CALAMR: Component ALignment for Abstract Meaning Representation

Implementation and Supplementary Guide

Paul Landes

1 Introduction

This document explains some of the implementation details of CALAMR. The API documentation is complete, but does not provide a high level guide to some aspects of the code base that are less than straightforward.

2 Graphs

The treatment of graphs is somewhat involved. There is s class structure with a more complex instance structure. We used igraph[1] for its push-relabel max flow algorithm [3], and for this reason, all graph operations and data structures use this module. The base class `GraphComponent` is the super class for all graph classes and has an instance of an igraph.

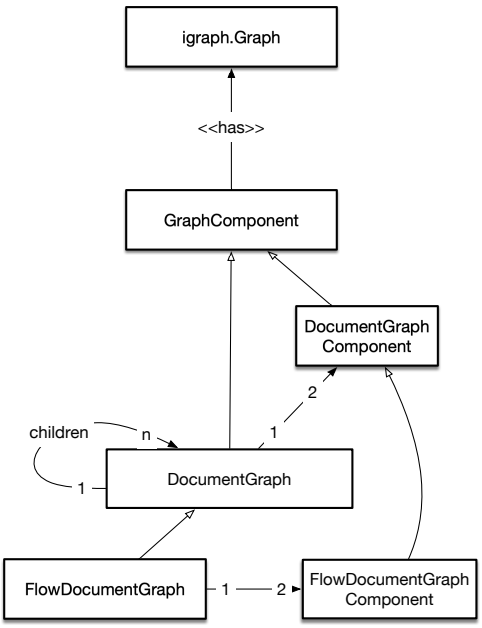

Figure 1: **Class diagram**. The graph component class hierarchy.

An instance of a `DocumentGraph` that represents the entire document to be aligned is passed between components to be aligned and inherits from `GraphComponent` to manage the source and summary components as a bipartite graph. However, it also maintains the disconnected source and summary graphs as instances of `DocumentGraphComponent` as shown by the *has a* arrow with "1" to "2" as shown in Figure 1. This makes it easy read the source and summary as individual graphs, such as alignment method iteration methods involving capacity tightening, alignment edge removals, etc.

Initially, an `AmrFeatureDocument` is either read from human annotations or parsed with an abstract meaning representation (AMR) parser such as *Symmetric PaRsIng aNd Generation* (SPRING) [1]. The

[1] https://igraph.org

`AmrFeatureDocument` has both the natural language features and the AMR graph. It is then given to the framework and used to create the nascent graph, which only has the disconnected components without alignments.

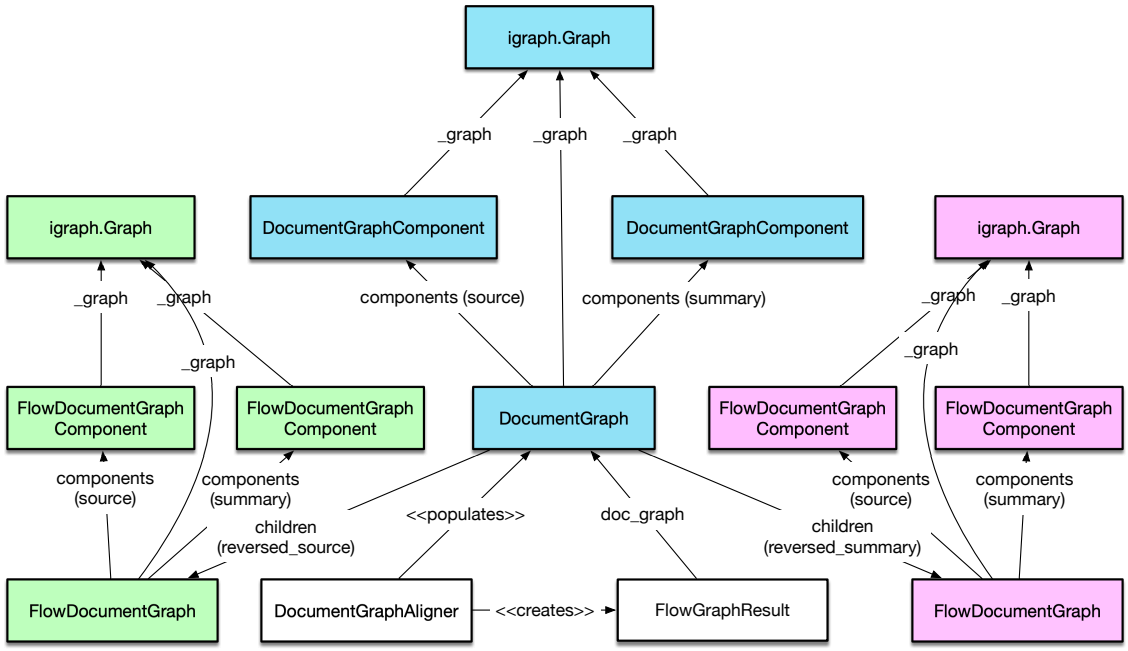

Figure 2: **Instance diagram.** The graph components in memory. The blue classes make up the nascent graph, the green make up the summary flow graph and the purple make up the source flow graph.

The first algorithm step implemented by the `DocumentGraphController` component clones the nascent graph. This object will be connected as a bipartite graph and will become the flow network. This clones the `DocumentGraph` as a `FlowDocumentGraph` and each `DocumentGraphComponent` as `FlowDocumentGraphComponent` instances. The `FlowDocumentGraph` clone with flow from the source to the summary and another for the summary to the source are then made children of the `DocumentGraph` following the GoF [2] composite pattern given in Figure 1 with the *children* self reference. After this step is complete, these classes and their respective igraph graphs are shown in Figure 2.

The `FlowDocumentGraph` children of `DocumentGraph` have the same structure of components as the nascent graph. Once these classes are created, terminal nodes are added to the `FlowDocumentGraph` and the max flow algorithm is run. Afterward, flow values assigned on all edge in the `FlowDocumentGraph` and `FlowDocumentGraphComponent` components. It is *imporant to understand* that all edges and nodes extend from `GraphAttribute` and are shared between all igraph instances. This means the flow values and capacities are the same across all graphs, including the nascent graph.

The flow graph children of the nascent graph will have the alignments, which is created by `DocumentGraphAligner` as a `FlowGraphResult`. The naming of the children are `reversed_source` and `reversed_summary` since the edges are reversed in each flow graph. Each are flow networks with `reversed_source` having the flow from the summary to the source and `reversed_source` having the

flow from the source to the summary.

`FlowGraphResult` if a container class for flow document results, which include the detailed data as dictionaries of statistics. The data dictionaries and statistics are created by the source component of the `reversed_source` flow graph and the `reversed_summary` of the summary component. It work out this way since each flow graph one component has the source feeding into infinity capacity edges, which is used to facilitate the flow on the other component that yields useful values. Before the graph building algorithm is complete, the source graph flow values are saved with `SnapshotDocumentGraphController`, then restored since they are overwritten by the final summary graph iteration. For this reason, the final flow graphs have the useful flow information from their respective run of the max flow algorithm.

Abbreviations

SPRING Symmetric PaRsIng aNd Generation p. 1

Definitions

AMR (abstract meaning representation) A semantic representation language that describes the abstract meaning of a sentence as a directed acyclic graph or a context free notation of Penman. p. 1

CALAMR (Component ALignment for Abstract Meaning Representation) An Abstract Meaning Representation Alignment Method p. 1

flow network A graph, or capacitance network, that associates a capacity and a flow with each edge of a graph. p. 2

max flow The maximum amount of flow available to traverse an s-t flow network given the capacities of the network. p. 1

nascent graph The initial graph with disconnected source and summary components. p. 2

References

[1] Michele Bevilacqua, Rexhina Blloshmi, and Roberto Navigli. “One SPRING to Rule Them Both: Symmetric AMR Semantic Parsing and Generation without a Complex Pipeline”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 14. Virtual, May 18, 2021, pp. 12564–12573 (cit. on p. 1).

[2] Erich Gamma et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. 1st ed. Addison-Wesley Professional, 1994. ISBN: 0-201-63361-2 (cit. on p. 2).

[3] Andrew V. Goldberg and Robert E. Tarjan. “A New Approach to the Maximum-Flow Problem”. In: *Journal of the ACM* 35.4 (Oct. 1, 1988), pp. 921–940. ISSN: 0004-5411. DOI: 10.1145/48014.61051 (cit. on p. 1).