Biological Insights Knowledge Graph: an Integrated Knowledge Graph to Support Drug Development

David Geleta
AstraZeneca
United Kingdom

Andriy Nikolov
AstraZeneca
United Kingdom

Gavin Edwards
AstraZeneca
United Kingdom

Anna Gogleva
AstraZeneca
United Kingdom

Richard Jackson
AstraZeneca
United Kingdom

Erik Jannson
AstraZeneca
Sweden

Andrej Lamov
AstraZeneca
Sweden

Sebastian Nilsson
AstraZeneca
Sweden

Marina Petersson
AstraZeneca
Sweden

Vladimir Poroshin
AstraZeneca
United Kingdom

Benedek Rozemberczki
AstraZeneca
United Kingdom

Timothy Scrivener
AstraZeneca
United Kingdom

Michael Ughetto
AstraZeneca
Sweden

Eliseo Papa
AstraZeneca
United Kingdom

ABSTRACT

The use of knowledge graphs as a data source for machine learning methods to solve complex problems in life sciences has rapidly become popular in recent years. Our Biological Insights Knowledge Graph (BIKG) combines relevant data for drug development from public as well as *COMPANY NAME* internal data sources to provide insights for a range of tasks: from identifying new targets to repurposing existing drugs. Besides the common requirements to organisational knowledge graphs such as being able to capture the domain precisely and give the users the ability to search and query the data, the focus on handling multiple use cases and supporting use case-specific machine learning models presents additional challenges: the data models must also be streamlined for the performance of downstream tasks; graph content must be easily customisable for different use cases; different projections of the graph content are required to support a wider range of different consumption modes. In this paper we describe our main design choices in implementation of the BIKG graph and discuss different aspects of its life cycle: from graph construction to exploitation.

ACM Reference Format:
David Geleta, Andriy Nikolov, Gavin Edwards, Anna Gogleva, Richard Jackson, Erik Jannson, Andrej Lamov, Sebastian Nilsson, Marina Petersson, Vladimir Poroshin, Benedek Rozemberczki, Timothy Scrivener, Michael Ughetto, and Eliseo Papa. 2018. Biological Insights Knowledge Graph: an Integrated Knowledge Graph to Support Drug Development. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/1122445.1122456

Woodstock '18, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Recent years have seen rapid growth in the popularity of knowledge graphs in the life sciences domain. Knowledge graphs often serve as a backbone for data integration within organisations, providing a common representation structure which enables querying across data sources. With the recent advances in machine learning (ML), knowledge graphs gained one more important purpose: to serve as training data for ML models, and graph machine learning models in particular. Their newfound use as training data has to be taken into account when constructing knowledge graphs and influences core design choices. For example, while a very expressive schema can be able to capture the most fine-grained aspects of domain data, it can at the same time hinder the application of machine learning due to scalability problems and diffusion of signal. Similarly, use as ML training data necessitates support for different modalities of data usage, beyond structured queries.

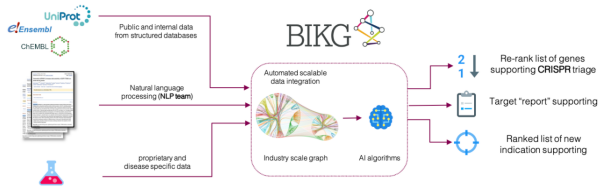

Figure 1: The Biological Insights Knowledge Graph project overview: data types, graph build, and example use cases.

In this paper we describe the Biological Insights Knowledge Graph (BIKG): an *AstraZeneca* project aimed at building a knowledge graph combining both public and internal data to facilitate knowledge discovery using machine learning. BIKG integrates knowledge from heterogeneous data sources including public databases like ChEMBL [18] or Ensembl [62], information extracted from full-text publications using Natural Language Processing (NLP) techniques, as well as diverse proprietary datasets collected as part of *AstraZeneca* drug development process and biological experimentation.

The rest of the paper is organised as follows: In Section 2 we discuss the main requirements to the BIKG knowledge graph and compares it with existing similar initiatives. In Section 3 we describe the process of building the knowledge graph by integrating heterogeneous sources. In Section 4 we overview different modalities of user interaction with BIKG data. Section 5 outlines the main applications of machine learning algorithms to graph data. In Section 6 we introduce the initial practical use case scenarios exploiting graph data. The paper concludes with Section 7 which discusses the main lessons learnt and lists directions for the future work.

2 MOTIVATION AND RELATED WORK

The life sciences domain has been an early adopter of ontologies and linked data technologies. The primary motivation, given the vast amount of knowledge to capture, was the need for standardisation of the vocabularies and taxonomies. This was essential for integration of data within and across large organisations and enabling data access. For this reason, the design of graph datasets focused on high granularity of the models and ability to capture the complexity of the domain in the most precise way. Such ontologies included, for example, the Gene Ontology (GO) [2] for gene functions, the Human Disease Ontology (DO) [26] for capturing different disease classifications, or BioPax [15] for pathway information.

With the growing number of datasets capturing separate domains, the focus shifted towards achieving interoperability and building integrated datasets that could serve as reference data sources across multiple interconnected topics. These included both *vertical* ontology integration of semantic alignment of multiple ontologies via common foundational schemas (e.g., the OBO Foundry [51]) and *horizontal* integration focusing on the data-level fusion of separate datasets into large interconnected graphs covering many domains that could be transparently queried (e.g., the Bio2RDF [4] triple database). Standardizing data structure and naming conventions helped to improve reusability of scientific data according to the FAIR data principles [59], making knowledge graphs a backbone for large-scale integration of data.

Recently, with advances in network medicine approaches [3] and in the area of AI, another role for knowledge graphs is quickly gaining importance, namely, facilitation of machine learning. Machine learning methods on graphs help to overcome sparsity of reliable data and reduce the need for expensive and time-consuming experiments by pre-selecting the most promising candidates for manual verification. Graph machine learning has been applied to a number of tasks [17] such as target identification [42], drug repurposing [20], or predicting polypharmacy side effects [65].

With the development of machine learning algorithms processing life science knowledge graphs, more work has focused on constructing integrated knowledge graphs to support such algorithms. There are several recent initiatives focusing on building knowledge graphs combining information from multiple data sources to use them for training machine learning models. For example, OpenBIOLink [7] was developed to provide a common benchmark to evaluate machine learning algorithms on life-science-related tasks. Drug Repurposing Knowledge Graph (DRKG) [24, 60] specifically combines the most relevant sources to support the drug repurposing task. CKG [47] focuses on bringing together multi-omics data to support precision medicine.

Constructing knowledge graphs to support graph analytics and machine learning has its own set of requirements that do not completely match the scenarios where the primary use of the graph is to enable complex cross-source querying. For example, a very precise and detailed data model can make it hard for an algorithm to learn essential relations between nodes, if they are not connected directly, but separated by several hops. High model granularity can also increase the size of the graph and make some complex models expensive to train.

The usability expectations also differ in this case. While the abilities to search and formulate expressive queries are still important, the possibility to customise and manipulate the graph for different use cases becomes particularly valuable. Depending on the task (e.g., drug repurposing or target selection) or the domain (e.g., specific disease like asthma), the users should be able to extract a custom relevant subgraph to apply statistical methods and train models. In general, it is important to combine for analysis both public reference datasets capturing state-of-the-art knowledge (e.g., Ensembl [62] for genes or ChEMBL [18] for drug compounds) as well as internal proprietary data generated inside the organisation over many years. Moreover, researchers often maintain their own task-specific datasets which they want to include in analysis, together with large-scale reference data sources. For this reason, integrating new data into the graph should be made as easy as possible, as well as the filtering of the graph according to custom criteria. Finally, the graph has to be easily consumable as input by popular machine learning software libraries.

With these requirements in mind, we developed the BIKG (Biological Insights Knowledge Graph): an internal *AstraZeneca* knowledge graph aimed at supporting analytics and machine learning tasks to help drug development. The graph construction process focuses on unifying the structure of multiple source datasets and the flexibility of the workflow: enabling quick incorporation of feedback and *democratisation* of the data integration by allowing the end users to bring in their own tasks-specific data. Multiple graph usage modalities, in turn, provide alternative ways of interacting with the graph optimised for different end-user tasks.

The BIKG development process involves supporting two stages:

- *Graph construction*, which involves bringing together diverse data sources and performing common data integration tasks to produce the graph and provide multiple access options.
- *Graph utilization*, which involves supporting end users with applying machine learning techniques to solve use case tasks.

3 KNOWLEDGE GRAPH CONSTRUCTION

The graph build pipeline integrates internal and external data, implemented in the cloud as a secure and scalable pipeline architecture, to create a consistent and coherent knowledge graph that is used for applying machine learning algorithms and gaining new insights.

3.1 Data sources

The latest BIKG combines over 50 years of biomedical information into a single resource, consisting of 10.9m nodes (of 22 types) and

over 118 million unique edges (of 59 types, forming 398 different triples) as shown in Table 1. Some of the ingested data sets are themselves integrate a number of data sources, such as *Hetionet* [23] and *Opentargets* [31].

At the core of the graph are reference datasets describing different topics of the drug discovery domain, such as genes, compounds, and diseases. These reference datasets are enriched with relations automatically extracted from literature using state-of-the-art natural language processing. Finally, in order to adjust to the needs of the end users and their tasks, the users have the ability to enrich the graph with their task-specific data. Using BIKG allows users to connect to *internal company data* and gain a unique advantage with respect to openly available graph datasets.

3.1.1 Backbone: static reference datasets. Standard datasets describing specific domains constitute the *backbone* of the BIKG graph: they both define the high-level structure for representation of each respective domain and the identification conventions for domain entities. This includes popular standard datasets commonly used in drug discovery [6], such as Ensembl [62] for genes, ChEMBL [18] for drug compounds, Mondo [38] for diseases, the Gene Ontology [2], and others. To model each domain of interest in the most complete way, in cases where there exist several alternative reference sources, they were integrated together and merged using mappings between the corresponding identifiers: e.g., having ChEMBL identifiers as canonical ones for drug compounds and expanding them with complementary external (such as PubChem [27]) as well as internal ones (e.g., covering experimental formulas not yet indexed by public repositories). Apart from the actual content data, integration of mapping sets provided by different sources represents added value in itself, allowing end users to switch quickly between the naming schema of their use case-specific source and the canonical one used by the graph.

Summary	Count
Nodes	11m
Node types	22
Node contexts keys	276
Edges (collapsed)	118m
Edges (uncollapsed)	1189m
Edge types	59
Edge evidence keys	154
Triple types	398
Ingested data sets	39+

Table 1: The graph content summary table provides a flavour for its size and compositional variety.

3.1.2 Natural language processing (NLP) for graph population. While the data imported from various structured sources provides the most reliable part of the graph, it is enriched using large-scale relation extraction from free-text sources such as scientific literature from PubMed [45]. The NLP aspects of BIKG is based on a series of pipelines, ranging from simple entity co-occurrence and traditional rule based dependency parsing, to state-of-the-art relationship classification with RBERT[61] and open information extraction with the OpenIE6 [30] neural information extraction system. In terms of quantity, this NLP-extracted data constitutes the largest component of the graph, providing around 80% of graph edges. Despite inherent uncertainty associated with these edges due to potential NLP errors, when aggregated, they provide clear added value for the output quality of machine learning models: e.g., NLP-extracted edges were found to be the most informative in a link prediction benchmark task focusing on protein-protein interaction (PPI) network population.

Extracting a large amount of structured information from biomedical literature remains a very complex task. The language used to describe biological entities and relationships varies across temporal and geographic dimensions, and is subject to various phenomena in language evolution (for instance, neologisms, synonyms, multi-entity constructs etc). Progress in developing a generalised knowledge base population solution is hampered by multiple factors: (i) Lack of sufficiently diverse training data covering the breath and depth of biomedical knowledge; (ii) Biases in existing training data sets (such as only covering a subset of diseases); (iii) Uncertainty/fluidity in our understanding of disease mechanisms, resulting in inconsistent language usage over time; (iv) Contradictory evidence and or non reproducible results; (v) Uncertainly in the interpretation of data, often resulting in authors hedging the assertions they make; (vi) Biases in literature to only report positive findings; (vii) Poor generalisability of ML models, research findings are generally overfit to small datasets and not suitable for production offerings.

Our NLP data is derived from a multi stage pipeline:

(1) **Abbreviation Expansion** - in order to improve our named entity recognition (NER), we preprocess text data to expand abbreviations. [48]
(2) **Named Entity Recognition** - to identify graph entities, we use the commercial Termite Tagger from Scibite. We supplement this with mutation information, based upon a modified form of SETH.
(3) **NER Post Processing** - in this stage, we normalise entity information from different NER tools back to a standard data class; enrich entities with BIKG ids where possible; and perform entity linking on concepts that otherwise lack this information
(4) **Relationship Extraction** - the relationship extraction engine is based on three techniques; First, a rules based dependency parsing application, based on the open source LINK software. Second, relationship classification using a neural network based upon BioBERT [34] and the RBERT [61] classification head. Third, open information extraction using the OIE6 software[30]. We limit the overall complexity of the input dataset by excluding those sentences containing more than 30 entities from any downstream processing. Complex sentences with more than 12 entities are not fed to our neural network inference engine.

LINK. Modern dependency parsers are incredibly accurate. However, while they are informative about the syntactic structure of sentences, they offer no guidance on how such structure should be manipulated in order to extract triples. Our current work in this area involves the refactoring of the LINK NLP pipeline [41] to make

use of our commercial NER tool, Termite[49], and the subsequent crafting of rules to produce triples suitable for ingestion into the graph. An advantage of dependency parsing approaches is that they are able to predict the type of relation (verb) between two entities, meaning that the results can be mapped to an ontology. The drawback is that they tend to be more error prone than neural network based methods.

RBERT. Taking inspiration from the latest wave of attention based transformer neural network, we have written our own relationship classifier implementation based upon the work of [61]. Our RBERT model is more powerful than LINK, but suffers from the significant disadvantage of only being able to suggest associations between two entities, as opposed to detecting the verb describing the relationship. Our deployed model is trained to predict binary relationships between the following entity types: Gene Target—Gene Target and Gene Target—Compound. To give an indication of performance, we use the BioInfer [43] and BioRelex [25] datasets.

3.1.3 User-contributed data. One of the main requirements of BIKG is the possibility to integrate internal use case-specific datasets on demand. Given the multitude of use cases, a purely centralised approach to data addition has limitations, because constant need to process new datasets creates a bottleneck on the side of the graph engineering team, while it is the end users and data owners who have the best knowledge of the data and the use case needs. For this reason, the approach chosen in BIKG is to enable and support the end users in integrating the data they need. This task was made easier by the fact that the majority of graph users are bioinformaticians who are familiar with script coding as well as tabular data manipulation techniques.

The *Bring Your Own Data (BYOD)* module, which is part of the Python API (used for accessing the graph, see Section 5.4), aids the ends users in extending the graph. This provides a command-line interface, code templates as well as tutorial notebooks, to aid users to transform their data into a pre-defined tabular format suitable for automated insertion into the graph. As the data might be syntactically well formed and its semantics need to conform to the BIKG Upper Level Ontology (Section 3.5), the users are also provided with a data verification tool, which checks the data format, relational structure, as well as the naming conventions for node identifiers. Using this API users can parse and integrate their own data to a copy of the graph, and can also share datasets to be integrated into to central build.

3.2 Unified data model

Similarly to the *Microsoft Academic Graph (MAG)* [50], the data is modelled and distributed as a set of tables. In the BIKG pipeline the graph data is processed and stored in *parquet* files [1]. There are two main tables, nodes (including mappings) and edges, described in Section 3.2.1 and Section 3.2.2, respectively. Both tables contain *standard columns* (representing mandatory table fields) and *dynamic columns* for representing the different node or edge background information found in the ingested sources.

3.2.1 Nodes. A node in BIKG is defined as the tuple:

$$\langle \text{Node ID, Label, Type, Context}(c_i ... c_j) \rangle.$$

[1] https://parquet.apache.org/

The fields of this tuple are defined as follows:

- Node ID: the preferred identifier for a given node, adhering to a unified ID schema (Data Set ID:Code, which is the internal node ID where the original ID is prefixed with a namespace to capture provenance e.g. $ENSEMBL : ENSG0001$ for the Ensembl node $ENSG0001$);
- Label: the preferred label for the node, for example for genes often the gene symbol label is used;
- Type: the classification of the node, this must conform to the Upper Level Ontology (see Subsection 3.5).

In parsed sources node tables also have a *Provenance* column to aid the build pipeline identity resolution step (see Section 3.6) in finding a canonical node for duplicate node instances (i.e. nodes with the with the same identifier appearing in different parsed sources). In addition to the above described *standard columns* that must be present in all parsed sources, the node table may have a set of context columns, Context$(c_i, \ldots, c_j)$. These columns record background information about the node such as labels, types or other data that may be a potential feature in ML training:

- Context Labels: All labels (preferred, synonym, description, definition, notes etc) associated with the concept
- Context Types: All types that are associated with a given node ID over all of its instances in the loaded data; e.g. a node can be a *Disease* but also a *Side Effect*, storing these enables for better filtering. The *type* value is a member of this set.

Node mappings are stored as a separate table in the build process, but materialised in graph projections for more convenient usage.

3.2.2 Edges. An edge in BIKG is defined as the tuple:

$$\langle \text{Edge ID , Source ID, Target ID, Relation, Provenance, Evidence}(e_i, \ldots, e_j) \rangle$$

The fields of this tuple are:

- Edge ID: the edge identifier is the base64 encoding of the fields Source ID, Target ID, Relation, and Provenance (note that this is not necessarily unique in a data set prior to graph compression (see Section 3.4);
- Source ID, Target ID: identifiers of the nodes connected by an edge;
- Relation: the type of the relationship between the source and the target node;
- Provenance: the source data set identifier;
- Evidence$(e_i, \ldots, e_j)$: zero or more meta data columns associated with the edge.

In a compressed edge table (see Section 3.4), standard edge columns (*Source ID, Target ID, Relation, Provenance*) are the same, but the evidences are merged.

3.3 Graph projections

The graph is built using several columnar data tables (node, mapping and edge table) that enable fast processing as well as reduce overall data size. The resulting output contains all data, such as mappings and alternative labels for nodes, as well as a large number of features (contextual attributes for nodes, and underpinning evidence for edges). In this form, the graph is not user friendly because it contains billions of rows and hundreds of columns. Unless

users are equipped with big data processing infrastructure, even simple tasks (for example querying the graph or creating edge sets) can pose a challenge. In order to support users and to make the graph more accessible for a wide range of audience within the company, several formats, denoted as *graph projections*, are produced by the pipeline. Section 4 describes how these projections are used. Projections may contain materialised data (e.g. node mappings are inferred and materialised in the nodes table), may differ in data structure (in the BIKG projection, the edge table is split into two tables), or uses a dedicated format (e.g. RDF for triple stores). The following projections are produced:

- The *BIKG* projection is the main distribution artefact, i.e. the source of truth, which is used by the Python API (see Section 5.4). This is a set of parquet files, where the edge table into two tables, one for the standard columns representing the edge ($SourceID$, $Relation$, $TargetID$, $Provenance$) and another containing all feature columns ($Evidence(e_i, \ldots, e_j)$); as a result users can pick and choose only desired features, ultimately reducing data size. The projection uses compressed edges (see Section 3.4).
- The *BROWSER* (see Subsection 4.1) projection is also serialised as parquet files, but it contains the uncompressed edges in order to aid indexing and data presentation, where standard edge information is merged with the feature columns.
- The *RDF* and *Neo4j* projections (see Subsection 4.2) both use the compressed edges. In the RDF format, nodes are materialised as classes, edges as triples, where the corresponding edge features are represented as triple annotations. In addition to the aforementioned graph database formats, another RDF projection is generated; this presents the high level schema, graph meta data and analysis (such as node and triple type counts) to support visual graph exploration [35].

3.4 Graph compression

The graph is compressed and filtered to limit its size. This is necessary because the raw data set is quite large (several terabytes); the data is noisy; also the purpose of the graph is not information retrieval but facilitating signal propagation, hence not all ingested data is preserved in the main graph projection (however, note that relevant data, e.g. publication references, is stored prior to the graph build and is made available separately as needed by users).

The graph is compressed by *merging duplicate edges*. Two edges are considered duplicates if they shares the same *Source ID, Target ID, Relation, Provenance* attributes, i.e. they have the same source and target node identifier reference, relation type and edge provenance, hence the only distinguishing feature is the difference in evidence attributes (e.g. publication source, experimental evidence score value etc.). When merging edges, the evidences are retained. However, this set can be rather large due to: skewed literature data (for example, several trivial edges are parsed from many publications, resulting in millions of noisy triples); and overlapping data source content. Therefore, to limit the size of the graph, the number of merged edges for each *edge duplication case are limited*. This reduces noise and retains most relevant information. The number of duplicate edges is retained with each merged edge.

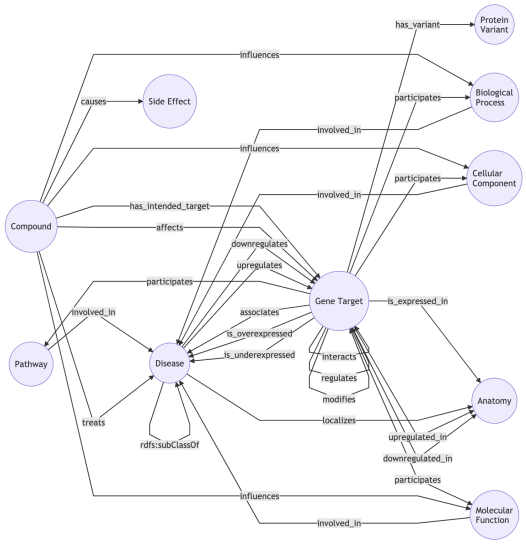

Figure 2: Core fragment of the BIKG Upper Level Ontology

3.5 Upper Level Ontology (ULO)

Different source datasets of BIKG use their own vocabularies to represent the same domains and model different aspects of the data with different levels of granularity. For this reason, a unification of models is required to produce an integrated graph with a common structure. The Upper-Level Ontology (ULO) of BIKG (see Figure 2) serves this purpose by defining a common schema over integrated sources. The second goal of the ULO ontology is to define common data constraints to verify the consistency of integrated data.

ULO aims at representing the node and edge types at the level of granularity that enables best signal propagation while, at the same time, providing sufficient coverage for representing the domain area. At the core of the ULO ontology are the most relevant concepts of the drug development domain: *Compound, Gene Target,* and *Disease* and relations between them. Design choices were motivated by the needs to achieve uniform representation of the source data and to facilitate the training of machine learning algorithms. For example, the notion of a *Target* is a key concept in drug development, usually representing a protein affected by a drug compound and encoded by some gene. However, different sources use different conventions for modelling targets: some refer to targets using the actual protein identifiers (e.g., codes from the UniProt database [56]), while many others use the identifier of the encoding gene (such as an Ensembl code [62]). Given that the gene-protein relations are in itself indirect (representing the gene → transcript → protein chain) and not always one to one, integration of different sources while preserving their full level of detail would necessarily break the uniformity of data and introduce extra complexity. For this reason, BIKG uses a *Gene Target* concept which is denoted by the gene ID, but merges the representation of the whole gene-transcript-protein subgraph.

To maintain the consistency of the model, concepts of the BIKG ULO ontology are mapped to the standard domain ontologies (such as Uberon [37] or Gene Ontology (GO) [2]) and, indirectly, to the foundational Basic Formal Ontology (BFO) [1]. Moreover, the BIKG

ULO defines ontological constraints on classes and properties to support quality assurance, most importantly, property domain/range and class disjointness: after building the integrated graph, automated tests verify whether the data conforms to the restrictions.

3.6 Identity resolution

One of the most typical and frequent data quality issue faced when ingesting heterogeneous data is *node duplication*, i.e. the same concept being represented in the graph by different node identifiers (e.g. HP:0030358 as Non-small cell lung cancer and UMLS:C0007131 as Non-small cell lung carcinoma). Node duplication causes to data fragmentation that degrades the quality of the data as well as leads to user confusion and data distrust. Duplication occurs for several reasons: different data sets use different vocabularies; one of the main data sources is the literature data, where the input data is evolving (by changing vocabularies used to tag nodes, new publications). Some node types are less ambiguous than others, for example we only encounter several gene vocabularies (which still requires extensive work on alignment), but there are dozens of disease vocabularies found in the ingested data sets (aligning disease nodes is a common problem when building large biomedical knowledge graphs, as there are many popular coding systems, where conversion is often not trivial [53]).

Node duplication is resolved by computing a stable set of mappings and merging the identified duplicate nodes. Each *duplicate node cluster* (set of nodes describing the same entity) is assigned a canonical, representative node, (e.g. for *Gene Target* nodes, Ensembl is preferred) and the node information of other non-representative nodes (such as types, labels, other context fields) is merged into the canonical node. For example node type *Side Effect* is preserved, even if it is classified as a *Disease* in the final graph; this information can be used for graph completion where the existing edge data (e.g. *Has side effect?* edge type) does not already imply this.

Mappings are collected from public sources (e.g. data sets, research papers), licensed sources, as well as produced internally in collaboration with other teams. A stable set of mappings is produced as an iterative process: first all available mappings relating to the nodes of ingested sources are aggregated and the set of inferred mappings computed. Next the produced merges are manually examined (sampled) and tuned with the help of user feedback. Due to the challenges of manual curation, this focuses is on several high priority areas supporting the use cases. Mappings are retained for determining node merges during the graph build process (the Node IDs in edges are updated using this table), but also for the purpose of adding data to the graph post build (as described in Subsection 3.1.3, users can add their own data to the produced graph using a a dedicated API).

3.7 Quality checking

Given the diversity of data sources and the volume of the data, data quality issues are inevitable. These are caused by either by imperfect data in the original sources (e.g. data entry errors, missing mappings) or by incorrect decisions taken by the data integration pipeline (e.g. a wrong canonical ID or label chosen during node merging). As noted by Stolios et al [53], several checks are required to be conducted in order to ensure the data quality of large biomedical knowledge graphs. This section first introduces the test framework (3.7.1) used in the pipeline, then it describes general data tests (3.7.2). Next it shows how node duplication is monitored (3.7.3), and finally it outlines how checking semantic constraints checking are implemented (3.7.4).

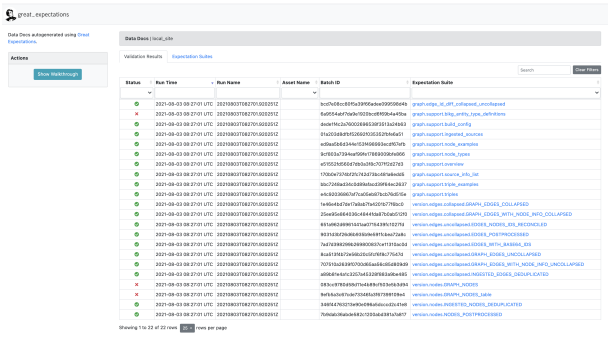

Figure 3: The Great Expectations BIKG data documentation.

3.7.1 Great Expectations. The knowledge graph build pipeline ingests and produces hundreds of different data files, such as parsed sources, intermediate build graph parts, different graph projections for distributing and accessing the graph (5.4), browsing (4.1), querying (via GraphDBs 4.2, Neo4j), and hosting graph analysis and content documentation. The *Great Expectations [9] (GE)* data test framework is used in the graph build pipeline to validate the input and output data, configuration and other files. GE is a Python-based open-source library for validating, documenting, and profiling the data. GE generates a *data documentation website* (as illustrated by Figure 3) that helps to maintain data quality and improve communication about data between teams. GE supports several environments (Spark, Pandas, SQL) for working with columnar data, which makes it suitable for use on BIKG.

GE provides a number of predefined simple (and some more complex) data tests for validating on columnar data [9]. Tests such as *Expect column values to be in set X.* can be used for checking the node *type* column content ULO conformance (see section 3.7.4). In addition, GE facilitates creating custom tests; for example ULO constraint checks (described in Section 3.7.4) require class subsumption reasoning, hence these are implemented as custom tests.

3.7.2 Data tests. A variety of data tests are run on the graph to ensure its quality. Some tests are more general such as validating whether all node IDs referenced in the edge table are present in the node table, or more data specific such as node and triple existence, or validating the existence of certain nodes (along with their preferred label, specified typed, merged equivalent nodes). Data specific test cases typically focus on gene targets, diseases and compounds (see Section 3.5) in certain therapeutic areas.

3.7.3 Node duplication check. In order to measure node duplication, node labels are compared for similarity. This can be challenging due to size (almost 11 million nodes) hence heuristics are employed to reduce the search space (e.g. Gene alterations are often wrongly named after the gene being altered using the gene symbol leading to duplication with the given gene node, therefore we can

exclude these cases as false positives). In addition, duplication cases can are also reported by users using internal reporting channels and issue tracking software. Some of the node duplication analysis results, as well as user reports are used as proposed mappings and fed back into the graph, or used for expanding the data tests.

3.7.4 ULO constraint check. The BIKG ULO (see Section 3.5) defines the class and property constraints used for validating the graph content. The ULO constraint checks ensure that: only permitted node and edge types are used, node types do not cause disjointness violations (see node *types* column), property domain and range violations are avoided and there are no restricted triple types. The checks are run on the tabular graph format, using Spark to materialise inferences. This is necessary due to scaling issues (memory requirements of triple stores), when using large RDF documents and running SHACL (Shapes Constraint Language, the W3C standard language for describing and validating RDF graphs [29]). Moreover, the RDF format is only one of many projections (different materialisation) of the graph, where the *main* projection is a set of parquet tables.

3.8 Pipeline overview

BIKG is built with a reproducible data pipeline that runs in the cloud and is capable of scaling to deal with large amount of data.

- **Build**: the set of sources specified in the configuration are loaded and merged into one table according to data type: node, mapping or edge. Each table has a set of standardised columns (3.2) and potentially other columns that are merged into a single table containing all columns (this results in a sparse table as the different node types have different contextual data). Edge compression (see Section 3.4), and node deduplication (see Section 3.6) takes place in this step.
- **Testing**: syntactic and semantic tests are run on the graph, as described in Section 3.7.
- **Analysis**: the graph content is analysed and documented. This includes computing simple graph metrics (node, edge and triple type breakdowns, counts), and producing small but representative examples of nodes and edges, for visual inspection.
- **Sampling**: due to the large size of the full graph, several samples are produced to facilitate testing and benchmarking.
- **Public**: this step produces a subset of the full graph. As a future work, we intend to open source a large portion of BIKG, that is composed of publicly available data.
- **Projections**: this step creates several projections of the graph. The different projections contain all, or most of the graph data but materialised in different file formats to serve different purpose (e.g. RDF for loading in a triple store and CSV for loading in Neo4j.).

Parsing of raw data sources takes place outside of the graph build pipeline. The raw data comes in many different formats (CSV, TSV, parquet, RDF, multiline JSON and JSON-LD etc.) and obtained from internal and external data base API calls and data dumps. The graph is rebuilt each time a new dataset is added, removed or updated; therefore it is important to manage graph size and subsequently the required graph build process time and compute effort.

3.9 Technology stack overview

The graph build takes place in the cloud and the graph is stored in a columnar format. In order to handle the large number, potentially billions of edges (rows) and hundreds of features (columns) the pipeline uses *Spark* [63]. Most of the data is stored as parquet files to reduce size and improve the data processing with Spark. Files are stored securely on a *Blob storage* [8] that complies with *AstraZeneca* data handling requirements for various types of data (public, licensed, company confidential and company restricted levels).

The code base is mostly implemented in Scala in order to optimise execution time. Python is also used in the project to allow for using a wider range of libraries, and to not limit developers and users to Scala. *Azure Datafactory* [28] is used as the ETL pipeline orchestrator. The compute clusters are managed by *Databricks* [16], where Python, Scala or R code is executed. In addition, the Databricks notebook environment is used for prototyping and collaborative development. Finally, *Azure Pipelines* [13] are used for running code base CI/CD, graph version, source analysis, data testing, creating and distributing releases and documentation.

4 DATA ACCESS AND EXPLORATION

Depending on the user profile and the task, different ways of interacting with the graph are required: e.g., keyword search, structured queries, or feeding data to machine learning models. There are different data storage, management, and access solutions that are optimised to tackle these interaction modalities: for example, indices for keyword search or graph databases for structured queries. However, it is difficult to select a single best solution that would be equally suitable for all. For this reason, after the BIKG graph is built, it is made available to the end users via several access routes.

4.1 Browser

The BIKG browser is a web interface tool giving the users the capabilities to search and browse the graph data in a convenient form. At the back-end side it is powered by a custom Elasticsearch index [19] which captures BIKG nodes as a set of multi-field documents, enabling faceted fuzzy keyword search. The browser index provides a GraphQL API [22] enabling expressive queries for nodes and edges. This GraphQL API both serves the web UI as well as gives the end users the ability to retrieve data directly from their scripts. The web UI is optimised for two tasks: retrieving the nodes and edges using a combination of keyword queries and logical filters and browsing the graph data by following incoming and outgoing edges. While this interaction mode provides a convenient entry point for graph exploration, it does not allow specifying expressive structural queries by defining conditions over a subgraph pattern.

4.2 Graph databases

In order to enable complex structural queries over the BIKG graph data, the graph is converted into a format suitable for loading into a graph database. The graph databases market includes two main streams of development: RDF triple stores and property graphs. Originally, RDF triple stores were primarily optimised for processing complex structural SPARQL queries, while property graphs

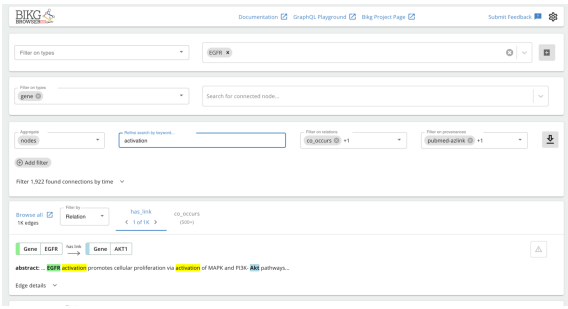

Figure 4: The BIKG Browser user interface, showing summary of the edges related to the EGFR gene [39].

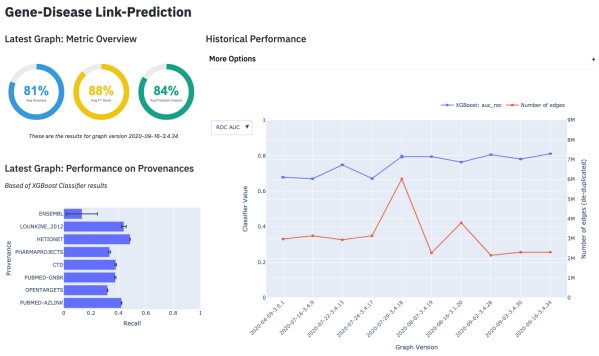

Figure 5: BIKG Benchmark Streamlit Application

commonly focused on the graph traversal algorithms. For this reason, BIKG graph gets exported into two formats suitable for these two different modalities: RDF [32] and Neo4j [58].

4.3 Columnar storage dumps

Given that the graph data is mainly used for data analytics and ML tasks, it is important to make the data easily accessible from Python scripts. To this end, providing a data dump in a compact format together with the corresponding Python API making the data consumption straightforward was found to be both faster and more convenient than retrieving the data by querying a graph database. For this reason, the graph is also released using the partitioned Apache Parquet format [57]. This format is then consumed by the Python API described in Section 5.4.

5 GRAPH MACHINE LEARNING ON BIKG

5.1 Knowledge graph embeddings

Recent years have seen an increase in methods that learns to encode the structural information of graphs as feature vectors. In this approach, called *representation learning* [5], each node in the graph is mapped to a point in a low-dimensional vector space, such that the graph topology is preserved. The node feature vectors can then be used as input to downstream machine learning tasks.

A part of the BIKG project has been to provide embeddings, both for use in internal ML models and as a resource for researchers across *COMPANY NAME*. Using Azure ML, we developed an automated pipeline for creating and analysing (see Section 5.2) embeddings after a new graph version is released. For a large-scale graph like the BIKG, a computationally efficient way of calculating graph embeddings is not only desirable but necessary. To this end, we have used DGLKE [64], a tool for generating embeddings developed by Amazon Web Services. DGLKE enables distributed computing of embeddings across multiple GPUs.

5.2 Benchmarks

With each new graph release, we run a set of benchmark tasks to gain insight into how changes to the graph anatomy affect the performance of downstream machine learning models. The benchmarks include common biomedical machine learning tasks such as predicting Gene-Disease association and Protein-Protein Interaction (PPI). For each benchmark, the graph is reduced in size to only contain relevant entities, e.g., Gene and Disease nodes for Gene-Disease link prediction. Then, using 5-fold cross-validation, node embeddings are trained with RESCAL [40] using 80% of the edges and evaluated on the remaining 20%.

For evaluation, the node embedding vectors are concatenated for each pair of nodes that share an edge in the training set. These pairs are labelled as true examples. In addition, an equal amount of concatenated embeddings are created for nodes that do not share an edge. These pairs are labelled as false examples. Finally, a downstream gradient boosted classification tree model [10], is trained to distinguish between true and false examples. The performance of the boosted model tells us how well the embeddings capture the proximities in the graph. To explore how the embeddings perform on different tasks with different data sources and how performance varies across graph releases, we created a Streamlit [55] application (Figure 5) displaying benchmark results from the ten latest releases.

5.3 Graph features

Graph-derived features, such as degree, betweenness, or transitivity contain rich information about nodes and edges in a graph [46] and it is valuable for downstream machine learning tasks.

Generating graph derived features on the scale of BIKG is complex. Traditional tools, such as NetworkX [21], iGraph [11] and Spark GraphFrames [12] fail to scale. After experimenting with tools it was found that GPU powered network analysis was far superior. Therefore GPU backed cuGraph library [54] was used to generated graph derived features. It runs most algorithms on a single GPU, for more demanding algorithms such as betweenness, it leverages multi-GPU batch processing. By using this approach graph features can be computed in a few hours on the full BIKG.

Graph features are provided to users via two methods: (i) By distributing pre-made features on the full BIKG graph. (ii) By providing the users with the example graph features notebook to create graph features on their own subgraphs. The end result of this is it improves downstream tasks performance and enables users to quickly leverage the rich information in BIKG for use cases.

5.4 Python library

To maximise the impact of BIKG in data science and machine learning, it is important to make data access intuitive and quick. This

enables users to experiment with the graph with minimal effort. We developed a Python package that is focused on loading the subset of BIKG data a user is interested in. We chose Python due to the wider ecosystem and its integration to other tools. The scope of the library is to make loading and cleaning subgraphs and their extra data as quick as possible. Once loaded, data is stored in Pandas DataFrames [36], giving users instant familiarity and maximal portability to other libraries. The package is accompanied by a documentation website with quick starts, tutorials and API explanations. For a code snippet example take a look at Listings 1.

```
import bikg
graph = bikg.Graph(version="latest",
                   graph_config="ppi-subgraph.json")
graph.edges.head()
```

Listings 1: An example of loading the latest version of BIKG with custom configuration and printing the head of the edges table.

The same Python library allows users to add their own data and enrich BIKG. This feature is especially important for sensitive data, since the data never has to leave the users' environment. Distributing a large dataset to end users via a Python library can be challenging. We were able to maintain acceptable speed and efficiency by partitioning the data, leveraging fast network connectivity and caching datasets near the users environments. Another challenge was limiting the scope of the library: users often requested additional pre-built features. We chose to actively limit the scope of our Python package to load and filter the data, rather than trying to address all downstream tasks with a single software library. We felt the integrative approach would have ultimately limited the users, given the rapidly evolving landscape of external downstream libraries. Instead, we produced a large set of example Jupyter notebooks [44] which demonstrate common graph tasks and that users could customise freely. The end result is users can quickly access, explore and apply BIKG data without any help.

5.5 Tutorial notebooks

Quickstart example notebooks are made available, that show users how to load BIKG via the python library then perform other tasks. There are a variety of examples. Some demonstrate loading BIKG into other libraries, such as NetworkX [21], Cytoscape [52] or cuGraph [54]. Others demonstrate how to apply BIKG, for example to do link prediction or drug re-positioning.

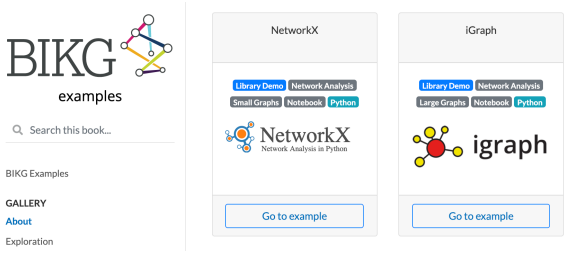

Figure 6: The BIKG example tutorials landing page.

Instead of limiting users by wrapping around libraries, we provide them examples they can customise. Currently we include tutorials for: exploratory data analysis with python and R, extracting a disease subgraph, examine literature trends, filter edge types based on metadata, year or Mesh clinical term, finding the best subgraph, link prediction, node classification, explainable learning on graphs.

We also include template workflows for the most common tasks we face when supporting decisions in the drug discovery pipeline: unsupervised recommendations with graph features (triaging hits), protein function prediction, drug repositioning, target identification, side effect prediction for combinations, gene ranking using tensor factorisation, target prediction with time slicing. There are high barriers getting started with BIKG due to unfamiliarity with (knowledge) graphs, the size of BIKG, new libraries and how to approach graph problems. The examples reduce the barrier of entry by bootstrapping new users and sharing the teams knowledge.

6 USE CASES

The goal of the BIKG is to serve as an asset for domain specialists in providing non-trivial insights for solving use case problems. In this section we just briefly reference two BIKG use cases: target identification and re-ranking of CRISPR screen hits.

6.1 Target identification

When searching for novel gene targets [17], interpreting vast quantities of data across multiple experiments is a time-consuming, but necessary procedure. The challenge is to reduce time and effort and provide a more unbiased approach to identify novel targets. To address this issue, we built a decentralised pipeline for integrating the experimental data important to the user. Once ingested, these data, along with information derived from BIKG, produces a ranked list of potential gene targets for the chosen disease area.

We employed the pipeline to rank potential targets for Chronic Obstructive Pulmonary Disease, Asthma and Lung Cancer among others with significant internal success. We also tested the pipeline on public data by trying to predict novel targets before their publication. For this, we used the data published before 2015 as a training set and used the newly discovered relevant targets for the disease of interest published between 2015 and 2020 as a test set. The model could predict 16 out of 95 relevant new targets appearing in the following 5 years. We observe higher performance when we use more data types and focus on disease-specific subgraphs.

6.2 Re-ranking of CRISPR screen hits

Acquired drug resistance is a major factor making development of lasting cancer treatments difficult. One strategy for determining key drivers of acquired resistance is based on functional genomic screens, such as CRISPR screens [33]. The output of these screens identifies thousands of potential targets. To narrow down the list to the most promising genes the scientists go through a lengthy and laborious procedure of manual validation, often prone to individual bias. There is a critical need for a standardised approach that integrates diverse types of knowledge to quickly identify valuable hits for experimental validation. We accelerated this decision-making process with a recommendation system built on top of BIKG.

The recommendation engine uses a multi-objective Pareto optimisation [14] producing a Pareto front of potentially optimal solutions not dominated by others. Resistance-specific hybrid feature set includes NLP-distilled, graph-derived clinical and preclinical features. Assigning different importance to features, users can re-rank results according to the specific use case requirements.

7 CONCLUSION

The focus of the BIKG project is to build an internal knowledge graph leveraging both internal and external BIKG data, such that it can be used for applying machine learning algorithms and gaining new insights. This approach has demonstrated its value in several use cases, such as CRISPR recommendation and target identification. Integrating various public datasets served as a backbone on top of which internal use case-specific data was added and contextualised, giving enough information for the trained algorithms to function.

Beyond the positive impact from achieving these goals, our experience has shown that such a knowledge graph also provides secondary benefits at the organisation level:

- Reduced time for data preparation work: common data preparation tasks like aligning the data formats and identity resolution are taken care when building the graph, which leaves scientists more time for actual research.
- Improved quality: dedicated APIs and templates enable quick addition of data on demand as well as stakeholder feedback.
- Reduced costs: the core graph as well as common task solutions are reusable across use cases and organisation units.

In our future work we are going to concentrate on the adaptation of the BIKG graph to new use cases and improving its suitability for novel machine learning techniques, such as graph neural networks, reinforcement learning, and explainable AI.

REFERENCES

[1] Robert Arp, Barry Smith, and Andrew D Spear. 2015. *Building ontologies with basic formal ontology.* Mit Press.

[2] M Ashburner, C A Ball, J A Blake, D Botstein, H Butler, J M Cherry, A P Davis, K Dolinski, S S Dwight, J T Eppig, M A Harris, D P Hill, L Issel-Tarver, A Kasarskis, S Lewis, J C Matese, J E Richardson, M Ringwald, G M Rubin, and G Sherlock. 2000. Gene ontology: tool for the unification of biology. The Gene Ontology Consortium. *Nat Genet* 25, 1 (May 2000), 25–29. https://doi.org/10.1038/75556

[3] A. L. Barabási, N. Gulbahce, and J. Loscalzo. 2011. Network medicine: a network-based approach to human disease. *Nat Rev Genet* 12, 1 (Jan 2011), 56–68.

[4] François Belleau, Marc-Alexandre Nolin, Nicole Tourigny, Philippe Rigault, and Jean Morissette. 2008. Bio2RDF: Towards a mashup to build bioinformatics knowledge systems. *Journal of Biomedical Informatics* 41, 5 (2008), 706–716. https://doi.org/10.1016/j.jbi.2008.03.004 Semantic Mashup of Biomedical Data.

[5] Yoshua Bengio, Aaron Courville, and Pascal Vincent. 2013. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence* 35, 8 (2013), 1798–1828.

[6] Stephen Bonner, Ian P Barrett, Cheng Ye, Rowan Swiers, Ola Engkvist, 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:2102.10062* (2021).

[7] Anna Breit, Simon Ott, Asan Agibetov, and Matthias Samwald. 2020. OpenBioLink: a benchmarking framework for large-scale biomedical link prediction. *Bioinformatics* 36, 13 (04 2020), 4097–4098. https://doi.org/10.1093/bioinformatics/btaa274 arXiv:https://academic.oup.com/bioinformatics/article-pdf/36/13/4097/33458979/btaa274.pdf

[8] Brad Calder, Ju Wang, Aaron Ogus, Niranjan Nilakantan, Arild Skjolsvold, Sam McKelvie, Yikang Xu, Shashwat Srivastav, Jiesheng Wu, Huseyin Simitci, et al. 2011. Windows Azure Storage: a highly available cloud storage service with strong consistency. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*. 143–157.

[9] James Campbell. 2020. Great Expectations. https://github.com/great-expectations/great_expectations.

[10] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 785–794.

[11] Gabor Csardi and Tamas Nepusz. 2006. The igraph software package for complex network research. *InterJournal* Complex Systems (2006), 1695. https://igraph.org

[12] Ankur Dave, Alekh Jindal, Li Erran Li, Reynold Xin, Joseph Gonzalez, and Matei Zaharia. 2016. GraphFrames: An Integrated API for Mixing Graph and Relational Queries. In *Proceedings of the Fourth International Workshop on Graph Data Management Experiences and Systems* (Redwood Shores, California) *(GRADES '16)*. Association for Computing Machinery, New York, NY, USA, Article 2, 8 pages. https://doi.org/10.1145/2960414.2960416

[13] Wouter De Kort. 2016. *DevOps on the Microsoft Stack*. Springer.

[14] Kalyanmoy Deb. 2011. Multi-objective optimisation using evolutionary algorithms: an introduction. In *Multi-objective evolutionary optimisation for product design and manufacturing*. Springer, 3–34.

[15] Emek Demir, Michael Cary, Suzanne Paley, Ken Fukuda, Christian Lemer, Imre Vastrik, Guanming Wu, Peter D'Eustachio, Carl Schaefer, Joanne Luciano, Frank Schacherer, Zhenjun Hu, Veronica Jimenez-Jacinto, Geeta Joshi-Tope, Richard Kumaran Kandasamy, Alejandra López-Fuentes, Huaiyu Mi, Elgar Pichler, and Gary Bader. 2010. BioPAX – A community standard for pathway data sharing. *Nature biotechnology* 28 (09 2010), 935–42. https://doi.org/10.1038/nbt1210-1308c

[16] Leila Etaati. 2019. Azure Databricks. In *Machine Learning with Microsoft Technologies*. Springer, 159–171.

[17] Thomas Gaudelet, Ben Day, Arian R. Jamasb, Jyothish Soman, Cristian Regep, Gertrude Liu, Jeremy B. R. Hayter, Richard Vickers, Charles Roberts, Jian Tang, David Roblin, Tom L. Blundell, Michael M. Bronstein, and Jake P. Taylor-King. 2021. Utilising Graph Machine Learning within Drug Discovery and Development. arXiv:2012.05716 [q-bio.QM]

[18] A. Gaulton, A. Hersey, M. Nowotka, A. P. Bento, J. Chambers, D. Mendez, P. Mutowo, F. Atkinson, L. J. Bellis, E. Cibrián-Uhalte, M. Davies, N. Dedman, A. Karlsson, M. P. Magariños, J. P. Overington, G. Papadatos, I. Smit, and A. R. Leach. 2017. The ChEMBL database in 2017. *Nucleic Acids Res* 45, D1 (01 2017), D945–D954.

[19] Clinton Gormley and Zachary Tong. 2015. *Elasticsearch: the definitive guide: a distributed real-time search and analytics engine.* " O'Reilly Media, Inc.".

[20] Deisy Morselli Gysi, Ítalo Do Valle, Marinka Zitnik, Asher Ameli, Xiao Gan, Onur Varol, Susan Dina Ghiassian, JJ Patten, Robert Davey, Joseph Loscalzo, and Albert-László Barabási. 2020. Network Medicine Framework for Identifying Drug Repurposing Opportunities for COVID-19. arXiv:2004.07229 [q-bio.MN]

[21] 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).

[22] Olaf Hartig and Jorge Pérez. 2018. Semantics and complexity of GraphQL. In *Proceedings of the 2018 World Wide Web Conference*. 1155–1164.

[23] Daniel Scott Himmelstein, Antoine Lizee, 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), e26726.

[24] Vassilis N. Ioannidis, Xiang Song, Saurav Manchanda, Mufei Li, Xiaoqin Pan, Da Zheng, Xia Ning, Xiangxiang Zeng, and George Karypis. 2020. DRKG - Drug Repurposing Knowledge Graph for Covid-19. https://github.com/gnn4dr/DRKG/.

[25] Hrant Khachatrian, Lilit Nersisyan, Karen Hambardzumyan, Tigran Galstyan, Anna Hakobyan, Arsen Arakelyan, Andrey Rzhetsky, and Aram Galstyan. 2019. BioRelEx 1.0: Biological Relation Extraction Benchmark. In *Proceedings of the 18th BioNLP Workshop and Shared Task*. 176–190.

[26] Warren Kibbe, Cesar Arze, Victor Felix, Elvira Mitraka, Evan Bolton, Gang Fu, Christopher Mungall, Janos Binder, James Malone, Drashtti Vasant, Helen Parkinson, and Schriml Lynn. 2014. Disease Ontology 2015 update: An expanded and updated database of Human diseases for linking biomedical knowledge through disease data. *Nucleic acids research* 43 (10 2014). https://doi.org/10.1093/nar/gku1011

[27] Sunghwan Kim, Jie Chen, Tiejun Cheng, Asta Gindulyte, Jia He, Siqian He, Qingliang Li, Benjamin A Shoemaker, Paul A Thiessen, Bo Yu, Leonid Zaslavsky, Jian Zhang, and Evan E Bolton. 2020. PubChem in 2021: new data content and improved web interfaces. *Nucleic Acids Research* 49, D1 (11 2020), D1388–D1395. https://doi.org/10.1093/nar/gkaa971 arXiv:https://academic.oup.com/nar/article-pdf/49/D1/D1388/35363961/gkaa971.pdf

[28] Scott Klein. 2017. Azure data factory. In *IoT Solutions in Microsoft's Azure IoT Suite*. Springer, 105–122.

[29] Holger Knublauch and Dimitris Kontokostas. 2017. Shapes Constraint Language (SHACL), W3C Recommendation 20 July 2017. *URL: https://www. w3. org/TR/shacl* (2017).

[30] Keshav Kolluru, Vaibhav Adlakha, Samarth Aggarwal, Mausam, and Soumen Chakrabarti. 2020. OpenIE6: Iterative Grid Labeling and Coordination Analysis for Open Information Extraction. arXiv:2010.03147 [cs.CL]

[31] Gautier Koscielny, Peter An, Denise Carvalho-Silva, Jennifer A Cham, Luca Fumis, Rippa Gasparyan, Samiul Hasan, Nikiforos Karamanis, Michael Maguire, Eliseo Papa, et al. 2017. Open Targets: a platform for therapeutic target identification

and validation. *Nucleic acids research* 45, D1 (2017), D985–D994.
[32] Ora Lassila, Ralph R Swick, et al. 1998. Resource description framework (RDF) model and syntax specification. (1998).
[33] Man-Tat Lau, Shila Ghazanfar, Ashleigh Parkin, Angela Chou, Jourdin R. Rouaen, Jamie B. Littleboy, Danielle Nessem, Thang M. Khuong, Damien Nevoltris, Peter Schofield, David Langley, Daniel Christ, Jean Yang, Marina Pajic, and G. Gregory Neely. 2020. Systematic functional identification of cancer multi-drug resistance genes. *Genome Biology* 21, 1 (07 Feb 2020), 27. https://doi.org/10.1186/s13059-020-1940-8
[34] Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, and Jaewoo Kang. 2020. BioBERT: a pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics* 36, 4 (2020), 1234–1240.
[35] Steffen Lohmann, Stefan Negru, Florian Haag, and Thomas Ertl. 2016. Visualizing Ontologies with VOWL. *Semantic Web* 7, 4 (2016), 399–419. https://doi.org/10.3233/SW-150200
[36] Wes 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.
[37] Christopher J. Mungall, Carlo Torniai, Georgios V. Gkoutos, Suzanna E. Lewis, and Melissa A. Haendel. 2012. Uberon, an integrative multi-species anatomy ontology. *Genome Biology* 13, 1 (31 Jan. 2012). https://doi.org/10.1186/gb-2012-13-1-r5 Copyright: Copyright 2012 Elsevier B.V., All rights reserved..
[38] Christopher J. Mungall, Julie A. McMurry, Sebastian Köhler, James P. Balhoff, Charles Borromeo, Matthew Brush, Seth Carbon, Tom Conlin, Nathan Dunn, Mark Engelstad, Erin Foster, J.P. Gourdine, Julius O.B. Jacobsen, Dan Keith, Bryan Laraway, Suzanna E. Lewis, Jeremy NguyenXuan, Kent Shefchek, Nicole Vasilevsky, Zhou Yuan, Nicole Washington, Harry Hochheiser, Tudor Groza, Damian Smedley, Peter N. Robinson, and Melissa A. Haendel. 2016. The Monarch Initiative: an integrative data and analytic platform connecting phenotypes to genotypes across species. *Nucleic Acids Research* 45, D1 (11 2016), D712–D722. https://doi.org/10.1093/nar/gkw1128 arXiv:https://academic.oup.com/nar/article-pdf/45/D1/D712/8846933/gkw1128.pdf
[39] Robert Ian Nicholson, Julia Margaret Wendy Gee, and Maureen Elaine Harper. 2001. EGFR and cancer prognosis. *European journal of cancer* 37 (2001), 9–15.
[40] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. 2011. A three-way model for collective learning on multi-relational data. In *Icml*.
[41] Andrea Pierleoni. 2018. Introducing LINK: the Open Targets Literature Knowledge Graph. https://blog.opentargets.org/link/
[42] Srivamshi Pittala, William Koehler, Jonathan Deans, Daniel Salinas, Martin Bringmann, Katharina Sophia Volz, and Berk Kapicioglu. 2020. Relation-weighted Link Prediction for Disease Gene Identification. arXiv:2011.05138 [cs.LG]
[43] Sampo Pyysalo, Filip Ginter, Juho Heimonen, Jari Björne, Jorma Boberg, Jouni Järvinen, and Tapio Salakoski. 2007. BioInfer: a corpus for information extraction in the biomedical domain. *BMC bioinformatics* 8, 1 (2007), 1–24.
[44] Bernadette M Randles, Irene V Pasquetto, Milena S Golshan, and Christine L Borgman. 2017. Using the Jupyter notebook as a tool for open science: An empirical study. In *2017 ACM/IEEE Joint Conference on Digital Libraries (JCDL)*. IEEE, 1–2.
[45] Richard J Roberts. 2001. PubMed Central: The GenBank of the published literature.
[46] Ryan A Rossi, Di Jin, Sungchul Kim, Nesreen K Ahmed, Danai Koutra, and John Boaz Lee. 2020. On proximity and structural role-based embeddings in networks: Misconceptions, techniques, and applications. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 14, 5 (2020), 1–37.
[47] Alberto Santos, Ana R. Colaço, Annelaura B. Nielsen, Lili Niu, Philipp E. Geyer, Fabian Coscia, Nicolai J Wewer Albrechtsen, Filip Mundt, Lars Juhl Jensen, and Matthias Mann. 2020. Clinical Knowledge Graph Integrates Proteomics Data into Clinical Decision-Making. *bioRxiv* (2020). https://doi.org/10.1101/2020.05.09.084897 arXiv:https://www.biorxiv.org/content/early/2020/05/10/2020.05.09.084897.full.pdf
[48] Ariel S Schwartz and Marti A Hearst. 2002. A simple algorithm for identifying abbreviation definitions in biomedical text. In *Biocomputing 2003*. World Scientific, 451–462.
[49] SciBite. 2021. Termite, https://www.scibite.com/platform/termite/. https://www.scibite.com/platform/termite/
[50] Arnab Sinha, Zhihong Shen, Yang Song, Hao Ma, Darrin Eide, 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*. 243–246.
[51] Barry Smith, M. Ashburner, C. Rosse, J. Bard, W. Bug, W. Ceusters, L. Goldberg, Karen Eilbeck, A. Ireland, C. Mungall, N. Leontis, P. Rocca-Serra, A. Ruttenberg, Susanna-Assunta Sansone, R. Scheuermann, N. Shah, Patricia L. Whetzel, and S. Lewis. 2007. The OBO Foundry: coordinated evolution of ontologies to support biomedical data integration. *Nature Biotechnology* 25 (2007), 1251–1255.
[52] Michael E Smoot, Keiichiro Ono, Johannes Ruscheinski, Peng-Liang Wang, and Trey Ideker. 2011. Cytoscape 2.8: new features for data integration and network visualization. *Bioinformatics* 27, 3 (2011), 431–432.
[53] Giorgos Stoilos, David Geleta, Szymon Wartak, Sheldon Hall, Mohammad Khodadadi, Yizheng Zhao, Ghadah Alghamdi, and Renate A Schmidt. 2018. Methods and Metrics for Knowledge Base Engineering and Integration.. In *WOP@ ISWC*. 72–86.
[54] RAPIDS Development Team. 2018. *RAPIDS: Collection of Libraries for End to End GPU Data Science*. https://rapids.ai
[55] Thiago Teixeira. 2018. Streamlit. https://github.com/streamlit/streamlit.
[56] The UniProt Consortium. 2020. UniProt: the universal protein knowledgebase in 2021. *Nucleic Acids Research* 49, D1 (11 2020), D480–D489. https://doi.org/10.1093/nar/gkaa1100 arXiv:https://academic.oup.com/nar/article-pdf/49/D1/D480/35364103/gkaa1100.pdf
[57] Deepak Vohra. 2016. Apache parquet. In *Practical Hadoop Ecosystem*. Springer, 325–335.
[58] Jim Webber. 2012. A programmatic introduction to neo4j. In *Proceedings of the 3rd annual conference on Systems, programming, and applications: software for humanity*. 217–218.
[59] Mark D Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten, Luiz Bonino da Silva Santos, Philip E Bourne, et al. 2016. The FAIR Guiding Principles for scientific data management and stewardship. *Scientific data* 3 (2016).
[60] Colby Wise, Vassilis N. Ioannidis, Miguel Romero Calvo, Xiang Song, George Price, Ninad Kulkarni, Ryan Brand, Parminder Bhatia, and George Karypis. 2020. COVID-19 Knowledge Graph: Accelerating Information Retrieval and Discovery for Scientific Literature. arXiv:2007.12731 [cs.IR]
[61] Shanchan Wu and Yifan He. 2019. Enriching pre-trained language model with entity information for relation classification. In *Proceedings of the 28th ACM international conference on information and knowledge management*. 2361–2364.
[62] Andrew D Yates, Premanand Achuthan, Wasiu Akanni, James Allen, Jamie Allen, Jorge Alvarez-Jarreta, M Ridwan Amode, Irina M Armean, Andrey G Azov, Ruth Bennett, Jyothish Bhai, Konstantinos Billis, Sanjay Boddu, José Carlos Marugán, Carla Cummins, Claire Davidson, Kamalkumar Dodiya, Reham Fatima, Astrid Gall, Carlos Garcia Giron, Laurent Gil, Tiago Grego, Leanne Haggerty, Erin Haskell, Thibaut Hourlier, Osagie G Izuogu, Sophie H Janacek, Thomas Juettemann, Mike Kay, Ilias Lavidas, Tuan Le, Diana Lemos, Jose Gonzalez Martinez, Thomas Maurel, Mark McDowall, Aoife McMahon, Shamika Mohanan, Benjamin Moore, Michael Nuhn, Denye N Oheh, Anne Parker, Andrew Parton, Mateus Patricio, Manoj Pandian Sakthivel, Ahamed Imran Abdul Salam, Bianca M Schmitt, Helen Schuilenburg, Dan Sheppard, Mira Sycheva, Marek Szuba, Kieron Taylor, Anja Thormann, Glen Threadgold, Alessandro Vullo, Brandon Walts, Andrea Winterbottom, Amonida Zadissa, Marc Chakiachvili, Bethany Flint, Adam Frankish, Sarah E Hunt, Garth IIsley, Myrto Kostadima, Nick Langridge, Jane E Loveland, Fergal J Martin, Joannella Morales, Jonathan M Mudge, Matthieu Muffato, Emily Perry, Magali Ruffier, Stephen J Trevanion, Fiona Cunningham, Kevin L Howe, Daniel R Zerbino, and Paul Flicek. 2019. Ensembl 2020. *Nucleic Acids Research* 48, D1 (11 2019), D682–D688. https://doi.org/10.1093/nar/gkz966 arXiv:https://academic.oup.com/nar/article-pdf/48/D1/D682/31697830/gkz966.pdf
[63] Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui 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
[64] Da Zheng, Xiang Song, Chao Ma, Zeyuan Tan, Zihao Ye, Jin Dong, Hao Xiong, Zheng Zhang, and George Karypis. 2020. Dgl-ke: Training knowledge graph embeddings at scale. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 739–748.
[65] Marinka Zitnik, Monica Agrawal, and Jure Leskovec. 2018. Modeling polypharmacy side effects with graph convolutional networks. *Bioinform.* 34, 13 (2018), i457–i466. https://doi.org/10.1093/bioinformatics/bty294