APPENDIX A

Proof of Theorem 1. When computing $N(e)$, we need to iterate through all vertices $u \in e$. Then, we use $E(u)$ to track every possible neighbour e' and the calculate $OD(e, e')$. It takes $O(\delta d)$ time to process neighbours for one hyperedge. We need to invoke up to m times the procedure to traverse the whole hypergraph. Therefore, the total time complexity is $O(m\delta d)$. □

Proof of Lemma 1. This can be easily derived according to the concept of Definition 7. □

Proof of Lemma 2. ($\Rightarrow$) If e_w transitively covers $\{u, e, t\}$, then $e_w \overset{t}{\rightsquigarrow} e$ and $O(e_w) < O(e)$. Since $e_w \overset{t}{\rightsquigarrow} e$ and $e \overset{t}{\rightsquigarrow} e_u$, we have $e_w \overset{t}{\rightsquigarrow} e_u$. Therefore, e_w transitively covers $e \overset{t}{\rightsquigarrow} e_u$.

($\Leftarrow$) If e_w transitively covers $e \overset{t}{\rightsquigarrow} e_u$, then $e \overset{t}{\rightsquigarrow} e_w$, $e_w \overset{t}{\rightsquigarrow} e_u$ and $O(e_w) < O(e)$. Since $e_w \overset{t}{\rightsquigarrow} e_u$ and $e_u \in E(u)$, we have $u \overset{t}{\rightsquigarrow} e_w$. Therefore, e_w transitively covers $\{u, e, t\}$. □

Proof of Lemma 3. *i*) If e_u is of the highest importance in $P(e, e_u)$, that is, $O(e_u) < O(e)$, clearly, e_u transitively covers $\{u, e, t\}$.

ii) If e_w is of the highest importance in $P(e, e_u)$ and $e_w \neq e$, $e_w \neq e_u$, then $O(e_w) < O(e)$ and $O(e_w) < O(e_u)$. Since $POD(P(e, e_u)) \geq t$, we have $e_u \overset{t}{\rightsquigarrow} e_w$ and $e_w \overset{t}{\rightsquigarrow} e$. Therefore, $u \overset{t}{\rightsquigarrow} e_w$, and e_w transitively covers $\{u, e, t\}$. □

Proof of Theorem 2. Suppose we have a pair of vertices u and v such that $MR(u, v) = k$ cannot be correctly answered based on $\mathcal{L}$. Since $MR(u, v) = k$, there should exist at least one hyperpath $P(e_v, e_u)$ such that $u \in e_u$, $v \in e_v$ and $POD(P(e_u, e_v)) = k$. Let e be the most important hyperedge across all such hyperpaths, we have $e \overset{t_u}{\rightsquigarrow} e_u$ and $e \overset{t_v}{\rightsquigarrow} e_v$ where $\min(t_u, t_v) = k$, and we have $(e, t_u) \notin \mathcal{L}(u)$, $(e, t_v) \notin \mathcal{L}(v)$ as $MR(u, v) = k$ cannot be correctly answered. Following Algorithm 2, we will not have (e, t_u) in $\mathcal{L}(u)$ only if any one of line 8 or line 14 holds. That is, either there exists a hyperedge e_w with higher importance than e such that $e_w \overset{t_u}{\rightsquigarrow} e$ and $e_w \overset{t_u}{\rightsquigarrow} e_u$, i.e., e_w is in a hyperpath $P(e, e_u)$ with path overlapping degree $POD(P(e, e_u)) \geq t_u$; or there exists a hyperedge e_v with higher importance than e in the hyperpath $P(e, e_u)$ with path overlapping degree $POD(P(e, e_u)) = t_u$. In both cases, e is not the most important hyperedge across all hyperpaths $P(e_v, e_u)$ with $POD(P(e_u, e_v)) = k$, which contradicts our assumption. Therefore, the MR queries between any pair of vertices $u, v \in \mathcal{V}$ can be correctly answered by $\mathcal{L}(u)$ and $\mathcal{L}(v)$. □

Proof of Lemma 4. ($\Rightarrow$) If there exists a hyperedge e_w with $O(e_w) < O(e)$ that transitively covers $e \overset{t}{\rightsquigarrow} e_u$, then $e_w \overset{t}{\rightsquigarrow} e$, that is, there exists a hyperpath $P(e_w, e)$ with $POD(P(e_w, e)) = t$. Therefore, following the definition of maximum cover degree, $MCD(e) \geq t$.

($\Leftarrow$) Suppose $MCD(e) = k \geq t$, based on the definition of maximum cover degree, there exists a hyperpath $P(e_w, e)$ with $O(e_w) < O(e)$ and $POD(P(e_w, e)) = k$. Given $e \overset{t}{\rightsquigarrow} e_u$, i.e., there exists a hyperpath $P(e, e_u)$ with $POD(P(e, e_u)) = t$, then we can form a hyperpath $P(e_w, e_u) = P(e_w, e) \oplus P(e, e_u)$ with $POD(P(e_w, e_u)) = \min(k, t) = t$, that is, $e_w \overset{t}{\rightsquigarrow} e_u$. Together with $e_w \overset{t}{\rightsquigarrow} e$, we can conclude $e \overset{t}{\rightsquigarrow} e_u$ is transitively covered by e_w. □

Proof of Lemma 5. Based on Lemmas 2, 3 and the construction workflow mentioned in Algorithm 2, we early terminated the hyperpath P that reaches hyperedge e when *i*) it only brings non-dominant tuples, and *ii*) it has been transitively covered by another hyperedge. Both cases indicate that we have explored another hyperpath P' that reaches e with $POD(P') \geq POD(P')$. Suppose there exists a hyperpath $P(e', e)$ that has been pruned in Algorithm 2, which affects the $MCD(e)$. According to Definition 8, this $P(e', e)$ has the highest path overlapping degree among all hyperpaths in $\mathcal{P}(e_w, e)$ for all hyperedges e_w that satisfy $O(e_w) < O(e)$, which contradicts the pruning conditions in Algorithm 2, so the lemma holds. □

Proof of Lemma 6. Suppose $POD(P(e', e_u)) = t_1$ and $OD(e_u, e_v) = t_2 \leq k$. Since $e \overset{k}{\rightsquigarrow} e_u$, there exists a hyperpath $P(e, e_u)$ with $POD(P(e, e_u)) = k$, and we can form two hyperpaths $P(e', e) = P(e', e_u) \oplus P(e_u, e)$ and $P(e, e_v) = P(e, e_u) \oplus \{e_v\}$. If $t_1 \geq t_2$, then we have $t_1 = t$, and $P(e, e') = \min(k, t_1)$ is t since $k \geq t$. Therefore, we have the lower bound of $MCD(e')$ is no smaller than t_2, so such $P(e', e_u) \oplus e_v$ will be early terminated at e_u. Another case is $t_1 < t_2$, in this case, the lower bound of $MCD(e')$ is t_1 derived from the hyperpath $P(e, e_u) \oplus P(e_u, e')$, and such $P(e, e_u)$ with $POD(P(e', e_u)) \leq MCD(e')$ will be early terminated. In both cases $e \overset{t}{\rightsquigarrow} e'$ and $e \overset{t}{\rightsquigarrow} e_v$, therefore $e' \overset{t}{\rightsquigarrow} e_v$ is transitively covered by e, so the lemma holds. □

Proof of Theorem 3. The time complexity of Algorithm 3 can be categorised into three parts, and we analyse them separately. *i*) Graph traversal: We start by analysing the total number of elements pushed onto the priority queues. During the index construction from a hyperedge e, a label $(e, t) \in \mathcal{L}(u)$ is inserted (line 12) when u is first visited from e, that is, a hyperedge e_u that contains u is popped from the priority queue and being explored. Since each e_u is explored at most once from e, in the worst case, we explore all hyperedges in $E(u)$ and are only able to insert one label into $\mathcal{L}$. Thus, the total number of elements pushed onto and popped from the priority queues is bounded by $O(ld)$, and each push/pop operation takes $O(\log(ld))$ time given that the size of priority queues is at most $O(ld)$. Therefore, lines 7 and 21 take $O(ld \cdot \log(ld))$ time in total. Each time a hyperedge e_u is popped from the priority queues, line 10 iterates through all vertices in e_u and line 9 scans the neighbour-index of e_u, which takes $O(ld \cdot (\delta + \alpha_e))$ time in total. *ii*) Index update: Line 12 inserts a new label (e, t) into $\mathcal{L}(u)$ by appending it to the end of $\mathcal{L}(u)$, which takes constant time for each insertion and the overall index updating time for $\mathcal{L}$ is $O(l)$. *iii*) Neighbour-index maintenance: The neighbour information for each hyperedge is computed exactly once (line 15), and the computation time is bounded by $O(m\delta d)$. The overall time complexity for deleting elements in neighbour-index is bounded by $O(\alpha \log \alpha_e)$. Therefore, the overall time complexity of Algorithm 3 is bounded by $O(ld \cdot (\log(ld) + \delta + \alpha_e) + m\delta d + \alpha \log \alpha_e)$. □

Proof of Theorem 4. According to Definition 5 and Lemma 3, we only add a label (e, t) into $\mathcal{L}(u)$ when e forms a hyperpath

$P(e, e_u)$ such that $u \in e_u$ and e has the highest importance among all hyperedges in $P(e, e_u)$. Note that, e_u can be e. Therefore, the number of labels in $\mathcal{L}(u)$ will be no greater than $|\mathcal{E}_{\leq u}|$ and the overall space complexity for our HL-index in both Algorithm 2 and 3 is bounded by $O(\sum_{u \in \mathcal{V}}(|\mathcal{E}_{\leq u}|))$. For a hyperedge e and the corresponding neighbour-index $\mathcal{M}(e)$, only those e_v with $OD(e, e_v) > MCD(e)$ will be maintained in $\mathcal{M}(e)$, so the number of neighbours of e maintained in $\mathcal{M}(e)$ will be a value α_e less than η_{max}. So the space complexity of the neighbour-index $\mathcal{M}$ is bounded by $O(m\alpha_e)$. □

Proof of Theorem 5. It is clear that the loops in line 4 and line 7 iterate l times since $\sum_{e \in \mathcal{E}} |\mathcal{D}(e)| \cdot |e| = l$, and the inner loop in line 5 iterates at most $O(l_v)$ times, Therefore, line 6 runs $O(l \cdot l_v)$ times in total to initialise $\mathcal{I}$. For each (u, t_u), the loop in line 9 iterates $O(l_v)$ times, the inner loop in line 10 iterates $O(\beta)$ times. The upper bound of β is derived from lines 4-6 of Algorithm 4, which is θ since the vertices in $\mathcal{I}(e)$ is the subset of vertices in $\mathcal{D}(e)$ based on Observation 1. Thus, lines 11-12 run $O(l \cdot l_v\beta)$ times in total, while each of them takes $O(\log \theta)$. So the overall time complexity of the loop in 9-13 is $O(l \cdot l_v\beta \cdot \log \theta)$. For lines 14-17, line 14 takes $O(\log \theta)$ time since it may search items in $\mathcal{D}(e)$. We can directly append (e, t_u) to the end of $\mathcal{L}^*(u)$ since we iterate over all hyperedges in descending importance order, as shown in line 15. line 16 takes $O(\theta \log \theta)$ time since we incur set interaction. Thus, lines 14-18 take $O(l \cdot (\log \theta + \theta \log \theta))$ time in total. Therefore, the overall time complexity of Algorithm 4 is bounded by $O(l \cdot (l_v\beta \log \theta + \theta \log \theta))$. □

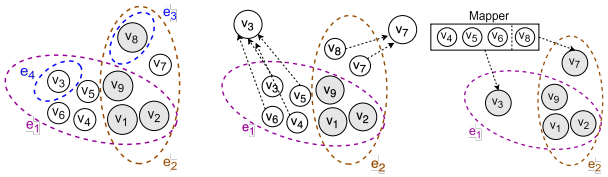
Proof of Theorem 6. Evidently, Algorithm 4 will remove a label (e, t) from $\mathcal{L}(u)$ if and only if $MR(u, v)$ query for any $v \in \mathcal{V}$ can be correctly answered with other labels in the current $\mathcal{L}$.

First, we prove the completeness of $\mathcal{L}^*$ as follows. Suppose $MR(u, v) = k > 0$ cannot be correctly determined through $\mathcal{L}^*$, that is, there does not exist a hyperedge that supports $MR(u, v)$ in $\mathcal{L}^*$. According to Theorem 2, the input $\mathcal{L}$ is complete, therefore, there exists at least one hyperedge e that supports $MR(u, v)$ in the input $\mathcal{L}$ and we denote the set of such hyperedges as $\mathcal{E}_{u,v}$. $MR(u, v)$ cannot be derived from $\mathcal{L}^*$ only if the support for $MR(u, v)$ from each hyperedge $e \in \mathcal{E}_{u,v}$ is disrupted in Algorithm 4 when its essential tuples are being verified and $(e, *)$ is deleted either from $\mathcal{L}(u)$ or from $\mathcal{L}(v)$. Let the hyperedge e' be the last hyperedge in $\mathcal{E}_{u,v}$ to have its essential tuples verified. Then, the support from e' is disrupted if and only if $MR(u, v)$ is still supported by other hyperedges from $\mathcal{E}_{u,v}$ in the current $\mathcal{L}$, which contradicts our assumption. Therefore, $\mathcal{L}^*$ is complete.

Next, we prove $\mathcal{L}^*$ satisfies the minimality property. Suppose there exists a label $(e, t) \in \mathcal{L}^*(u)$ such that, for all $v \in \mathcal{V}$, $\mathcal{L}^*$ can still correctly answer $MR(u, v)$ after we (e, t) remove from $\mathcal{L}^*(u)$. Then, Algorithm 4 would have already removed (e, t) from $\mathcal{L}$, which contradicts our assumption. Therefore, $\mathcal{L}^*$ is minimal. □

Proof of Lemma 8. It can be easily derived according to the completeness property of our HL-index. □

Proof of Theorem 7. It is clear that to determine the existence of common hyperedge for both $\mathcal{L}(u)$ and $\mathcal{L}(v)$, we invoke a merge-sort-liked algorithm that need to iterate through elements in both $\mathcal{L}(u)$ and $\mathcal{L}(v)$ for exactly the once. Therefore, the time complexity of Algorithm 5 is $|\mathcal{L}(u)| + |\mathcal{L}(v)|$. □



(a) Original Graph (b) Compacting (c) Final Graph

Figure 5: Graph Compacting Procedure

Proof of Obasevation 2. For every hyperedge $e = \{v\} \in \mathcal{E}$, e cannot transitively cover any other hyperedge since it has the lowest importance among all hyperedges in a hyperpath that contains e. Besides, every hyperpath P that contains e s.t. $P = \{e_1, e, e_2, ..., e_k\}$ can be simplified to $P' = \{e_1, e_2, ..., e_k\}$ while $POD(P')$ remains unaffected, since we have $v \in e_2$. □

Proof of Observation 3. The proof for this observation is trivial and hence omitted.

□

APPENDIX B

Graph compacting. We notice that some hyperedges and vertices carry repeated or meaningless information, which could be a waste for both space and time efficiency. Therefore, we introduce the following steps to compact an input hypergraph, reducing the search space and accelerating the index constructing and querying stage. For a given hypergraph, we have the following two observations.

Observation 2. *For a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, all hyperedges with only one vertex inside carry no useful information.*

For a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, We assume that every $v \in \mathcal{V}$ must exist in at least one hyperedge, so the result of MR query for the same vertex will always be a non-zero value.

Observation 3. *For a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, if there are more than one vertices that exist and only exist in the same hyperedge, then those vertices carry the same information.*

Based on Observations 2 and 3, we propose the following graph compacting techniques. Given an input hypergraph, we first remove all hyperedges with the size of 1. Then for each hyperedge e, we scan through every vertex inside. All vertices inside e that do not exist in any other hyperedge are marked as isolated, which will then be mapped to the isolated vertex in e that has the smallest ID.

Example 8. *Given the initial input hypergraph shown in Figure 5(a), we observe that there exist two hyperedges e_3, e_4 with the size of 1, so we first remove e_3 and e_4 from the hypergraph, and v_8 becomes an isolated vertex after that. We then use grey colour to represent all un-isolated vertices. According to the example of Figure 5(b). For each hyperedge, we map those isolated vertices into the one with the smallest ID. For example, all isolate vertices $\{v_3, v_4, v_5, v_6\} \in e_1$ are mapped to v_3, and $\{v_7, v_8\} \in e_2$ are mapped into v_7. In the final stage, all vertices are marked as un-isolated, and the hypergraph after compacting is shown in Figure 5(c).*