Graph-Based Context Optimization for RAG

Overview: Given a query Q and retrieved document chunks $D = \{D_1, \ldots, D_n\}$, we build a knowledge graph $G = (V, E)$ where nodes represent either document chunks or extracted entities, and edges encode semantic links (e.g., common entities or hyperlinks). The graph is anchored by a special query node Q.

Mathematical Formalization

Assign each node v a relevance weight $w(v)$ and token cost $\ell(v)$. We aim to select a connected subgraph $S \subseteq V$ that maximizes total weight under a token budget L:

$$\max_{S \subseteq V} \sum_{v \in S} w(v) \quad \text{s.t.} \quad \sum_{v \in S} \ell(v) \leq L,$$

with connectivity enforced by requiring every $v \in S$, $v \neq Q$ to be adjacent to at least one other selected node:

$$x_v \ \leq \ \sum_{(u,v) \in E} x_u.$$

This is equivalent to a prize-collecting Steiner tree problem, known to be NP-hard.

Algorithmic Pipeline

- **Graph Construction:** Retrieve documents D_i, extract entities via a lightweight NLP pipeline, and assemble G by connecting document nodes to entity nodes (and inter-document nodes via shared entities), with Q linked to relevant nodes.
- **Graph Optimization:**
  - **Exact Method:** Solve an Ineger Linear Program (ILP) for small graphs.
  - **Heuristics:** Use greedy/approximation algorithms that incrementally add nodes based on utility $\frac{w(v)}{\ell(v)}$ while preserving connectivity.
- **Context Linearization:** Order the selected subgraph S^* (e.g., placing highest-$w(v)$ nodes at the beginning and end to counteract positional bias) and convert nodes into text (using structured triples or bullet points) for LLM input.

This pipeline is encapsulated in an open-source library to integrate seamlessly with RAG systems.

Advantages & Challenges

- **Primary Advantage:**
  - This pipeline, when operational, makes it possible to interact with massive codebases and adjust them dynamically, exploiting the natural multi-hop capabilities of graph structures.
- **Primary Challenge:**
  - ILP-based optimization is computationally taxing.