Project 2: Truss-Based Structural Diversity Search

Sun Yuxiang

May 2025

1 Simple-Index

1.1 Algorithm Idea

1.1.1 Index Construction

The index is implicit: for every vertex v we materialise its ego–network $G_{N(v)}$ once, store it on disk, and record the trussness of each edge. Figure 1 illustrates the construction steps on the running example from [1].

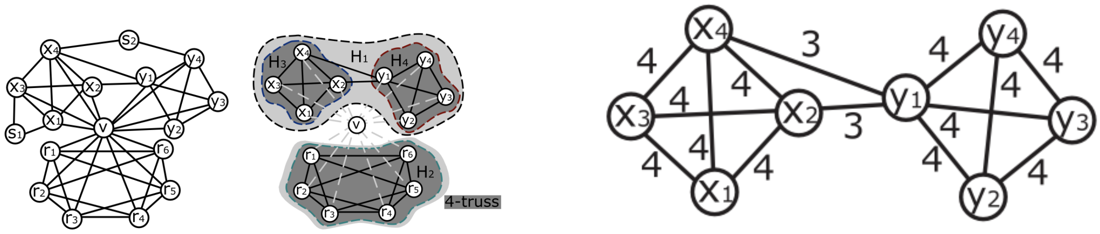

Figure 1: Ego–network extraction and edge–trussness labelling used by *Simple-Index*.

Algorithm 1 Simple-Index: Ego-Network Construction

Require: Undirected graph $G = (V, E)$
Ensure: For every vertex v, its ego–network $G_{N(v)}$ with edge-truss numbers $\tau_{G_{N(v)}}(e)$

/* One-shot ego–network extraction */
for all $v \in V$ **do** ▷ initialise
  $G_{N(v)} \leftarrow$ empty graph
end for
for all edge $e = (u, v) \in E$ **do**
  for all vertex $w \in N(u) \cap N(v)$ **do**
    add edge e into $G_{N(w)}$
  end for
end for
/* Compute trussness for every ego–network */
put all edges into a min-priority queue ordered by $S(e)$
while the queue is not empty **do**
  $(u, v) \leftarrow$ edge with smallest support s
  $\tau(u, v) \leftarrow s + 2$ ▷ final trussness of this edge
  mark (u, v) as removed
  for all common neighbours $w \in N_H(u) \cap N_H(v)$ still alive **do**
    if $S(u, w) > s$ **then**
      $S(u, w) -= 1$;
    end if
    if $S(v, w) > s$ **then**
      $S(v, w) -= 1$;
    end if
  end for
end while

1.1.2 Query Processing

Given a query threshold k, we reload the ego–network, drop every edge whose trussness $< k$, and simply count connected components. The count is the diversity score $\textsc{score}(v)$.

Algorithm 2 Simple-Index: Online Query (v, k)

load $G_{N(v)}$ and edge support
for each e with support$(e) < k$ **do**
    delete e
end for
$s \leftarrow$ number of connected components in the remaining graph
return s

1.2 Algorithm Analysis

Time complexity

- **Construction** Triangle listing on an ego–network with m_v edges costs $O(m_v^{1.5})$ in worst case; summed over all vertices it is $O(T)$, the global triangle count [1].
- **Query** For a single vertex: edge-deletion $O(m_v)$ and BFS/DFS $O(m_v + n_v)$. For all vertices sequentially the total becomes $O(m + n)$.

Space complexity We store every ego–network explicitly: $\sum_v m_v = 3T$ edges, hence $O(T)$ space.

Empirical performance On the provided dataset the whole query phase takes **6.5s**.

2 GCT-Index(Final Solution)

2.1 Algorithm Idea

GCT-Index is the compressed index proposed by Huang et.al. [1]. For each ego–network we:

1. Build all ego-networks using Algorithm 1;
2. Generate a **maximum spanning forest**; edges of identical trussness are contracted into a *supernode* S with label $\tau(S)$;
3. Edges between supernodes keep the larger τ as weight (Figure 2).

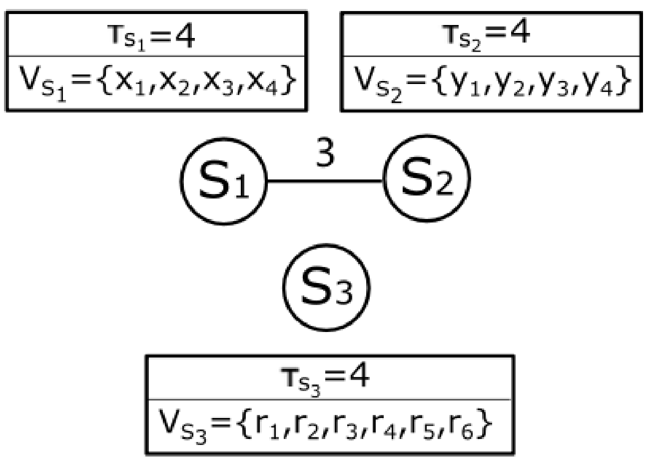

Figure 2: GCT-Index for the running example: circles = supernodes, edge label = trussness.

2.1.1 Index Construction

Algorithm 3 Global GCT-Index Construction

Require: Graph $G = (V, E)$
Ensure: Compressed index $\{\mathrm{GCT}_v\}_{v \in V}$
  Phase I – reuse Algorithm 1
  Build *all* ego–networks and their edge trussness with Algorithm 1.
  Phase II – compress per vertex (Fig. 3 lines 1–16)
  for all vertex $v \in V$ **do**
    $\mathrm{GCT}_v \leftarrow \textsc{BuildGCT}(G_{N(v)})$ ▷ Alg. 4
  end for

Algorithm 4 BuildGCT for one ego–network (Fig. 3)

Require: Ego–network $G_{N(v)}$ with $\tau_{G_{N(v)}}(e)$
Ensure: $\mathrm{GCT}_v = (\mathcal{V}_v, \mathcal{E}_v)$
  $\mathcal{V}_v \leftarrow \emptyset,\ \mathcal{E}_v \leftarrow \emptyset$
  for all vertex $u \in N(v)$ **do** ▷ Lines 2–4
    create super-node S_u with $\tau(S_u) = \tau_{G_{N(v)}}(u)$
    $\mathcal{V}_v \leftarrow \mathcal{V}_v \cup \{S_u\}$
  end for
  $L \leftarrow E(G_{N(v)})$ ▷ Line 5
  while $L \neq \emptyset$ **do** ▷ Lines 6–15
    pop (u, w) of *largest* $\tau_{G_{N(v)}}(u, w)$
    identify super-nodes S_u, S_w
    if $S_u = S_w$ **or** S_u and S_w already connected **then**
      continue
    else if $\tau(S_u) = \tau(S_w) = \tau_{G_{N(v)}}(u, w)$ **then**
      merge: $V_{S_u} \leftarrow V_{S_u} \cup V_{S_w}$; redirect edges; delete S_w
    else
      $\mathcal{E}_v \leftarrow \mathcal{E}_v \cup \big\{(S_u, S_w)\big\}$; $w\big((S_u, S_w)\big) \leftarrow \tau_{G_{N(v)}}(u, w)$
    end if
  end while
  return $\mathrm{GCT}_v = (\mathcal{V}_v, \mathcal{E}_v)$

2.1.2 Query Processing

Lemma 3 in [1] states that for any k,

$$\textsc{score}(v) = N_k - M_k,$$

where N_k (resp. M_k) is the number of supernodes (resp. superedges) with trussness $\geq k$ in GCT_v.

Algorithm 5 GCT-Index Query (v, k)

  $N_k \leftarrow$ count of supernodes with $\tau \geq k$
  $M_k \leftarrow$ count of superedges with $\tau \geq k$
  return $N_k - M_k$

2.2 Algorithm Analysis

Construction Our implementation performs a standard support–peeling truss decomposition on every ego–network $G_{N(v)}$. For an ego containing m_v edges we spend $O\big(m_v \log m_v\big)$ time—$O(m_v)$ extractions from the heap, each costing $O(\log m_v)$, plus constant-time support updates for the two remaining incident edges of every destroyed triangle. Summing over all vertices, $\sum_{v \in V} m_v = 3T$, where T is the number of triangles in the whole graph, so the total cost is $O\big(T \log d_{\max}\big) \subseteq O\big(T \log m\big)$, with $d_{\max}$ the maximum degree. The space footprint is $O(m)$ for storing all ego edge records plus $O(m)$ temporary heap memory.

After truss numbers are ready, the super-node / super-edge construction scans each ego once, performing at most one *union* or *edge insertion* per original edge. Hence it is linear in the ego size, $O(m_v)$ time and $O(m_v)$ memory. Aggregated over all vertices this is $O(T)$ time and $O(m)$ space.

Overall construction complexity therefore becomes $O\big(T \log m\big)$ time, $O(m)$ space.

Query Counting arrays of size $O(\tau_{\max})$ gives $O(1)$ per vertex, thus answering all vertices costs $O(n)$ time and $O(1)$ extra space.

Empirical performance Total query phase time: **0.005s**, a 1300× speed-up over *Simple-Index* due to index reuse and $O(1)$ per-vertex counting.

References

[1] J. Huang, X. Huang, and J. Xu, "Truss-based structural diversity search in large graphs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 8, pp. 4037–4051, 2020.