

# DynDiff: A Tool for Comparing Versions of Large Ontologies

Sara Diaz Benavides<sup>1,†</sup>, Silvio Domingos Cardoso<sup>2,†</sup>, Marcos Da Silveira<sup>3,\*,†</sup> and Cédric Pruski<sup>3,†</sup>

<sup>1</sup>The Data Analysis Bureau 41-47 OLD ST, LONDON, EC1V 9AE

<sup>2</sup>Dynaccurate 9, Avenue des Hauts-Fourneaux, 4362 Esch-sur-Alzette, Luxembourg

<sup>3</sup>Luxembourg Institute of Science and Technology

5, avenue des Hauts-Fourneaux, L-4362 Esch-sur-Alzette, Luxembourg

## Abstract

Ontologies have been widely adopted to represent domain knowledge. The dynamic nature of knowledge requires frequent changes in the ontologies to keep them up-to-date. Understanding and managing these changes and their impact on other artefacts become important for the semantic web community, due to the growing volume of data annotated with ontologies and the limited documentation describing their changes. In this paper, we present a method to automatically detect and classify the changes between different versions of ontologies. We also built an ontology of changes (DynDiffOnto) that we use to classify the changes and provide a context that makes them more comprehensible for machines and humans. We evaluate the algorithm with different ontologies from the biomedical domain (i.e. ICD9-CM, MeSH, NCI, SNOMED-CT, GO, IOBC, and CIDO) and we compare the results with COnTo-Diff. We observed that for small ontologies, COnTo-Diff computes the Diff faster, but the opposite is observed for larger ontologies. The higher granularity of DynDiffOnto requires more rules to compute the Diff. It can partially justify the lower performance for small ontologies, but DynDiff provides a richer documentation for end-users.

## Keywords

Diff Computation, Ontology Management, Ontology Evolution, Knowledge Graph

## 1. Introduction

The large amount of semantically annotated knowledge publicly available on the Web makes the need for efficient and automatic solutions to keep this knowledge consistent and updated more evident. This is especially true for Knowledge Graphs (KGs) that use concepts of ontologies to define the elements that they describe (objects, events, situations, etc.). In the fields of life sciences, providers of controlled terminologies quickly adopted ontologies to better represent

---

*SeWebMeDa'22: Workshop on Semantic Web solutions for large-scale biomedical data analytics, May 29th, 2022, Hersonissos, Greece*

\*Corresponding author.

†These authors contributed equally.


✉ sara.diaz052@gmail.com (S. D. Benavides); silvio.cardoso@dynaccurate.com (S. D. Cardoso);

marcos.dasilveira@list.lu (M. D. Silveira); cedric.pruski@list.lu (C. Pruski)

🌐 <https://marcao02.github.io/> (M. D. Silveira)

🆔 0000-0002-2604-3645 (M. D. Silveira); 0000-0002-2103-0431 (C. Pruski)

© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

their vocabularies. For instance, there are more than 800 biomedical ontologies published in Bioportal [1], including SNOMED-CT, ICD9-CM, MeSH, NCI, gene ontology, CIDO, and IOBC. They constitute an important source of semantic information for reporting on diagnoses, medical procedures, medications, body characteristics, etc. and are widely used by healthcare professionals and institutions involved in providing health services to patients [2, 3, 4]. These ontologies are also used for data-intensive tasks, such as analysing treatments/drug interactions [5, 6], enabling semantic interoperability between information systems [7], and combining and disseminating pandemic information [8], etc. Advances in biomedical knowledge need to be translated into the underlying ontologies, which requires the addition/removal of information to clarify or correct the description of existing concepts [9]. The impact of these changes propagates to all other artefacts that use these modified ontologies. For instance, annotations of medical reports (metadata) can become invalid or not findable if the concept used is modified or removed in subsequent versions. The reproduction of experiments can lead to different outcomes if the metadata of documents are not findable, etc. A significant manual effort is necessary to keep metadata updated. The three main causes are: the huge size of the ontologies (e.g. SNOMED CT has more than 340,000 concepts while IOBC has more than 120,000 concepts and 60,000 individuals); the high frequency of changes; and the lack of machine-understandable documentation regarding the ontology evolution.

In our previous work, we studied the impact of ontology changes on associated mappings [10] and semantic annotations [11]. Based on these findings, we have designed methods to (semi-)automatically maintain mappings [12] and annotations [13] rendered invalid by the evolution of underlying ontologies. A key element for assuring the quality of the outcomes of these methods is the *Diff* tool, a tool used to compute and reason on the changes between two ontology versions [14, 15, 16]. We looked for tools that provided detailed classification of changes. The two main goals were: to search for change classification patterns that facilitate inferring the consequences of these changes on other artefacts; and to recommend maintenance actions to resolve some critical problems (e.g. broken mappings, invalid semantic annotations, etc.) caused by these changes. We first based our solution on the outcomes of COnTo-Diff, a flexible and efficient tool to compute Diffs between ontology versions. Other tools were evaluated during the selection process, like the widely adopted Promptdiff [14] that focuses on changes in the labels or the work by Gonçalves et al. [16] that focuses on the changes in axioms. Despite the relevance of their work, their approach did not provide the level of details that we needed for our analysis nor did their tool were flexible enough to allow new specific rules to be written to identify more change types. Moreover, we were looking for tools that accepted inputs in OWL and OBO, provided an outcome in a reasonable time for large ontologies ( $\leq 70.000$  classes), computed diffs between classes and between individuals, could be specific on the property that was changed within the concept/individual, could present a more abstract/understandable description of changes (e.g. present a Move concept instead of two different actions: Del relation + Add relation), and that could decouple deterministic changes (e.g. addLeaf) from undeterministic changes (based on heuristics, like for a split), giving end-users the possibility of benefiting from advances in matching heuristics to improve the Diff outcomes. COnTo-Diff was the tool that satisfied almost all our criteria. Inspired by the best features of COnTo-Diff and by the characterization model presented in [17], we decided to develop DynDiff with the following characteristics:

- **Rule-based system:** It is based on specific change detection rules, which make up a change language that classifies the changed entities into different categories. These rules are created by a revision of the state-of-the art, aiming to gather the strongest points from existing approaches [15, 17].
- **Human and machine-interpretable:** The classification of changes provided by our approach is intuitive for humans while at the same time being granular enough to be useful for the machines. This classification is represented as an ontology of changes (DynDiffOnto) and is made available to the community at <https://git.list.lu/dynaccurate/change-ontology>.
- **Decoupled from heuristic semantics:** End-users have the possibility of configuring the matching algorithm used to compute some types of changes. This option allows a detachment from heuristic strategies that add a randomness to the results and allow end-users to benefit from the latest advances in the domain without changing the DynDiff code.
- **Scalable and robust:** The solution is able to work with different sized inputs (i.e. amount of ontological elements) and outputs (i.e. quantity of change types) in an efficient manner. Our experiments show that DynDiff is able to compute Diffs between large ontologies in a reasonable time (less than 1,000 sec to compute more than 70,000 changes)

Based on these characteristics, we made the following contributions:

1. A change language that extends those proposed by Hartung et al. and Papavasileiou et al. [15, 17] by proposing rules for individuals and making the distinction between the properties of type relationships and attributes. As evoked, many tools are restricted to classes only, limiting the exploitation of the data. The proposed change language is presented as an ontology of changes (DynDiffOnto) and the instantiation of the ontology's classes represents the changes that were detected between the different versions.
2. A change detection algorithm that provides a classification of changes based on DynDiffOnto classes. This structure provides transparency for the user in terms of the description of the results.

The remainder of the paper is structured as follows: In Section 2, we describe the various notions that are necessary to understand this work and we review related works. Section 3 presents our approach for comparing large ontologies. We start by introducing the DynDiff approach and continue with a description of the ontology we have designed to define the changes and the method used to identify them. Section 4 contains the experimental evaluation of our approach and Section 5 comprises concluding remarks and outlines future work.

## 2. Background

The granularity of the description of the changes is an important aspect that provides rich information about the evolution of the ontology. The change description can be low-level, when composed of add and remove changes only, or high-level, when low-level changes are regrouped in order to report a more intuitive interpretation of these changes. In DynDiffOnto,

the *Basic changes* class belongs to the low-level changes. These are easier to detect and provide detailed information for machine processes. However, the quantity of basic changes can easily grow and overwhelm human users with details that are not easy to interpret. Thus, we propose combining some of these basic changes to compose two types of high-level changes: *Complex changes* and *Heuristic changes*. These describe more concise deltas in an understandable format. The main challenge of this approach is to make the distinction between structural changes (e.g. add a sub-graph) and semantic changes (e.g. splitting a concept). In DynDiffOnto, all high-level changes that can be detected using logical rules are grouped into *Complex changes* and the other changes, detected based on non-deterministic methods, are grouped into *Heuristic changes*. In the next section, we detail the problem and the notations used.

## 2.1. Problem statement

Ontology Diff approaches look for a set of rules or heuristics that enable the detection and description of changes between two versions of the same ontology  $\text{Diff}(O_{old}, O_{new})$ . In our work, an ontology is a set of RDF triples

$$\mathcal{T} = \mathbf{U} \times \mathbf{U} \times (\mathbf{U} \cup \mathbf{L})$$

[17] where  $\mathbf{U}$  denotes resources (all things described by RDF) and  $\mathbf{L}$  denotes literals (values such as strings and integers). The general problem that we address in our work can be described as follows: Given two versions of an ontology ( $O_{old}$  and  $O_{new}$ ), we search for:

1. Determining the set of all **basic changes**  $BC$  computed according to the definition given by [17]:

$$BC = \{t \in \mathcal{T} \mid t \in (O_{new} \setminus O_{old}) \text{ or } t \in (O_{old} \setminus O_{new})\} \quad (1)$$

2. Determining the sets of **complex changes**  $CC \subseteq BC$  and **heuristic changes**  $HC \subseteq BC$  where  $CC$  and  $HC$  are disjoint, i.e.  $CC \cap HC = \emptyset$ .
3. Determining the subsets of **basic changes**  $BC^*$  that has all basic changes that were not used to compose neither  $HC$  nor  $CC$ .

## 2.2. Related work

The Diff problem is not new and has already being approached in different ways. Logging or *structural diff* [18, 19, 20] is the simplest and most used diff method. The *logging* method makes use of a log file, often generated manually by experts, where changes are added incrementally. The file is updated every time a new modification takes place, and the changes can be searched for by using temporal queries in this log file. An automatic log generator was also proposed, where the changes were calculated *a posteriori* in two input versions. This solution solved some inconsistency problems detected in the manual log generation process. One of the first implementations of this type of approach was PromptDiff [14], where changes are computed by inputting the two versions of the ontology and applying a set of rules. The outcomes are presented as instances of the Changes and Annotation Ontology (ChAO) [21], but this ontology was designed to facilitate collaborations between humans and to generate a manual annotation of users' actions (e.g. whether s/he accepted a change and providing a way of commenting on

why), a lot of information was expressed in the comments, and types of changes were limited to just a few classes. This format was widely adopted by Protégé users, but the format (i.e., semantics of changes) remains proprietary since there is no consensus on a standard description language for the changes. Another type of approach proposes using the axioms of the ontologies to detect the consequences of the changes in the semantics of the new version of the ontology [16]. The resulting changes are complementary to the previous approaches, but they are still not intuitive for end-users. The size of the ontologies and the frequency of changes push for more elaborated solutions to compute the Diff. *COnto-Diff* (Complex Ontology Diff) [15] is a tool that proposes a two-phase *diff* approach; during the first phase, there is a matching algorithm to determine conceptual mappings between versions. The second phase makes use of these mappings to generate the high-level changes (e.g. merges and splits). The authors evaluated the tool with ontologies described in OWL and OBO, demonstrating the efficiency of the tool to detect changes in these languages. Another relevant approach is presented by Papavasileiou et al. [17]. Their work was designed to compute the diff between RDFS datasets. The authors propose a language of changes with 132 change actions, complete application semantics and the proof of completeness, consistency, uniqueness and non-overlapping classes. Their algorithm starts by computing a simple delta to identify all triples that change (i.e., added and deleted triples) and then regroups these triples to compose more human-representative changes (Basic, Composite, and Heuristic Changes). The approach prevents triples being re-used to compose more than one type of change. One of the innovative aspects of this language is that it takes into account the schema (i.e. classes and properties), instance level (i.e. individuals) and group of these schema entities (i.e. meta-classes and meta-properties). These five groups allow them to focus on the nodes rather than the edges of the graph and they consider that this makes the language more human-intuitive. The approach was designed to apply to RDFS, limiting its use to more sophisticated axioms like those provided by OWL. The *matcher* is an optional feature to map the changes and establish potential semantic links between them.

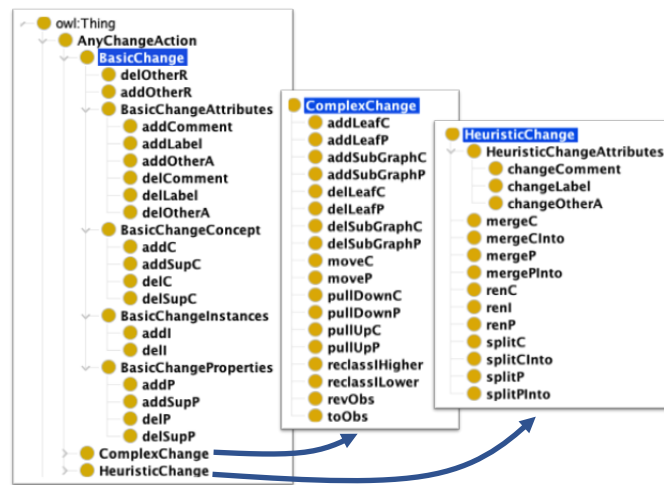
Regarding existing work reviewed in this section, our approach differs in the following way. On the one hand, we are going beyond the capabilities offered by *COnto-Diff* in the sense that *DynDiff* is able to consider Properties and Individuals while *COnto-Diff* focused on Concepts. Moreover, *DynDiff* shows more stability regarding its execution time independently of the ontology given as input which improve the scalability of the algorithm. On the other hand, our approach added more granularity in the changes' classification (or change actions) for the same amount of change triples (e.g. we do not only report `addRelationship` but this is split into `addSubClass`, `addSubProperty` and `addOtherRelationship`) than the approach of Papavasileiou et al. [17]. In the following section we will detail our method and compare it experimentally with related work.

### 3. *DynDiff* for comparing large and dynamic ontologies

In this section we describe our approach for comparing large and dynamic ontologies and producing the change actions. We start by introducing the main concepts of *DynDiffOnto*. It is an ontology that we defined to represent the semantics of detectable change actions. Later, we will describe the *DynDiff* algorithm. The outcomes of this algorithm are the instances of

DynDiffOnto.

The DynDiffOnto represents 60 different **change actions** (9 abstract classes and 51 leaf classes). The selection process started by applying two different methods that compute Diffs (i.e. those presented in [17, 15]) to a set of biomedical ontologies. We compared the outcomes and regrouped the changes with the same semantic meaning. This first set of changes all becomes part of DynDiffOnto. For the others, we analysed them one-by-one and select the classes according to the precision of the changes or its intuitiveness. The main inspiration comes from the work of Papavasileiou et al. [17], and we extended the definitions of some rules according to the work of [15], and added new rules according to our own experience of evaluating the impact of changes in other artefacts presented in [12, 11]. Table 1 presents part of the outcomes of our analysis and Figure 1 illustrates the organization of DynDiffOnto. A complete table of change actions and the ontology can be downloaded from the project repository<sup>1</sup>.



**Figure 1:** Illustration of the main classes of DynDiffOnto

As figure 1 shows, the change actions were grouped into three different classes: Basic, Complex, and Heuristic changes.

- **Basic changes:** this is the set of the low-level changes that are meant to be simple and straightforward to be machine-understandable. Examples would be addition of Concept (abbreviated as addC).
- **Heuristic changes:** It is composed of at least one basic change and the main characteristic of these high-level changes are that they depend on the mappings (see Fig. 2) outputted by the user-selected matcher, which is based on heuristics which gives them their name. An example of these changes would be a split concept (abbreviated as splitC ) operation which required there to have a mapping between the origin concept and the other concept nodes its split into. As their name implies, this changes highly depend on the matcher used and thus are non-deterministic depending on this variable.

<sup>1</sup><https://git.list.lu/dynaccurate/change-ontology>



Change Action	[Papavasileiou,2013]	[Hartung,2013]
addC	Add_Type_Class	addC
addLeafC	-	addLeaf
addLeafP	-	-
addSupC	Add_Superclass	$\subseteq$ addR
addComment	Add_Comment	$\subseteq$ addA
addOtherA	$\subseteq$ Add_Property_Instance	$\subseteq$ addA
renI	Rename_Individual	-
mergeC	Merge_Classes	$\subseteq$ merge
mergeP	Merge_Properties	-
changeOtherA	-	$\subseteq$ chgAttValue
moveC	$\supseteq$ Move_Classes	move
pullUpC	Pull_up_Class	-
reclassHigher	Reclassify_Individual_Higher	-
toObs	-	toObsolete

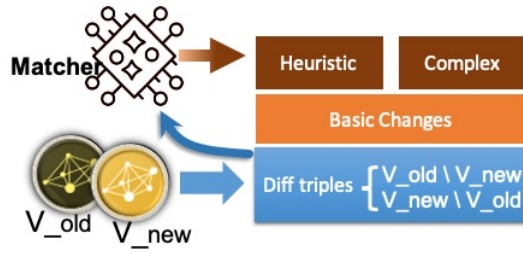
**Table 1**

A subset of DynDiffOnto’s change actions and the closest change actions from [17, 15]. The symbol “ $\subseteq$ ” indicates a more general definition and “ $\supseteq$ ” indicates a more specific definition.

- **Complex changes:** in contrast to the previous, these high-level changes are deterministic, independent of the matcher, and equally apply rules over to compose basic changes. This is why we separated them from the heuristic changes, as it would make the determinism more transparent to the user. A type of change to highlight here would be the subgraph related changes.

By combining the idea of having the class change attributes (from [15]) and the classes change label and comments (from [17]), we created the BasicChangeAttributes and HeuristicChangeAttributes classes that regroup the changes that can be made into comments, labels, and other attributes describing classes, individuals, or properties. For instance, assuming that the following modifications were made in the old version of an ontology: adding the property `rdfs:comment` and assigning a textual description to comment each class, DinDiff will detect two different types of change actions: (1) the *addP* (add property) from the BasicChangeProperties; and (2) the *addA* (add the text describing the class) from the BasicChangeAttribute. If one existing comment is slightly modified in a future version of the ontology, then DinDiff will detect a HeuristicChange (i.e., *changeComment*). These distinctions are important when we want to analyse the impact of changes in Semantic Mappings and Semantic Annotations because we use lexical and semantic similarity measures for this analysis [13]. The lexical similarity measures use information from attributes while semantic similarity measures take the relationships into account. We are currently working on a new representation format for the history of an ontology [22] that tracks changes in OtherA and OtherR classes of DynDiffOnto.

Figure 2 gives an overview of how DynDiff works and Algorithm 1 shows the sequence of actions implemented. In this work we mainly implemented the algorithms proposed by Papavasileiou et al. [17] to generate the low-level triples, basic changes and complex changes. We adopted the approach of Hartung et al. [15] to compute the heuristic changes. This decision ensures that all proofs presented in [17] also apply to this work. Readers can find a detailed



**Figure 2:** DynDiff overview.  $V_{old}$  and  $V_{new}$  are used to identify the changed triples. The matcher build mappings between triples that are used to detect heuristic change actions. Deterministic rules are used to compute basic and complex change.

explanation of the algorithm in their respective referenced articles. In this paper, we will introduce our algorithm and explain in more detail the experiments we did and the comparison with other approaches.

The blue box (figure 2) represents the set of low-level changes, computed as described by Equation (1) (algorithm 1, line 1). This set includes all added and removed triples and is implemented in the same way that is described in the work by Papavasileiou et al.[17]. These triples are used as the input for the Matcher (algorithm 1, line 3) and to compute the basic changes (algorithm 1, line 4). The *Basic changes* box consumes the triples to compose simpler changes (e.g. add class or instance). The set of basic changes is reduced when some of them have been consumed by more complex changes (algorithm 2, line 9). This follows the principle of unambiguity (avoiding overlapped changes) proposed by Papavasileiou et al. [17]. The Matcher searches for potential links between the set of triples and expresses these links as Mappings. For this work, we adopted the same Matcher used by COnTo-Diff, but with a small modification that will also allow mappings to be generated for properties and instances. The reasons for adopting the same Matcher are: (1) To compare the performance of both approaches fairly; (2) To demonstrate good efficacy during our analysis. The comparison of Matchers is out of the scope of this paper (to respect the page limit), but we intend to publish it at a later date. We note that the difference in the mapping rule is:

COnTo-Diff rule:  $r \in R(Relationships) \wedge a, r \in O_{old} \wedge b \in O_{new} \wedge r_{source} = a \wedge r_{target} = b \wedge r \in \{“part\_of”, “is\_a”, “synonym”\} \rightarrow \mathbf{match(a,b)}$

DynDiff rule:  $a \in O_{old} \wedge b \in O_{new} \wedge matchC(a,b) \wedge a \neq b \wedge \neg isObsolete(a) \wedge \neg isObsolete(b) \rightarrow \mathbf{create [mapC(a,b)]}$

Similar DynDiff rules for properties and instances were defined. Since COnTo-Diff uses the OBO format to compare ontologies, the restriction on the relationships is justified. DynDiff uses subsumption relationships defined by RDFS (e.g. rdfs:subClassOf) and several formats for synonyms (e.g. skos:altLabel).

Giving the mappings and the basic changes, the *Heuristic changes* box can be computed (algorithm 1, line 5). These mappings are used to identify probabilistic (underterministic) relations between the basic changes in order to represent more abstract changes such as split, merge, change label, etc. (algorithm 2, line 6). The *Complex changes* box consumes the basic changes according to a set of deterministic rules in order to represent more abstract change



actions (e.g. moveC consumes addSupC and delSupC) (algorithm 2, line 8). The set of all rules used in this project can be downloaded from the project repository. The algorithms below will show how these rules were used to implement DynDiff.

---

**Algorithm 1** Compute diff:

---

**Input:**  $V_{old}, V_{new}$

**Output:** *diff* (BC, HC, CC)

- 1:  $T \leftarrow LowlevelChgs(V_{new}, V_{old})$
  - 2:  $BC, HC, CC \leftarrow \emptyset$
  - 3:  $M \leftarrow Matcher(V_{new}, V_{old})$
  - 4:  $BC \leftarrow findBasicChanges(T, M)$
  - 5:  $(BC, HC, CC) \leftarrow findHighLevelChanges(BC, M)$
- 

---

**Algorithm 2** Finding High-Level Changes

---

**Input:**  $BC, M$

**Output:** *findHighLevelChanges* (HC; CC, BC)

- 1: import *change\_Rules* ▷ this is our rules' library
  - 2:  $rule\_types = [heuristic\_change\_Rules, Complex\_change\_Rules]$
  - 3: **for**  $TRule \in rule\_types$  **do**
  - 4:     **for**  $rule \in changeRule.TRule$  **do** ▷ each rule from this type
  - 5:         **if**  $TRule = heuristic\_change\_Rules$  **then**
  - 6:              $HC, used\_basic\_changes \leftarrow applyRule(rule, BC, M)$
  - 7:         **else**
  - 8:              $CC, used\_basic\_changes \leftarrow applyRule(rule, BC, M)$
  - 9:          $BC \leftarrow updateBC(used\_basic\_changes)$  ▷ avoid reusing a BC
- 

## 4. Experimental evaluation

DynDiff has been experimentally evaluated on large ontologies. Our goal was to assess the performance of the algorithm on state-of-the-art tools in terms of execution time, as well as its ability to identify the right change actions.

### 4.1. Materials

During this evaluation, we used seventy-three versions from seven different ontologies. Table 2 provides some general information about them, such as the average number of classes (C), properties (P), individuals (I), subsumption relationships (R) and attributes (A).

We collected these ontologies from BioPortal [1]. For each of the dataset versions, the initial and last ID used are listed in the “versions” column, as well as the number of versions (between parentheses). We used the notation AA to indicate that for these ontologies we selected the latest version published in January of each year. For GO, IOBC, and CIDO we indicate in the

Acronym	Versions (#)	Average number of				
		<b>C</b>	<b>P</b>	<b>I</b>	<b>R</b>	<b>A</b>
ICD9CM	2005AA - 2016AA (12)	21,765	2	0	21,765	54,572
MESH	2005AA - 2016AA (12)	25,820	2	0	34,899	215,971
NCIT	2005AA - 2016AA (12)	70,396	2	0	79,525	179,983
SNOMEDCT	2005AA - 2016AA (12)	309,572	2	0	496,084	800,158
GO	01072017 - 21072020 (4)	52,574	54	0	492,651	395,285
IOBC	1.0.0 - 1.4.0 (7)	115,114	84	59,346	707,970	904,443
CIDO	01262020 - 06142020 (14)	3,599	165	294	15,368	25,407

**Table 2**

Average number of concepts **C**, properties **P**, individuals **I**, subsumption relationships **R** and attributes **A** in the different versions of each of the datasets. The *Versions* column also shows the number of versions collected.

table the first and last version used. We will only present the comparison with *COnto-Diff*, since the execution time of *COnto-Diff* was shorter than the approach of Papavasileiou et al. [17] for all ontologies.

## 4.2. Experimental method and metrics

We executed *DynDiff* and *COnto-Diff* algorithms on the same computing environment and used the following metrics to analyse the results obtained.

- Execution time: The objective was to analyse the behaviour and find trends in the execution time, given the characteristics (e.g. input size, output size and results composition in terms of changes) of the ontology; The aim of this analysis was to evaluate the scalability of the algorithm.
- % change tool comparison: The following metric was used to compare the results of the *DynDiff* with that of *COnto-Diff* [15]:

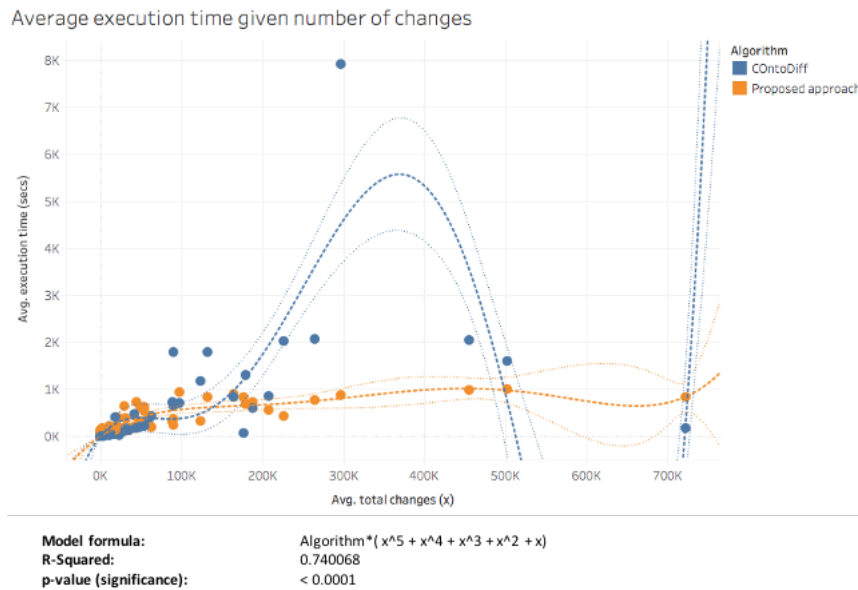
$$\%ChangeTools_{type} = \frac{AVG(N_{PA,type} - N_{CO,type})}{AVG(N_{CO,type})} \quad (2)$$

where  $type \in \{Basic, Composite\}$  (which includes both complex and heuristic),  $N$  is the number of changes of a given tool. The *COnto-Diff* serves as a *ground-truth* given that it was also tested against other existing solutions like *Promptdiff*. We thus compared the execution time metric and evaluate the proposed %changeTools metric. The aim of this last analysis will also to find an explanation for the differences in the high-to-low-level ratio.

## 4.3. Results

*DynDiff* was designed to have a balanced set of changes, avoiding a very detailed set (as for [17]), but with more detail than that of *COnto-Diff*. Thus, *DynDiff* is not a technical improvement of *COnto-Diff*, since they produce different outcomes (impacting the execution time). Before the analysis, we expected that the execution time for *COnto-Diff* to be lower.

Fig. 3 shows the average execution time of ten iterations of *COnto-Diff* and *DynDiff* for all versions of each ontology described in table 2. A polynomial trend of degree 5 (dashed lines), with their respective 95% confidence interval bands, show the possible adjusted behaviour of the algorithm. The model can be considered significant as  $\rho \leq 0.05$  and R-squared is close to 1. On average, *COnto-Diff* fluctuates more than *DynDiff* for the different computations. This is more evident for larger ontologies. There are also very evident outliers, as was observed when parsed ontologies were used. We only used OWL ontologies in this analysis, which required us to transform some ontologies that were originally in OBO into OWL. This transformation added some noises that impacted on the *COnto-Diff* algorithm. As we expected, both algorithms presented good results and close execution times, with a small advantage for *COnto-Diff* in the later criterion for small ontologies. However, the higher number of change actions did not avoid *DynDiff* outperforming *COnto-Diff* for larger ontologies.



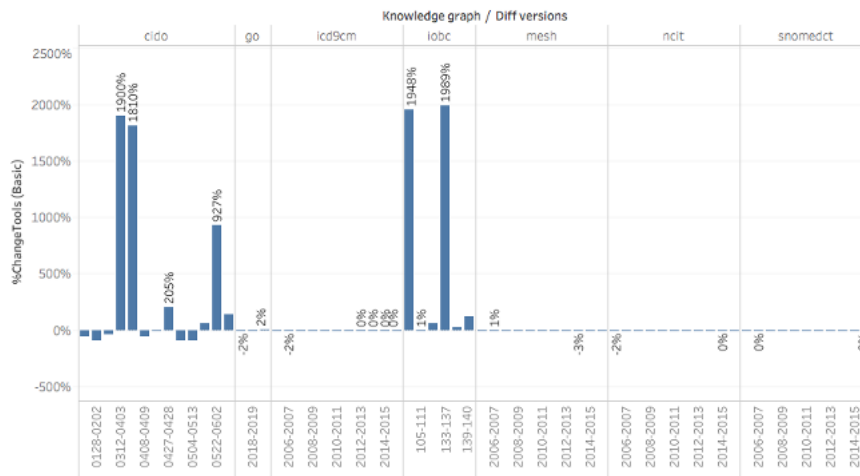
**Figure 3:** Comparison of average execution time (secs) vs total number of changes

From this analysis, we briefly conclude that *COnto-Diff* potentially performs better with small and middle-sized ontologies ( $\leq 70K$  classes), but it shows a high variability in performance when dealing with larger and parsed ontologies, as was the case for GO. This shows the ability of *DynDiff* to be scalable and provide robust results for ontologies of any size.

As a measure of accuracy, we used  $\%changeTools_{type}$  rate to compare *DynDiff* with *COnto-Diff* (see Fig. 4 and 5). From the basic change analysis (Fig. 4), we observed that the results are equivalent for almost all ontologies that we evaluated (i.e., the rate is close to 0%). Exceptions are observed for IOBC and CIDO. This can be explained by the inability of *COnto-Diff* to process labels in Japanese or to identify changes in individuals and properties. *DynDiff* can handle diffs of instances and properties encoded in non-utf-8<sup>2</sup> format.

<sup>2</sup>Universal Coded Character Set Transformation Format – 8-bit.

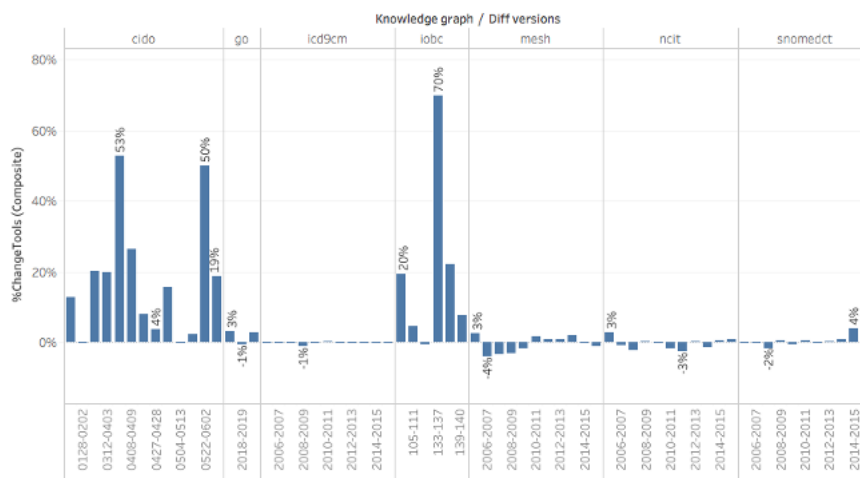
### Basic change comparison



**Figure 4:** Comparison of the computed basic changes by DynDiff vs *COnto-Diff*

We compared the tools wrt heuristic and complex changes (see Fig. 5). The conclusions are similar to the previous analysis, i.e. the performance of both tools is close, except for CIDO and IOBC. The slight differences of 2-4% for the ontologies that contain almost no properties or instances result from the difference in levels of granularity of each approach.

### Complex change comparison



**Figure 5:** Comparison of the computed composite changes by DynDiff vs *COnto-Diff*.

Furthermore, negative percentages are due to the capacity of DynDiff to represent more abstract changes (e.g. PullUpC), which reduces the total number of composite changes. This highlights that the way the information is presented and grouped has an effect on the comparison between tools and the interpretability of the results by humans.

## 5. Conclusion

Computing ontology diff in a scalable and efficient manner is still a problem, especially for the ever-growing size of ontologies. In this paper, we presented DynDiffOnto, an ontology that defines a balanced set of change actions to support the interpretation of how ontologies evolve. We also present the DynDiff tool designed to compute the Diff between versions of ontologies and classify them according to DynDiffOnto. Our analysis show that DynDiff is efficient for handling large ontologies, even those with dynamic instances and properties. We finally compared our approach with the state-of-the-art tools used in this field and observed a very close performance in terms of execution time and capacity to identify changes. This tool was evaluated with ontologies from the medical domain; we are now evaluating it with ontologies from other domains and plan to review DynDiffOnto in order to also meet the needs of other domains.

## Acknowledgments

The research reported in this publication was supported by the Luxembourg National Research Fund (FNR), <https://www.fnr.lu>, project HKG4Covid, grant number COVID-19/2020-2/14841581.

## References

- [1] N. F. Noy, N. H. Shah, P. L. Whetzel, B. Dai, M. Dorf, N. Griffith, C. Jonquet, D. L. Rubin, M.-A. Storey, C. G. Chute, et al., Bioportal: ontologies and integrated data resources at the click of a mouse, *Nucleic acids research* 37 (2009) W170–W173.
- [2] F. Belleau, M.-A. Nolin, N. Tourigny, P. Rigault, J. Morissette, Bio2rdf: towards a mashup to build bioinformatics knowledge systems, *Journal of biomedical informatics* 41 (2008) 706–716.
- [3] J. Xu, S. Kim, M. Song, M. Jeong, D. Kim, J. Kang, J. F. Rousseau, X. Li, W. Xu, V. I. Torvik, et al., Building a pubmed knowledge graph, *Scientific Data* 7 (2020) 205. doi:10.1038/s41597-020-0543-2.
- [4] M. Biryukov, V. Groues, V. Satagopam, R. Schneider, Biokb - text mining and semantic technologies for biomedical content discovery, in: *Semantic Web Applications and Tools for Healthcare and Life Sciences*, 2018. doi:10.6084/m9.figshare.6994121.v1.
- [5] V. Zamborlini, R. Hoekstra, M. Da Silveira, C. Pruski, A. ten Teije, F. van Harmelen, A conceptual model for detecting interactions among medical recommendations in clinical guidelines, in: *International Conference on Knowledge Engineering and Knowledge Management*, Springer, 2014, pp. 591–606.
- [6] M. Herrero-Zazo, I. Segura-Bedmar, J. Hastings, P. Martínez, Dinto: Using owl ontologies and swrl rules to infer drug–drug interactions and their mechanisms, *Journal of Chemical Information and Modeling* 55 (2015) 1698–1707. doi:10.1021/acs.jcim.5b00119, PMID: 26147071.
- [7] S. Garde, P. Knaup, E. J. Hovenga, S. Heard, Towards semantic interoperability for electronic health records, *Methods of information in medicine* 46 (2007) 332–343.

- [8] Y. He, H. Yu, E. Ong, Y. Wang, Y. Liu, A. Huffman, H.-h. Huang, J. Beverley, J. Hur, X. Yang, et al., Cido, a community-based ontology for coronavirus disease knowledge and data integration, sharing, and analysis, *Scientific Data* 7 (2020) 181. doi:10.1038/s41597-020-0523-6.
- [9] Y. Roussakis, I. Chrysakis, K. Stefanidis, G. Flouris, Y. Stavrakas, A flexible framework for understanding the dynamics of evolving rdf datasets, in: *International Semantic Web Conference*, Springer, 2015, pp. 495–512.
- [10] J. C. Dos Reis, C. Pruski, M. Da Silveira, C. Reynaud-Delaître, Understanding semantic mapping evolution by observing changes in biomedical ontologies, *Journal of biomedical informatics* 47 (2014) 71–82.
- [11] S. D. Cardoso, C. Pruski, M. Da Silveira, Y.-C. Lin, A. Groß, E. Rahm, C. Reynaud-Delaître, Leveraging the impact of ontology evolution on semantic annotations, in: *European Knowledge Acquisition Workshop*, Springer, 2016, pp. 68–82.
- [12] J. C. Dos Reis, C. Pruski, M. Da Silveira, C. Reynaud-Delaître, Dykosmap: A framework for mapping adaptation between biomedical knowledge organization systems, *Journal of biomedical informatics* 55 (2015) 153–173.
- [13] S. D. Cardoso, C. Reynaud-Delaître, M. Da Silveira, C. Pruski, Combining rules, background knowledge and change patterns to maintain semantic annotations, in: *AMIA Annual Symposium Proceedings*, volume 2017, American Medical Informatics Association, 2017, p. 505.
- [14] N. F. Noy, M. A. Musen, et al., Promptdiff: A fixed-point algorithm for comparing ontology versions, *AAAI/IAAI 2002* (2002) 744–750.
- [15] M. Hartung, A. Groß, E. Rahm, Conto-diff: generation of complex evolution mappings for life science ontologies, *Journal of biomedical informatics* 46 (2013) 15–32.
- [16] R. S. Gonçalves, B. Parsia, U. Sattler, Categorising logical differences between owl ontologies, in: *Proceedings of the 20th ACM international conference on Information and knowledge management*, 2011, pp. 1541–1546.
- [17] V. Papavasileiou, G. Flouris, I. Fundulaki, D. Kotzinos, V. Christophides, High-level change detection in rdf (s) kbs, *ACM Transactions on Database Systems (TODS)* 38 (2013) 1–42.
- [18] N. F. Noy, A. Chugh, W. Liu, M. A. Musen, A framework for ontology evolution in collaborative environments, in: *International semantic web conference*, Springer, 2006, pp. 544–558.
- [19] A. M. Khattak, K. Latif, S. Lee, Change management in evolving web ontologies, *Knowledge-Based Systems* 37 (2013) 1–18.
- [20] P. Plessers, O. De Troyer, Ontology change detection using a version log, in: *International Semantic Web Conference*, Springer, 2005, pp. 578–592.
- [21] M. Strohmaier, S. Walk, J. Pöschko, D. Lamprecht, T. Tudorache, C. Nyulas, M. A. Musen, N. F. Noy, How ontologies are made: Studying the hidden social dynamics behind collaborative ontology engineering projects, *Web semantics* 20 (2013) 18–34.
- [22] S. D. Cardoso, M. Da Silveira, C. Pruski, Construction and exploitation of an historical knowledge graph to deal with the evolution of ontologies, *Knowledge-Based Systems* 194 (2020) 105508.