Appendix: LMSC: Local Sketch Modularity Optimisation for Size-Constrained Community Search in Networks

A Proof of Hardness of LMSC

PROOF. We reduce the set-cover problem to LMSC with $\tau = 0$ to prove its hardness. Figure 1 presents a reduction from the set-cover problem to the specific case of the LMSC problem.

Consider an instance of the set-cover problem $I_{SC} = (I, S)$, where I is a set of items and S is a collection of itemsets. We construct a new instance with $|I| = |S|$ by setting $I' = I$ and $S' = S$, and connecting each $i \in I'$ to $s \in S'$ if $i \in s$. If $|I'| < |S'|$, we add $|S'| - |I'|$ dummy items to I', each connected to all itemsets in S'. If $|I'| > |S'|$, we add $|I'| - |S'| + 1$ dummy itemsets to S' and one dummy item to I', linking only these two. This yields the graph $G' = (V', E')$.

- $V' = I' \cup S' \cup R' \cup T'$
- $E' = E_{S',I'} \cup E_{R',S'} \cup E_{S',T'} \cup E_{T'} \cup E_{R'}$
- $Q' = I' \cup R' \cup S'_{dummy}$

Each component is defined as follows:

- R' is a set of nodes in a bipartite clique which contains n nodes on one side, and n^2 nodes on the other side.
- Each layer of T' consists of n^2 nodes. Each node in S' is connected to a distinct set of n nodes in the first layer of T', denoted T^1. Each node in T^1 is then connected in a 1:1 manner to n^2 nodes in the second layer T^2. This pattern continues up to layer $n + 1$.

In this reduction, we set the edge weights as follows.

- The weight of each edge from $s \in S'$ to $i \in I'$ is $\frac{1}{|N(s,I')|}$, where $N(s, I')$ denotes the set of neighbours of s in I' .
- The weight of each edge from T^i to T^{i+1} is n^i
- The weight of the other edges is 1. (S' to R', S' to T', and R')

Suppose that the lower bound l is $n^2 + 2n + 1 + |S'_{dummy}|$, and the upper bound h is $n^2 + 3n$. Note that $n^2 + 2n + |S'_{dummy}|$ is the size of all the query nodes. In S', we have to choose at least one node to guarantee covering the nodes in I', and its maximum size is n. Based on the solution type, we categorise the solutions. Let C denote an identified query-centric community in G'.

Scenario #1. Let X be the selected subset of S', including the dummy set. We compute:

$$l_C = |E_{R'}| + |E_{R',X}| + |E_{X,I'}| = n^3 + |X| + |X| = n^3 + 2|X|$$
$$o_C = |E_{R',S'\setminus X}| + |E_{S'\setminus X,I'}| + |E_{X,T'}| = 2(n - |X|) + n|X|$$

Let x be the size of the nodes in X, i.e., $x = |X|$.

$$\mathsf{LSM}(G', C, \tau = 0) = \frac{n^3 + 2|X|}{2(n - |X|) + n|X|} \Rightarrow \frac{n^3 + 2x}{2(n - x) + nx}$$

Since $1 \le x \le n$, we analyse the trend. Thus, we compute $\frac{\partial}{\partial x}\mathsf{LSM}(G', C, 0)$.

$$\frac{\partial}{\partial x}\mathsf{LSM}(G', C, 0) = -\frac{n(n^3 - 2n^2 - 4)}{((n - 2)x + 2n)^2}$$

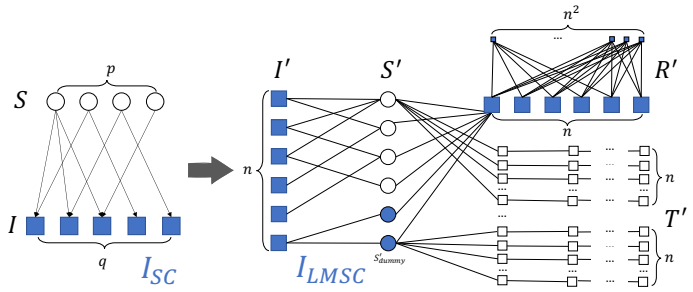


Fig. 1. Reduction procedure

We notice that when $n \geq 3$, $\frac{\partial}{\partial x}\mathsf{LSM}(G', C, 0)$ is always negative, which directly implies that when the value x increases, $\mathsf{LSM}(G', C, 0)$ decreases. From this result, we notice that the nodes $X' = X \setminus S'_{dummy}$ could be a solution of the instance of I_{SC}. This is because the nodes X' cover all the nodes in I' and its size is minimised. Thus, we notice that LMSC problem is NP-hard due to the polynomial reduction from the set cover problem.

Scenario #2. Compared to Scenario #1, we now consider the case of adding a set of nodes in T'. Note that in order to satisfy the connectivity constraint, the inclusion of at least one node from $S' \setminus S'_{dummy}$ is mandatory. Suppose we add only one node, v, from T' to X. Focusing on the simple case of adding a single node from T is sufficient because, in any scenario, the number of internal nodes is smaller than the number of external nodes. The detailed computation of LSM is as follows.

$$l_{C\cup\{v\}} = n^3 + 2|X| + 1$$
$$o_{C\cup\{v\}} = 3n + (n-2)x - 1$$

We then get the following result.

$$\mathsf{LSM}(G', C \cup \{v\}, 0) = \frac{n^3 + 2x + 1}{3n + (n-2)x - 1}$$

Compared with the LSM values in Scenario #1, the $\mathsf{LSM}(G', C, 0)$ is always larger than $\mathsf{LSM}(G', C \cup \{v\}, 0)$, since ΔLSM always has a positive result when $n \geq 2$.

$$\Delta\mathsf{LSM} = \frac{n^3 + 2x}{2(n-x) + nx} - \frac{n^3 + 2x + 1}{3n + (n-2)x - 1} > 0$$

We may also consider adding nodes from deeper layers T^i, but this increases o_C due to high-weight outgoing edges, thereby decreasing the overall LSM score. Hence, communities that include nodes from T' are suboptimal. Consequently, the optimal solution must consist only of nodes in S', and the best such solution minimises $|X|$, subject to covering all nodes in I'. Since $X \setminus S'_{dummy}$ corresponds to a valid minimum set cover of I_{SC}, this completes the polynomial-time reduction and establishes that the LMSC problem is NP-hard. □

B Pseudo code of Greedy Expansion Procedure

Algorithm description. Algorithm 1 details the GREEDYEXPANSION procedure, which incrementally constructs a query-centric community under an upper size constraint. At each iteration, the algorithm considers all neighbouring nodes outside the current set C and selects the node v with the highest marginal gain in local sketch modularity (LSM).

Algorithm 1: Step #1: `greedyExpansion` (G, C, τ, h)

for $|C| \leq h - 1$ **do**
  $v \leftarrow \arg\max_{w \in N(C,G) \setminus C} \Delta\mathsf{LSM}(G, C \cup \{w\}, \tau)$;
  if $\Delta\mathsf{LSM}(G, C \cup \{v\}, \tau) > 0$ **then**
    $C \leftarrow C \cup \{v\}$;
  else
    break;
return C';

Importantly, in contrast to IGA approaches that may allow the inclusion of nodes with zero or negative gain, this method strictly incorporates a node only if it results in a positive improvement in the LSM score. This ensures that the expansion is focused on discovering the core structure of the community—adding only nodes that genuinely enhance internal cohesion according to the objective. The process continues until no further positive gain is possible or the community reaches the maximum size h, and the final locally optimal set C' is returned.

C Pseudo code of Chain Identification Procedure

Algorithm 2: Step #2: `findChains` (G, C, h)

$h' \leftarrow h - |C|$, $\mathcal{P} \leftarrow N(C, G) \setminus C$;
foreach $u \in \mathcal{P}$ **do**
  Z.`append` (u);
  while $|Z| < h'$ **do**
    $cand = N(Z, G) \setminus Z \setminus C$;
    if $cand = \emptyset$ **then**
      break;
    $v \leftarrow \arg\max_{w \in cand} \Delta\mathsf{CLSM}(G, Z \cup \{w\}, C)$;
    if $\Delta\mathsf{CLSM}(G, Z \cup \{v\}, C) < 0$ **then**
      break;
    Z.`append` (v);
  $\mathcal{Z}$.`append` (Z);
return $\mathcal{Z}, \mathcal{P}$

Algorithm description. Algorithm 2 presents the FINDCHAINS procedure, which identifies candidate chains to enable group-wise community expansion. For each node u in the set of neighbours $\mathcal{P}$ (i.e., nodes adjacent to the current community C but not in C), the algorithm initiates a chain Z starting from u. It then iteratively appends nodes to Z, each time selecting the node v from the set of valid candidates that maximises the marginal gain in the chain-based local sketch modularity (CLSM) relative to C. Expansion proceeds only as long as the addition of the next node results in a non-negative gain and the chain does not exceed the remaining size budget h'. Once the chain can no longer be expanded without reducing CLSM, it is added to the collection of chains $\mathcal{Z}$. The process repeats for all pivot nodes in $\mathcal{P}$, after which the algorithm returns the set of identified chains $\mathcal{Z}$ and the pivot set $\mathcal{P}$. This step allows the algorithm to explore cohesive substructures that may be merged with the core community in subsequent optimisation steps.

Algorithm 3: Step #3: `getSubchain` $(G, C, \mathcal{Z}, h, \tau)$

$S \leftarrow \emptyset, LSM_{max} \leftarrow -1$;
foreach $Z \in \mathcal{Z}$ **do**
  $S_{cur} \leftarrow C, LSM_{cur} \leftarrow -1$;
  foreach $v \in Z$ **do**
    S_{cur}.`append` (v) ;
    $LSM_{cur} \leftarrow \mathsf{LSM}(G, S_{cur}, \tau)$;
    if $LSM_{max} < LSM_{cur}$ **then**
      $LSM_{max} \leftarrow LSM_{cur}$;
      $S \leftarrow S_{cur}$
    if $|S_{cur}| = h$ **then**
      break;
return $S \setminus C$

D Pseudo Code of Sub-chain Merge Procedure

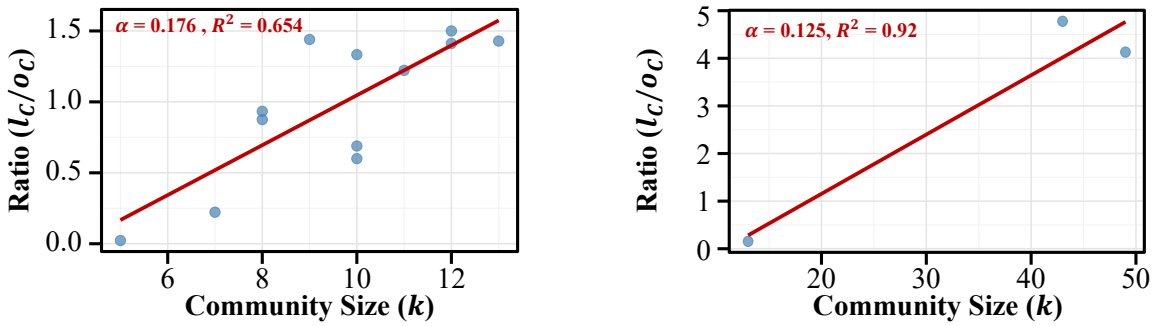
Algorithm description. Algorithm 3 describes the GETSUBCHAIN procedure, whose main objective is to identify the most beneficial sub-chain for merging into the current community C. For each candidate chain Z in the set $\mathcal{Z}$, the algorithm incrementally builds a sequence of sub-chains by appending nodes from Z to C. At every extension, it computes the local sketch modularity (LSM) for the resulting set. Whenever a higher LSM score is obtained, the algorithm updates both the score and the corresponding sub-chain. This process is repeated for every chain in $\mathcal{Z}$, considering all possible sub-chains, and terminates early if the community size constraint h is reached. After evaluating all candidates, the sub-chain that achieves the highest LSM score is chosen as the optimal segment to merge. The procedure then returns the nodes in this sub-chain that are not already part of C. These nodes are subsequently incorporated into the community in the next step, thus allowing group-wise expansion in a manner that directly maximises the LSM objective.

E Pseudo Code of Chain Update Procedure

Algorithm description. Algorithm 4 describes the UPDATE procedure, which maintains the validity and optimality of candidate chains following the merging of a selected sub-chain S into the current community C. For each chain Z in the current set $\mathcal{Z}$, the algorithm applies a series of update rules to ensure that Z remains disjoint from both the updated community and S, and still respects the size constraint h'. Chains that now overlap with S are truncated at the point of overlap or discarded if their pivot node was merged. Additional adjustments are made if neighbouring or connectivity conditions are violated, guaranteeing that only valid, extendable portions of chain are preserved. The algorithm then attempts to expand truncated chains further, greedily appending nodes that yield a positive marginal gain in the chain-based local sketch modularity (CLSM) until the size limit is met or no further gain is possible. After processing all affected chains, the set of new pivot nodes is updated to reflect the expanded boundary of C, and new chains are initiated as needed. The updated collection of candidate chains $\mathcal{Z}'$ and pivot set $\mathcal{P}'$ are then returned, enabling the framework to efficiently maintain and reuse local structures in subsequent optimisation steps.

Algorithm 4: Step #4: `update` $(G, C, S, \mathcal{Z}, \mathcal{P}, h)$

$\mathcal{Z}' \leftarrow \emptyset$;
foreach $Z \in \mathcal{Z}$ **do**
  $Z' \leftarrow \emptyset$; `// New chain`
  $index \leftarrow Z$.`length()`;
  if $Z[0] \in S$ **then**
    continue;
  if $|Z| > h'$ **then**
    $Z = Z[v_0, \ldots, v_{h'-1}]$
  if $Z \cap S \neq \emptyset$ **then**
    $index \leftarrow \min\{j \mid Z[j] \in S\}$;
    $Z' \leftarrow Z[v_0, \ldots, v_{index-1}]$;
  $R \leftarrow (N(Z, G) \cap N(S, G)) \setminus S \setminus C$;
  if $R \neq \emptyset$ **then**
    $cand \leftarrow \{u \in Z \mid v \in R,\ v \in N(u, G)\}$;
    $i \leftarrow \min_{u \in cand} Z$.`indexOf(`$u$`)`;
    $index \leftarrow \min\{index - 1, i\}$;
    $Z' \leftarrow Z[v_0, \ldots, v_{index}]$;
  $N_s \leftarrow Z \cap N(S, G) \setminus C$;
  if $N_s \neq \emptyset$ **then**
    $i \leftarrow \min\{j \mid Z[j] \in N_s\}$;
    $index \leftarrow \min\{index, i\}$;
    $Z' \leftarrow Z[v_0, \ldots, v_{index}]$;
  if $Z' = \emptyset$ **then**
    $\mathcal{Z}'$.`append(`Z`)`;
    continue;
  while $|Z'| < h'$ **do**
    $cand \leftarrow N(Z', G) \setminus (C \cup Z')$;
    if $cand = \emptyset$ **then**
      break
    $v \leftarrow \texttt{argmax}_{w \in cand} \Delta\mathsf{CLSM}(G, Z' \cup \{w\}, C)$;
    if $\Delta\mathsf{CLSM}(G, Z' \cup \{v\}, C) < 0$ **then**
      break;
    Z'.`append(`v`)`;
  $\mathcal{Z}'$.`append(`Z'`)`;
$\mathcal{P} \leftarrow N(S, G) \setminus C \setminus \mathcal{P}'$;
$\mathcal{P}'$.`append(`$\mathcal{P}$`)`;
`/* Lines 2-12 in Algorithm 2 */`
  $\mathcal{Z}'$.`append(`Z'`)`;
return $\mathcal{Z}', \mathcal{P}'$;



(a) Football network (b) Polbooks network

Fig. 2. Scatter plots of α for different networks.

F Parameter τ Setting

Motivation. The sketch threshold τ controls the trade-off between *compactness* and *separability* in LSM. Although Theorem 3 provides a lower bound for avoiding the local free-rider effect,

$$\tau > \log_{1+\frac{1}{h}}\left(\frac{h(h-1)}{2}\right), \tag{1}$$

this bound exceeds 10^2 even for $h = 50$ and is therefore unsuitable for real-world applications.

Empirical scaling of internal and external edges. To derive a more practical threshold for τ, we assume that the ratio of internal to external edges, l_C/o_C, scales linearly with the community size $k = |C|$, i.e., $l_C/o_C \propto \alpha k$ for some constant α. We empirically validate this in Figure 2 for the American college football and Polbooks networks. The football network exhibits an R^2 coefficient of 0.65 for the linear fit, indicating a moderate linear relationship, while the Polbooks network achieves $R^2 = 0.92$, providing strong evidence that l_C/o_C grows linearly with k. This supports the use of a linearly scaling model for τ selection.

From Theorem 3 to the working rule $\tau > 1$. Theorem 3 gives the general lower bound

$$\tau > \frac{-\log(l_C/o_C) + \log(l_{C\cup C'}/o_{C\cup C'})}{\log(1 + \frac{|C'|-|C\cap C'|}{|C|})} \tag{2}$$

Under the *weak-sense* assumption ($l_C \geq o_C$) and empirical scaling $l_C/o_C \propto \alpha k$, consider merging a singleton $C' = \{v\}$:

- **Internal vs external scaling:** $\frac{l_C}{o_C} \approx \alpha k$ and $\frac{l_{C\cup\{v\}}}{o_{C\cup\{v\}}} \approx \alpha(k+1)$.
- **Numerator:** $-\log(\alpha k) + \log(\alpha(k+1)) = \log(1+\frac{1}{k})$.
- **Denominator:** Only one vertex is added, thus, $\log(1+\frac{1}{k})$.

As a result, both the numerator and denominator are equal, so the bound reduces to $\tau \approx 1$.

Accordingly, we set $\tau = 1$ in all experiments. This value is consistent with the empirical scaling behaviour observed in real-world networks and satisfies the theoretical lower bound established above. While the choice of $\tau = 1$ may not be strictly optimal for every possible instance, in our setting, values of τ in the range $[1, 1.5]$ consistently yielded the best performance (refer to EQ3), providing a robust balance between compactness and separability without the need for extensive parameter tuning.

G Proof of Correctness for the Chain Update Procedure

We prove that local update strategy (described in Section 6.5) is sufficient. That is, for any community C and any merged sub-chain S, the set of chains produced by local update rules is identical to the set of chains that would be generated by global computation from the new community $C' = C \cup S$.

Preliminaries. Let $\mathcal{Z}_{\text{pre}}$, $\mathcal{Z}_{\text{local}}$, and $\mathcal{Z}_{\text{global}}$ denote, respectively, the sets of chains before merging, after local updates, and after global recomputation. Let $Z = [v_0, v_1, \ldots, v_{|Z|-1}]$ be a chain, and let $Z_{\leq t} = [v_0, v_1, \ldots, v_t]$ denote its prefix up to index t. For a pivot v_0, $\texttt{findChain}(C, v_0)$ denotes the deterministic greedy procedure in Section 6.3. At each step t, the node v_{t+1} is selected as : $v_{t+1} = \arg\max_{x \in N(Z_{\leq t}, G) \setminus (C \cup Z_{\leq t})} \mathsf{CLSM}(C, Z_{\leq t} \cup \{x\})$. We use identical tie-breaking to ensure determinism. Formally, a node v is considered valid if $v \notin C'$ and invalid if $x \in C'$.

G.1 Type of Affected Chain

A chain is considered to have changed if either the set of nodes or their sequence differs. Given a chain $Z = [v_0, \ldots, v_t, v_{t+1}, \ldots, v_{|Z|-1}]$, a change occurs if, i) the pivot node v_0 becomes invalid, or ii) the size of the chain exceeds the allowable size, or iii) at some step v_t, a different node is selected instead of the v_{t+1}. For the chain to remain unchanged according to i) and ii), the following conditions must be satisfied: (1) The pivot v_0 must not be part of the merged sub-chain S, and (2) The size constraint must be satisfied, i.e., $|Z| \leq h - |C'|$.

Now, consider case iii) where another node could be selected during chain construction. This can happen only if either iii-1) node v_{t+1} is no longer a valid candidate of chain, or iii-2) some other node $x \in N(A, G)$, where $A = \{v_0, \ldots, v_t\}$, achieves a higher CLSM score than v_{t+1}. We first consider the invalidity of v_{t+1}. Only nodes satisfying the chain constraints (connectivity and disjointness) can be included. Connectivity is always satisfied, as nodes in chain are selected from $N(A, G)$, so there is no risk of violation. We then focus on disjointness: to be valid, (3) v_{t+1} must not be included in S.

Next, consider the case where another node $x \in N(A, G)$ achieves a higher CLSM score than v_{t+1} under C'. Recall the CLSM score is defined as: $\mathsf{CLSM}(C, Z) = \frac{l_Z + l_{C,Z}}{o_Z - l_{C,Z}}$. For x to be chosen over v_{t+1} under the new community C', it must hold that: $\mathsf{CLSM}(C', A \cup \{x\}) > \mathsf{CLSM}(C', A \cup \{v_{t+1}\})$. However, since x was not selected in the original construction under C, we must have:$\mathsf{CLSM}(C, A \cup \{v_{t+1}\}) > \mathsf{CLSM}(C, A \cup \{x\})$. The CLSM score under the updated community C' can be expressed in terms of C. Since $C' = C \cup S$ and $C \cap S = \emptyset$, for any node x, we have $l_{C',x} = l_{C,x} + l_{S,x}$. Accordingly, the CLSM score of chain Z under C' becomes: $\mathsf{CLSM}(C', Z) = \frac{l_Z + (l_{C,Z} + l_{S,Z})}{o_Z - (l_{C,Z} + l_{S,Z})}$. Now, when comparing $\mathsf{CLSM}(C', A \cup \{x\})$ and $\mathsf{CLSM}(C', A \cup \{v_{t+1}\})$, $l_{A \cup \{x\}}$ and $l_{A \cup \{v_{t+1}\}}$, $o_{A \cup \{x\}}$ and $o_{A \cup \{v_{t+1}\}}$, $l_{C, A \cup \{x\}}$ and $l_{C, A \cup \{v_{t+1}\}}$ cannot change unless new edges are added or removed in the graph. Therefore, the only terms that can vary due to the merge are $l_{S, A \cup \{x\}}$ and $l_{S, A \cup \{v_{t+1}\}}$. This directly corresponds to the condition that (4) node x must not be adjacent to S. The CLSM score can increase only if x is a neighbour of S. Hence, a chain is *unaffected* if none of the following conditions are violated:

(1) The pivot node is not included in the merged sub-chain S (corresponding to Update Rule 2).

(2) The chain satisfies the size constraint: $|Z| \leq h - |C'|$ (corresponding to Update Rule 1).

(3) Any node in Z is not included in S (corresponding to Update Rule 3).

(4) Any node among the neighbours of chain Z must not be connected to S (corresponding to Update Rule 4).

Global Update Equivalence (Proof by Contradiction). Suppose $Z = \texttt{findChain}(C, v_0)$ is an unaffected chain. Then, under local update procedure, the result is preserved as $Z_{\text{local}} = Z$. Now assume, for contradiction, that the result of a global computation differs: $Z_{\text{global}} \neq Z_{\text{local}}$. However, as analysed earlier, if all four conditions are satisfied, the chain cannot change. This guarantees that the greedy construction process remains unchanged even after merging S into C. Since the `findChain` function is deterministic, its output is entirely determined by this environment. Given an identical starting pivot v_0 and identical algorithmic conditions, the function must produce an identical output chain. This implies

$Z = Z_{\text{global}} \implies Z_{\text{local}} = Z_{\text{global}}$ This contradicts initial assumption that $Z_{\text{global}} \neq Z_{\text{local}}$. Hence, the global computation must produce the same result as the local update.

G.2 Affected Chains and Rule Correctness

A chain is considered *affected* when it violates any of the above conditions. For each such case, we show that the corresponding update rule produces the same result as a global recomputation.

(1) *Pivot invalidation:* If pivot node $v_0 \in S$ in chain Z, the chain Z is removed (Rule 2). Since global recomputation only builds chains from $v \in N(C', G) \setminus C'$, this removal is correct.

(2) *Size violation:* If $|Z| > h - |C'|$, Update Rule 1 truncates Z to length $h - |C'|$. As it is a prerequisite for a chain, a chain that does not satisfy this condition does not exist. Consequently, a chain produced through global recomputation also satisfies the same constraint.

(3) *Partial overlap:* If any node $v_i \in Z \cap S$, Update Rule 3 truncates Z before v_i and rebuilds greedily. The prefix $Z_{\leq i-1}$ remains unaffected; hence, the local and global recomputations share the same prefix $Z_{\leq i-1}$. Subsequently, since both local and global are extended by the same process, it follows that $Z_{\text{local}} = Z_{\text{global}}$.

(4) *Score reordering:* If a node $x \in (N(Z, G) \cap N(S, G)) \setminus C'$ exists, Update Rule 4 truncate Z up to v_i, where i is the smallest index such that $v_i \in Z$ is adjacent to x, and then rebuilds the rest. Both processes share the same prefix and neighbour set, yielding identical continuation.

(5) *New pivots:* Update Rule 5 starts new chain constructions from pivot nodes in $N(S, G) \setminus (C' \cup \mathcal{P})$, which corresponds to newly pivot nodes not previously included in $N(C, G)$. Since this set equals $N(C', G) \setminus N(C, G)$, all new chains are correctly identified.

This section confirms the sufficiency of our rule set in handling isolated cases. However, a chain may satisfy conditions for multiple update rules. In such cases, the order in which rules are applied can influence the outcome. Therefore, we also prove in Appendix L that the proposed rule priority ensures correctness under multiple simultaneous rule conditions.

H Proof of Sufficiency for the Update Rule Priority Order

Reminder of Update Rules (Section 6.5–6.6). Before proving the sufficiency of the priority order, we briefly recall the local update rules defined in the main text:

- **Update Rule 1 (Size constraint)**: If $|Z| > h - |C'|$, truncate the chain to $Z_{\leq k-1}$, where $k = h - |C'|$.
- **Update Rule 2 (Pivot removal)**: If the pivot node $v_0 \in S$ in chain Z, remove the chain Z entirely.
- **Update Rule 3 (Partial overlap)**: If the pivot node $v_0 \notin S$ but $Z \cap S \neq \emptyset$, let $i = \min\{t : v_t \in S\}$ and keep $Z_{\leq i-1}$.
- **Update Rule 4 (Score change)**: If a node $x \in (N(Z, G) \cap N(S, G)) \setminus C'$, let i be the smallest index such that $v_i \in Z$ is adjacent to x, and truncate the chain to $Z_{\leq i} = \{v_0, \ldots, v_i\}$.
- **Update Rule 5 (New pivots)**: For each $v \in N(S, G) \setminus (C' \cup \mathcal{P})$, initiate a new chain with v as the pivot.

The following proof concerns the relative priority order among Update Rules 1–4. Note that Update Rule 5 simply introduces a new chain without affecting the existing chains; therefore, it is excluded from the priority analysis.

Setup. After merging sub-chain S into community C, each applicable update rule for a chain Z identifies a prefix boundary and truncates Z accordingly, followed by a greedy rebuild (Section 6.5). For $k \in \{-1, 0, 1, \ldots\}$, let $T_k(Z)$ denote the prefix operator of chain Z that returns $Z_{\leq k}$ if $k \geq 0$, and the empty chain if $k = -1$. Each rule induces the following k indices:

- *Update Rule 2 (removal):* $k_{\text{remove}} = -1$ if $v_0 \in S$.
- *Update Rule 1 (size):* $k_{\text{size}} = h - |C'| - 1$.

- *Update Rule 3 (overlap with S):* $k_{\text{overlap}} = i_3 - 1$, where $i_3 = \min\{t : v_t \in S\}$.
- *Update Rule 4 (score reordering):* $k_{\text{score}} = i_4$, where i_4 is the smallest index such that $v_{i_4} \in Z$ is adjacent to any node $x \in (N(Z, G) \cap N(S, G)) \setminus C'$.

We write $\mathsf{i}_r(Z)$ for the index contributed by rule r and use the shorthand $k_r = \mathsf{i}_r(Z)$.

Prefix calculus. For any $k, m \in \{-1, 0, 1, \ldots\}$ and chain Z, we have $T_k(T_m(Z)) = T_{\min\{k,m\}}(Z)$. The prefix operator $T_k(Z)$satisfies idempotence, commutativity, and associativity.

PROOF. If either index is -1 the result is empty on both sides. Otherwise, $T_m(Z)$ keeps the first $m+1$ nodes. Applying T_k afterwards keeps the first $\min\{k + 1, m + 1\}$ nodes. The right-hand side does the same in one step. □

H.1 Justification for Priority 1 (Rule 2 over all others)

Claim. For any chain Z, if pivot node $v_0 \in S$, the final outcome is an empty chain irrespective of the order.

PROOF. Update Rule 2 corresponds to the prefix operator T_{-1}. For any k, $T_{-1} \circ T_k = T_k \circ T_{-1} = T_{-1}$ by the prefix calculus, so removal is absorbing. Hence applying Rule 2 first (our priority) or last yields the same outcome $\emptyset$. □

H.2 Justification for Priority 2 (Rule 1 before Rules 3 & 4)

Claim. When both the size constraint and a structural rule (Rule 3 or 4) apply, applying Rule 1 first yields the same final result as any alternate ordering and avoids unnecessary truncation steps.

PROOF. Let $k_\star = \min\{k_{\text{size}}, k_{\text{overlap}}, k_{\text{score}}\}$. By the prefix calculus, any composition of the corresponding truncations equals $T_{k_\star}$. Suppose an ordering produced a different preserved prefix. Then some composition would equal $T_{k'}$ with $k' \neq k_\star$, contradicting $T_{k_{\text{size}}} \circ T_{k_{\text{overlap}}} \circ T_{k_{\text{score}}} = T_{k_\star}$. Therefore the outcome is order-independent; applying Rule 1 first simply discards any suffix that would be removed later, mirroring a global process that never exceeds the budget. □

H.3 Justification for Priority 3 (Handling Rules 3 & 4)

Claim. If both rules 3 and 4 apply, truncation at the earlier boundary yields an order-independent and sufficient result.

PROOF. Let $k_{\text{overlap}} = i_3 - 1$ and $k_{\text{score}} = i_4$. By the prefix calculus, then $T_{k_{\text{overlap}}} \circ T_{k_{\text{score}}} = T_{k_{\text{score}}} \circ T_{k_{\text{overlap}}} = T_{\min\{k_{\text{overlap}}, k_{\text{score}}\}}$. Hence truncating at the earlier boundary and rebuilding greedily from the common prefix yields a unique *canonical prefix state*. □

H.4 Canonical Boundary and Rebuild Determinism

Let the *canonical boundary* be $\mathsf{i}^*(Z) = \min\{k_{\text{remove}}, k_{\text{size}}, k_{\text{overlap}}, k_{\text{score}}\}$. By the above, any order of applying the rules is equivalent to a single truncation $T_{\mathsf{i}^*(Z)}$. Since the rebuild is the same deterministic greedy procedure as in Section 6.3, the chain reconstructed from this prefix is unique.

H.5 Overall conclusion

Every affected chain first maps to the unique canonical prefix $T_{\mathsf{i}^*(Z)}(Z)$, independent of rule ordering under the stated priority (Rule 2 → Rule 1 → Rules 3/4). Combining this with the single-merge equivalence in Section G yields that the priority order is sufficient and correct for all cases. It follows that $\mathcal{Z}_{\text{local}} = \mathcal{Z}_{\text{global}}$.