Conceptual Model of Data and Information for Conservation Decision Making

Author: Brian Wee (ORCID: 0000-0002-0038-9381)

Organizational affiliations: Founder and Managing Director, Massive Connections, LLC

Date released: 2021-03-11

Version: 1.0

Citation: Wee, Brian (2021): Conceptual Model of Data and Information for Conservation Decision Making using the Neo4j labeled property graph database. figshare. Online resource. https://doi.org/10.6084/m9.figshare.14199767.v2

Team members:
Jennifer Burnett (ORCID: 0000-0002-0896-5099)
Scott Anderson (ORCID: 0000-0002-8580-5582)
William Teng (ORCID: 0000-0001-5642-6359)
Steve Aulenbach (ORCID: 0000-0002-0172-6538)
Rustem ‘Arif’ Albayrak (ORCID: 0000-0001-6060-9844)

ESIP Lab project title: Modeling data and information needs for avian conservation using Neo4j.

Acknowledgment: This work is based on team collaboration tools provided by the ESIP Lab with support from the National Aeronautics and Space Administration (NASA), National Oceanic and Atmospheric Administration (NOAA), and the United States Geologic Survey (USGS).

Citation BibTeX: @misc{wee_2021, title={Conceptual Model of Data and Information for Conservation Decision Making using the Neo4j labeled property graph database}, url={https://figshare.com/articles/online_resource/Conceptual_Model_of_Data_and_Information_for_Conservation_Decision_Making_using_the_Neo4j_labeled_property_graph_database/14199767/2}, DOI={10.6084/m9.figshare.14199767.v2}, publisher={figshare}, author={Wee, Brian}, year={2021}, month={Mar} }

Table of Contents

Table of Figures

1.0 Introduction

1.1 The Challenge

Conservation organizations set priorities as a balance of their overall mission and the larger conservation needs. As we craft conservation strategies across ever-larger landscape scales, we need to understand and incorporate conservation priorities that span different organizations, geographies, management plans, and jurisdictional boundaries.

An integrative assessment of the linkages among organizational conservation priorities provides opportunities for effective planning to increase the impact of conservation investments. This project is an experiment in that type of integrative assessment using the graph database Neo4j.

1.2 Long-term Objectives

Several long-term objectives undergird this ESIP lab project. We would ultimately like to:

- Provide a framework for connecting conservation priorities within and across conservation-relevant scales.
- Streamline and improve the FAIRness (Findability, Accessibility, Interoperability, and Reuse of digital assets) of the processes for populating and writing SWAPs (State Wildlife Action Plans).
- Synthesize data and information sources to populate locally- and contextually-relevant tools, like fact sheets, or species summary documents.
- Use a database as an exploratory research tool for identifying temporal and spatial patterns in conservation management and planning activities

1.3 Immediate Project Objectives

The objectives include:

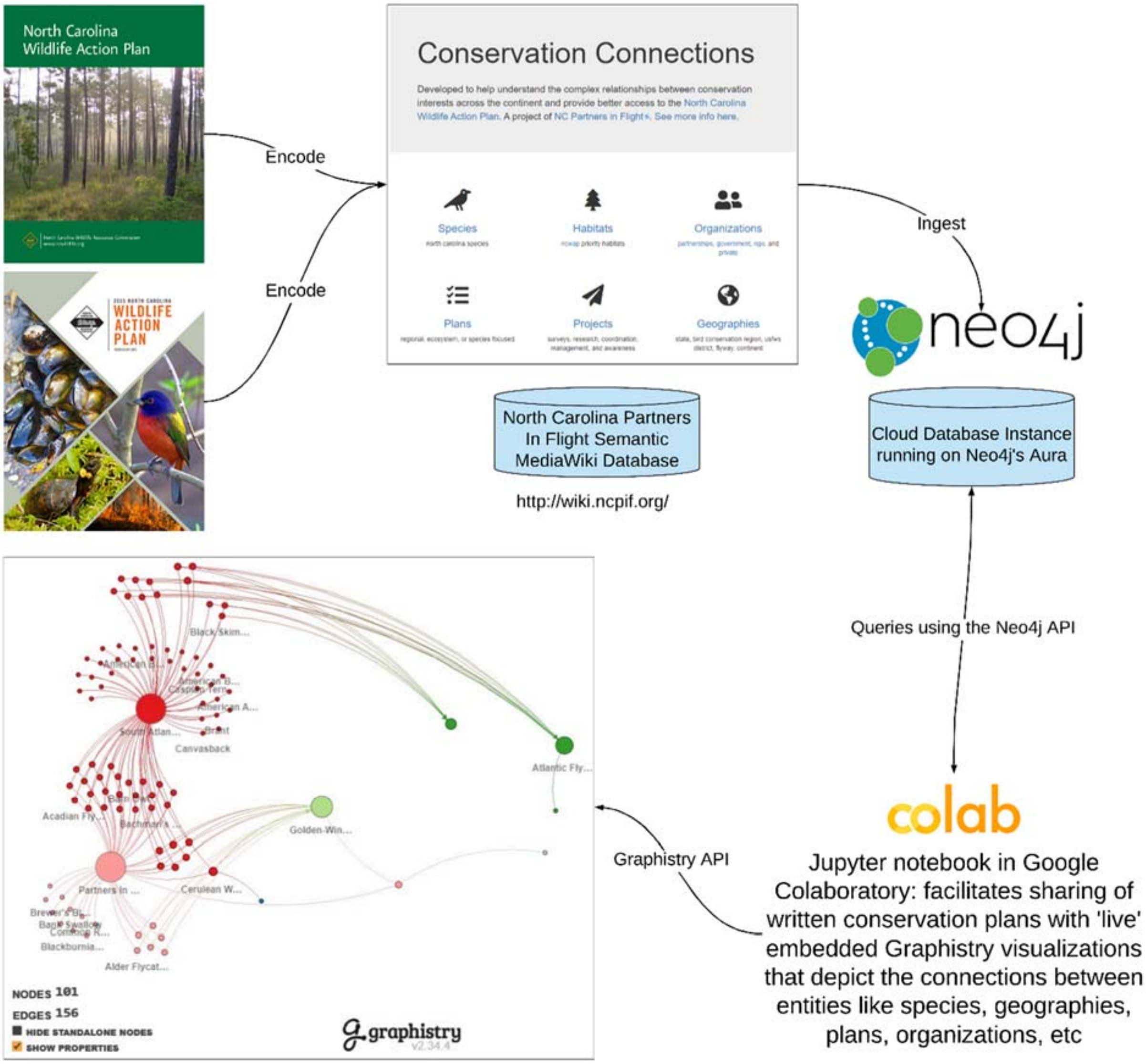
- Explore the use of graph databases for connecting data, information, and knowledge relevant to conservation planning and management activities.
- Deliver a prototype that provides a way for both data scientists and conservation practitioners to work together in real-time and in a collaborative environment, where management decisions are supported by data-driven visualization.

1.4 Conceptual Approach

Figure 1 shows the conceptual approach adopted for this project.

Structured data characterizing selected avian conservation aspects of North Carolina's Wildlife Action Plans were already encoded in a Semantic MediaWiki database (http://wiki.ncpif.org/). That database was created, and is maintained by, the North Carolina Partners in Flight (NC PIF) program, which is a program of the North Carolina Wildlife Resources Commission.

The NC PIF wiki database was ported into a Neo4j labeled property graph database. A Jupyter notebook was used to execute queries against the Neo4j database. Query results, embedded within the notebook, were visualized through a Graphistry cloud service API. The vision is to use Jupyter notebooks as intermediate work-products that inform the formulation of SWAPs. Our approach uses Google Colaboratory (Colab), which is a free, cloud-based resource that hosts Jupyter notebooks. Colab notebooks can be shared between members of the team: some focused on drafting notebook sections that focus on strategy and policy for conservation action, while others focused on adjacent sections that visualize connections between conservation entities.



Jupyter notebook in Google Colaboratory: facilitates sharing of written conservation plans with 'live' embedded Graphistry visualizations that depict the connections between entities like species, geographies, plans, organizations, etc

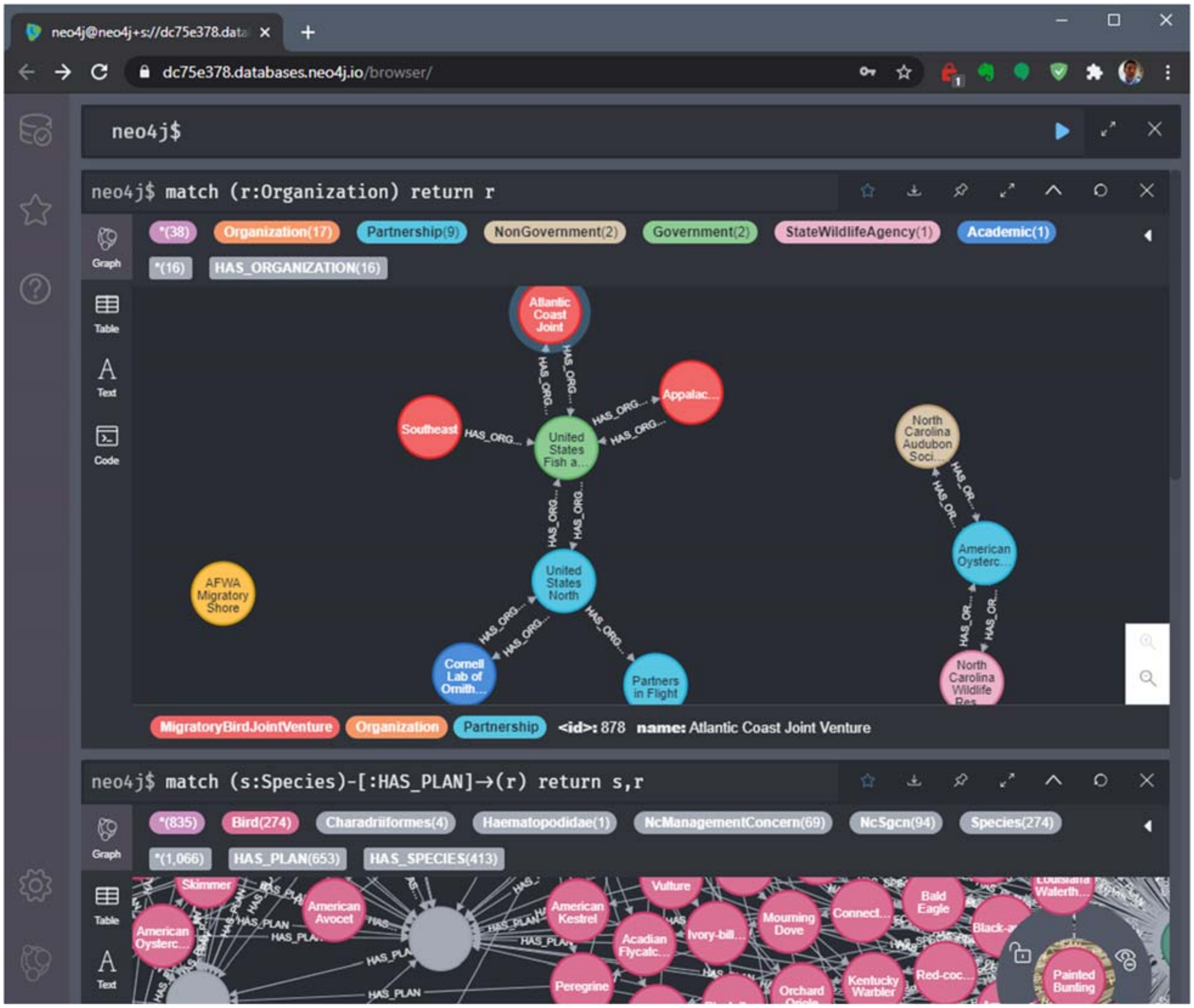
Figure 1: Conceptual approach for modeling and visualizing avian conservation relationship data.

2.0 Technical Platform

2.1 Database

Neo4, a labeled property graph database was selected as the database platform of choice. The reasons for this selection included:

- **Relatively low barrier to experimentation: Sandbox in the cloud.** For individuals who are looking to "test-drive" the database, Neo4j offers a free Neo4j instance in the cloud. The user can get a feel of Neo4j very rapidly without downloading any executables to one's machine.
- **Relatively low barrier to experimentation: Easy to prototype on a local machine.** Neo4j is available as a free download for a desktop machine running MacOS 10.10 (Yosemite)+, Windows 7+, Ubuntu 12.04+, Fedora 21, or Debian 8. The download comprises the database server and Neo4j apps. Interacting with the Neo4j server is accomplished via a web browser interface. Although the interface requires familiarity with the Neo4j database query language Cypher, the interface is relatively easy to use, and supports visualizing query results from within a browser interface, as shown below.



```
neo4j$ match (r:Organization) return r
```

```
neo4j$ match (s:Species)-[:HAS_PLAN]→(r) return s,r
```

MigratoryBirdJointVenture Organization Partnership <id>: 878 name: Atlantic Coast Joint Venture

Figure 2: Browser-based Cypher query interface to the Neo4j database server.

- **Relatively low barrier to experimentation: Visual interface for exploring data.** The Neo4j app, Bloom, is a powerful visual interface for exploring a graph database. It supports a near-natural-language query mode. The Bloom interface runs within a browser and can be used to make minor edits to the database.

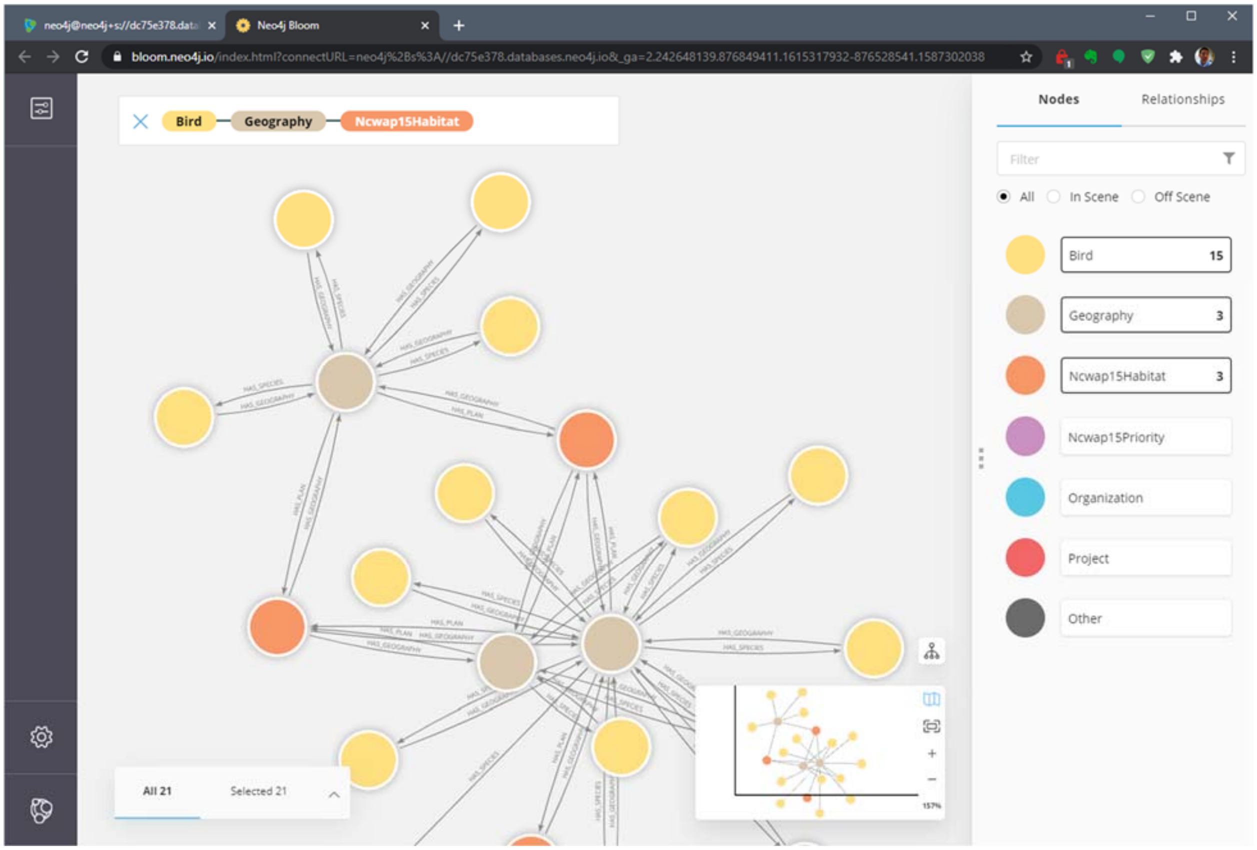


Figure 3: Browser-based Bloom query interface to the Neo4j database server.

- **Cloud database instances on Neo4j Aura.** Aura is Neo4j's cloud-hosted database platform. Although we eventually installed a free version of Neo4j on an Ubuntu machine running on a virtual machine on a Network Attached Storage (NAS) server, having access to the Neo4j-managed Aura instance alleviates the need to maintain the database server. We initially purchased a low-capacity Aura instance costing approximately $70 per month. Neo4j ultimately offered us a free instance as this project drew to a close. We intend to continue experimenting using this free Aura instance for as long as Neo4j offers it.

2.2 Collaboration Platform

We used Jupyter Python notebooks hosted on the free Google Colaboratory (Colab) cloud platform to prototype an online, collaborative, work-product that members of a SWAP team can use to write an action plan for bird conservation.

More generally, we envision such notebooks as a useful mechanism to write human-readable documents that incorporate graph visualizations. This is fully concordant with the broader goals of reproducible and traceable intellectual products that include provenance information.

Colab notebooks can be shared between members of the team. Some team members would be focused on drafting notebook sections that focus on strategy and policy for conservation action. Other members will be focused on adjacent sections that support those strategies and policies with graph visualizations.

The APIs used in our Python code were:

- Neo4j's APIs for querying the Neo4j database server (https://neo4j.com/docs/api/python-driver/current/).
- Graphistry's (https://www.graphistry.com/) APIs for plotting graphs based on data returned by the Neo4j database server and reformatted for Graphistry. Graphistry is an open-source cloud-based service that offers Python APIs for the visualization of graph data. Data is sent by the Python program to the Graphistry server for rendering, after which the graph (vector) objects are displayed within the Python notebook. The rendered objects can also be displayed in-line via an iframe in an HTML webpage.

The documentation for the Graphistry APIs would benefit from the API author(s) being more descriptive. There is a general lack of details with assumptions not explicitly specified, requiring copious amounts of development by trial-and-error.

Several Python graph visualization libraries were assessed for use, but ultimately not selected because of various shortcomings. These libraries include:

- py2neo (https://py2neo.org/)
- neo4jupyter (https://github.com/merqurio/neo4jupyter)
- Py3Plex (https://github.com/SkBlaz/Py3plex)

3.0 Neo4j Graph Schemas

The Neo4j graph schema is depicted in Figure 4, Figure 5, and Figure 6.

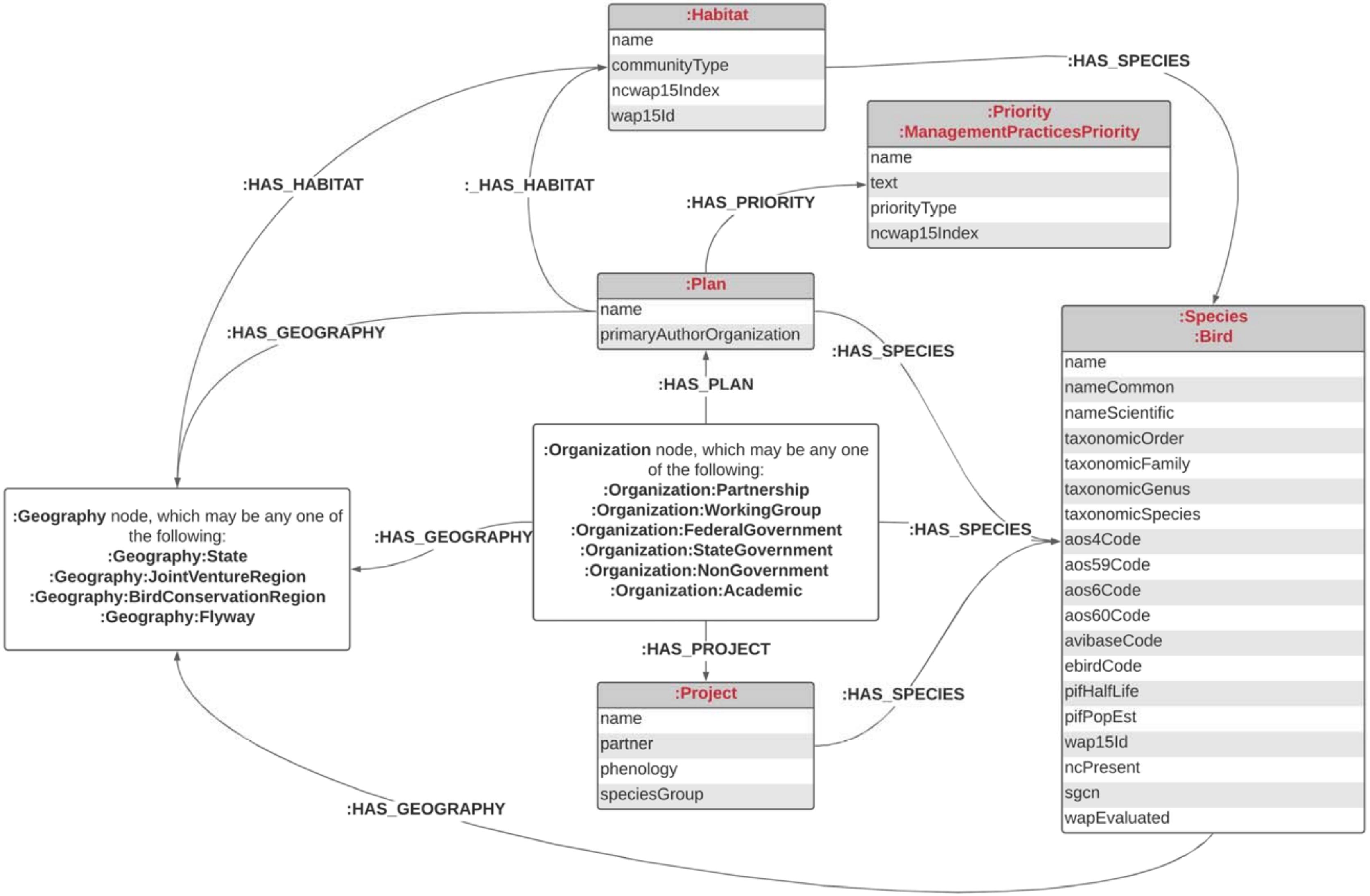


Figure 4: Top level Neo4j schema.

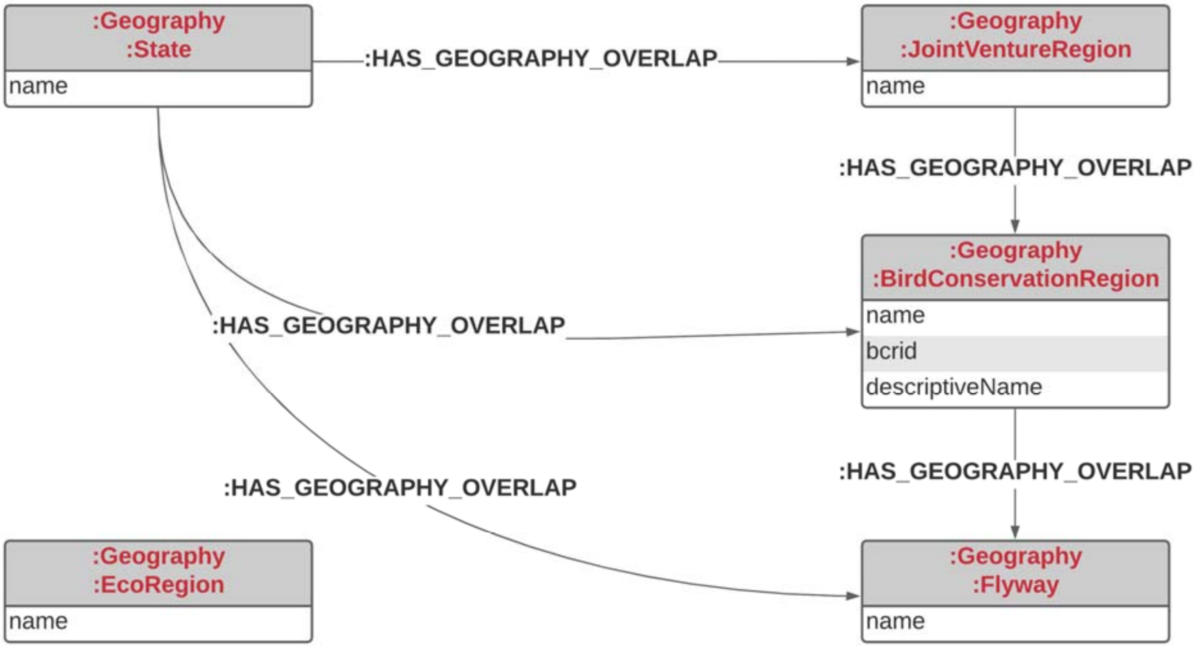


Figure 5: Neo4j schema for nodes with the :Geography label.

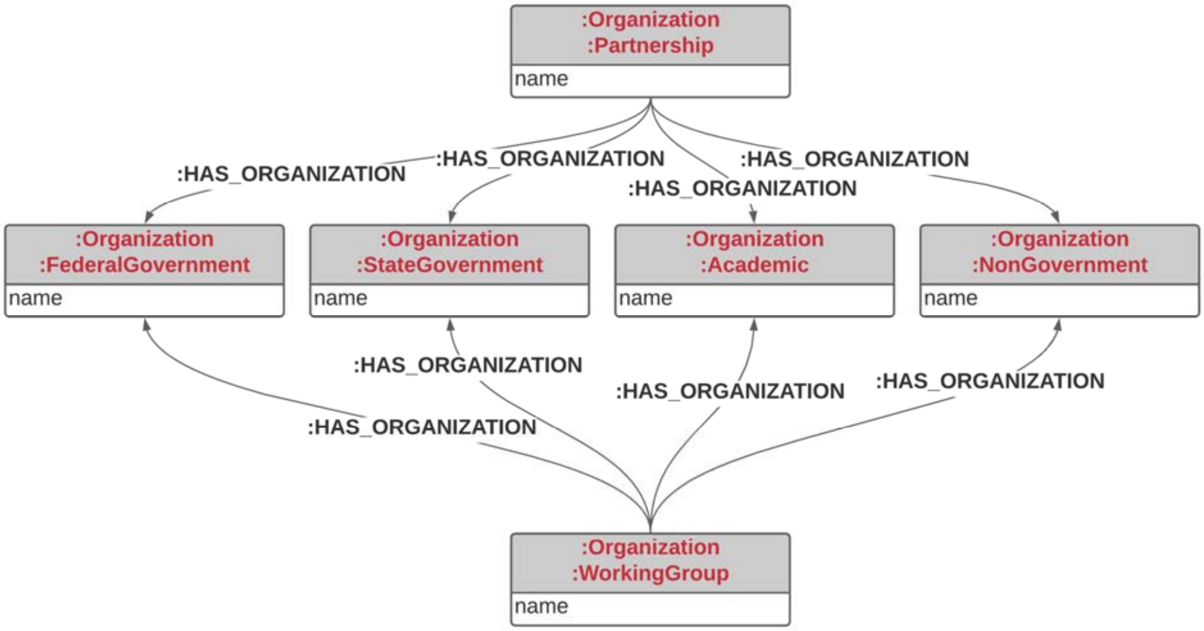


Figure 6: Neo4j schema for nodes with the :Organization label.

Neo4j's Cypher query language syntax is used in this document. The syntax is:

- Neo4j nodes are written as camel-case expressions that begin with an upper-case character, are preceded by a colon, and embedded in parentheses, for example:

  (:Bird)
- Neo4j relationships as written as upper-case expressions that are preceded by a colon and embedded in square brackets, for example:

  [:HAS_GEOGRAPHY]

The schema is based largely on the North Carolina Partners in Flight wiki database. We took advantage of Neo4j capabilities when designing the schema. Notable schema design decisions are noted in the following subsections.

3.1 Multiple labels for each graph node

Neo4j allows each node to be associated with one or multiple labels. Examples of labels in Figure 4 include ":Habitat" (a singular label) and ":Species:Bird" (two labels). The ability to attach multiple labels to a given node makes it possible to include (:Species:Amphibian), (:Species:Fish), (:Species:Mammals), or (:Species:Reptile) nodes in the future.

3.2 Unidirectional relationships

Consider the following example:

(:Bird:Species)-[:HAS_GEOGRAPHY]->(:Geography:State)

The Neo4j database currently includes symmetrical relationships in the form of:

(:Geography:State)-[:HAS_GEOGRAPHY]->(:Bird:Species)

The latter symmetrical relationship is unnecessary and not depicted in the following pages, even though the symmetrical relationships exist in the Neo4j database. Symmetrical relationships need not be explicitly asserted in this particular (and other analogous) case because:

(1) Given any (:Geography:State) node, the Neo4j Cypher query is capable of retrieving a (:Bird:Species) node. This is despite the cardinality (direction) of the [:HAS_GEOGRAPHY] relationship being explicitly asserted that originates from (:Bird:Species) to (:Geography:State). The symmetrical relationship is therefore unnecessary and makes it harder to ensure that the database exhibits relational integrity. That is, over time as the database evolves, it is harder to ascertain that there is a [:HAS_GEOGRAPHY] relationship going in both directions for every pair of nodes that is related by the [:HAS_GEOGRAPHY] relationship.
(2) Conceptually, one might state, as a narrative, that "a bird has a certain geography that is this US state), but not "this US state has a geography that is this bird".

The symmetrical relationships are not currently used in Neo4j Cypher queries. To ease maintenance overheads, these relationships will be removed in the next major version of the Neo4j database.

3.3 Inferencing geographical relationships

Figure 7 shows selected south-eastern Bird Conservation Regions (BCRs) overlaid on US states

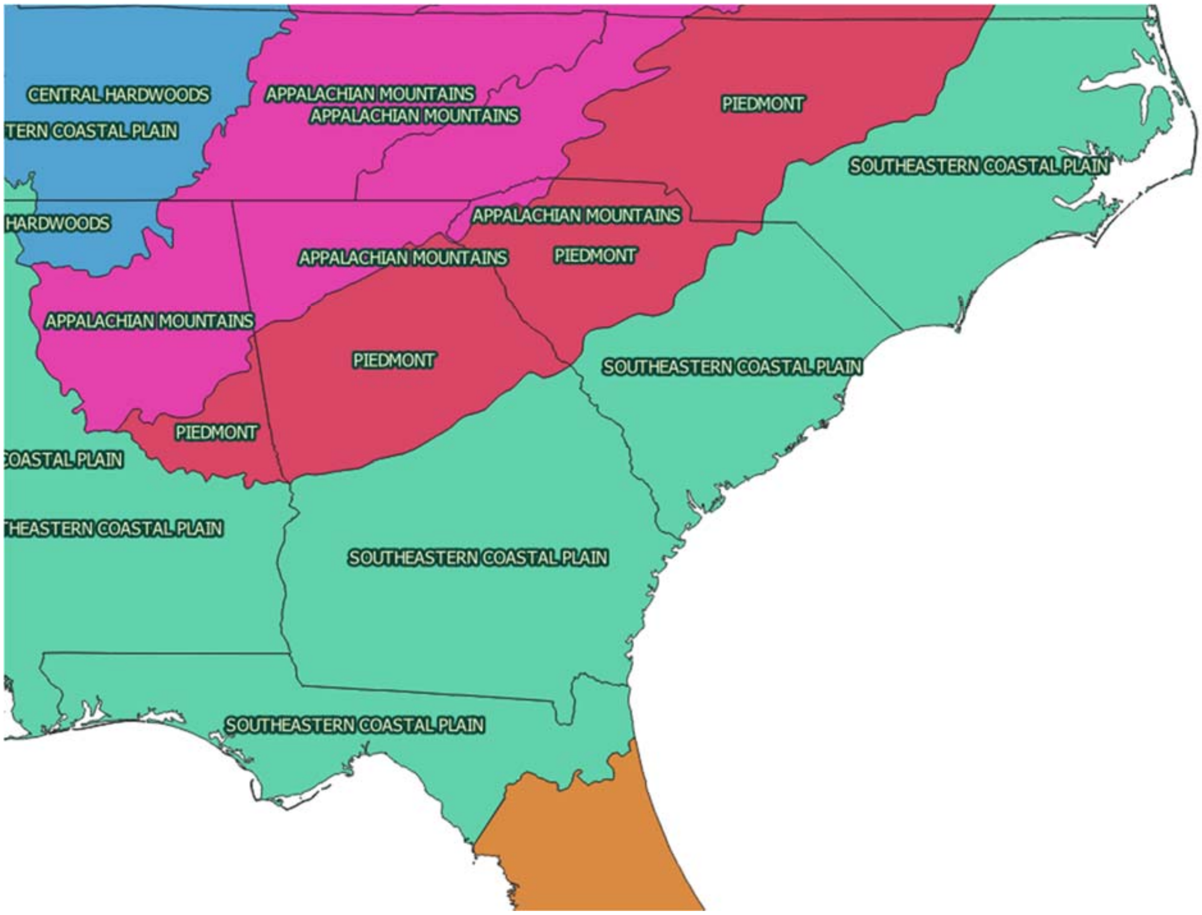


Figure 7: Selected south-eastern Bird Conservation Regions (BCRs) overlaid on US states.

Figure 5 shows the following relationship that is currently encoded in the Neo4j database:

(:Geography:State)-[:HAS_GEOGRAPHY_OVERLAP]->(:Geography:BirdConservationRegion)

Given any bird species that is found in a US state, is it easy to ask Neo4j to return all the BCRs that intersect with that state through the [:HAS_GEOGRAPHY_OVERLAP] relationship. Similarly, given any bird species that is found in a BCR, one can query Neo4j to return all the US states that intersect with the geography of the BCR.

That is about the extent to which any geographical relationships can be inferred. See Figure 8. If Geography_A overlaps with Geography_B, and Geography_B overlaps with Geography_C, one cannot infer that there is necessarily overlap between Geography_A and Geography_C.

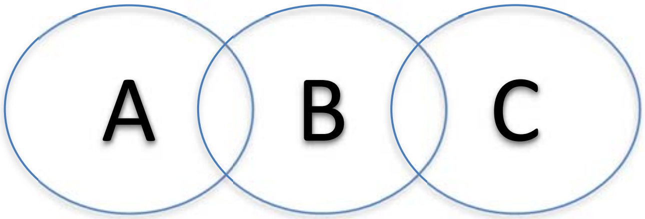
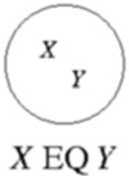
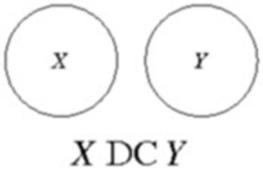


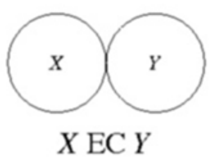
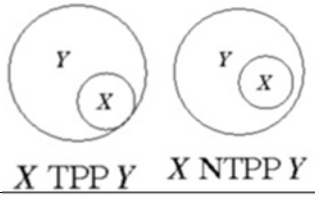
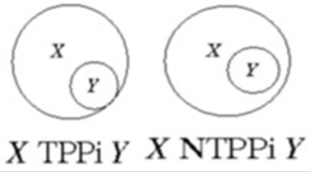
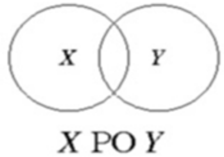
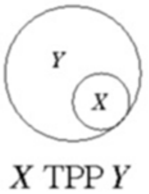
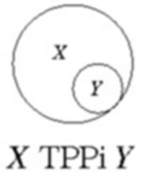
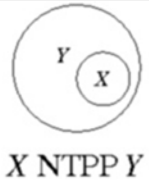
Figure 8: A simple example of what cannot be inferred from overlapping geographies.

Other than the [:HAS_GEOGRAPHY_OVERLAP] relationship, what other topographical relationships can we use to encode information in the Neo4j database to enhance our ability to infer bird conservation actions that are not explicitly asserted in the database? We consulted with three different sources to ascertain what those other relationships may be:

(1) Open Geospatial Consortium's "GeoSPARQL - A Geographic Query Language for RDF Data" (document # 11-052r4).
(2) Wikipedia's "Region connection calculus" on https://en.wikipedia.org/wiki/Region_connection_calculus (retrieved 2020-12-14).
(3) Battle, Robert, and Dave Kolas. "Geosparql: enabling a geospatial semantic web." Semantic Web Journal 3.4 (2011): 355-370.

The table below summarizes our findings. At some later juncture, in addition to "overlaps", the "within" and "contains" relationships could be considered for encoding into the Neo4j database.

Simple features (Source: 3)	Egenhofer (Source: 3)	RCC8 code (Source: 3)	RCC8 description (Source: 1)	Diagram (Source: 2)
equals	equal	EQ	equals	
disjoint	disjoint	DC	disconnected	
intersects	~disjoint	~DC		

Simple features (Source: 3)	**Egenhofer (Source: 3)**	**RCC8 code (Source: 3)**	**RCC8 description (Source: 1)**	**Diagram (Source: 2)**
touches	meet	EC		
within	inside + overedBy	NTPP + TPP		
contains	contains + covers	NTPPi + TPPi		
overlaps	overlap	PO	partially overlapping	
crosses				
		TPP	tangential proper part	
		TPPi	tangential proper part inverse	
		NTPP	non-tangential proper part	
		NTPPi	non-tangential proper part inverse