OntoMerger: An Ontology Integration Library for Deduplicating and Connecting Knowledge Graph Nodes

David Geleta
AstraZeneca
United Kingdom

Andriy Nikolov
AstraZeneca
United Kingdom

Mark O’Donoghue
AstraZeneca
United Kingdom

Benedek Rozemberczki
AstraZeneca
United Kingdom

Anna Gogleva
AstraZeneca
United Kingdom

Valentina Tamma
The University of Liverpool
United Kingdom

Terry R. Payne
The University of Liverpool
United Kingdom

ABSTRACT
Duplication of nodes
takes a set of KG nodes, mappings and disconnected hierarchies and generates a set of merged nodes together with a connected hierarchy. In addition, the library provides analytic and data testing functionalities that can be used to fine-tune the inputs, further reducing duplication, and to increase connectivity of the output graph. OntoMerger can be applied to a wide variety of ontologies and KGs. In this paper we introduce OntoMerger and illustrate its functionality on a real-world biomedical KG.

ACM Reference Format:
David Geleta, Andriy Nikolov, Mark O’Donoghue, Benedek Rozemberczki, Anna Gogleva, Valentina Tamma, and Terry R. Payne. 2022. OntoMerger: An Ontology Integration Library for Deduplicating and Connecting Knowledge Graph Nodes. In Proceedings of ACM Conference (Conference’17). ACM, New York, NY, USA, 5 pages.

1 INTRODUCTION
Complex knowledge graphs (KGs) [6, 21] are often built from multiple overlapping but heterogeneous data sets. For example, a biomedical KG can include separate data sets providing information about genes, diseases, compounds and relations between these entities [1, 3, 6, 10, 28, 31]. Typically, these data sets rely on diverse ontologies to refer to the same concepts with no obvious mapping or alignment between them [2]. Furthermore, data version discrepancy can also lead to duplication when concepts get deprecated and identifiers reassigned. On a large scale, these issues may cause syntax and semantic heterogeneity in the resulting graph; which in turn can lead to fragmented knowledge that can negatively impact downstream applications [9] due to possible incompatibilities between the hierarchies, that in turn may lead to logical inconsistencies. Thus, organising concepts of the same domain into a hierarchy [29] might improve the accuracy of machine learning models, e.g. pattern recognition.

OntoMerger tackles these challenges by using a scaffolding ontology, together with mappings and concept hierarchies, to produce an integrated domain ontology with a single DAG (Directed Acyclic Graph) hierarchy. By design, this ontology is not intended to retain complete knowledge for all the integrated concepts (such as hierarchical connections and the granularity of the original conceptualisation), but rather relies on additional mappings provided by the users, that are then used to infer new ones. The library is open source, implemented in Python [27], and is available to download as a package via PyPi. The library processes data in tabular format using Pandas [16]. OntoMerger produces the hierarchy by converting hierarchy triples into graph objects and by running queries for finding paths (using NetworKit [23]), as well as for counting sub-graphs (using NetworkX [8]). To ensure stable performance, input and output data is tested using the Great Expectations [7] validation framework. In addition, the library provides an in-depth.
analysis by using the the Pandas Profiling package\(^4\) to profile all the data that is consumed and produced. Insights about the (i) input, (ii) intermediate and produced data (i.e. the domain ontology), (iii) results of data profiling and data validation, and (iv) metrics about the inner workings of the library are presented in an HTML report, shown in Figure 1.

2 RELATED WORK

Ontology integration has been the focus of the research community for several years [17]. Whilst ontology matching [4] focuses on the automated alignment, or generation of sets of mappings between the corresponding elements of different ontologies, the use of these mappings to produce a coherent, integrated ontology is non-trivial. This is due to the need to satisfy merge requirements [19] such as: preserving correctness and coherence; maintaining acyclicity; and avoiding redundancy. To satisfy these requirements, existing algorithms either modify the generated alignments [13, 22, 24, 26] or the ontologies themselves [5, 12, 20].

Despite the significant effort invested in the ontology integration problem, existing methods do not always satisfy the requirements of real-world use cases in the context of large scale data science, which can compromises their pragmatic use. For example, existing systems usually rely on OWL semantics and on the explicit constraints defined in source ontologies. However, implicit constraints may also exist; for example, two concepts in the same ontology may not be explicitly declared as disjoint, but this may be implicitly desirable to avoid them being merged as a result of ontology integration.

3 PRELIMINARIES

In this paper, we assume familiarity with basic notions of RDF [15], ontology alignment and integration [4].

**Definition 1 (Concept Name Set).** The symbol \(C\) denotes a set of concept names \(C = \{c_1, ..., c_n\}\) where \(|C| = n\). A set of concept names from the same ontology is denoted as \(C_O\).

**Definition 2 (Mapping and Mapping Set).** A concept mapping is a triple \(m = \langle c, r, c' \rangle\), asserting that some relation \(r\) holds between concepts \(c\) and \(c'\), where either \(\langle c, c' \rangle \in C_O\), or \(c \in C_O\) and \(c' \in C_O'\), given the ontologies \(O\) and \(O'\). The symbol \(M\) denotes a set of concept mappings \(M = \{m_1, ..., m_n\}\), \(|M| = n\).

The semantics of a mapping relation \(r\) may vary due to the diversity of input mappings that originate from different sources in real world scenarios. The may be formally defined [4]: \(r \in \{\equiv, \subseteq, \top, \bot\}\), or may be loosely defined, such as \(r \in \{\equiv, \subseteq, \top, \bot\}\). The concepts \(c\) and \(c'\) refer to the source and target concepts respectively.

**Definition 3 (Hierarchy Edge and Hierarchy Edge Set).** Given two ontologies, \(O\) and \(O'\), a concept hierarchy edge is a triple \(h = \langle c, c' \rangle\), which asserts that a subsumption relation \(\subseteq\) holds between concepts \(c\) and \(c'\), where either \(\langle c, c' \rangle \in O\), or \(c \in O\) and \(c' \in O'\). The symbol \(H\) denotes the concept hierarchy edge set \(H = \{h_1, ..., h_n\}\), \(|H| = n\).

**Definition 4 (Merge and Merge Set).** A concept merge is a triple \(x = \langle c, \Rightarrow, c' \rangle\), asserting that a concept \(c\) can be replaced by \(c'\) in a concept, mapping or in a hierarchy edge set. The symbol \(X\) denotes the set of merges \(X = \{x_1, ..., x_n\}\), with \(|X| = n\).

**Definition 5 (Stable Merge Set).** A merge set is \(X\) is stable if a concept only appears in one merge \(x \in X\) and only as a source concept.

In the definitions of mapping, hierarchy edge, and merge triples, we refer to \(c\) as the source and \(c'\) as the target concept. The function \(\text{Sig}\) returns the source (\(\text{Sig}_\text{source}\)), target (\(\text{Sig}_\text{target}\)) or both (\(\text{Sig}\)) set of concept names from either an individual mapping or from a set of mappings, merges, hierarchy edges or paths.

4 INTEGRATION DESIGN

The integration design process requires the following input sets: concept names \(C_{in}\), obsolete (i.e. deprecated) concept names \(C_{obs}\), mappings \(M_{in}\), and hierarchy edges \(H_{in}\), where each set may contain members from different ontologies. In addition, a configuration must specify the concept names in a \(\text{SEED}\) ontology, such that \(C_{\text{SEED in}} \subseteq C_{in}\) and \(H_{\text{SEED in}} \subseteq H_{in}\). No other ontology must be fully contained in the input sets other than the \(\text{SEED}\) ontology.

The optional preference order in which the concepts are de-duplicated and connected (denoted as \(\text{SOURCES}\)) can also be specified. If it is omitted, it will be computed by taking the frequency of each ontology concept set of \(C_{in}\) in decreasing order. In either case, the \(\text{SEED}\) ontology will be the first item in \(\text{SOURCES}\). The configuration must also contain the \(\text{MAPPINGGROUPS}\) list that categorises the mapping types of \(M_{in}\) into a minimum of two categories: equivalence and reference (where the equivalence group must have at least one member, and the other groups can be empty sets). These categories represent the strength of the mapping relations and are handled in sequence (in decreasing order starting from the strongest group) during the deduplication stage.

| Input | \(C_{in}\): input concepts; \(C_{obs}\): obsolete concepts of \(C_{in}\); \(M_{in}\): input mappings; \(H_{in}\): input hierarchy edges; \(\text{SEED}\): name of the seed ontology (i.e.\(\text{O}_{\text{SEED}}\)); \(\text{SOURCES}\): preference order; \(\text{MAPPINGTYPES}\): mapping type groups; |
|-------|----------------------------------------------------------|
| Output | \(C_{dom}\): domain concepts; \(M_{dom}\): domain mappings; \(X_{can}\): concept merges with canonical target concepts; \(H_{dom}\): domain hierarchy edge set; |
|       | 1 \(C_{dom}, C_{unm}, X_{can}, M_{dom} \leftarrow \text{DEDUPLICATECONCEPTS}(\) \(C_{in}, C_{obs}, M_{in}, \text{SEED}, \text{SOURCES}, \text{MAPPINGTYPES})\) |
|       | 2 \(H_{dom} \leftarrow \text{CONNECTCONCEPTS}(\) \(C_{unm}, X_{can}, H_{in}, \text{SEED, SOURCES})\) |
|       | 3 return \(C_{dom}, M_{dom}, X_{can} - H_{dom}\) |

**Algorithm 1: ComputeDomainOntology**

The main part of \(\text{OntoMerger}\) produces the domain ontology by first deduplicating concepts (Section 4.1) and then connecting the remaining unmerged concepts (i.e. concepts without an accepted mapping) together to form a single DAG, i.e. concept hierarchy (4.2). This process is formalised by Algorithm 1.
4.1 Deduplication

```
Input : C_in: C_obs; M_in; SEED; SOURCES; MappingTypes
Output: C_dom: C_umn: unmerged concepts; X_can; M_dom:
/* Pre-processing */
1 X_obs ← M_internal ← ComputeObsoleteMerges(M_in, C_obs)
2 M_in′ ← UpdateMappings((M_in \ M_internal), X_obs)
/* Deduplication */
3 X ← X_obs \ ComputeMerges(
    C_in, M_in′, SEED, SOURCES, MappingTypes)
/* Post-processing */
4 X_can ← AggregateMerges(X, SOURCES)
5 C_dom ← ApplyMerges(C_in, X_can)
6 M_dom ← ApplyMerges(M_in, X_can)
7 C_umn ← C_in \ (C_obs \ SigSource(X_can))
8 return C_dom, C_umn, X_can, M_dom
```

Algorithm 2: DeduplicateConcepts

The deduplication stage, formalised by Algorithm 2, reduces the input set of concepts (C_in) by processing mappings to create a stable concept merge set (X_can).

4.1.1 Pre-processing. First, we compute the merges for obsolete concepts. Some mappings assert a relation between concepts of the same ontology to describe internal concept renaming, i.e. when a concept is deprecated and assigned to a new name. Using the set of obsolete concepts, the function ComputeObsoleteMerges computes merges X_obs from the relevant input mappings \((m = (c, \Rightarrow, c′) \in M_{in}|c \in C_{obs} \land c′ \not\in C_{obs})\). Next, we update the mappings by applying the obsolete concept merges X_obs, so the output M_in now only contains current concept names.

4.1.2 Alignment. Algorithm 3 computes the stable merge set. The set of concepts C_umn keeps track of unmerged (i.e. not merged) concepts; this initially contains all input concepts, excluding concepts of the seed ontology. The alignment process loops through the mapping type groups and the ontology alignment order. In each iteration, we attempt to align every remaining unmerged concept to concepts of the given Source \( \in \) SOURCES. The function GetMps returns all mappings that are in the specified type group M_Type and map concepts of the given Source; this (M_Source) is filtered by the function Filter, so the resulting mapping set M_Source only contains those mappings that align unmerged concepts. These mappings focus on the current Source such that the target of mappings is always from M_Source. Finally, the oriented mapping set M_Source is further filtered by multiplicity with the function Get1Or*SourceTo1Target such that in the produced mapping set (M_Source) every source concept is mapped to exactly one target concept (but one target may be mapped to many source concepts). The mappings are then converted to merges and stored in X, and the mapped concepts are removed from C_umn.

```
Input : C_in; M_updated: mappings with current concept names; X_obs; SEED; SOURCES; MappingTypes;
Output: X: concept merges
1 C_umn ← C_in \ GetConcepts(C_in, SEED)
2 X ← ∅
3 for each M_Type ∈ MappingTypes do
    4 for each Source ∈ SOURCES do
        5 M_Source ← GetMps(M_updated, M_Type, Source)
        6 M′ Source ← Filter(M_Source, C_umn)
        7 M′′ Source ← OrientTo(M′ Source, Source)
        8 M′′′ Source ← Get1Or*SourceTo1Target(M′′ Source)
        9 X ← X ∪ ConvertToMerges(M′′′ Source)
        10 C_umn ← C_umn \ SigSource(X)
    end
end
return X
```

Algorithm 3: ComputeMerges

Figure 2: Integrating an unmapped concept via its original hierarchy path and a mapped parent.

After concept deduplication, the remaining concepts (H_umn) are organised into a DAG, i.e. a concept hierarchy using the input hierarchy edge set (H_in). This process is formalised by Algorithm 4, and the main idea, which is a simplified version of the approach presented in [14, 24], is depicted in Figure 1. First, we add the seed ontology hierarchy to the domain hierarchy set (H_dom) and initialise the set of connected nodes with the seed hierarchy concepts and the concepts that were merged to seed concepts.
we iterate through each source in the priority order and filter out the corresponding hierarchy fragment from the input set with the function GetHierarchy(\text{Source}, \text{H}_{\text{in}}) (because path finding performs better on smaller inputs). Next, for each unmerged concept in \text{H}_{\text{SOURCE}}, we check whether there is a path to the root concept that contains a connected concept (i.e., an anchor point that is either a seed concept, is merged to a seed concept, or has already been connected to \text{H}_{\text{dom}}). If there is, we prune the path by dropping those intermediate concepts that are not in \text{C}_{UNMAPPEDSOURCE} and convert the pruned path to a set of hierarchy edges. Finally, we update \text{C}_{\text{connected}} and \text{H}_{\text{dom}} before proceeding to the next unmerged concept. The result is a crude approximation where we may lose concept classification granularity (i.e., the dropped intermediate concepts) in order to establish a baseline that does not increase the size of the domain concept set but maximises the hierarchy coverage of \text{C}_{\text{unn}} by \text{H}_{\text{dom}}.

5 EXPERIMENTAL EVALUATION

We conducted an experimental evaluation on a real-life data set: the disease concepts of BIKG [6] in order to assess the effectiveness of our approach in reducing duplication and creating connectivity. A comparative evaluation was not possible since currently there are no other systems that operate with the same requirements as OntoMerger. Here we only include a brief evaluation; however, the library produces an in-depth analysis report [8], as depicted in Figure 1, using several metrics [25] to evaluate the KG integration. Table 1 shows a comparison of the input and output data, while Table 2 presents the results of the deduplication of concepts and the creation of the concept hierarchy. 30.79% of input concepts (of 16 sources) were merged, and 41.11% were successfully integrated into a DAG using only 4 hierarchies. The overall run time of the process was 25 minutes on an average machine (6 core CPU, 32GB memory). The majority of the run time was the generation of the concept hierarchy (72%) and the result analysis (13%).

### Table 2: Concept deduplication and connectivity status

| Metric Count (Input) Count (Output) |
|-------------------------------------|
| Metric                          | Count (Input) | Count (Output) |
| Concept concepts (ontologies)     | 16            | 16             |
| Concepts                         | 187,717       | 152,021        |
| Concept merges                    | n/a           | 35,696         |
| Mappings                         | 439,389       | 133,496        |
| Connected subgraphs               | 1             | 4              |
| Hierarchy edges                   | 272,824       | 60,621         |

### Table 1: Input and output (domain ontology) comparison

Future Directions.

In this paper we have presented OntoMerger, an ontology integration library that deduplicates KG concepts of the same domain, and organises them into a single DAG, i.e., a concept hierarchy. Furthermore, we have empirically evaluated the approaches that perform deduplication of concepts and the formation of the concept hierarchy. In this section, we address the limitations of the library and outline directions for our future work.

**Limitations.** The library does not directly consume traditional RDF formats but requires the additional step of parsing and transforming concept names, mappings, and hierarchy edges into a columnar data structure. Moreover, the output is also in a columnar format. The library handles all data in-memory, posing several limitations regarding the scalability of the system. Firstly, processing tables with Pandas becomes challenging for large datasets. Secondly, hierarchy path computation slows down significantly due to using in-memory graphs. Currently, only the shortest path is computed for each concept when we attempt to connect them to the domain hierarchy. However, in many ontologies, there are often multiple possible paths to the root of the ontology. Using only one path could lead to missing out on paths that contain an anchor point, i.e., a connected concept.

**Future Directions.** In order to get a better view of the effectiveness of this approach, we plan to conduct a comparative evaluation of our system with several available (Java) ontology integration libraries. The library could use a mediator step to consume RDF formats directly and translate them into the internal columnar data structures. There are several libraries available that perform RDF to table translation, for instance KGTK [11].

To address scalability when handling large tables, the system could use distributed data processing such as Spark [30]. Path computation could be scaled by using an external triple store, where path computation is optimised (this is often achieved by the full materialisation of hierarchy triples) [18]. Offloading path computation would also enable us to use more than one path during the hierarchy formation process. However, this could also be improved.
by either implementing a greedy strategy (keep looking for paths until a suitable one is found) or providing the ability to specify the path computation strategy in the OntoMerger configuration file. Finally, OntoMerger may merge concepts of the same ontology when it may be more appropriate to convert these merges into hierarchy edges.

REFERENCES

[1] Gintaras Barisevičius, Martin Coste, David Geleta, Damir Juric, Mohammad Khodadadi, Giorgos Stoulos, and Ilya Zahirayev. 2018. Supporting Digital Healthcare Services Using Semantic Web Technologies. In: The Semantic Web – ISWC 2018. Denny Vrandečić, Kalina Boncheva, Mari Carmen Suárez-Figueroa, Valentina Presutti, Irene Celino, Marta Sabou, Lucie-Aimée Kaffee, and Elena Simperl (Eds.). Springer International Publishing, Cham, 291–306.

[2] Stephen Bonner, Ian P. Barrett, Cheng Ye, Rowan Swers, Ola Engvist, Andreas Bender, Charles Tapley Hoyt, and William Hamilton. 2021. A review of biomedical datasets relating to drug discovery: A knowledge graph perspective. arXiv preprint arXiv:2012.10662 abs/2012.10662 (2021), 1–34.

[3] Anna Breet, Simon Intt. Asan Agibetov and Matthias Samwald. 2020. OpenBiolink: a benchmarking framework for large-scale biomedical link prediction. Bioinformatics 36, 13 (2020), 4097–4098.

[4] Jérôme Euzenat, Pavel Shvaiko, et al. 2013. Ontology matching. Springer, Berlin, Heidelberg.

[5] Muhammad Fahad, Nejib Moalla, and Abdelaziz Bouras. 2012. Detection and resolution of semantic inconsistency and redundancy in an automatic ontology merging system. J. Intell. Inform. Syst. 39, 2 (2012), 535–557. https://doi.org/10.1007/s10844-011-0202-9

[6] David Geleta, Andry Nikolov, Gavin Edwards, Anna Gogleva, Richard Jackson, Erik Jansson, Andrej Lamov, Sebastian Nilsson, Marina Pettersson, Vladimir Poroshin, et al. 2021. Biological Insights Knowledge Graph: an integrated knowledge graph to support drug development. bioRxiv bioRxiv 2021.10.28.466262 (2021), 1–24. https://doi.org/10.1101/2021.10.28.466262 arXiv:https://www.biorxiv.org/content/early/2021/11/01/2021.10.28.466262.full.pdf

[7] Abe Gong and James Campbell. 2021. Great Expectations. https://github.com/ great-expectations/great_expectations.

[8] Aric Hagberg, Pieter Swart, and Daniel S Chult. 2008. Exploring network structure, dynamics, and function using NetworkX. Technical Report. Los Alamos National Lab.(LANL). Los Alamos, NM (United States).

[9] William L. Hamilton. 2020. Graph representation learning. Synthesis Lectures on Artificial Intelligence and Machine Learning 14, 3 (2020), 1–159.

[10] Daniel Scott Himmelstein, Antoine Lizere, Christine Hessler, Leo Brueggeman, Sabrina L. Chen, Dexter Hadley, Ari Green, Pouya Khankhanian, and Sergio E Baranzini. 2017. Systematic integration of biomedical knowledge prioritizes drugs for repurposing. Elife 6 (2017), e07026.

[11] Filip Iievski, Daniel Garjo, Hans Chalupsky, Naren Teja Divvala, Yixiang Yao, Craig Milo Rogers, Kousheng Li, Jun Lin, Ambadeep Singh, Daniel Schabes, and Pedro A. Szekely. 2020. KGTK: A Toolkit for Large Knowledge Graph Manipulation and Analysis. CoRR abs/2006.00088 (2020), 16 pages. https://doi.org/10.48550/arXiv.2006.00088

[12] Ernesto Jiménez-Ruiz, Bernardo Cuenca Grau, Ian Horrocks, and Rafael Berlanga. 2007. Leveraging the Cell Ontology to classify unseen cell types. In: The Semantic Web: ISWC 2007. W3C Working Group Note 74 (2007), 1–9.

[13] Ernesto Jiménez-Ruiz, Terry R. Payne, Alessandro Solimando, and Valentina Tamma. 2016. Limiting Logical Violations in Ontology Alignment Through Negotiation. In Proceedings of the 15th International Conference on Principles of Knowledge Representation and Reasoning (KR). AAAI Press, Cambridge, MA, USA, 217–226.

[14] Damir Juric, David Geleta, Gregory McKay, and Giorgos Stoulos. 2021. A Platform and Algorithms for Interoperability Between Clinical Coding Systems. Procedia Computer Science 192 (2021), 563–572.

[15] Graham Klyne 2004. Resource description framework (RDF): Concepts and abstract syntax. http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/.

[16] W. McKinney et al. 2011. pandas: a foundational Python library for data analysis and statistics. Python for high performance and scientific computing 14, 9 (2011), 1–9.

[17] Inés Osman, Sadek Ben Yahia, and Gayo Diallo. 2021. Ontology Integration: Approaches and Challenging Issues. Information Fusion 71 (2021), 38–63. https://doi.org/10.1016/j.inffus.2021.01.007

[18] Jaroslav Pokorný. 2015. Graph Databases: Their Power and Limitations. In Computer Information Systems and Industrial Management, Khalid Saeed and Wladyslaw Homenda (Eds.). Springer International Publishing, Cham, 58–69.

[19] Rachel A. Pottinger and Philip A. Bernstein. 2003. Merging Models Based on Given Correspondences. In Proceedings 2003 VLDDB Conference, Johann-Christoph Freytag, Peter Lockemann, Serge Abiteboul, Michael Carey, Patricia Selinger, and Andreas Huer (Eds.). Morgan Kaufmann, San Francisco, 862–873. https://doi.org/10.978/9-12722442-8/50088-1

[20] Salvatore Raunich and Erhard Rahm. 2014. Target-driven merging of taxonomies with Atom. Information Systems 42 (2014), 1–14. https://doi.org/10.1016/j.cis.2013.11.001

[21] Arnah Sinha, Zhihong Shen, Yang Song, Hao Ma, Darrin Ede, Bo-June Hsu, and Kuansan Wang. 2015. An Overview of Microsoft Academic Service (MAS) and Applications. In Proceedings of the 24th international conference on world wide web Association for Computing Machinery, New York, NY, USA, 243–246.

[22] Alessandro Solimando, Ernesto Jiménez-Ruiz, and Giovanna Guerrieri. 2017. Minimizing conservatism violations in ontology alignments: algorithms and evaluation. Knowledge and Information Systems 51, 3 (2017), 775–819. https://doi.org/10.1007/s10115-016-0983-3

[23] Christian L Staudt, Alekssejs Sazonovs, and Henning Meyerhenke. 2016. NetworKit: A tool suite for large-scale complex network analysis. Network Science 4, 4 (2016), 508–530.

[24] Giorgos Stoulos, David Geleta, Jentend Shamsadani, and Mohammad Khodadadi. 2018. A novel approach and practical algorithms for ontology integration. In The Semantic Web – ISWC 2018. Springer International Publishing, Cham, 458–476.

[25] Giorgos Stoulos, David Geleta, Szymon Wartak, Sheldon Hall, Mohammad Khodadadi, Yizheng Zhao, Ghadah Alghabani, and Renate A Schmidt. 2018. Methods and Metrics for Knowledge Base Engineering and Integration. In Proceedings of the 9th Workshop on Ontology Design and Patterns (WODP 2018) co-located with 17th International Semantic Web Conference (ISWC 2018), Monterey, USA, October 9th, 2018. CEUR-WS.org, Monterey, USA, 72–86.

[26] Octavian Udrea, Lise Getso, and René J. Miller. 2007. Leveraging data and structure in ontology integration. In Proceedings of the ACM SIGMOD International Conference on Management of Data, Beijing, China, June 12–14, 2007, Chee Yong Chan, Beng Chin Ooi, and Asyou Zhou (Eds.). Association for Computing Machinery, New York, NY, USA, 449–460. https://doi.org/10.1145/1247480.1247531

[27] Guado Van Rossum and Fred L. Drake. 2009. Python 3 Reference Manual. CreateSpace, Scotts Valley, CA.

[28] Brian Walsh, Sameh K Mohamed, and Vit Nováček. 2020. Biokg: A knowledge graph for relational learning on biological data. In Proceedings of the 29th ACM International Conference on Information & Knowledge Management. Association for Computing Machinery, New York, NY, USA, 3173–3180.

[29] Sheng Wang, Angela Oliveira Pisco, Aaron McGeever, Maria Brbic, Marinka Zitnik, Spyros Darmanis, Jure Leskovec, Jim Karkanias, and Russ B Altman. 2021. Leveraging the Cell Ontology to classify unseen cell types. Nature communications 12, 1 (2021), 1–11.

[30] Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangru Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, and Ion Stoica. 2016. Apache Spark: A Unified Engine for Big Data Processing. Commun. ACM 59, 11 (oct 2016), 56–65. https://doi.org/10.1145/2934664

[31] Marinka Zitnik, Rok Sošic, Sagar Maheshwari, and Jure Leskovec. 2018. BioSNAP Datasets: Stanford Biomedical Network Dataset Collection. http://snap.stanford.edu/biodata.