}\}$ and $\{s'_{pq}\}$. Denote $\{s_{kt}\}$ and $\{s'_{kt}\}$ with $\{s_{pq}\} - \{s_{kt}\} = \{s'_{pq}\} - \{s'_{kt}\}$ and $\exists s \in \{s_{kt}\}, s' \in \{s'_{kt}\}, s = s'$. As $\{s_{pq}\} - \{s_{kt}\} = \{s_{pq}\} - \{s'_{kt}\}$, we have $\bigcup\{s_{kt}\} = \bigcup\{s'_{kt}\}$. 

\[ \square \]
Obviously, $|\{s_{kt}\}| \geq 2$ and $|\{s'_{kt}\}| \geq 2$. Denote $\{v\}$ as all the connecting vertices of $s \in \{s_{kt}\}$, and $\{v'\}$ for $\{s'_{kt}\}$. Obviously $\{v\} \neq \emptyset$ and $\{v'\} \neq \emptyset$. As $s_{ij}$ is not Branch IS, $\{v\} \cup \{v, v_j\} \neq \emptyset$ and $\{v'\} \cup \{v, v_j\} \neq \emptyset$.

Suppose $s_{ab}, s_{bc} \in \{s_{kt}\}$, according to Lemma 5, there’s at least 1 other member IS that has $v_b$ as connecting vertex. Suppose the other connecting vertex of this member IS is $v_d$. Let’s discuss whether $v_b \in \{v'\}$ and whether $v_d \in \cup\{s_{kt}\}$.

If $v_d \notin \cup\{s_{kt}\}$, then $v_b$ must occur in $\{v'\}$. Otherwise, $v_b$ would be inside an IS which would be violated by $v_d$. Given $v_b \in \{v\}$, as $s_{ab} \notin \{s'_{kt}\}$, suppose $s_{eb} \in \{s'_{kt}\}$ and $s_{ab} \cap s_{eb} \neq \emptyset$. If $v_a \in s_{eb}$, from Lemma 1, $s_{ab}$ is invalid IS. If $v_e \in s_{ab}$, from Lemma 5, $s_{ab}$ is invalid IS. In this case, there cannot exist another different division.

If $v_d \in \cup\{s_{kt}\}$, then $s_{ab} \notin \{s_{kt}\}$. If $v_b \in \{v'\}$, we can use the same logic above to show this is impossible. Therefore, $v_b \notin \{v'\}$ and $s_{pq}$ is contained by an IS $s$. From Lemma 6, this is impossible. In this case, there cannot exist another division.

In all the cases, there cannot exist another division. Therefore, the division of Complicate IS is unique.

2.3. Completeness of Division Tree

Similar with the uniqueness, the completeness of division tree is equivalent to the completeness of the division of an IS. To prove this completeness, we simply discuss the completeness of division of Linear IS, Branch IS, and Complicate IS.

An equivalent statement of the completeness of the division is: there doesn’t exist an IS whose source vertex is in one member IS of the division and whose target vertex is in another member IS of the division.

2.3.1 Completeness of Division of Linear IS

Proof. Suppose there exists an IS $s_{pq}$ whose source vertex $v_p$ is in one member IS $s_1$ and whose target vertex $v_q$ is in another member IS $s_2$.

If $v_p$ is not an endpoint vertex of $s_1$, then according to Lemma 3, $v_p$ is also a linear splitting vertex in $s_1$ and can break $s_1$ into smaller IS, which makes $v_p$ also the linear splitting vertex of the whole IS $s_{ij}$. However, $v_p$ is not the splitting vertex of the whole IS $s_{ij}$. This also applies to $v_q$. Therefore, the division of Linear IS is complete.

2.3.2 Completeness of Division of Branch IS

Proof. Suppose there exists an IS $s_{pq}$ whose source vertex $v_p$ is in one branch $s^1_{ij}$ and whose target vertex $v_q$ is in another branch $s^2_{ij}$. As $s_{pq}$ crosses $s^1_{ij}$ and $s^2_{ij}$, there exists a boundary vertex $v$ in $s_{pq}$, which belongs to $s^1_{ij}$ and has direct connection with a vertex outside $s^2_{ij}$. If $v$ is not $v_i$ or $v_j$, it will violate the independence of $s_{ij}$. If $v = v_i$, as $v_i$ is the source vertex of both $s^1_{ij}$ and $s^2_{ij}$, it cannot be the boundary vertex, same when $v = v_j$. Therefore, there cannot exist such an IS $s_{ij}$. The division of Branch IS is complete. □

2.3.3 Completeness of Division of Closed Set Type 3

Proof. Suppose there exists an IS $s_{pq}$ whose source vertex $v_p$ is in one member IS $s_1$ and whose target vertex $v_q$ is in another member IS $s_2$. Same with Branch IS, the boundary vertex $v$ has to be the endpoint vertex of $s_1$ or the independence of $s_1$ will be violated. According to Lemma 5, $v$ is the connecting vertex of at least 3 members, meaning that $v$ will at least have 1 connection with another IS $s_3$. To maintain the independence of $s_{pq}$, $s_{pq}$ has to contain $s_3$ as well. However, $s_3$ also has its endpoint vertices. This will propagate until $s_{pq}$ becomes the whole closed set $s_{ij}$. Therefore, there cannot exist such an IS $s_{pq}$. The division of Complicate IS is complete. □

3. Complexity Analysis

3.1. Algorithm 1

Suppose there are $|V|$ vertices and $|E|$ edges in the computation graph. There are $O(|V|^2)$ vertex pairs. For each vertex pair, the time cost is mainly on constructing accessibility graph and finding the shortest path. Denote the source vertex of the whole computation graph as $v_0$. To construct an accessibility graph, first we traverse the linear computation graph, record the accumulated sum $l(v_0, v_i)$ for each vertex $v_i$, and form a table of $l(v_i, v_j) = l(v_0, v_j) - l(v_0, v_i) - l(v_i)$. These steps will cost $O(|V|^2)$ and will only run once. Then we traverse each $(v_i, v_j)$ pair to form the edges of the accessibility graph, which also cost $O(|E|)$. Solving the shortest path problem in accessibility graph will cost $O(|V||\log|V| + |E|)$. Therefore, the overall time complexity of Algorithm 1 would be $O(|V|^2|E| + |V|^3\log|V|)$.

The space complexity would be $O(|V|^2)$ for the table of $l(v_i, v_j)$ and the accessibility graph itself.

3.2. Algorithm 2

Suppose there are $|V^{ij}|$ vertices and $|E^{ij}|$ edges in the IS $s_{ij}$. In step 2, getting $\{v_{in}\}$ and $\{v_{out}\}$ will cost $O(|V^{ij}|)$ time for traversing the ancestors and descendants of $v_i$. In our implementation, an array $a$ of length $|V^{ij}|$ is used to represent $\{v_{in}\}$ and $\{v_{out}\}$: $a_i = 1$ indicates $v_i \in \{v_{in}\}$, $a_i = 2$ indicates $v_i \in \{v_{out}\}$ and $a_i = 0$ indicates $v_i \notin \{v_{in}\} \cup \{v_{out}\}$. Then the union check and intersection check in step 3 can be done in $O(|V^{ij}|)$. The
connection check in step 3 traverses the edges and costs \(O(|E^{ij}|)\). Other steps are \(O(1)\). Therefore, the overall time complexity of Algorithm 2 would be \(O(|V^{ij}|^3)\).

The space complexity would be \(O(|V^{ij}|)\) for the array to represent \(\{v_{in}\}\) and \(\{v_{out}\}\).

### 3.3. Algorithm 3

Suppose there are \(|V^{ij}|\) vertices in the IS \(s_{ij}\). The most time consuming part will be from step 7 to step 18. Other steps are \(O(1)\). In step 7 to step 18, every edge between two vertices in \(s_{ij}\) is visited once and there are \(O(E^{ij})\) edges. Therefore, the overall time complexity of Algorithm 3 would be \(O(E^{ij})\).

In our implementation, an array of length \(|V^{ij}|\) is used to represent the vertex set \(s = \{v_k\}\). Therefore, the space complexity would be \(O(|V^{ij}|)\).

### 3.4. Algorithm 4

Suppose there are \(|V^{ij}|\) vertices and \(|E^{ij}|\) edges in the IS \(s_{ij}\) and there are \(O(|V^{ij}|^2)\) vertex pairs. For each vertex pair, the connection check in step 2-4 will cost \(O(|E^{ij}|)\), similar to the connection check in Algorithm 2. Thus step 1-4 will cost \(O(|V^{ij}|^2|E^{ij}|)\). In our implementation, for each vertex in the IS \(s_{ij}\), we select the largest formed IS \(s_{k\ell}\) that contains this vertex. This IS number is reduced to \(O(|V^{ij}|)\) and step 5-6 can be done in \(O(|V^{ij}|^3)\). Therefore, the overall time complexity of Algorithm 4 would be \(O(|V^{ij}|^2|E^{ij}| + |V^{ij}|^3)\).

As \(O(|V^{ij}|^2)\) IS can be formed in step 1-4 and each closed set is a smaller DAG with \(O(|V^{ij}|)\) vertices and cost \(O(|V^{ij}|^2)\) space, the space complexity would be \(O(|V^{ij}|^4)\) for all these closed sets.

### 3.5. Algorithm 5

Given a max term \(C\), the actual time consuming part in the recursion will be step 9 which calls the LCG solver under constraint \(C\), and other steps would be \(O(1)\). Suppose the LCG solver is called \(k\) times, solving problems of \(a_1, a_2, ..., a_k\) vertices and \(e_1, e_2, ..., e_k\) edges. The total complexity of this would be \(O(a_1\log a_1 + e_1) + O(a_2\log a_2 + e_2) + ... + O(a_k\log a_k + e_k)\). Notice that \(a_1 + a_2 + ... + a_k \leq |V|\) for the fact that any vertex in the computation graph would not be solved twice by LCG solver, and \(e_1 + e_2 + ... + e_k \leq |E|\), we have \(a_1\log a_1 + e_1 + a_2\log a_2 + e_2 + ... + a_k\log a_k + e_k \leq |V|\log |V| + |E|\). Therefore the time complexity of Algorithm 5 is \(O(|V|\log |V| + |E|)\).

### 3.6. Algorithm 6

Step 1 is similar to step 1-4 in Algorithm 4 with \(s_{ij}\) being the whole computation graph. Therefore, the overall time complexity for step 1 is \(O(|V|\log |V| + |E|)\).

In step 2, the complexity of building division tree is related to the complexity of getting the division of an IS. For Linear IS, getting the division cost \(O(|V|^3)\) time. For Branch IS, Algorithm 3 is used to get its division and costs \(O(|E|)\) time. For Complicate IS, Algorithm 4 is called to solve for its division. Notice that we have already stored all possible IS in step 1, step 1-4 in Algorithm 4 can be skipped thus the time complexity of getting the division of Complicate IS is reduced to \(O(|V|^3)\). Therefore, getting the division of an arbitrary IS costs \(O(|V|^3)\) time. In depth \(i\) of the division tree, suppose there are \(k\) IS, and the number of vertices of \(j\)th closed sets is \(a_j\). To build depth \(i + 1\) of the division tree, we need to get the division of all these IS, which will cost \(\sum_j O(a_j^3)\). As \(\sum_j a_j \leq |V|\), we have \(\sum_j O(a_j^3) \leq O(|V|^3)\). As the depth of division tree is \(\log |V|\), the overall time complexity of step 2 would be \(O(|V|^3\log |V|)\).

For the loop, the length of \(\{e\}\) would be \(O(|V|^2)\) for there are \(O(|V|^2)\) vertex pair, and the recursion costs \(O(|V|\log |V| + |E|)\). Therefore the time complexity of the loop is \(O(|V|^2|E| + |V|^3\log |V|)\).

Step 1 would cost \(O(|V|^4)\) space to store all the possible closed sets. Step 2 would cost \(O(|V|^2)\) space for the division tree. the loop would cost \(O(|V|^2)\) space for calling LCG solver.

In conclusion, the overall time complexity of Algorithm 6 is \(O(|V|^2|E| + |V|^3\log |V|)\) and the overall space complexity of Algorithm 6 is \(O(|V|^4)\).

### 4. Runtime Analysis

The number of vertices in the computation graph and the preprocess time of ACG Solver (Algorithm 6) for each network are listed in Table 1. All the preprocess time were measured on a desktop.

Although it might be concerning that the preprocess time is too much for some deep networks, it is still relatively small compared to training processes which might cost days or even weeks. More importantly, solving the optimal solution for a network is an one-time effort. The optimal solutions for all popular networks will be released online for people to use without taking the time to run ACG solver.
5. Visualization

We visualize the computation graph of Alexnet, vgg11, vgg13, vgg16, vgg19, and CustomNet and the solution of our approach (in green) and the solution of Chen’s approach (in red). In the computation graphs, the cost of each vertex and the actual operation of each edge are also marked. The cost of each vertex is the size of this tensor during forward given the input as $[1, 3, 224, 224]$ ([1, 3, 300, 300] for inception v3). For example, in Alexnet, the input is $[1, 3, 224, 224]$ and thus the source vertex has the cost $150528 = 1 \times 3 \times 224 \times 224$. After 2D convolution and relu, the tensor becomes $[1, 64, 55, 55]$ and thus the second vertex has the cost $193600 = 1 \times 64 \times 55 \times 55$. 

<table>
<thead>
<tr>
<th>Linear network</th>
<th>Number of vertices</th>
<th>Preprocess Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alexnet</td>
<td>12</td>
<td>0.097</td>
</tr>
<tr>
<td>Vgg11</td>
<td>17</td>
<td>0.198</td>
</tr>
<tr>
<td>Vgg13</td>
<td>19</td>
<td>0.274</td>
</tr>
<tr>
<td>Vgg16</td>
<td>22</td>
<td>0.416</td>
</tr>
<tr>
<td>Vgg19</td>
<td>25</td>
<td>0.531</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Non-linear network</th>
<th>Number of vertices</th>
<th>Preprocess Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resnet18</td>
<td>51</td>
<td>0.290</td>
</tr>
<tr>
<td>Resnet34</td>
<td>91</td>
<td>0.502</td>
</tr>
<tr>
<td>Resnet50</td>
<td>125</td>
<td>0.910</td>
</tr>
<tr>
<td>Resnet101</td>
<td>244</td>
<td>2.690</td>
</tr>
<tr>
<td>Resnet152</td>
<td>363</td>
<td>6.576</td>
</tr>
<tr>
<td>Densenet121</td>
<td>306</td>
<td>35.024</td>
</tr>
<tr>
<td>Densenet161</td>
<td>406</td>
<td>79.754</td>
</tr>
<tr>
<td>Densenet169</td>
<td>426</td>
<td>87.111</td>
</tr>
<tr>
<td>Densenet201</td>
<td>506</td>
<td>160.808</td>
</tr>
<tr>
<td>Inceptionv3</td>
<td>219</td>
<td>4.344</td>
</tr>
<tr>
<td>NASNet</td>
<td>1149</td>
<td>39.098</td>
</tr>
<tr>
<td>AmoebaNet</td>
<td>1015</td>
<td>30.250</td>
</tr>
<tr>
<td>DARTS</td>
<td>1073</td>
<td>28.483</td>
</tr>
</tbody>
</table>
Figure 5: Gradient checkpoints found on Alexnet

Figure 6: Gradient checkpoints found on vgg11

Figure 7: Gradient checkpoints found on vgg13
Figure 8: Gradient checkpoints found on vgg16