Discovering Evolving Regions in Life Science Ontologies

Michael Hartung[1,2], Anika Gross[1,2], Toralf Kirsten[1,3], and Erhard Rahm[1,2]

[1] Interdisciplinary Centre for Bioinformatics, University of Leipzig
[2] Department of Computer Science, University of Leipzig
[3] Institute for Medical Informatics, Statistics and Epidemiology, University of Leipzig
{hartung, tkirsten}@izbi.uni-leipzig.de,
{gross, rahm}@informatik.uni-leipzig.de

Abstract. Ontologies are heavily used in life sciences and evolve continuously to incorporate new or changed insights. Often ontology changes affect only specific parts (regions) of ontologies making it valuable for ontology users and applications to know the heavily changed regions on the one hand and stable regions on the other hand. However, the size and complexity of life science ontologies renders manual approaches to localize changing or stable regions impossible. We therefore propose an approach to automatically discover evolving or stable ontology regions. We evaluate the approach by studying evolving regions in the Gene Ontology and the NCI Thesaurus.

Keywords: ontology evolution, ontology changes, ontology regions.

1 Introduction

Ontologies are heavily used in life sciences, especially to consistently describe or annotate objects of an application domain [1, 14]. For instance, SwissProt [2] and Ensembl [10] are two frequently used data sources in which proteins are annotated (associated) with concepts of the Gene Ontology (GO) [7] to describe their molecular functions as well as their involvement in biological processes. The high importance of ontologies is reflected in their growing number and size. Currently, there are about 70 ontologies available in the Open Biomedical Ontology (OBO) foundry [23]. These ontologies usually underlie a continuous evolution to incorporate the latest requirements and insights of a particular domain [9]. For instance, the GO or the NCI Thesaurus [22] have nearly doubled their size since 2004 [8]. Ontology providers continuously release new versions of changed ontologies. For example, changes for GO are released on a daily basis, and for NCI Thesaurus every month.

As a consequence of this evolution ontology users need to cope with these changes. To determine whether applications or data sources need to be adapted for the newest ontology versions it is valuable to know what parts of an ontology have significantly changed or remained unchanged in a specific period of time. Such information can be utilized in different ways. On the one hand, analysis applications such as functional profiling [3, 21] that used a heavily changed ontology region should be rerun to determine how analysis results are affected by the ontology changes. On the other hand, algorithms may use the information that specific ontology parts remained unchanged for a more efficient computation since they can reuse previous results. For example, algorithms to match different ontologies [5] can then reuse match results of previous versions for improved efficiency. The information on stable or changing ontology regions is also a good indicator where little or much further development is to be expected. So, unstable ontology regions are a good indicator for ontology developers to participate within a collaborative ontology development. Furthermore, project coordinators may use the information about regions to plan future development steps.

The manual discovery of stable and changing ontology regions is not feasible for large ontologies so that automatic techniques are required. So far only little and preliminary work has been performed in this direction. Previous research in the area of ontology change (see [6] for a survey) focused on ontology versioning [12, 18], the ontology evolution process [15, 24, 25] or the change detection between ontology versions [16, 17, 19, 20]. In our own previous work we quantitatively evaluated evolution of life science ontologies [9]. Furthermore, we designed a web application [8] which allows access to information about changes in life science ontologies. However, to our best knowledge no current work determines the location (region) where changes occurred in an ontology. We therefore make the following contributions in this paper:

- We introduce and define the notion of ontology regions and corresponding measures to classify ontology regions according to their change intensity.
- We propose an algorithm for the discovery of stable and unstable ontology regions. The algorithm is customizable to meet the requirements of different applications. It (1) considers different change types, (2) uses an extensible set of measures for regions and (3) allows region discovery over different time periods. Hence, we can support various application scenarios, e.g., finding small and unstable, or large and stable ontology regions.
- We evaluate the approach for the Gene Ontology and NCI Thesaurus. Results show that in both cases unstable and stable regions exist and hence indicate that the proposed approach is applicable for automatic discovery of evolving regions in large life science ontologies.

The rest of the paper is organized as follows. In Section 2 we present our models for ontologies as well as ontology changes and introduce the notion of ontology regions. Section 3 describes the discovery algorithm. We evaluate the approach in Section 4. We finally conclude and outline possibilities for future work.

2 Preliminaries and Models

We first outline our ontology model including versioning. Next, we describe which kinds of ontology changes are considered and introduce a corresponding change cost model. Finally, we define ontology regions and outline possible measures to quantify the change intensity of regions.

2.1 Ontology model and versioning

An ontology $O = (C, R)$ consists of concepts C which are interconnected by relationships in R. Together they form a so-called directed acyclic graph (DAG) representing the structure of O. Special concepts of C called *roots* are the topmost concepts of O, i.e., they have no relationship to any parent concept. If the number of *roots* is greater than one, we introduce a *virtual root* which acts as a single entry point for the ontology. Thus, we can define all *roots* of the ontology as children of the *virtual root*.

A concept $c \in C$ of an ontology is defined by a set of single-valued or multi-valued attributes. The accession number c_{acc} is a special attribute to unambiguously identify ontology concepts. Further typical attributes include the name/label, a definition or synonyms of concepts. Relationships $r \in R$ can be separated into two groups: (1) is_a relationships and (2) other relationships. Is_a relationships usually form the base structure of an ontology, hence we will utilize these relationships to define our ontology regions (see Section 2.3) and make use of them in our discovery algorithm (see Section 3). Other relationships extend the basic is_a structure by more specific relationships, e.g., part_of or has_parts. . The used ontology model represents well existing life science ontologies, in particular the ones in the OBO Foundry [23].

An ontology version $O_v = (C, R, t)$ of version v is a snapshot of an ontology at a specific point in time t. The concepts C and relationships R of O_v are valid until a newer ontology version is released. We assume that versions of an ontology follow a linear versioning scheme, i.e., each ontology version O_i has at most one successor O_{i+1} and one predecessor version O_{i-1}. The first / last ontology versions have no predecessor / successor version, respectively.

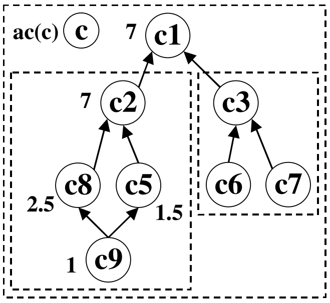
2.2 Ontology changes and cost model

The evolution from an old ontology version O_{old} to a newer ontology version O_{new} can be described by a set of ontology changes. We distinguish between the basic change types addition (*add*), deletion (*del*) and update (*upd*) for *concepts*, *relationships* and *attributes* of an ontology:

concept		**relationship**		**attribute**		
add	*del*	*add*	*del*	*add*	*del*	*upd*

Particularly, concepts, relationships and attributes can be added or deleted. In case of attributes we further use the update change type for attribute value changes in concepts, e.g., the modification of a concept's name or definition. Note that at the current stage we do not include complex changes such as merge or split of concepts, since these changes are typically composed of basic changes that we already cover. However, complex changes can be included in the future to achieve a more fine-grained and semantically richer distinction between different changes.

To reflect the impact of changes we introduce a *cost model* for ontology changes. Particularly, we assign *change costs* to the different kinds of ontology changes to determine their impact on an ontology. For instance, we can assign higher change costs for *delConcept* compared to *addConcept* to consider a higher change impact for concept deletions vs. concept additions. The individual costs can be assigned to on-



OR	*abs_size*	*rel_size*	*abs_costs*	*avg_costs*
c1	8	8/8=1	7	7/8=0.875
c2	4	4/8=0.5	7	7/4=1.75
c3	3	3/8=0.375	0	0/3=0

Figure 1: Sample ontology with regions, aggregated costs (left) and corresponding region measures (right)

tology concepts affected by an ontology change. Particularly, we distinguish between two types of costs for an ontology concept: local costs *lc* and aggregated costs *ac*. Local costs *lc(c)* cover the impact of ontology changes that directly affect an ontology concept *c*, i.e., changes on the concept itself as well as changes on its relationships and attributes. For instance, the addition/deletion of a child concept or an attribute value change have a direct impact. We will later (Section 3.1.1) discuss how local costs are assigned to ontology concepts based on the change type. We further use aggregated costs *ac(c)* to reflect all changes occurring in the is_a descendants of a concept *c*. For instance, leaf concept additions/deletions have an indirect impact on corresponding ancestor concepts in the ontology. In Section 3.1.2 we describe how aggregated costs are derived from local costs. The sample (changed) ontology version in Fig. 1 (left) contains aggregated costs (numbers next to a concept) for each concept, e.g., concept *c2* has aggregated costs of 7 while its sibling *c3* has no aggregated costs *ac(c3)* = 0.

2.3 Ontology regions and measures

An ontology region *OR* is a subgraph of an ontology consisting of a single root concept *rc*. A region contains all concepts located in the is_a subgraph of *rc*, i.e., there exists at least one is_a-path from every concept $c \in OR$ to *rc*. We will aggregate the concept change costs per ontology region to identify change-intensive or stable regions. Our notion of an ontology region observes that changes often occur in the boundary of an ontology, e.g., addition of leaves or subgraphs to extend the knowledge of a specific topic. Of course an ontology region also covers changes on inner concepts since all intermediate concepts between the root and the leaves are part of the region. In the sample ontology of Fig. 1 (left) several ontology regions are marked. For instance, the region with root concept *c2* consists of the four concepts *c2, c5*, *c8* and *c9*. The complete ontology with root *c1* can also be seen as a region.

The change intensity of an ontology region *OR* and other characteristics can be described by *region measures* incorporating aspects such as the local/aggregated costs or the region size. We will later use these measures in our algorithm for the discovery of specific ontology regions. We define the following exemplary measures for an ontology region *OR*:

- absolute region size *abs_size(OR)*: number of concepts in an ontology region *OR*
- relative region size *rel_size(OR)*: relative size of *OR* compared to the overall size of the ontology *O* defined by *abs_size(OR) / abs_size(O)*
- absolute change costs *abs_costs(OR)*: the absolute costs of *OR* represented by its root's aggregated costs *ac(rc)*
- average change costs *avg_costs(OR)*: the average costs per concept in *OR* defined by *abs_costs(OR) / abs_size(OR)*

Note that these measures are only examples, i.e., we can extend the set of measures depending on application requirements. For instance, one may consider other characteristics such as the depth or the compactness of a region. The example regions *c1*, *c2* and *c3* of the sample ontology in Fig. 1 show different characteristics based on our example measures, as shown in the table on the right side of Fig. 1. For instance, regions *c2* and *c3* have a similar size but differ largely in their change intensity (measures *abs_costs* and *avg_costs).* While region *c3* has not been changed (*avg_costs* of 0), region *c2* exhibits average costs of 1.75. We will now (Section 3) explain how we determine aggregated costs (ac) of concepts in general and for our example ontology of Fig. 1.

3 Ontology Region Discovery

In this section we present the algorithm for discovering evolving ontology regions. We first show how the aggregated costs of concepts are computed for two succeeding ontology versions. We then present the algorithm for the computation of region measures. Finally, we combine both algorithms to discover ontology regions for multiple ontology versions released in a specific period of time.

3.1 Computation of aggregated costs for two ontology versions

The algorithm for determining aggregated costs in two succeeding ontology versions takes as input an old ontology version O_{old} and a new ontology version O_{new} as well as change costs σ for ontology changes (see Section 2.2). Note that we use dedicated concept attributes to store local (*lc*) and aggregated costs (*ac*) of concepts, i.e., we internally extend the given ontology versions to capture assigned costs in each concept. The algorithm computeAggregatedCosts consists of four steps as follows:

Algorithm 1: computeAggregatedCosts (ontology versions O_{old}, O_{new}, change costs σ)

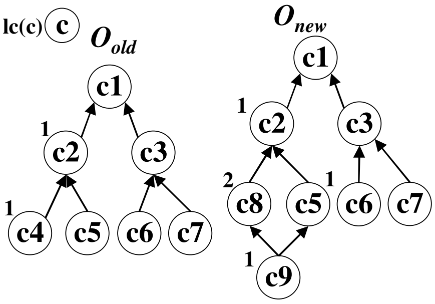
```
$\Delta O_{old}$-$O_{new}$ := diff ($O_{old}$, $O_{new}$) computes changes between ontology versions (both directions)
assignLocalCosts ($\Delta O_{old}$-$O_{new}$, $\sigma$, $O_{old}$, $O_{new}$)
$O_{old}$:=aggregateCosts ($O_{old}$)
$O_{new}$:=aggregateCosts ($O_{new}$)
transferCosts ($O_{old}$, $O_{new}$)
return $O_{new}$
```

We first compute the changes between the input versions (diff). Next we assign local costs to affected concepts (assignLocalCosts) to determine the added, deleted and modified ontology elements. Depending on the change type local costs are assigned either to concepts of the older or the newer ontology version. For instance, the deletion of a concept can only be captured in the older version since the concept is not available in the newer one and vice versa for added concepts. Afterwards the local costs are propagated upwards in each ontology version (aggregateCosts) according to the respective ontology structure. This step ensures that costs from deeper ontology parts are aggregated within inner ontology concepts and finally in the ontology root. Since we like to discover regions based on the latest ontology version we need to transfer aggregated costs of older versions to newer ones (transferCosts). The transfer guarantees that costs originated in older ontology versions such as deletes are also reflected in the newest ontology version. Finally, the newer ontology version including the computed aggregated costs is returned. We then can use this ontology version for applying our region measures (see Section 3.2). We also use this enriched version in the iterative algorithm for dealing with more than two ontology versions (see Section 3.3). We will explain the steps of computeAggregatedCosts in more detail in the following sub sections. A simple yet comprehensive example will be used for illustration.

3.1.1 Change detection and assignment of local costs

Change detection between the two ontology versions O_{old} and O_{new} is based on the comparison of concept accession numbers which are typically used in life science ontologies for unambiguous concept identification. Particularly, we determine ontology changes by comparing elements of O_{old} with those of O_{new}: diff(O_{old}, O_{new}). In this process we distinguish between concept, relationship and attribute changes. Added elements (*add*) are only present in the newer version O_{new} while deleted elements (*del*) only exist in the older version O_{old}. Furthermore, we detect updates (*upd*) on attributes, e.g., when the name of a concept has been changed. Thus, we cover all changes described in Section 2.2. Note that these changes represent the basic change types in ontology evolution and complex changes such as split or merge can be seen as a composition of these.

The example in Fig. 2 shows two ontology versions O_{old} and O_{new} including changes in concepts and relationships (for simplicity we omit attribute changes and focus on is_a relationships). Particularly, from O_{old} to O_{new} two new concepts (*c8*, *c9*)



diff(O_{old},O_{new}) → ΔO_{old}-O_{new}:
addConcept{c8, c9}
delConcept{c4}
addRelationship{(c8,c2), (c9,c5), (c9,c8)}
delRelationship{(c4,c2)}

assignLocalCosts(ΔO_{old}-O_{new}, O_{old}, O_{new}):

O_{old}: lc(c2) =1; lc(c4)=1
O_{new}: lc(c2)=1; lc(c5)=1; lc(c8) =2; lc(c9)=1

Figure 2: Diff and assignment of local costs for two ontology versions

were introduced while one concept (*c4*) was deleted. Corresponding relationships were inserted ((*c8,c2*), (*c9,c5*), (*c9,c8*)) and removed ((*c4,c2*)).

The changes in the diff result and the specified change costs are used to assign local costs (*lc*) to affected concepts in both ontology versions. We assign local costs to concepts using the assignLocalCosts method in the following way. Costs of additions and updates are always captured in the new ontology version. In contrast, costs of deletions are captured in the old ontology version since the affected elements (e.g., a deleted concept) are only present in this version. The costs of concept and attribute changes are directly assigned to the affected concept. For relationship changes the costs are assigned to the source and target concept of a relationship. Note that different costs for the source and target concept can be used.

In Fig. 2 the numbers next to the concepts refer to the associated local costs for the changes found by diff(O_{old}, O_{new}). For simplicity, we assume uniform change costs of 1 per change. Furthermore, we only assign costs to the target of a changed relationship. In our example the deletion of *c4* causes the assignment of local costs 1 to *c4* (*delConcept*) and *c2* (*delRelationship*) in O_{old}. The insertion of *c8* and *c9* (*addConcept*) leads to the assignment of local costs 1 to both concepts. Concept *c8* receives additional costs 1 caused by the insertion of the (*c9,c8*) relationship, thus its overall local costs are 2 (*lc(c8)*=2). Due to the addition of the relationships (*c9,c5*) and (*c8,c2*) concepts *c2* and *c5* of O_{new} are both assigned local cost 1.

3.1.2 Aggregation of local costs

We propagate local costs (*lc*) of concepts via is_a paths upwards (in root direction) and hence aggregate costs of subgraphs in corresponding inner ontology concepts (aggregated costs (*ac*) of concepts). The aggregation is applied on the old version as well as the new version with the intention, that the sum of all assigned local costs is equal to the aggregated costs of the root, i.e., the root subsumes all costs assigned to an ontology version.

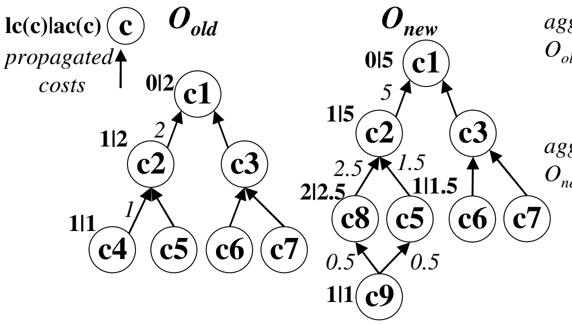
The aggregation of costs follows one rule. The aggregated costs of a concept is the sum of the aggregated costs of its direct children plus the local costs of itself:

$$ac(c) = \sum_{\text{direct children c' of c}} \frac{ac(c')}{|\,parents(c')\,|} + lc(c)$$

If a concept *c* has more than one parent the *costs* are split into |*parents*| portions so that *costs/*|*parents*| costs are propagated to each parent. The algorithm aggregateCosts uses an ontology version O_v with associated local costs and propagates them through the ontology using the given structure of O_v:

Algorithm 2: aggregateCosts (ontology version O$_v$)

```
for all concepts c in Ov do
    if local costs lc(c) > 0 then
        aggregate (c, Ov, lc(c))
    end if
end for
return Ov
```



aggregateLocalCosts(O_{old}):
O_{old}: ac(c4)=1+0=1
ac(c2)=1+1=2
ac(c1)=0+2=2

aggregateLocalCosts(O_{new}):
O_{new}: ac(c9)=1+0=1
ac(c8)=2+0.5=2.5
ac(c5)=1+0.5=1.5
ac(c2)=1+2.5+1.5=5
ac(c1)=0+5=5

Figure 3: Aggregation of costs in both ontology versions

Particularly, the local costs of an ontology concept are propagated along its root paths to the root of the ontology. The recursive algorithm aggregate is responsible for one propagation step along the path. Thereby aggregated costs *ac(c)* are updated (summed up) with the incoming costs from a child. The new costs are calculated based on *c*'s number of parents and are finally propagated to all parents of *c* (recursive call of aggregate). Since a concept may have multiple children, the concept can be updated multiple times to aggregate all local costs assigned to its is_a descendants.

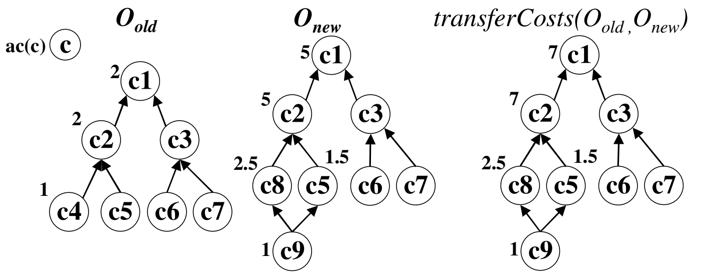
Algorithm 3: aggregate (concept c, ontology version O_v, change costs σ)

```
aggregated costs ac(c) := ac(c) + $\sigma$
parent concepts $C_{parent(c)}$ := getParents($O_v$, c)
normalized costs $\sigma_{norm}$ := $\sigma$ / $|C_{parent(c)}|$
for all concepts c' in $C_{parent(c)}$ do
    aggregate(c', $O_v$, $\sigma_{norm}$)
end for
```

Fig. 3 shows the aggregation of local costs in our running example for ontology versions O_{old} and O_{new}. Each concept is displayed with its local and aggregated costs (*lc(c)*|*ac(c)*), paths are annotated with the propagated costs. For instance, in O_{new} *c9*'s aggregated costs are equal to *lc(c9)* since *c9* has no children. The relationships (*c9,c5*) and (*c9,c8*) are utilized to propagate *c9*'s costs to the corresponding parents. Since two parents exist, *ac(c9)* is split into two portions of 0.5 which are propagated to *c5* and *c8*, respectively. Thus, the aggregated costs of *c5* are composed of *ac(c9)/2* and *lc(c5)*: *ac(c5)* = *ac(c9)/2*+*lc(c5)* = 0.5+1 = 1.5. The same holds for *c8*: *ac(c8)* = *ac(c9)/2*+*lc(c8)* = 0.5+2 = 2.5. In the next step *ac(c5)* and *ac(c8)* are propagated to *c2* and aggregated with *lc(c2)*: *ac(c2)* = *ac(c5)*+*ac(c8)*+*lc(c2)* = 1.5+2.5+1 = 5. Having propagated all costs, the aggregated costs of both roots are equal to the sum of all assigned local costs: 2 for O_{old} and 5 for O_{new}, respectively.

3.1.3 Transfer of aggregated costs

After separate aggregation of costs in the old and new version the results are now transferred to the newer version. The transfer ensures that change costs of the old version are reflected in the new version as well since we like to discover regions of interest based on the new version. The transferCosts algorithm transfers aggregated



transferCosts(O_{old},O_{new}):

	ac(c1)	ac(c2)	ac(c3)	ac(c4)	ac(c5)	ac(c6)	ac(c7)	ac(c8)	ac(c9)
O_{old}	2	2	0	1	0	0	0		
O_{new}	5	5	0		1.5	0	0	2.5	1
transfer	7	7	0		1.5	0	0	2.5	1

Figure 4: Transfer of aggregated costs from old to new ontology version

costs of concepts from the old version into the new version. In particular, the method sums up the aggregated costs of equal concepts in both versions and stores the result in the new version:

Algorithm 4: transferCosts (ontology versions O_{old}, O_{new})

```
for all concepts c in $O_{old}$ do
    if c $\in O_{new}$ then
        ac(c) $\in O_{new}$ += ac(c) $\in O_{old}$
    end if
end for
```

The transfer of costs for our running example is displayed in Fig. 4. The table below shows how the costs of O_{old} and O_{new} are summed up in O_{new}. Since concepts *c1*, *c2*, *c3*, *c5, c6* and *c7* are present in the old and new ontology version their aggregated costs of both versions are fused, e.g., after the transfer *c2*'s aggregated costs is 7 (2 from O_{old} and 5 from O_{new}). If a concept is only present in the new version its aggregated costs remain unchanged (e.g., for *c8* and *c9* in O_{new}). In contrast, aggregated costs of deleted concepts can not directly be transferred to the new version (e.g., the costs of *c4*). However, the cost aggregation described in Section 3.1.2 ensures that costs of deletions are indirectly transferred. In our case *c2*'s aggregated costs which are transferred to O_{new} contain the costs of *c4*'s deletion. Thus, changes on *c4* are indirectly reflected in the new version as well.

3.2 Computation of measures and discovery of ontology regions

To compute the proposed region measures of Section 2.3 we apply an algorithm computeRegionMeasures which uses available information such as aggregated costs or the ontology structure. As an example, in case of the *rel_size* measure we iterate over all ontology concepts and compute the ratio between the region size of each concept and the overall ontology size. *c1* as the root of our running example exhibits a *rel_size* of 1.0 while *c2* (*c3*) show a *rel_size* of 0.5 (0.375). As one may notice the sample ontology displayed in Fig. 1 is equal to the result of the transfer of our running example discussed in Section 3.1.3. Hence, the results of our example are equal to the ones presented in Section 2.3.

Based on the results we can discover regions of interest in the new ontology version. Particularly, we define constraints on the results and thus select the regions that satisfy the criteria. Depending on the application different criteria (e.g., relative or absolute size/cost measures) can be considered and combined. For instance, "large stable regions" may be defined with the constraints: *rel_size(OR)*>0.2 and *avg_costs(OR)*=0. In our case region *c3* is the only region satisfying these constraints. In contrast, one may use *rel_size(OR)*>0.2 and *avg_costs(OR)*>1 to select "large unstable regions", e.g., region *c2* in our running example. Note that we can eliminate sub-regions of a larger ontology region for a compact result, i.e., only regions satisfying the given constraints and which are not contained in another selected region are returned. For instance, the region covered by *c8* would also satisfy the constraints of an unstable region (*avg_costs(c8)*>1 and *rel_size(c8)*>0.2). However, *c8* is contained in region *c2* and thus we only return *c2* as an identified region.

3.3 Discovery algorithm for multiple ontology versions

Based on computeAggregatedCosts and computeRegionMeasures we now define the generalized findRegions algorithm which works on multiple ontology versions released in a specific time period. The idea of the combined algorithm is the following. Having *n* released ontology versions (O_1, …, O_n) we iterate over all releases and apply computeAggregatedCosts on each pair (O_i,O_{i+1}). Thus, we cover all version changes between succeeding ontology versions and transfer costs from older ontology versions to the latest ontology version O_n where the region discovery is applied (computeRegionMeasures). The algorithm findRegions for *n* ontology versions looks as follows:

Algorithm 5: findRegions(ontology versions O_1 … O_n, change costs σ)

```
for all succeeding ontology versions $O_i$ – $O_{i+1}$ do
    $O_{i+1}$ := computeAggregatedCosts($O_i$, $O_{i+1}$, σ)
end for
computeRegionMeasures($O_n$)
```

4 Evaluation

We evaluated the proposed region discovery algorithm for the well-known Gene Ontology (GO) and the National Cancer Institute Thesaurus (NCIT). After the description of the evaluation setup we first comparatively analyze the overall ontology stability for different periods. In Section 4.3 we analyze the distribution of ontology regions w.r.t. their stability and present how the most (un)stable ontology regions can be discovered. We finally show how the algorithm can be used to track the stability of ontology regions over time.

4.1 Evaluation setup

The two considered ontologies are heavily used in different projects and underlie continuous changes. GO is widely used for the annotation of proteins w.r.t. Biological Processes (BP), Molecular Functions (MF) and Cellular Components (CC). NCIT maintained at the National Cancer Institute consists of 20 main categories which cover cancer-related topics such as drugs, tissues or anatomical structures. It is utilized in US-wide projects such as the Cancer Biomedical Informatics Grid (caBIG) [4] and its underlying infrastructure caCORE [13].

We integrated available ontology versions between 2004 and 2009 on a monthly basis in a repository [11]. Note that we include at most one version per month, if there is more than one version available we use the first release. The repository allows for the efficient retrieval of versioned ontology information. Thus, we can compare ontology versions of a specified time period to determine the ontology changes in our algorithm. The latest considered GO version of December 2009 consists of 30,304 concepts (GO-BP: 18,108; GO-MF: 9,459; GO-CC: 2,737) while the latest NCIT version of December 2009 contains 77,465 concepts.

For all evaluation studies we apply the following change costs:

concept		**relationship**		**attribute**		
add	*del*	*add*	*del*	*add*	*del*	*upd*
1.0	2.0	1.0	2.0	0.5	0.5	0.5

In general concept changes have the biggest impact followed by relationship and attribute changes. Furthermore, we give concept and relationship deletions more impact, attribute changes are weighted equally. In case of relationships we assign half of the costs to the target and the other half to the source concept of a changed relationship. The used values are for illustration only and can be changed to meet specific application characteristics.

4.2 Overall ontology stability

We apply our region measures to the root of an ontology for assessing its overall stability. Particularly, we utilize released versions of a specific time period and assess the overall stability by taking the measures *abs_size(root)*, *abs_costs(root)* and *avg_costs(root)* into account. Table 1 lists the overall stability of GO (including its sub ontologies) and NCIT for 2008 and 2009, respectively.

In 2008 GO and NCIT exhibit similar absolute costs (GO: ~24,200; NCIT: ~23,200) but the average change intensity was much higher for GO (*avg_costs* 0.87 for GO vs. 0.32 for NCIT). In 2009, the change intensity increased for NCIT but decreased for GO, but GO still retained an increased change activity (*avg_costs* 0.64 vs. 0.47). Within the GO sub ontologies GO-BP possesses the highest absolute and average costs in both periods. In contrast GO-MF can be seen as the most stable sub

	abs_size(root)		*abs_costs(root)*		*avg_costs(root)*	
	2008	*2009*	*2008*	*2009*	*2008*	*2009*
GO	27,799	30,304	24,242	19,412	0.87	0.64
– MF	9,205	9,459	4,636	3,002	0.50	0.32
– BP	16,231	18,108	17,594	14,557	1.08	0.80
– CC	2,363	2,737	2,011	1,854	0.85	0.68
NCIT	71,337	77,455	23,165	36,562	0.32	0.47

Table 1: Overall stability of ontologies in 2008 and 2009

ontology of GO (≤0.5 *avg_costs* in 2008 and 2009). Between 2008 and 2009 the average costs decreased especially for GO-MF (from 0.5 to 0.32) underlining the improved stability compared to GO-BP and GO-CC.

4.3 Discovery of (un)stable regions

To discover the most stable and unstable regions of an ontology we analyze the distribution of ontology regions w.r.t. their *avg_costs*. Figure 5 shows such a distribution for GO-BP changes in 2009. We consider ontology regions with a minimum *rel_size* of 0.3% (~ 50 concepts) and group them according to their average costs into intervals of size 0.05. Overall we classified 518 regions in 36 intervals (0.00:0.05 to 1.75:1.80). Most of the regions (~430 regions; ~83%) exhibit average costs between 0 and 0.5, 60 out of which (~12%) have average costs lower than 0.05 and are thus largely stable. In contrast about 53 ontology regions (~10%) show average costs above 0.65.

We can thus determine the most stable and unstable ontology regions by focusing on the two ends of the cost-based distribution. Depending on the application needs we may use either absolute thresholds (e.g., *avg_costs* < 0.01 or *avg_costs* > 0.8) or percentiles of a distribution to classify regions as stable or unstable. For the following

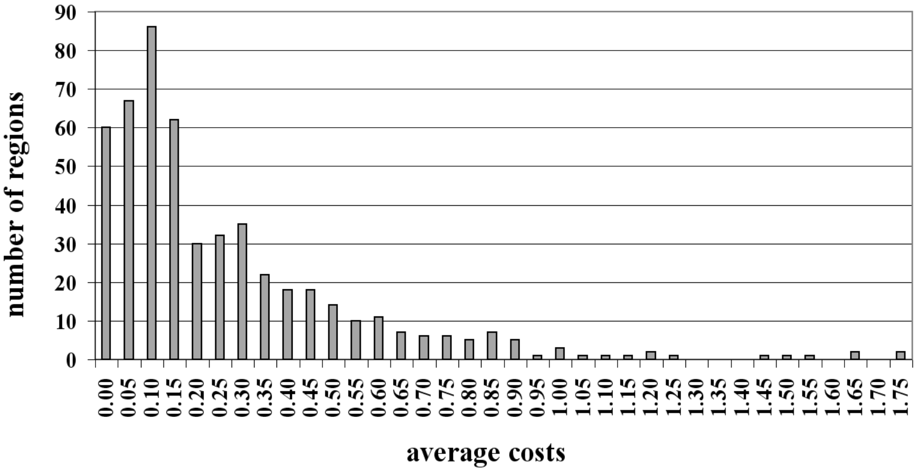


Figure 5: Distribution of regions w.r.t. average costs for GO-BP in 2009

		accession	name	abs_size	rel_size	avg_costs
GO	unstable	GO:0005102	receptor binding	408	4.31%	0.95
		GO:0009653	anatomical structure morphogenesis	583	3.22%	1.22
		GO:0048856	anatomical structure development	566	3.13%	0.91
		GO:0033643	host cell part	77	2.81%	1.90
		GO:0003676	nucleic acid binding	241	2.55%	0.86
		GO:0048646	anatomical structure formation involved in morphogenesis	253	1.40%	0.92
	stable	GO:0031300	intrinsic to organelle membrane	36	1.32%	0.000
		GO:0030054	cell junction	31	1.13%	0.000
		GO:0050865	regulation of cell activation	184	1.02%	0.012
		GO:0075136	response to host	181	1.00%	0.019
		GO:0000151	ubiquitin ligase complex	25	0.91%	0.000
		GO:0016860	intramolecular oxidoreductase activity	71	0.75%	0.000
NCIT	unstable	C28428	Retired Concept	3,264	4.21%	3.49
		C53791	Adverse Event Associated with Infection	1,186	1.53%	2.36
		C45678	Industrial Aid	889	1.15%	1.40
		C74944	Clinical Pathology Procedure	747	0.96%	0.84
		C66892	Natural Product	708	0.91%	1.35
		C53543	Rare Non-Neoplastic Disorder	504	0.65%	1.22
	stable	C64389	Genomic Feature Physical Location	1,026	1.32%	0.000
		C23988	Mouse Neoplasms	886	1.14%	0.000
		C48232	Cancer TNM Finding	742	0.96%	0.000
		C53798	Adverse Event Associated with Surgery & Intra-Operative Injury	707	0.91%	0.000
		C43877	American Indian	555	0.72%	0.000
		C53832	Infection Adverse Event with Unknown Absolute Neutrophil Count	386	0.50%	0.000

Table 2: Largest (un)stable ontology regions in 2009

analysis, we regard all ontology regions of a certain minimal size below the 5%-percentile as stable and all ontology regions above the 95%-percentile as unstable.

Table 2 displays the six largest (un)stable ontology regions of GO and NCIT in 2009. The relative region sizes vary between 0.5% and 5% of the overall ontology size. In GO the relative sizes of the six largest unstable regions are higher than the stable ones. Particularly, the largest stable region in GO exhibits a relative size of 1.32% (GO:0031300) whereas the 6th largest unstable region (GO:0048646) has 1.4% relative size. The largest stable regions regarding absolute size can be found in NCIT consisting of more than 400 concepts. Furthermore, all stable regions of NCIT exhibit no average costs, i.e., in these regions no changes occurred. In contrast, some stable regions of GO show slight average costs, e.g., GO:0050865 or GO:0075136. We further observed that in GO-BP "anatomical structure" topics were highly modified in 2009 (see GO:0009653, GO:0048856 or GO:0048646). Furthermore, in GO-MF the change focus was on special binding functions such as "receptor binding" and "nucleic acid binding". Particularly, "receptor binding" is the largest unstable region of GO (*rel_size*=4.31%). In NCIT "Retired Concept" is the largest unstable region (*rel_size*=4.21%). Note that this ontology region is utilized to collect all ontology concepts that have been retired. Other regions of high interest concern "Drugs and Chemicals" topics such as "Industrial Aid" or "Natural Product".

4.4 Tracking the stability of ontology regions

A sample application of our discovery algorithm is tracking the stability of ontology regions over time. Particularly, we apply our region measures for different time periods to determine the change intensity of different regions over time. We can thus

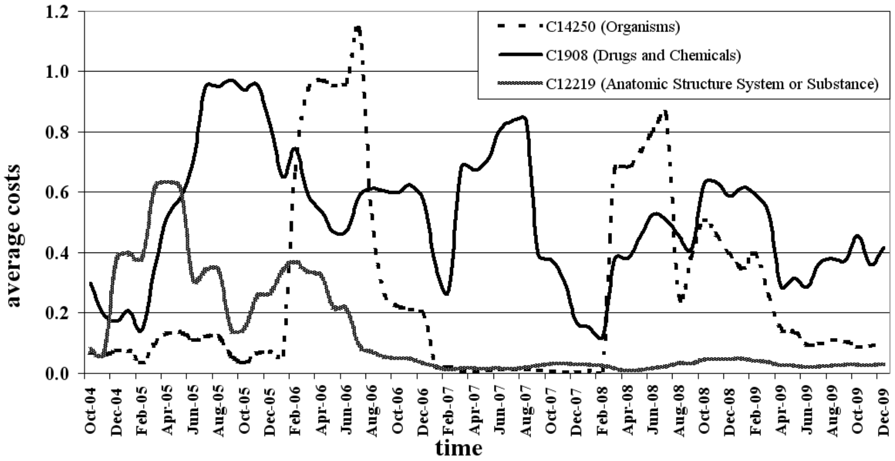


Figure 6: Tracking of *avg_costs* for sample regions in NCIT (2004-2009)

observe certain trends in the evolution of ontologies that are of interest to ontology users.

As an example we applied region tracking on NCIT between 2004 and 2009 for its 20 main categories. The computation uses a sliding window in the following way. We apply our algorithm for a window of size ‘half year’ (window step: 1 month), i.e., for each window we compute region measures for the selected categories and consider them for a final trend analysis. Hence, we can study variances in the measured results over time, e.g., to find out where and when massive development took place or not.

The chart in Fig. 6 shows the tracking of average costs for three selected main categories of NCIT between 2004 and 2009. We can distinguish different patterns. First, we observe regions, such as “Drugs and Chemicals”, that are always unstable, i.e., they experience higher average costs due to frequent modifications. Such regions represent active research fields, and will likely be modified in the near future as well. Furthermore, there are regions such as “Organisms” which exhibit both, periods of high stability mixed with periods of substantial instability. Its instability peaks (Mar 2006-Feb 2007, Mar 2008-Mar 2009) may be caused by new research findings or restructuring decisions by the project consortium which coordinates the ontology development. Finally, there are regions which have become stable over time. For instance, “Anatomic Structure System or Substance” had change activities until the end of 2006, but remained largely stable since 2007. Hence, such a region can be considered as almost finished, i.e., the probability for dramatic changes in the near future is low. This observation especially holds for ontology regions covering accepted / standardized knowledge, e.g., anatomy in the life sciences.

5 Conclusion and Future Work

We introduced the notion of ontology regions and corresponding measures to determine the change intensity or stability of ontology parts. Based on this notion we proposed an algorithm to discover evolving (un)stable regions in life science ontologies by taking ontology changes and the ontology structure into account. The presented algorithm utilizes an adaptable change cost model to reflect the impact of different ontology changes. Our approach can be used in different scenarios, e.g., by ontology users to find out the need to rerun analysis applications or by ontology engineers to notice past and ongoing work in regions of an ontology. We applied our algorithm in a comparative study for two large life science ontologies for different time periods. We observed that the algorithm is able to discover (un)stable ontology regions. The tracking of ontology region stability over time showed different evolution patterns, e.g., ontology regions which are always heavily modified or others that have become stable over the past years.

We see several directions for future work. First, we can consider high-level ontology changes such as merge or split of concepts to achieve a more fine-grained representation of ontology evolution. Second, we plan to integrate the region discovery algorithm into our OnEX system [8]. Finally, we will investigate how algorithms for ontology matching can utilize information about (un)stable regions to determine new ontology mappings in a more efficient way.

Acknowledgments. This work is supported by the German Research Foundation (DFG), grant RA 497/18-1 ("Evolution of Ontologies and Mappings").

References

1. Bodenreider, O., Stevens, R.: Bio-ontologies: current trends and future directions. Briefings in Bioinformatics 7(3), 256-274 (2006)
2. Boutet, E., Lieberherr, D., Tognolli, M.: UniProtKB/Swiss-Prot. Methods in Molecular Biology 406, 89-112 (2007)
3. Boyle, E.I., Weng, S., Gollub, J. et al.: GO::TermFinder - open source software for accessing Gene Ontology information and finding significantly enriched Gene Ontology terms associated with a list of genes. Bioinformatics 20(18), 3710-3715 (2004)
4. caBIG Strategic Planning Workspace: The Cancer Biomedical Informatics Grid (caBIG): infrastructure and applications for a worldwide research community. Studies Health Technology and Informatics 129, 330-334 (2007)
5. Euzenat, J., Shvaiko, P.: Ontology matching. Springer (2007)
6. Floris, G., Manakanatas, D., Kondylakis, H. et. al: Ontology change: classification and survey. The Knowledge Engineering Review 23(2), 117-152 (2008)
7. The Gene Ontology Consortium: The Gene Ontology project in 2008. Nucleic Acids Research 36(Database issue), D440-D444 (2008)
8. Hartung, M., Kirsten, T., Gross, A., Rahm, E.: OnEX – Exploring changes in life science ontologies. BMC Bioinformatics 10, 250 (2009)

9. Hartung, M., Kirsten, T., Rahm, E.: Analyzing the Evolution of Life Science Ontologies and Mappings. In: Bairoch, A., Cohen-Boulakia, S., Froidevaux, C. (eds.) DILS 2008. LNCS (LNBI), vol. 5109, pp. 11-27. Springer Heidelberg (2008)
10. Hubbard, T.J., Aken, B.L, Ayling, S. et al.: Ensembl 2009. Nucleic Acids Research 37(Database issue), D690-D697 (2009)
11. Kirsten, T., Hartung, M., Gross, A., Rahm, E.: Efficient Management of Biomedical Ontology Versions. In: Meersman, R., Herrero, P., Dillon, T.S. (eds.): On the Move to Meaningful Internet Systems Workshops. Proceedings. LNCS, vol. 5872, pp. 574-583. Springer (2009)
12. Klein, M., Fensel, D.: Ontology versioning on the Semantic Web. In: Proceedings of the International Semantic Web Working Symposium (SWWS), pp. 75-91 (2001)
13. Komatsoulis, G.A., Warzel, D.B., Hartel, F.W. et al.: caCORE version 3: Implementation of a model driven, service-oriented architecture for semantic interoperability. Journal of Biomedical Informatics 41(1), 106-123 (2008)
14. Lambrix, P., Tan, H., Jakoniene, V., Strömbäck, L.: Biological Ontologies. In Semantic Web: Revolutionizing Knowledge Discovery in the Life Sciences, pp. 85-99 (2007)
15. Noy, N., Chugh, A., Liu, W. et al.: A Framework for Ontology Evolution in Collaborative Environments. In: Proc. of the 5th Intl. Semantic Web Conference (ISWC), pp. 544-558 (2006)
16. Noy, N., Klein, M.: Ontology evolution: Not the same as schema evolution. Knowledge and Information Systems 6(4), 428–440 (2004)
17. Noy, N., Musen, M.: Promptdiff: a fixed-point algorithm for comparing ontology versions. In Proc. 18th Intl. Conference On Artificial Intelligence, pp. 744-750 (2002)
18. Noy, N., Musen, M.: Ontology versioning in an ontology management framework. IEEE Intelligent Systems 19(4), 6-13 (2004)
19. Papavassiliou, V., Flouris, G., Fundulaki, I. et al.: On Detecting High-Level Changes in RDF/S KBs. In: Proc. of the 8th Intl. Semantic Web Conference (ISWC), pp. 473-488 (2009)
20. Plessers, P., De Troyer, O.: Ontology Change Detection Using a Version Log. In: Proc. of the 4th Intl. Semantic Web Conference (ISWC), pp. 578-592 (2005)
21. Prüfer, K., Muetzel, B., Do, H.H. et al.: FUNC: a package for detecting significant associations between gene sets and ontological annotations. BMC Bioinformatics 8, 41 (2007)
22. Sioutos, N., de Coronado, S., Haber, M.W. et al.: NCI Thesaurus: A semantic model integrating cancer-related clinical and molecular information. Journal of Biomedical Informatics 40(1), 30-43 (2007)
23. Smith, B., Ashburner, M., Rosse, C. et al.: The OBO Foundry: coordinated evolution of ontologies to support biomedical data integration. Nature Biotechnology 25(11), 1251-1255 (2007)
24. Stojanovic, L., Maedche, A., Motik, B. et al.: User-driven ontology evolution management. In: Proc. of 13th International Conference On Knowledge Engineering and Knowledge Management (EKAW), pp. 285-300 (2002)
25. Stojanovic, L., Motik, B.: Ontology evolution within ontology editors. In: Proceedings of the International Workshop on Evaluation of Ontology-based Tools, pp. 53-62 (2002)