Taxonomy Package (TaxPac): An Experimental Mathematics Environment for Knowledge Systems Analysis

Cliff Joslyn and Amanda White

Version July, 2009

Abstract

This document presents a primer and class specification for the Taxonomy Package (`TaxPac`) system developed by the Battelle Memorial Institute at the Pacific Northwest National Laboratory. `TaxPac` is an experimental mathematics platform available in Python for the manipulation and measurement of semantic hierarchies represented as ordered sets. `TaxPac` is built as an extension of the NetworkX system for graph analysis developed by the Los Alamos National Laboratory.

Contents

Chapter 1

Introduction

Semantic typing systems for modern knowledge systems such as ontological databases are dominated by structures characterized as **semantic hierarchies:** collections of objects (classes or linguistic concepts) which are organized into hierarchies such as taxonomic (subsumption, "is-a"), meronomic (compositional, "has-part"), and/or implication (logical, "follows-from") relations. Prominent examples include WordNet in the computational linguistics community [10][1] and the Gene Ontology in the computational genomics community (GO [2][2]).

Long a bullwark of both object-oriented programming and artificial intelligence [20, 24], such ontologies are increasingly seen as critical for facilitating large-scale knowledgebase integration [3, 4, 29]. Tasks include the construction of semantic hierarchies by authors and curators, automatically identifying hierarchical relations in data, annotation of semantic categories with instance data, alignment, linkage, and mapping of multiple structures together into federated ontologies, and search, navigation, clustering, and visualization.

Fig. 1.1 shows a portion of the GO. Black text indicates nodes representing biological processes, and are arranged in a subsumption hierarchy ("DNA repair" is a kind of "DNA metabolism"). The names of genes from three species of model organism as "annotations" to thse nodes, indicating that those genes perform those functions. Note that genes can appear at multiple nodes.

While Fig. 1.1 shows only a fragment of one portion of the GO, the GO is typical of many such real-world semantic hierarchies in growing quite large, currently over twenty-five thousand nodes. As the number and size of semantic hierarchies grows, it is becoming critical to have computer systems which are appropriate for managing them, and especially important to complement manual methods with algorithmic approaches to tasks such as construction, alignment, annotation, and visualization. Thus it is essential to have a solid mathematical grounding for the analysis of semantic hierarchies, and especially concepts such as distances, sizes of regions, and the vertical structure of levels and rank separation.

As mathemtical data objects, semantic hierarchies resemble in many aspects top-rooted trees. But with the presence of significant amounts of "multiple inheritance" (nodes having more than one parent), and also the possible inclusion of transitive links, they must in general be represented at most as directed acyclic graphs (DAGs). And since the primary semantic categories of subsumption and composition are transitive, the proper mathematical grounding for such algorithms and

[1] `wordnet.princeton.edu`
[2] `www.geneontology.org`

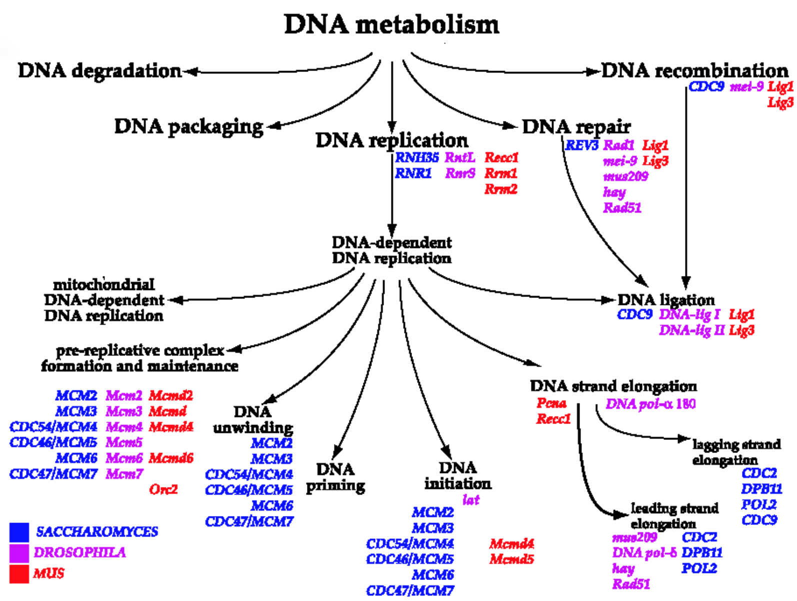

Figure 1.1: A portion of the Biological Process branch of the Gene Ontology (adapted from [2]). The database is structured as a large, top-rooted directed acyclic graph of genomic functional categories, labeled with the genes of multiple species.

measures is **order theory**, or the mathematics of directed acyclic graphs, lattices, and partially ordered sets (posets) [8].

Order theory provides the fundamental formalization of hierarchy in the most general sense, and provides or promises the needed tools for managing semantic hierarchies. But order theory has been largely neglected in knowledge systems technology, and the most prominent approaches to managing the GO and Wordnet are effectively *ad hoc* and *de novo* (e.g. [6]). Our prior work [13, 14, 15, 16, 17, 18, 28] has advanced the use of measures of distance and similarity, characterizations of structures and levels, and characterizations of mappings and linkages within semantic hierarchies.

This document presents a primer and class specification for the Taxonomy Package (`TaxPac`) experimental mathematics platform for measuring, manipulating, and displaying taxonomic structures for knowledge systems analysis. `TaxPac` is built as an extension of the NetworkX[3] [12] system for graph analysis developed by the Los Alamos National Laboratory.

We begin with a terse primer on the fundamentals of order theory for application to semantic hierarchies. We then describe the `TaxPac` system and present the Python class hierarchy to complement the Javadoc in the distribution, with interleaved examples. We conclude with a reference table of the mathematical notation.

[3]http://networkx.lanl.gov

Chapter 2

Order Theory for Knowledge Systems

Here we tersely introduce the mathematical notation we use for ordered sets and lattices. For general references on order and lattice theory, see [5, 8, 23, 26].

2.1 Directed Graphs

Let $\mathcal{G} := \langle P, E \rangle$ be a **directed graph**, with $E \subseteq P^2$ a set of directed **edges** on a nonempty, finite set P of **nodes**. Let $e := \langle a, b \rangle$ be a **link** if $e \in E$, and denote $a \prec b, b \succ a$. Let $\succ(a) := \{b \succ a\} \subseteq P$ be the **parents** of $a \in P$, and $\prec(a) := \{b \prec a\} \subseteq P$ its **children**. A node $a \in P$ is a **root** if $\succ(a) = \emptyset$, and a **leaf** if $\prec(a) = \emptyset$.

A graph $\mathcal{G}$ is **node weighted** if there exists a function $w_P \colon P \to \mathbb{R} \cup \emptyset$, and link-weighted if there exists a function $w_E \colon E \to \mathbb{R} \cup \emptyset$.

Let a vector (ordered set, possibly containing duplicates) of n nodes $\vec{C} := \langle a_i \rangle_{i=1}^n \subseteq P$ be a **directed path** (or just **path**) in $\mathcal{G}$ if $n \geq 3$, and if $\forall a_i \in \vec{C}, a_i \prec a_{i+1}$ or $i = n$, where $a \in \vec{C}$ means that $\exists a \in P, \exists 1 \leq i \leq n, a_i = a$. Note that therefore for our purposes, a link is not a path, since a path has at least two links. Denote $a_1 \prec a_2 \prec \ldots \prec a_n$, and $\vec{C} \subseteq \mathcal{G}$. We say that an edge is included in a path, denoted $e \in \vec{C}$ or $a \prec b \in \vec{C}$, if $\exists a_i, a_{i+1} \in \vec{C}, a = a_i, b = a_{i+1}$. **(not yet implemented, caj)**

For any two nodes $a, b \in P$, we say that a is **reachable** from b if either $a \prec b$ or there is a path $\vec{C} = \langle a, \ldots, b \rangle \subseteq \mathcal{G}$. We then denote

$$a \leq b \tag{2.1}$$

We say $a \leq b \in \mathcal{G}$ to mean that $a, b \in P$ and $a \leq b$.

A link $a \prec b$ is called **transitive** if there exists a path (in our sense) $\vec{C} = \langle a, \ldots, b \rangle \subseteq P$. A graph $\mathcal{G}$ is **transitively reduced** if it contains no transitive links, and **transitively closed** if it contains all possible transitive links, that is, for all paths $\vec{C} = \langle a_1, a_2, \ldots, a_n \rangle \subseteq \mathcal{G}, a_1 \prec a_n$. Note that our definition of transitively reduced is slightly different from that of [1]. Our definition corresponds to [1] for acyclic graphs, and for that reason we only discuss or implement transitive reduction for acyclic graphs.

We deal secondarily with undirected (simple) graphs as the symmetric closures of digraphs. Given a digraph $\mathcal{G} = \langle P, E \rangle$, its **symmetric closure** is the digraph $\mathcal{U}(\mathcal{G}) := \langle P, \mathcal{U}(E) \rangle$, where $\mathcal{U}(E) := \{\langle a, b \rangle \in P^2 : a \prec b \text{ or } b \prec a\}$. A vector of nodes $\vec{C}^U := \langle a_i \rangle_{i=1}^n \subseteq P$ is an **undirected path in $\mathcal{G}$**

if $\vec{C}^U$ is a path in $\mathcal{U}(\mathcal{G})$. We say that an edge is included in an undirected path, denoted $e \in \vec{C}^U$ or $a \prec b \in \vec{C}^U$, if $\exists a_i, a_{i+1} \in \vec{C}^U, a = a_i, b = a_{i+1}$. **(not yet implemented, caj)**

The number of **turns** $T(\vec{C}^U)$ of an undirected path $\vec{C}^U$ is the minimum number of sub-vectors of $\vec{C}^U$ which are directed paths, and whose union is $\vec{C}^U$. Note that $T(\vec{C}^U) \geq 0$, and $T(\vec{C}^U) = 0$ only if $\vec{C}^U$ is a directed path. **(not yet implemented, caj)**

2.2 Directed Acyclic Graphs (DAGs)

A path $\vec{C} = \langle a_1, a_2, \ldots, a_n \rangle \subseteq \mathcal{G}$ is a **cycle** when C has no duplicates and $a_n \prec a_1$. A set of nodes $S \subseteq P$ is a **strongly connected component (SCC)** [21] if there is a path from every node $a \in S$ to every other. It follows that S is a union of cycles. If $\mathcal{G}$ is connected and contains no cycles, then we call $\mathcal{G}$ a **directed acyclic graph (DAG)**, denoted $\mathcal{D}$. The **cyclic closure** of a graph $\mathcal{G}$ is a DAG $\mathcal{D}$ created from $\mathcal{G}$ by contracting all its SCCs to single nodes, and connecting those to the base graph according to the links into the SCC [7].

Given a DAG $\mathcal{D}$, for any subset of nodes $Q \subseteq P$, define its maximal and minimal elements as

$$\mathrm{Max}(Q) := \{a \in Q : \nexists b \in Q, a \prec b\} \subseteq Q$$

$$\mathrm{Min}(Q) := \{a \in Q : \nexists b \in Q, b \prec a\} \subseteq Q,$$

called the **roots** and **leaves** respectively.

In a DAG $\mathcal{D}$, for any node $a \in P$, define its **up-set** $\uparrow a := \{b \in P : b \geq a\} \subseteq P$, its **down-set** $\downarrow a := \{b \in P : b \leq a\} \subseteq P$, and its **hourglass** $\Xi(a) := \uparrow a \cup \downarrow a \subseteq P$.

For any two nodes $a, b \in P$, define their **upper cone** as $a \nabla b := \uparrow a \cap \uparrow b \subseteq P$ and **lower cone** as $a \Delta b := \downarrow a \cap \downarrow b \subseteq P$. Their **join** is $a \vee b := \mathrm{Min}(a \nabla b) \subseteq P$, and **meet** is $a \wedge b := \mathrm{Max}(a \Delta b) \subseteq P$. Further define (asymmetric) **implication** $a \rightarrow b := \uparrow a \setminus \uparrow b$ and **difference** $a - b := \downarrow a \setminus \downarrow b$, yielding the identities $\forall a, b \in P$

$$\uparrow a = (a \rightarrow b) \cup (a \nabla b), \quad \uparrow a \cup \uparrow b = (a \rightarrow b) \cup (b \rightarrow a) \cup (a \nabla b),$$

$$\downarrow a = (a - b) \cup (a \Delta b), \quad \downarrow a \cup \downarrow b = (a - b) \cup (b - a) \cup (a \Delta b).$$

Note, however, that in general $\uparrow a \cup \uparrow b \neq a \nabla b, \downarrow a \cup \downarrow b \neq a \Delta b$. This does hold, however, at least if $\mathcal{P}$ is a Boolean lattice. **(not yet implemented, caj)**

2.3 Intervals and Bounded DAGs

For $a \leq b \in \mathcal{G}$, define the **interval** $[a, b] := \{c \in P : a \leq c \leq b\} = \uparrow a \cap \downarrow b$. Then we have that

$$a \nabla b = \uparrow b, \quad a \vee b = \mathrm{Max}([a, b]) = \{b\}, \qquad a \Delta b = \downarrow a, \quad a \wedge b = \mathrm{Min}([a, b]) = \{a\}.$$

$\mathcal{D}$ is **upper-bounded** if $|\mathrm{Max}(P)| = 1$, so that $\mathrm{Max}(P) = \{1\}$ with $1 \in P$; and **lower-bounded** if $|\mathrm{Min}(P)| = 1$, so that $\mathrm{Min}(P) = \{0\}$ with $0 \in P$. $\mathcal{D}$ is **bounded** (we denote $\mathcal{B}$) if it is both upper- and lower-bounded. If $\mathcal{D}$ is not upper (lower) bounded, then it can be made so by inserting a node $1 \in P$ ($0 \in P$) and inserting all links $\{a \prec 1 : a \in \mathrm{Max}(P)\} \in E$ ($\{0 \prec a : a \in \mathrm{Min}(P)\} \in E$).

Note that $\forall a \le b \in P$, the interval $[a,b]$ is a sub-poset bounded above by b and below by a. Thus in particular, if $\mathcal{B}$ is bounded, then $P = [0,1]$, and $\forall a \in P, \uparrow a = [a,1], \downarrow a = [0,a], \Xi(a) = [0,a] \cup [a,1]$, and all are also bounded. Additionally, $\forall a,b \in P, a \nabla b, a \vee b, a \Delta b$ and $a \wedge b$ are all non-empty.

Given a bounded DAG $\mathcal{B}$, then define its **height** as $\mathcal{H}(\mathcal{B}) := \max_{\vec{C} \subseteq \mathcal{B}} |\vec{C}| - 1$, the size of its largest directed path. For a node $a \in P$, define its **top rank** as $r^t(a) := \mathcal{H}([a,1])$, and its **bottom rank** as $r^b(a) := \mathcal{H}(\mathcal{B}) - \mathcal{H}([0,a])$. It can be shown that $\forall a \in P, r^t(a) \le r^b(a)$, so define its **interval rank** as the interval $R(a) := [r^t(a), r^b(a)]$, the **midrank** as the midpoint of that interval $\frac{r^t(a)+r^b(a)}{2}$, and the **rank width** as the width of that interval $r^b(a) - r^t(a)$.

Given a bounded DAG $\mathcal{B}$, then define it **atoms** as $\bot(\mathcal{B}) := \succ(0)$, and its **co-atoms** as $\top(\mathcal{D}) := \prec(1)$. For any node $a \in P$, define its **complement**

$$\overline{a} := P \setminus \left(\bigcup_{b \in \top(\uparrow a)} \downarrow b \right) \cup \left(\bigcup_{b \in \bot(\downarrow a)} \uparrow b \right) \subseteq P.$$

Note $\overline{0} = \{1\}, \overline{1} = \{0\}$. For any subset of nodes $Q \subseteq P$, define its complement as

$$\overline{Q} := \bigcap_{a \in Q} \overline{a}.$$

While $\forall a \in P, \overline{a} \neq \emptyset$, it is common for any $Q \subseteq P, \overline{Q} = \emptyset$. **(not yet implemented, caj)**

Given a bounded DAG $\mathcal{D}$ equipped with a complement operator $\bar{\cdot}$, we can define the following Boolean-like operations **(not yet implemented, caj)** .

Difference: $a - b := \mathrm{Max}(a \Delta \overline{b})$

Implication: $a \to b := \mathrm{Min}(a \nabla \overline{b})$

Symmetric Difference: $a \bowtie b := \mathrm{Min}((a-b) \nabla (b-a))$

2.3.1 Node Comparisons

If $\mathcal{B}$ is node-weighted, then we have its **upper weight** $F^*(a) := \sum_{b \ge a} w_P(b)$ and **lower weight** $F_*(a) := \sum_{b \le a} w_P(b)$. Denote

$$F^\vee(a,b) := 2 \max_{c \in a \vee b} F^*(c), \qquad F_\wedge(a,b) := 2 \max_{c \in a \wedge b} F_*(c).$$

For any two nodes $a, b \in P$, define their **upper distance** and **lower distance**

$$d^*(a,b) := F^*(a) + F^*(b) - 2F^\vee(a,b), \qquad d_*(a,b) := F_*(a) + F_*(b) - 2F_\wedge(a,b)$$

respectively. Finally, if $a \le b \in \mathcal{G}$ then $d^*(a,b) = F^*(b) - F^*(a)$ and $d_*(a,b) = F_*(a) - F_*(b)$.

The **Tversky parameterized ratio** [27] is generalized in ordered sets to be

$$S^*_{\alpha,\beta}(a,b) := \frac{F^\vee(a,b)}{F_\wedge(a,b) + \alpha}$$

2.4 Posets and Covers

A structure $\mathcal{P} = \langle P, \leq \rangle$ is called an **ordered set, partially-ordered set**, or **poset** if $\leq \subseteq P^2$ is a binary relation on P which is reflexive, anti-symmetric, and transitive. We say $a \leq b \in \mathcal{P}$ to mean that $a, b \in P$ and $a \leq b$.

If a DAG $\mathcal{D}$ is transitively closed, then $\langle P, \leq \rangle$ is a poset, where $\leq$ is the relation from (2.1). Given a DAG $\mathcal{D}$, then let $\mathcal{P}(\mathcal{D})$ be its **transitive closure**, the DAG produced by including all possible transitive links consistent with its paths. Thus $a \leq b \in \mathcal{G} \rightarrow a \leq b \in \mathcal{P}$. The graph $\mathcal{V}(\mathcal{D})$ produced from a DAG $\mathcal{D}$ by removing all its transitive links determines a **cover relation** or **Hasse diagram**.

In this way, each cover relation $\mathcal{V}$ determines a unique poset $\mathcal{P}(\mathcal{V})$, and *vice versa* $\mathcal{P}$ determines a unique cover $\mathcal{V}(\mathcal{P})$; each DAG $\mathcal{D}$ determines a unique poset $\mathcal{P}(\mathcal{D})$ and cover $\mathcal{V}(\mathcal{D})$; and each unique poset-cover pair determines a class of DAGs equivalent by transitive links. Thus the **degree of transitivity** of a DAG can be measured as

$$TR(\mathcal{D}) := \frac{|\mathcal{D} \setminus \mathcal{V}(\mathcal{D})|}{|\mathcal{P}(\mathcal{D}) \setminus \mathcal{V}(\mathcal{D})|}.$$

Given a poset $\mathcal{P} = \langle P, \leq \rangle$, two nodes $a, b \in P$ are called **comparable** if $a \leq b$ or $b \leq a$ (we denote $a \sim b$), and **noncomparable** otherwise (we denote $a \not\sim b$). A **chain** is a set of nodes $C \subseteq P$ which are pairwise comparable, so that $\forall a, b \in C, a \sim b$. A chain $C \subseteq P$ is called **saturated** if in addition $\forall a, b \in C, a \prec b$ or $b \prec a$. Note that thereby the nodes of any directed path $\vec{C} \subseteq \mathcal{G}$ form a saturated chain $C \subseteq P$. An **antichain** is a set of nodes $A \subseteq P$ which are pairwise noncomparable, so that $\forall a, b \in A, a \not\sim b$. Define the **width** $\mathcal{W}(\mathcal{P}) := \max_{A \subseteq P} |A|$ of a poset $\mathcal{P}$ as the size of its largest antichain.

2.5 Lattices and Trees

A poset $\mathcal{P} = \langle P, \leq \rangle$ is a **join semi-lattice** $\mathcal{L}^\vee$ if $\forall a, b \in P, |a \vee b| = 1$, and is a **meet semi-lattice** $\mathcal{L}^\wedge$ if $\forall a, b \in P, |a \wedge b| = 1$. In these cases we denote $a \vee b = c, a \wedge b = d \in P$ for those unique nodes, respectively.

A poset $\mathcal{P}$ is a **lattice** $\mathcal{L}$ if it is both a join- and a meet-semilattice. In a lattice $\mathcal{L}$, we have that

$$d^*(a,b) = F^*(a) + F^*(b) - 2F^*(a \vee b), \qquad d_*(a,b) = F_*(a) + F_*(b) - 2F_*(a \wedge b).$$

Every poset $\mathcal{P}$ can be embedded homomorphically into a lattice $\mathcal{L}$ through the **Dedekind-MacNeille completion** process [8].

A join semi-lattice $\mathcal{L}^\vee$ is an **upper tree** $\mathcal{R}^\vee$ if $\forall a \neq 0 \in P, |\succ(a)| = 1$. A meet semi-lattice $\mathcal{L}^\wedge$ is a **lower tree** $\mathcal{R}^\wedge$ if $\forall a \neq 1 \in P, |\prec(a)| = 1$.

In this work, we will consider a **concept lattice** to be a doubly-labeled, bounded lattice. This work is not complete yet. See [8, 11] for more information about the mathematical definitions.

Chapter 3

Class Specification

We now provide a specification of the `TaxPac` classes and methods, with interleaved examples, to complement the HTML documentation which lists every class and function with documentation. The `TaxPac` class hierarchy is shown in Fig. 3.1. Classes and methods are described below. Methods that are not currently implemented are indicated by *.

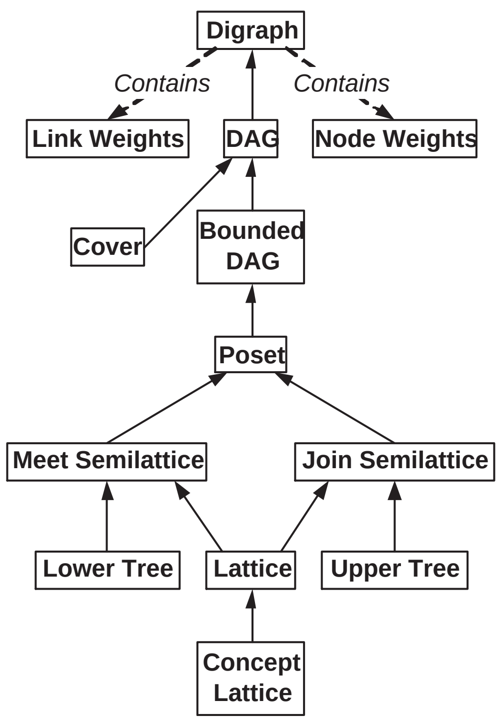

Figure 3.1: Class hierarchy for `TaxPac`.

Each `TaxPac` object and method is illustrated with examples drawn from a digraph $\mathcal{G}$ shown in Fig. 3.2, with its components identified in Fig. 3.3, its cyclic closure shown in Fig. 3.5, and the bounded version of its cyclic closure in Fig. 3.6. We additionally sometimes use a DAG shown in Fig. 3.4. We also repeat the mathematical notation developed in Sec. 2, which is then compiled in the appendix.

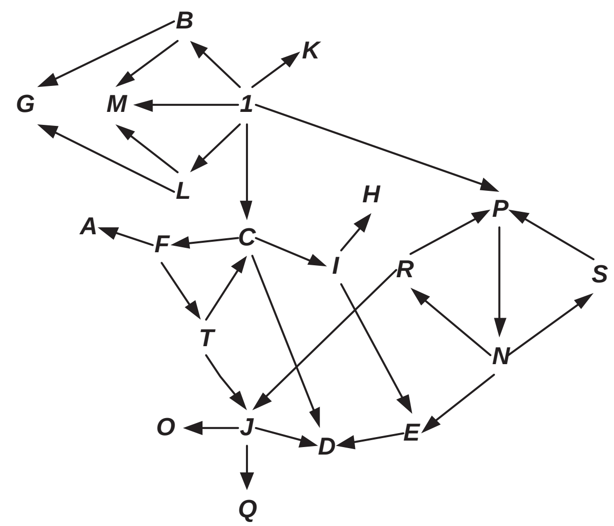

Figure 3.2: Example digraph $\mathcal{G}$.

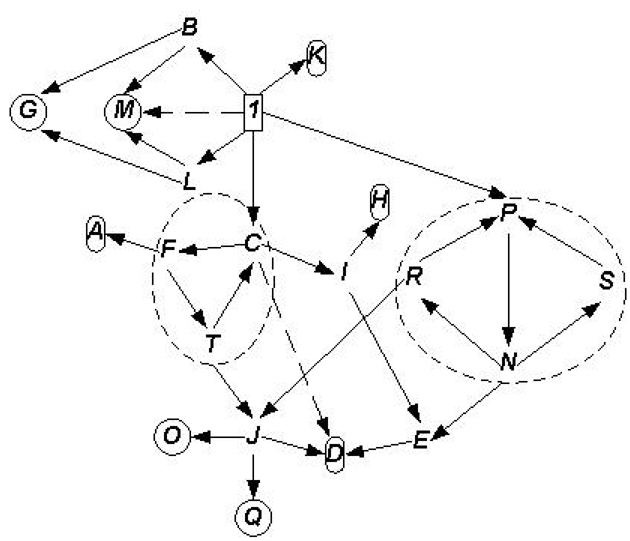

Figure 3.3: $\mathcal{G}$ with its components identified.

3.1 Class Digraph

The `TaxPac` class `DiGraph` extends the NetworkX class `MultiDiGraph`.

3.1.1 Objects

Digraph: Let $\mathcal{G} := \langle P, E \rangle$ be a directed graph, with $E \subseteq P^2$ a set of directed edges on a finite set P of nodes. Fig. 3.2 shows a digraph.

Link: Let $e := \langle a, b \rangle$ be a link if $e \in E$. Denote $a \prec b, b \succ a$. We have $C \prec T, I \succ E$.

Directed Path: Let a vector (ordered set, possibly containing duplicates) of nodes $\vec{C} := \langle a_i \rangle_{i=1}^n \subseteq P$ be a **directed path in** $\mathcal{G}$ if $n \geq 3$ and $\forall a_i \in \vec{C}, a_i \prec a_{i+1}$ or $i = n$. Note that for our purposes, a link is not a path, a path has at least two links! Denote $\vec{C} \subseteq \mathcal{G}$. We have $C \prec T \prec J \prec Q \subseteq \mathcal{D}$.

Undirected Path: A vector of nodes $\vec{C}^U := \langle a_i \rangle_{i=1}^n \subseteq P$ is an **undirected path in** $\mathcal{G}$ if $\vec{C}^U$ is a path in $\mathcal{U}(\mathcal{G})$. $C \succ D \prec E$ is an undirected path. **(not yet implemented, caj)**

Cycle: A directed path $\vec{C} = a_1 \prec a_2 \prec \ldots \prec a_n$ is a **cycle** when C has no duplicates and $a_n \prec a_1$. $F \prec C \prec T$ is a cycle.

Weight Set: Let $W = \mathbb{R} \cup \{\emptyset\}$ be a weight set.

3.1.2 Methods

3.1.2.1 Weights

bool = `is_node_weighted(`$\mathcal{G}$`)`**:** Returns `true` if $\exists w_P \colon P \to W$.

$\mathcal{G}'$ = `set_node_weights(`$\mathcal{G}, w_P$`)`**:** Returns $\mathcal{G}$ now equipped with the node weight function $w_P \colon P \to W$. There are four special cases:

Null Node Weighting: $\forall a \in P, w_P(a) \equiv \emptyset$

Zero Node Weighting: $\forall a \in P, w_P(a) \equiv 0$

Unit Node Weighting: $\forall a \in P, w_P(a) \equiv 1$.

Probabilistic Node Weighting: $\forall a \in P, w_P(a) \in [0, 1], \sum_{a \in P} w_P(a) = 1$.

Information Content Node Weighting: $\forall a \in P, w_P(a) = \log(p(a)) \in [0, \infty)$, where $p(a)$ is a probabilistic node weight. **(not yet implemented, caj)**

double = `get_node_weight(`$\mathcal{G}$`, a)`**:** Returns the node weight $w_P(a)$ for the given node, or null if no weight.

bool = `is_edge_weighted(`$\mathcal{G}$`)`**:** Returns `true` if $\exists w_E \colon E \to W$.

$\mathcal{G}'$ = `set_edge_weights(`$\mathcal{G}, w_E$`)`**:** Returns $\mathcal{G}$ now equipped with the edge weight function $w_E \colon E \to W$. There are two special cases:

Null Link Weighting: $\forall e \in E, w_E(e) \equiv \emptyset$

Zero Link Weighting: $\forall e \in E, w_E(e) \equiv 0$

double $=$ `get_edge_weight(`$\mathcal{G}$`, e)`**:** Returns the edge weight $w_E(e)$ for the given edge, or null if no weight.

3.1.2.2 Paths and Connectiveness

bool $=$ `is_connected(`$\mathcal{G}$`)`**:** Returns true if $\mathcal{G}$ is connected [9]. `is_connected`$(\mathcal{G}) =$ `true`

bool $=$ `is_link`(a, b)**:** Returns true if $a \prec b$. $A \prec F$.

$P' =$ `parents`(a)**:** Returns $\{b \succ a\}$. Denote $\succ(a)$. $\succ(J) = \{G, R\}$.

$Q' =$ `roots`$(Q \subseteq P)$**:** Returns $\{a \in Q : \succ(a) = \emptyset\} \subseteq Q$. Denote $\mathrm{Max}(Q)$. $\mathrm{Max}(P) = \{1\}, \mathrm{Max}(\{G, R, J, O, Q\}) = \{G, R\}, \mathrm{Max}(\{F, C, T\}) = \emptyset$.

bool $=$ `is_root`(a)**:** Returns `true` if a has no parents.

$P' =$ `children`(a)**:** Returns $\{b \prec a\}$. Denote $\prec(a)$. $\prec(J) = \{O, Q\}$.

$Q' =$ `leaves`$(Q \subseteq P)$**:** Returns $\{a \in Q : \prec(a) = \emptyset\} \subseteq Q$. Denote $\mathrm{Min}(Q)$. $\mathrm{Min}(P) = \{G, M, K, A, H, O, D, Q\}, \mathrm{Min}$ $\{O, Q\}, \mathrm{Min}(\{F, C, T\}) = \emptyset$.

bool $=$ `is_leaf`(a)**:** Returns `true` if a has no children.

$\{\vec{C}\} =$ `paths`(a, b)**:** Returns the set of all noncyclic directed paths $\vec{C} = \langle a, \ldots, b\rangle \subseteq \mathcal{G}$. `paths`$(D, C) = \{D \prec C, D \prec E \prec I \prec C\}$. **(not yet implemented, caj)**

int $=$ `turns`$(\vec{C}^U)$**:** Returns the numbder of turns in the undirected path $\vec{C}^U$. `turns`$(1 \succ M \prec B \succ G) = 2$. **(not yet implemented, caj)**

$\{\vec{C}^U\} =$ `unidrected_paths(a,b)`**:** Returns the set of all noncyclic undirected paths $\vec{C}^U = \langle a, \ldots, b\rangle \subseteq \mathcal{G}$. `undirected_paths`$(G, 1) = \{G \prec B \prec 1, G \prec B \succ M \prec 1, G \prec B \succ M \prec L \prec 1, G \prec L \succ M \prec B \prec 1, G \prec L \succ M \prec 1, G \prec M \prec 1\}$. **(not yet implemented, caj)**

bool $=$ `has_directed_path`(a, b)**:** Returns true if `paths`(a, b) is non-empty. `has_directed_path`(J, N)=`true`, `has_directed_path`(J, R)=`false`. **(not yet implemented, caj)**

double$\cup\{\emptyset\}$ $=$ `path_weight`$(\vec{C})$**:** Returns $\sum_{e \in \vec{C}} w_E(e)$. Under unitary weighting, `path_weight`$(\langle D, E, I, C\rangle) = 4$. (By convention for every $r \in R, r+$ NULL $=$ NULL.) **(not yet implemented, caj)**

double$\cup\{\emptyset\}$ $=$ `undirected_path_weight`$(\vec{C}^U)$**:** Returns $\sum_{e \in \vec{C}^U} w_E(e)$. Under unitary weighting, `path_weight`$(\langle C, T, J\rangle) = 3$. (By convention for every $r \in R, r+$ NULL $=$ NULL.) **(not yet implemented, caj)**

$\langle$**double** $\cup \{\emptyset\}\rangle$ =`path_weights`(a, b))**:** Returns the vector of `path_weight` for all directed paths from a to b. Under unitary weighting, `path_weights`$(D, C) = \langle 2, 4\rangle$. **(not yet implemented, caj)**

$\langle$**double** $\cup \{\emptyset\}\rangle$ =`undirected_path_weights`(a, b))**:** Returns the vector of `undirected_path_weight` for all undirected paths from a to b. Under unitary weighting, `undirected_path_weights`$(G, 1) = \langle 3, 4, 5, 5, 4, 3\rangle$ **(not yet implemented, caj)**

double$\cup\{\emptyset\}$ = `min_directed_path(`a, b`)`: Returns minimum path length if `has_directed_path(`a, b`)`, otherwise returns NULL. Under unitary weighting, `min_path_length`$(D, C) = 2$, `min_path_length`$(J, N) =$ NULL. **(not yet implemented, caj)**

double$\cup\{\emptyset\}$ = `min_undirected_path(`a, b`)`: Returns minimum undirected path length if `has_directed_path(`$a,$ otherwise returns NULL. Under unitary weighting, `min_path_length`$(G, 1) = 3$. **(not yet implemented, caj)**

bool = `is_connected(`a, b`)`: Returns `true` if $a = b$ or `is_link(`a, b`)` or `has_directed_path(`a, b`)`. Note polymorphism with `is_connected(`$\mathcal{G}$`)`. `is_connected`(J, N)=`true`, `is_connected`(J, R)=`true`.

3.1.2.3 Transitivity and Cycles

bool = `is_transitive(`a, b`)`: Returns `true` if `has_directed_path(`a, b`)`. `is_transitive`(D, C) = `is_transitive`$(M, 1)$=`true`.

bool = `is_cycle(`$\vec{C}$`)`: Returns `true` if $\vec{C} \subseteq \mathcal{G}$ is a cycle. `is_cycle`$(\{P, R, N\})$ =`is_cycle`$(\{F, C, T\})$ =`true`.

$\{\vec{C}\}$ = `get_cycles()`: Returns the set of all cycles in the graph.

$\mathcal{G}'$ = `transitive_closure(`$\mathcal{G}$`)`: Returns the transitive closure of $\mathcal{G}$ [19, 25]. Denote $\mathcal{P}(\mathcal{G})$. In Fig. 3.4, for the digraph $\mathcal{D}$ shown on top, $\mathcal{P}(\mathcal{D})$ is shown on the right.

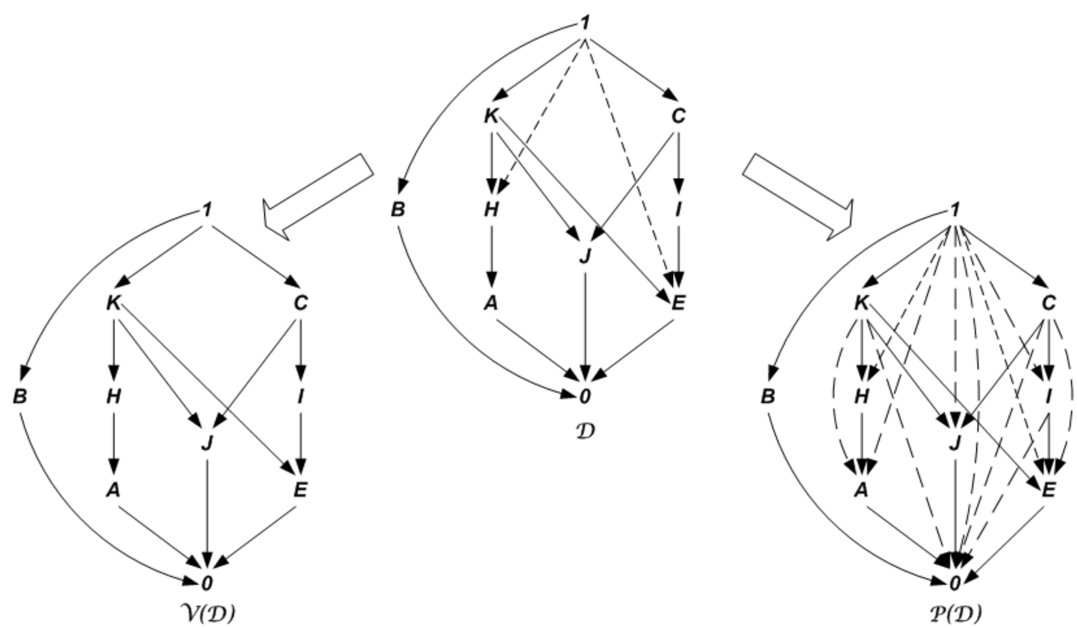

Figure 3.4: (Top) A DAG $\mathcal{D}$. (Left) Transitive reduction $\mathcal{V}(\mathcal{D})$. (Right) Transitive closure $\mathcal{P}(\mathcal{D})$.

bool = `is_transitively_closed(`$\mathcal{G}$`)`: Returns `true` if $\mathcal{P}(\mathcal{G}) = \mathcal{G}$. For any digraph $\mathcal{G}$, `is_transitively_closed`$(\mathcal{P}(\mathcal{G}))$ = `true`.

3.2 Class DAG : Digraph

We note that many of the mathematical methods described in Sec. 2 which are available on DAGs are implemented in `TaxPac` in the BoundedDAG class documented in Sec. 3.3 below. This is for the purposes of engineering convenience and efficiency. In practice, the methods for which `TaxPac` is used require bounded DAGs.

3.2.1 Objects

DAG: Let $\mathcal{D} := \mathcal{G}$ be a directed graph where `is_connected`$(\mathcal{G})$ and $\nexists C \subseteq \mathcal{G}$, `is_cycle`$(C)$.

3.2.2 Methods

$\mathcal{D} =$ `cyclic_closure_constructor`$(\mathcal{G})$: Constructs the DAG $\mathcal{D}$ as the cyclic closure of the digraph $\mathcal{G}$ by the algorithm:

1. Input $\mathcal{G} = \langle P, E \rangle$.
2. Let $\mathbf{S} := \{S_j\}_{j=1}^N, S_j \subseteq P$ be the set of all strongly connected components (SCCs) of G [21].
3. Let $P^\circ := \bigcup_{j=1}^M S_j \subseteq P$ be the set of all nodes in any SCC.
4. Let $P' := P \setminus P^\circ$
5. For each SCC $S_j \in \mathbf{S}$, insert into P' a new node a_j mapping uniquely to that strongly connected component S_j.
6. Let
$$E^\downarrow := \{a \prec b : a \notin P^\circ, b \in P^\circ\} \subseteq E$$
be the links entering an SCC, but not leaving one.
7. Let
$$E^\uparrow := \{a \prec b : a \in P^\circ, b \notin P^\circ)\} \subseteq E$$
be the links leaving an SCC, but not entering one.
8. Let
$$E^- := \{a \prec b : \exists S_j, S_{j'} : a \in S_j, b \in S_{j'}\} \subseteq E$$
be the links leaving one SCC and entering another.
9. Let $E' := E \setminus (E^\uparrow \cup E^\downarrow \cup E^-)$.
10. For each edge $a \prec b \in E^\downarrow$, insert into E' an edge $a \prec a_j$, where $b \in S_j$.
11. For each edge $a \prec b \in E^\uparrow$, insert into E' an edge $a_j \prec b$, where $a \in S_j$.
12. For each edge $a \prec b \in E^\circ$, insert into E' an edge $a_j \prec a_{j'}$, where $a \in S_j, b \in S_{j'}$.
13. Output $\mathcal{D} = \langle P', E' \rangle$.

The DAG produced from `cyclic_closure_constructor`(`transitive_reduction`$(\mathcal{G})$) is shown in Fig. 3.5, where $X = \{F, C, T\}, Y = \{R, P, S, N\}$. Note that $|$`cyclic_closure_constructor`$(\mathcal{G})| \leq |\mathcal{G}|$.

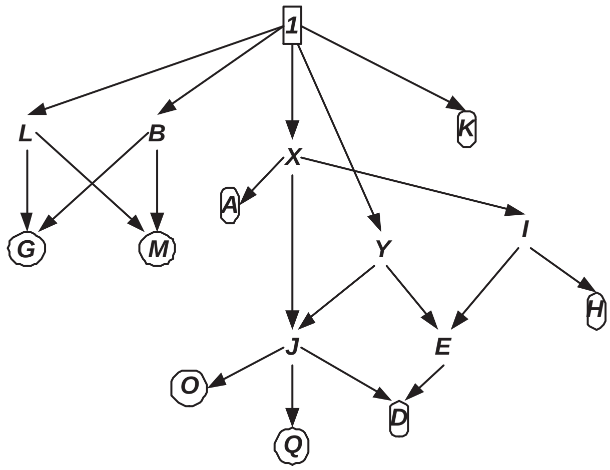

Figure 3.5: DAG $\mathcal{D} = \texttt{cyclic\_closure\_constructor}(\texttt{transitive\_reduction}(\mathcal{G}))$.

*$\mathcal{D} = \texttt{union\_constructor}(\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_n)$**:** Return $\langle \bigcup_{i=1}^n P_i, \bigcup_{i=1}^n E_i \rangle$. Denote $\bigcup_{i=1}^n \mathcal{D}_i$. No example needed, but note that $\mathrm{Max}\left(\bigcup_{i=1}^n \mathcal{D}_i\right) = \bigcup_{i=1}^n \mathrm{Max}(\mathcal{D}_i), \mathrm{Min}(\bigcup_{i=1}^n \mathcal{D}_i) = \bigcup_{i=1}^n \mathrm{Min}(\mathcal{D}_i)$.

bool $=$ `below`(a, b)**:** Returns `true` if `is_connected`(a, b). Denote $a \leq b, b \geq a$. $D \leq X$.

bool $=$ `comparable`(a, b)**:** Returns `true` if $a \leq b$ or $a \geq b$. Denote $a \sim b$. $D \sim X, I \sim D$.

bool $=$ `non_comparable`(a, b)**:** Returns `true` if not `comparable`(a, b). Denote $a \not\sim b$. $J \not\sim E$.

$P' =$ `downset`(a)**:** Returns $\{b : b \leq a\}$. Denote $P' = \downarrow a = \downarrow(a)$. $\downarrow J = \{J, O, Q, D\}$.

$P' =$ `upset`(a)**:** Returns $\{b : b \geq a\}$. Denote $P' = \uparrow a = \uparrow(a)$. $\uparrow J = \{1, X, Y, J\}$.

$P' =$ `hourglass`(a)**:** Returns $\uparrow a \cup \downarrow a$. Denote $\Xi(a)$. $\Xi(J) = \{J, O, Q, D, 1, X, Y\}$.

$P' =$ `lower_bounds`$(\mathcal{D})$**:** Returns $\mathrm{Min}(P)$. $\mathrm{Min}(\mathcal{D}) = \{G, M, K, A, H, O, D, Q\}$

bool $=$ `is_lower_bounded`$(\mathcal{D})$**:** Returns `true` if $|\mathrm{Min}(\mathcal{D})| = 1$. Denote $0 \in \mathcal{D}$. $0 \notin \mathcal{D}$.

$\mathcal{D}' =$ `make_lower_bounded`$(\mathcal{D})$**:** Construct $\mathcal{D}'$ by the algorithm:

1. Let $\mathcal{D}' := \mathcal{D}$.
2. If `is_lower_bounded`$(\mathcal{D})$ then return.
3. Let $P' := P' \cup 0$.
4. For each node $a \in \mathrm{Min}(\mathcal{D})$ insert into E' the link $0 \prec a$.

$P' =$ `upper_bounds`$(\mathcal{D})$**:** Returns $\mathrm{Max}(\mathcal{D})$. $\mathrm{Max}(\mathcal{D}) = \{1\}$.

bool $=$ `is_upper_bounded`$(\mathcal{D})$**:** Returns `true` if $|\mathrm{Max}(\mathcal{D})| = 1$. Denote $1 \in \mathcal{D}$. $1 \in \mathcal{D}$.

$\mathcal{D}' =$ `make_upper_bounded`$(\mathcal{D})$**:** Construct $\mathcal{D}'$ by the algorithm:

1. Let $\mathcal{D}' := \mathcal{D}$.

2. If $\texttt{is\_upper\_bounded}(\mathcal{D})$ then return.
3. Let $P' := P' \cup 1$.
4. For each node $a \in \text{Max}(\mathcal{D})$ insert into E' the link $a \prec 1$.

bool = $\texttt{is\_bounded}(\mathcal{D})$**:** Returns $\texttt{true}$ if $\texttt{is\_upper\_bounded}(\mathcal{D})$ and $\texttt{is\_lower\_bounded}(\mathcal{D})$. $\texttt{is\_bounded}(\mathcal{D})$=fa

$\mathcal{D}'$ = $\texttt{make\_bounded}(\mathcal{D})$**:** Return $\texttt{make\_upper\_bounded(make\_lower\_bounded(}\mathcal{D}\texttt{))}$. $\texttt{make\_bounded}(\mathcal{D})$ is shown in Fig. 3.6. In the sequeal, let $\mathcal{D}'$ =$\texttt{make\_bounded}(\mathcal{D})$.

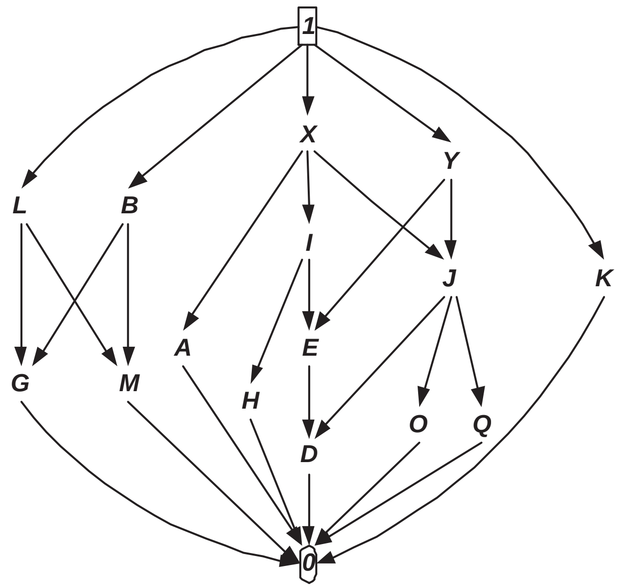

Figure 3.6: $\mathcal{D}'$ =$\texttt{make\_bounded}(\mathcal{D})$.

$\mathcal{G}'$ = $\texttt{transitive\_reduction}(\mathcal{G})$**:** Returns the transitive reduction of $\mathcal{G}$. Denote $\mathcal{V}(\mathcal{G})$. In Fig. 3.4, for the digraph $\mathcal{D}$ shown on top, $\mathcal{V}(\mathcal{D})$ is shown on the left.

bool = $\texttt{is\_transitively\_reduced}(\mathcal{G})$**:** Returns $\texttt{true}$ if $\mathcal{V}(\mathcal{G}) = \mathcal{G}$. For any digraph $\mathcal{G}$, $\texttt{is\_transitively\_reduced}(\mathcal{V}(\mathcal{G})) = \texttt{true}$.

real = $\texttt{transitivity\_degree}(\mathcal{D})$**:** Return

$$TR(\mathcal{D}) := \frac{|\mathcal{D} \setminus \mathcal{V}(\mathcal{D})|}{|\mathcal{P}(\mathcal{D}) \setminus \mathcal{V}(\mathcal{D})|}.$$

In Fig. 3.4, we have $|\mathcal{D} \setminus \mathcal{V}(\mathcal{D})| = 2, |\mathcal{P}(\mathcal{D}) \setminus \mathcal{V}(\mathcal{D})| = 9, TR(\mathcal{D}) = 2/9$.

real = $\texttt{upper\_additive}(a)$**:** Return $F^*(a) := \sum_{b \geq a} w_P(a)$. Note that for unit node weighting, $F^*(a) = |\uparrow a|$. For unit node weighting, $F^*(J) = |\uparrow J| = |\{1, X, Y, J\}| = 4$.

real = $\texttt{lower\_additive}(a)$**:** Return $F_*(a) := \sum_{b \leq a} w_P(a)$. Note that for unit node weighting, $F_*(a) = |\downarrow a|$. For unit node weighting, $F_*(J) = |\downarrow J| = |\{0, O, Q, D, J\}| = 5$.

P' = $\texttt{upper\_cone}(a, b)$**:** Returns $\uparrow a \cap \uparrow b$. Denote $a \nabla b$. $H \nabla J = \{1, X\}$.

$P' =$ `join`(a, b)**:** Returns $\mathrm{Min}(a \nabla b)$. Denote $a \vee b$. Note that $a \vee b \neq \emptyset$ necessarily only if `LowerBounded`$(\mathcal{D})$. $H \vee J = \{X\}$.

$P' =$ `lower_cone`(a, b)**:** Returns $\downarrow a \cap \downarrow b$. Denote $a \Delta b$. $L \Delta B = \{G, M, 0\}$.

$P' =$ `meet`(a, b)**:** Returns $\mathrm{Max}(a \Delta b)$. Denote $a \wedge b$. Note that $a \wedge b \neq \emptyset$ necessarily only if `UpperBounded`$(\mathcal{D})$. $L \wedge B = \{G, M\}$.

$P' =$ `interval`(a, b)**:** If $a \leq b$, returns $\{c \in P : a \leq c \leq b\}$; otherwise returns `null`. Denote $[a, b]$. $[E, 1] = \{E, I, Y, X, 1\}$.

bool $=$ `is_saturated`$(C \subseteq P)$**:** Returns true if $C \subseteq \mathcal{V}(\mathcal{P})$, that is, if C is also a chain in the cover relation. If $C = a_1 \leq a_2 \leq \ldots \leq a_n$ is saturated, then $\forall C' = a_1 \leq \ldots \leq a_n, C' \subseteq C$. Denote $a_1 \prec a_2 \prec \ldots \prec a_n$.

$\{C\} =$ `chains`(a, b)**:** If $a \leq b$, returns $\{C \subseteq [a, b] :$ `saturated`$(C)\}$. Otherwise returns `null`. Denote $\mathcal{C}(a, b)$. $\mathcal{C}(E, 1) = \{E \prec I \prec X \prec 1, E \prec Y \prec 1\}$.

int $=$ `chain_density`(a)**:** Returns $|\mathcal{C}(a, 1)| \times |\mathcal{C}(0, a)|$. `chain_density`$(Y) = 1 \times 4 = 4$.

3.3 Class BoundedDAG : DAG

3.3.1 Objects

BoundedDAG: Let $\mathcal{B} := \mathcal{D}$ be a DAG where `is_bounded`$(\mathcal{D}) =$ `true`.

3.3.2 Methods

3.3.2.1 Vertical Ranks

int $=$ `height`$(\mathcal{B})$**:** Returns $\max_{C \subseteq \mathcal{B}} |C| - 1$. Denote $\mathcal{H}(\mathcal{B})$. In $\mathcal{P}(\mathcal{D})$, the largest path is $0 \prec D \prec E \prec I \prec X \prec 1$, so $\mathcal{H}(\mathcal{D}') = 5$.

int $=$ `top_rank`(a)**:** Returns $\mathcal{H}([a, 1])$. Denote $r^t(a)$. $r^t(Y) = \mathcal{H}([Y, 1]) = |Y \prec 1| - 1 = 1$.

int $=$ `bottom_rank`(a)**:** Returns $\mathcal{H}(\mathcal{B}) - \mathcal{H}([0, b])$. Denote $r^b(a)$.

$$r^b(Y) = \mathcal{H}(\mathcal{D}) - \mathcal{H}([0, Y]) = 5 - \mathcal{H}(\{0, D, O, Q, E, J, Y\}) = 5 - (|0 \prec D \prec J \prec Y| - 1) = 2.$$

real $=$ `mid_rank`(a)**:** Returns $\frac{r^t(a) + r^b(a)}{2}$. `mid_rank`$(Y) = 1.5$

[int,int] $=$ `interval_rank`(a)**:** Returns $[r^t(a), r^b(a)]$. Denote $R(a)$. $R(Y) = [1, 2]$.

int $=$ `rank_width`(a)**:** Returns $r^b(a) - r^t(a)$. `rank_width`$(Y) = 1$.

3.3.2.2 Complementation (not yet implemented, caj)

$P=$ `atoms`$(\mathcal{B})$**:** Returns $\succ(0)$. Denote $\bot(Q)$. `coatoms`$(\mathcal{B})=\{G,M,A,H,D,O,Q,K\}$.

$P=$ `coatoms`$(\mathcal{B})$**:** Returns $\prec(1)$. Denote $\top(Q)$. `atoms`$(\mathcal{B})=\{L,B,X,Y,Q\}$.

$P=$ `complement`(a)**:** Returns

$$P\setminus\left(\bigcup_{b\in\top(\uparrow a)}\downarrow b\right)\cup\left(\bigcup_{b\in\bot(\downarrow a)}\uparrow b\right).$$

Denote $\overline{a}$. $\overline{A}=P\setminus\downarrow X\cup\uparrow A=\{L,G,B,M,Y,K\}$.

3.3.2.3 Boolean-Like Operations

$P=$ `difference`(a,b)**:** Retrns $\mathrm{Max}(a\Delta\overline{b})$.

Implication: $a\rightarrow b:=\mathrm{Min}(a\nabla\overline{b})$

Symmetric Difference: $a\bowtie b:=\mathrm{Min}((a-b)\nabla(b-a))$

3.3.2.4 Node Comparisons

3.3.2.4.1 Distances

real = `upper_distance`(a,b)**:** Return

$$d^*(a,b)=F^*(a)+F^*(b)-2\max_{c\in a\vee b}F^*(c).$$

Under unit weighting, for $\mathcal{D}'$,

$$d^*(H,J)=F^*(H)+F^*(J)-2\max_{c\in H\vee J}F^*(c)=4+4-2\max(F^*(X),F^*(1))=4+4-2\max(2,1)=4.$$

$$d^*(L,B)=F^*(L)+F^*(B)-2\max_{c\in L\vee B}F^*(c)=2+2-2F^*(1)=2+2-2\times 1=2.$$

real = `lower_distance`(a,b)**:** Return

$$d_*(a,b)=F_*(a)+F_*(b)-2\max_{c\in a\wedge b}F_*(c)$$

Under unit weighting, for $\mathcal{D}'$,

$$d_*(H,J)=F_*(H)+F_*(J)-2\max_{c\in H\wedge J}F_*(c)=2+5-2F_*(0)=2+5-2\times 1=5.$$

$$d_*(L,B)=F_*(L)+F_*(B)-2\max_{c\in L\wedge B}F_*(c)=4+4-2\max(F_*(G),F_*(M))=4+4-2\max(2,2)=4.$$

int = `upper_diameter()`**:** Upper diameter is the upper distance between leaf and root.

int = `lower_diameter()`**:** Lower diameter is the lower distance between leaf and root.

3.3.2.4.2 Tversky Measures (not yet implemented, caj)

3.3.2.4.3 Semantic Similarities (not yet implemented, caj)

Resnik Semantic Similarity:

Lin Semantic Similarity:

Jian and Contrath Semantic Similarity:

3.3.2.4.4 Vector Space Model (not yet implemented, caj)

Cosine Measure:

3.3.2.5 Node Set Characterization (not yet implemented, caj)

DAG Width:

3.3.2.5.1 Rank Methods

Top Rank Statistics:

Bottom Rank Statistics:

Rank Width Statistics:

3.3.2.5.2 Metric Methods

Diameter:

Centroid:

Dispersion:

3.3.2.5.3 POSOC Scores

3.3.2.6 Compare Node Sets (not yet implemented, caj)

Hausdorff Distance:

Hierarchical Precision and Recall:

3.4 Class Cover : DAG

3.4.1 Objects

Cover: Let $\mathcal{V} := \mathcal{D}$ be a DAG where `is_transitively_reduced`$(\mathcal{D})$. Denote $\mathcal{P} = \{P, \prec\}$.

3.4.2 Methods

$\mathcal{V}$ = `transitive_reduction_constructor`($\mathcal{D}$)**:** Returns `transitive_reduction(`$\mathcal{D}$`)`.

3.4.2.1 Node Comparisons (not yet implemented, caj)

3.4.2.1.1 Interval Chain Decomposition Methods

3.4.2.1.2 Path-Length Methods

Wu and Palmer:

Hisrt and St. Onge:

3.5 Class Poset : BoundedDAG

3.5.1 Objects

Poset: Let $\mathcal{P} := \mathcal{D}$ be a DAG where `is_transitively_closed`($\mathcal{D}$). Denote $\mathcal{P} = \{P, \leq\}$.

Chain: Denote $C \subseteq \mathcal{P}$ where C is a path in $\mathcal{P}$. Ordering the $a_i \in C$ so that $a_i \leq a_{i+1}, 1 \leq i \leq |C|-1$, then denote $C = \{a, b, \ldots, p_{|C|}\} = a_1 \leq a_2 \leq \ldots \leq a_{|C|}$. $C = \{E, X, 1\} = E \leq X \leq 1$.

Antichain: Denote $A \subseteq \mathcal{P}$ where $\forall a, b \in A, a \not\sim b$. $A = \{H, E, O, Q\}$.

3.5.2 Methods

$\mathcal{P}$ = `transitive_closure_constructor`($\mathcal{D}$)**:** Returns `transitive_closure(`$\mathcal{D}$`)`.

***int** = `width`($\mathcal{P}$)**:** Returns $\max_{A \subseteq \mathcal{D}} |A|$. Denote $\mathcal{W}(\mathcal{P})$ [22]. The maximal antichain is $\{G, M, A, H, D, O, Q, K\}$, so that $\mathcal{W}(\mathcal{D}') = 8$.

3.6 Class Meet Semilattice : Poset (not yet implemented)

3.6.1 Objects

Meet Semilattice: Let $\mathcal{L}^\wedge := \mathcal{P}$ where `is_lower_bounded`($\mathcal{P}$) and $\forall a, b \in P, |a \wedge b| = 1$.

3.6.2 Methods

bool = `is_meet_semilattice(`$\mathcal{P}$`)`**:** Returns `true` if `is_lower_bounded`($\mathcal{P}$) and $\forall a, b \in P, |a \wedge b| = 1$.

c = `meet`(a, b)**:** Returns $c \in P$ for which $a \wedge b = \{c\}$. Note polymorphism with `DAG.meet()`.

3.7 Class Lower Tree : Meet Semilattice (not yet implemented)

3.7.1 Objects

Lower Tree: Let $\mathcal{R}^\wedge := \mathcal{L}^\wedge$ where $\forall a \neq 1 \in P, |{\prec}(a)| = 1$.

3.7.2 Methods

bool = `is_lower_tree(`$\mathcal{L}^\wedge$`)`**:** Returns `true` if $\forall a \neq 0 \in P, |{\prec}(a)| = 1$.

3.8 Class Join Semilattice : Poset (not yet implemented)

3.8.1 Objects

Join Semilattice: Let $\mathcal{L}^\vee := \mathcal{P}$ where `is_upper_bounded`($\mathcal{P}$) and $\forall a, b \in P, |a \vee b| = 1$.

3.8.2 Methods

bool = `is_join_semilattice(`$\mathcal{P}$`)`**:** Returns `true` if `is_upper_bounded`($\mathcal{P}$) and $\forall a, b \in P, |a \vee b| = 1$.

$c =$ `join`(a, b)**:** Returns $c \in P$ for which $a \vee b = \{c\}$. Note polymorphism with `DAG.join()`.

3.9 Class Upper Tree : Join Semilattice (not yet implemented)

3.9.1 Objects

Upper Tree: Let $\mathcal{R}^\vee := \mathcal{L}^\vee$ where $\forall a \in P, |{\succ}(a)| = 1$.

3.9.2 Methods

bool = `is_upper_tree(`$\mathcal{L}^\vee$`)`**:** Returns `true` if $\forall a \in P, |{\succ}(a)| = 1$.

3.10 Class Lattice : Meet Semilattice, Join Semilattice (not yet implemented)

3.10.1 Objects

Lattice: Let $\mathcal{L} := \mathcal{P}$ where `is_meet_semilattice`($\mathcal{P}$) and `is_join_semilattice`($\mathcal{P}$).

3.10.2 Methods

bool = `is_lattice(`$\mathcal{P}$`)`: Returns `true` if `is_meet_semilattice`($\mathcal{P}$) and `is_join_semilattice`($\mathcal{P}$).

$\mathcal{L}$ = `CompletionConstructor`($\mathcal{P}$): Returns $\mathcal{L}$ as the Dedekind-MacNeille completion of $\mathcal{P}$ [8]. Requires `is_bounded`($\mathcal{P}$).

real = `upper_distance`(a, b): Return

$$d^*(a,b) = F^*(a) + F^*(b) - 2F^*(a \vee b)$$

real = `lower_distance`(a, b): Return

$$d_*(a,b) = F_*(a) + F_*(b) - 2F_*(a \wedge b)$$

3.11 Class Concept Lattice : Lattice (not yet implemented)

Chapter 4

Acknowledgements

Valerie Cross at Miami U. of Ohio made substantial contributions to this material. Thanks also to Alex Donaldson, Patrick Paulson, and Joshua Short at PNNL for supporting this work. This work was funded by the Battelle Memorial Institute through the Threat Anticipation Initiative.

Appendix A

Reference Sheets

Class	**Method**	**Notation**
Directed Graph	Digraph	$\mathcal{G} = \langle P, E \rangle$
	Node set	P
	Node	$a \in P$
	Edge set	$E \subseteq P^2$
	Link	$a \prec b = e \in E$
	Parents	$\succ(a) \subseteq P$
	Children	$\prec(a) \subseteq P$
	Weight set	$W = \mathbb{R} \cup \emptyset$
	Node weight function	$w_P \colon P \to W$
	Link weight function	$w_E \colon E \to W$
	Directed path	$\vec{C} = a_1 \prec a_2 \prec \ldots \prec a_n \subseteq \mathcal{G}$
	Symmetric closure	$\mathcal{U}(\mathcal{G}) = \langle P, \mathcal{U}(E) \rangle$
	Undirected path	$\vec{C}^U = a_1 \prec a_2 \succ \ldots \prec a_n \subseteq \mathcal{G}$
	Number of turns	$T(\vec{C}^U) \in \mathcal{W}$
Directed Acyclic Graph	DAG	$\mathcal{D}$
	Downset	$\downarrow a \subseteq P$
	Upset	$\uparrow a \subseteq P$
	Hourglass	$\Xi(a) \subseteq P$
	Roots	$\mathrm{Max}(Q) \subseteq P$
	Leaves	$\mathrm{Min}(Q) \subseteq P$
	Lower bound	$0 \in \mathcal{D}$
	Upper bound	$1 \in \mathcal{D}$
	Upper cone	$a \nabla b \subseteq P$
	Join	$a \vee b \subseteq P$
	Lower cone	$a \Delta b \subseteq P$
	Meet	$a \wedge b \subseteq P$
	Transitive closure	$\mathcal{P}(\mathcal{G})$
	Transitive reduction	$\mathcal{V}(\mathcal{G})$
	One node below another	$a \leq b$
	Interval	$[a, b]$

Class	**Method**	**Notation**
Bounded DAG	Bounded DAG	$\mathcal{B}$
	Height	$\mathcal{H}(\mathcal{B}) \in \mathcal{W}$
	Top rank	$r^t(a) \in \mathcal{W}$
	Bottom rank	$r^b(a) \in \mathcal{W}$
	Interval rank	$R(a)$
	Co-atoms	$\bot(\mathcal{B}) \subseteq P$
	Atoms	$\top(\mathcal{B}) \subseteq P$
	Complement	$\overline{a} \subseteq P, \overline{Q} \subseteq P$
	Upper additive	$F^*(a) \in W$
	Lower additive	$F_*(a) \in W$
	Upper distance	$d^*(a,b) \in \mathcal{W}$
	Lower distance	$d_*(a,b) \in \mathcal{W}$
	Degree of transitivity	$TR(\mathcal{D}) \in [0,1]$
Cover	Cover	$\mathcal{V}$
Poset	Poset	$\mathcal{P}$
	Comparable nodes	$a \sim b$
	Noncomparable nodes	$a \not\sim b$
	Antichain	$A \subseteq \mathcal{P}$
	Width	$\mathcal{W}$
Meet Semilattice	Meet Semilattice	$\mathcal{L}^\wedge$
	Meet	$a \wedge b$
	Chain	$C = a_1 \leq a_2 \leq \ldots \leq a_n \subseteq \mathcal{P}$
	Saturated Chain	$C = a_1 \prec a_2 \prec \ldots \prec a_n \subseteq \mathcal{P}$
	Saturated Chains	$\mathcal{C}(a,b)$
Lower Tree	Lower Tree	$\mathcal{R}^\wedge$
Join Semilattice	Join Semilattice	$\mathcal{L}^\vee$
	Join	$a \vee b$
Upper Tree	Upper Tree	$\mathcal{R}^\vee$
Lattice	Lattice	$\mathcal{L}$

Table A.1: Notation

References

[1] Aho, AV; Garey, MR; and Ullman, JD: (1972) "The Transitive Reduction of a Directed Graph", *SIAM Journal of Computing*, v. **1**:2, pp. 131-137

[2] Ashburner, M; Ball, CA; and Blake, JA et al.: (2000) "Gene Ontology: Tool For the Unification of Biology", *Nature Genetics*, v. **25**:1, pp. 25-29

[3] Berners-Lee, Tim; Hendler, J; and Lassila, O: (2001) "Semantic Web", *Scientific American*, v. **284**:5, pp. 34+

[4] Bernstein, Philip A and Haas, Laura M: (2008) "Information Integration in the Enterprise", *Communications of the ACM*, v. **51**:9, pp. 72-79

[5] Birkhoff, Garrett: (1967) *Lattice Theory*, Am. Math. Soc., Providence RI, 3rd edition

[6] Butanitsky, Alexander and Hirst, Graeme: (2006) "Evaluating WordNet-based Measures of Lexical Semantic Relatedness", *Computational Linguistics*, v. **32**:1, pp. 13-47

[7] Cormen, Thomas T; Leiserson, CE; and Rivest, Ronald L: (1990) *Introduction to Algorithms*, MIT Press, Cambridge MA

[8] Davey, BA and Priestly, HA: (1990) *Introduction to Lattices and Order*, Cambridge UP, Cambridge UK, 2nd Edition

[9] Diestel, Reinhard: (1997) *Graph Theory*, Springer-Verlag, New York

[10] Fellbaum, Christiane, ed.: (1998) *Wordnet: An Electronic Lexical Database*, MIT Press, Cambridge, MA

[11] Ganter, Bernhard and Wille, Rudolf: (1999) *Formal Concept Analysis*, Springer-Verlag

[12] Hagberg, Aric A; Schult, Daniel A; and Swart, Peter J: (2008) "Exploring Network Structure, Dynamics, and Function Using NetworkX", in: *Proc. 7th Python in Science Conf. (SciPy2008)*, ed. G Varoquaux et al., pp. 11-15

[13] Joslyn, Cliff: (2004) "Poset Ontologies and Concept Lattices as Semantic Hierarchies", in: *Conceptual Structures at Work, Lecture Notes in Artificial Intelligence*, v. **3127**, ed. Wolff, Pfeiffer and Delugach, pp. 287-302, Springer-Verlag, Berlin, `ftp://ftp.c3.lanl.gov/pub/users/joslyn/iccs04.pdf`

[14] Joslyn, Cliff: (2009) "Hierarchy Analysis of Knowledge Networks", in: *IJCAI Int. Wshop. on Graph Structures for Knowledge Representation and Reasoning*

[15] Joslyn, Cliff; Donaldson, Alex; and Paulson, Patrick: (2008) "Evaluating the Structural Quality of Semantic Hierarchy Alignments", *Int. Semantic Web Conf. (ISWC 08)*, `http://dblp.uni-trier.de/db/conf/semweb/iswc2008p.html#JoslynDP08`

[16] Joslyn, Cliff; Mniszewski, Susan; Fulmer, Andy; and Heaton, G: (2004) "The Gene Ontology Categorizer", *Bioinformatics*, v. **20**:s1, pp. 169-177, `ftp://ftp.c3.lanl.gov/pub/users/joslyn/ismb04.pdf`,

[17] Joslyn, Cliff; Mniszewski, SM; Smith, SA; and Weber, PM: (2006) "SpindleViz: A Three Dimensional, Order Theoretical Visualization Environment for the Gene Ontology", in: *Joint BioLINK and 9th Bio-Ontologies Meeting (JBB 06)*, `http://www.bio-ontologies.org.uk/2006/download/Joslyn2EtAlSpindleviz.pdf`

[18] Kaiser, Tim; Schmidt, Stefan; and Joslyn, Cliff: (2008) "Adjusting Annotated Taxonomies", in: *Int. J. of Foundations of Computer Science*, v. **19**:2, pp. 345-358

[19] Lee, Yugyung and Geller, James: (2002) "Efficient Transitive Closure Reasoning in a Combined Class/Part/Containment Hierarchy", *Knowledge and Information Systems*, v. **4**, pp. 305-328

[20] Lenat, Douglas B; Ramanathan, V Guha; and Pittman, K et al.: (1990) "Cyc: Towards Programming with Common Sense", *Communications of the ACM*, v. **33**:8, pp. 30-49,

[21] Nuutila, Esko and Soesalon, Eljas: (1994) "On Finding the Strongly Connected Components in a Directed Graph", *Information Processing Letters*, v. **49**, pp. 9-14

[22] Obdržálek, Jan: (2006) "DAG-width – Connectivity Measure for Directed Graphs", in: *SODA 06*, pp. 814-821

[23] Schröder, Bernd SW: (2003) *Ordered Sets*, Birkhauser, Boston

[24] Sowa, John F: (2000) *Knowledge Representation: Logical, Philosophical, and Computational Foundations*, Brooks/Cole, Pacific Grove

[25] Trissl, Silke and Leser, Ulf: (2005) "Querying Ontologies in Relational Database Systems", in: *DILS 2005, LNBI*, v. **3615**, pp. 63-79

[26] Trotter, William T: (1992) *Combinatorics and Partially Ordered Sets: Dimension Theory*, Johns Hopkins U Pres, Baltimore

[27] Tversky, Amos: Features of Similarity, Psychological Review, Volume 84, Number 4, pp. 327:352, 1977

[28] Verspoor, KM; Cohn, JD; Mniszewski, SM; and Joslyn, CA: (2006) "A Categorization Approach to Automated Ontological Function Annotation", *Protein Science*, v. **15**, pp. 1544-1549

[29] Zhao, Huimin: (2007) "Semantic Matching Across Heterogeneous Data Sources", *Communications of the ACM*, v. **50**:1, pp. 45-50