Large-Scale Ontology Matching: State-of-the-Art Analysis

PETER OCHIENG and SWAIB KYANDA, Makerere University, Uganda

Ontologies have become a popular means of knowledge sharing and reuse. This has motivated the development of large-sized independent ontologies within the same or different domains with some overlapping information among them. To integrate such large ontologies, automatic matchers become an inevitable solution. However, the process of matching large ontologies has high space and time complexities. Therefore, for a tool to efficiently and accurately match these large ontologies within the limited computing resources, it must have techniques that can significantly reduce the high space and time complexities associated with the ontology matching process. This article provides a review of the state-of-the-art techniques being applied by ontology matching tools to achieve scalability and produce high-quality mappings when matching large ontologies. In addition, we provide a direct comparison of the techniques to gauge their effectiveness in achieving scalability. A review of the state-of-the-art ontology matching tools that employ each strategy is also provided. We also evaluate the state-of-the-art tools to gauge the progress they have made over the years in improving alignment's quality and reduction of execution time when matching large ontologies.

CCS Concepts: • **Information systems → Entity resolution**;

Additional Key Words and Phrases: Survey, ontology mapping, mapping repair, repair, scalability

ACM Reference format:
Peter Ochieng and Swaib Kyanda. 2018. Large-Scale Ontology Matching: State-of-the-Art Analysis. *ACM Comput. Surv.* 51, 4, Article 75 (July 2018), 35 pages.
https://doi.org/10.1145/3211871

1 INTRODUCTION

Ontologies play a key role in the sharing and reusing of knowledge among software agents [104]. Due to their importance, a number of ontologies have been developed with each ontology describing a given domain from its subjective view. This has resulted in the existence of multiple ontologies in the same or different domains with some level of heterogeneity among them [31]. To resolve this heterogeneity, ontology matching is usually performed to find correspondences between semantically related entities of different ontologies [31]. Consequently, many tools have been developed to perform ontology matching [79] [99]. However, with the increased pervasiveness of ontologies, challenges have emerged that the ontology matching tools have to address to establish high-quality correspondences between ontologies within limited computing resources [82]. In this article, we focus on reviewing tools and techniques that have been developed toward matching large ontologies. Large ontologies such as Foundational Model of Anatomy (FMA) [87], National Cancer Institute (NCI) [35], Systematized Nomenclature of Medicine-Clinical Terms

Authors' addresses: P. Ochieng and S. Kyanda, Makerere University P.O BOX 7062, Kampala Uganda; emails: {onexpeters, kswaibk}@gmail.com.

https://doi.org/10.1145/3211871

(SNOMED CT) [95], Gene Ontology (GO) [10], Cyc [68], Sumo [83], and the like, present scalability challenge to the tools matching them. Therefore, for a tool to handle the task of matching such large ontologies, they require additional techniques beyond those required to match medium- and small-sized ontologies. Some of the key challenges that large ontologies pose to the tools matching them include:

Increased Complexity of the Matching Process. Ontology matching tools should generate mappings that have high precision and recall regardless of the size or type of the input ontologies. However, as the sizes of the input ontologies increase, so does the number of axioms that an ontology mapping tool has to reason over to generate accurate and complete mappings. This demand for more reasoning power increases the overall complexity of the matching process, resulting in considerable decrease in mapping quality with respect to standard measures of precision and recall. Therefore, an increase in the sizes of input ontologies penalizes the quality of mappings a tool generates [88].

Demand for More Memory. To match entities of two input ontologies, a naive approach entails comparing each entity of the source ontology against all entities of the target ontology. This Cartesian product approach of matching entities of two input ontologies results in a space complexity of $O(n^2)$ (assuming that each input ontology has n entities). A space complexity of $O(n^2)$ entails maintaining in memory several similarity values between the entities of the source and the target ontologies. This problem is further compounded by the fact that most ontology matching tools integrate multiple matchers to improve the quality of mappings they produce. Consequently, the number of similarity values that will have to be maintained in memory will be $k(n^2)$, with k being the number of matchers composed in a tool. An ontology matching process with a space complexity of $O(n^2)$ can easily lead to an out-of-memory error in case of a large n.

Increased Execution Time of the Mapping Process. Similar to the space complexity, the matching of two input ontologies in a Cartesian product fashion has a time complexity of $O(n^2)$ if each ontology has n entities. If we consider the efficiency of a matcher and assume that for each similarity computation the matcher takes time t, then the overall time complexity of the matching process is $O(n^2 \times t)$. Unfortunately, users are generally impatient and a process that has a time complexity $O(n^2 \times t)$ would require them to significantly wait for the final mappings even for a modest number of n entities.

Therefore, for a tool to be fit for the task of matching large ontologies, it has to incorporate strategies to handle the complex matching process and reduce the search space and time complexities. These challenges help us in focusing this review. A number of articles already exist in literature that provide the-state-of-the-art review in the ontology matching domain. Key among the papers include References [79], [86], and [99]. This review is mainly focused on two key aspects of matching large ontologies. First, we discuss scalability techniques being employed by different ontology matching tools to reduce high time and space complexities associated with matching large ontologies. Second, we provide a discussion on techniques that ontology matching tools are employing to establish high quality mappings when matching large ontologies. Unlike the review in Reference [86], which also provides a discussion on large ontology matching, we provide a low-level review on each scalability technique being discussed. Further, we provide a direct comparison of a given set of tools applying a given scalability technique with a goal of highlighting how their implementations differ. Table 1 provides a summary of our contributions as compared to Shvaiko and Euzenat [99], Rahm [86], and Otero-Cerdeira et al. [79]. It summarizes four key aspects covered in the article, i.e., techniques for reducing space and time complexities, techniques for ensuring quality mappings, and the progress tools have made so far.

Table 1. Showing the Contributions of This Article

Technique	Shvaiko and Euzenat (2013)	Rahm (2011)	Otero-Cerdeira et al. (2015)	This Article
Ontology Partitioning				
Module extraction	✗	✗	✗	✓
Complete ontology partitioning	✗	✓	✗	✓
Use of Data Structures				
Indexes (structural and lexical)	✗	✗	✗	✓
HashMap or Hashtable	✗	✗	✗	✓
Use Ontology structure				
Exploiting ontology's structural relationships to reduce search space	✗	✓	✗	✓
Self Tuning Match				
Self tuning match	✗	✓	✗	✗
Parallel Matching				
Parallelazation based on instruction vs data relationship	✗	✗	✗	✓
Parallelazation based on matcher implementation.	✗	✓	✗	✓
Progress in large ontology matching	✓	✗	✗	✓
Mapping Repair				
Pay as you go technique	✗	✗	✗	✓
Automatic repair	✗	✗	✗	✓

1.1 Definition of Terms

In this section, we define key terms used in this article to make it self-sufficient.

Ontology Matching. The process of finding semantic relationships that exist between entities of two ontologies. For example, in Figure 1, ontology matching would entail establishing the relationships that exists between concepts of Ontology 1 and Ontology 2. In ontology matching, key relationships that a matching tool seeks to establish between entities of two ontologies are equivalence (≡), subsumption(⊑), and Disjointness ($A \sqcap B \sqsubseteq \perp$). In Figure 1, an ontology matching tool may establish that {Person ≡ Agent}, {Author ⊑ Person}, and {Documents ⊓ Person ⊑ ⊥}.

Correspondence. Given two ontologies O_T and O_S, a correspondence between O_T and O_S is a triple $\langle e_1, e_2, r\rangle$ with the entity $e_1 \in O_S$, $e_2 \in O_T$ and r is the semantic relationship that exists between e_1 and e_2. It can also be referred to as a mapping. In some instances, a correspondence can be represented as a 4-uple; in such a case, the degree of confidence of a correspondence is also included, i.e., $\langle e_1, e_2, r, v\rangle$ where $v \in [0, 1]$ is the confidence level of the semantic relationship r. A possible correspondence from Figure 1 is ⟨ Person,Agent,≡,0.77⟩.

Alignment. This is a set of correspondences between two ontologies. For example, in Figure 1, a possible alignment A is A = {⟨Person,Agent,≡,0.77⟩, ⟨Document,Documents,≡,0.96⟩, ⟨WrittenBy,hasWritten,≡,0.90⟩, ⟨PaperReview,Review,≡,0.84⟩}.

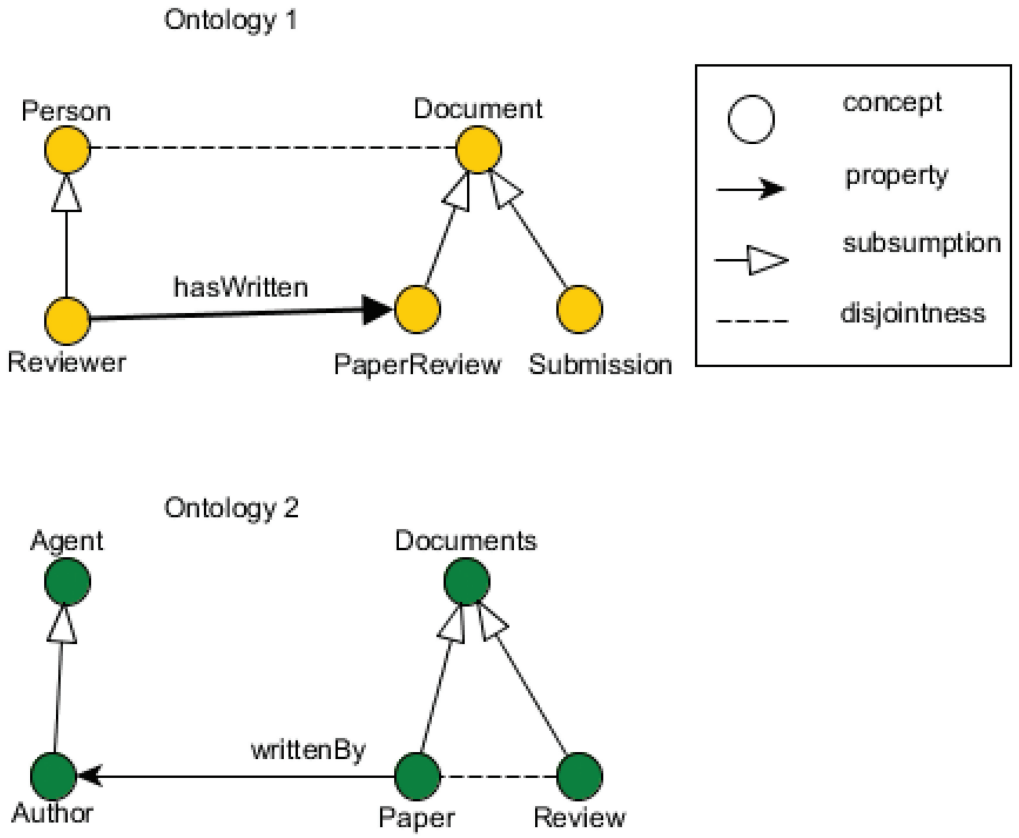

Fig. 1. Fragments of ontologies.

Matcher. An automatic matching algorithm employed by an ontology matching tool to establish an alignment between two ontologies.

Similarity Matrix. Given a source and target ontology S and T with n and m entities, respectively, a similarity matrix $M = (s_{ij})_{i=1,\ldots n, j=1,\ldots m}$ generated by a matcher L is a matrix where each entry s_{ij} shows the similarity value between entities i and j with $i \in S$ and $j \in T$.

Precision. A measure of the ratio of correctly found correspondences over the total number of returned correspondences as shown in Equation (1). Precision is used to check the degree of correctness of the ontology matching algorithm:

$$precision = \frac{\text{correct correspondences}}{\text{total returned correspondences}}. \tag{1}$$

For example, in Figure 1, if a matcher L generates a final alignment $A_1 = \{\langle$Person,Agent,≡,0.77$\rangle$, $\langle$Document,Documents,≡,0.96$\rangle$, $\langle$Reviewer,Review,≡,0.84$\rangle\}$, taking the alignment A as the reference alignment, then the mapping $\langle$ Reviewer,Review,≡,0.84$\rangle\}$ is flagged as a wrong mapping. The precision of the alignment A_1 is

$$precision = \frac{2}{3} = 0.6667.$$

Recall. It measures the ratio of correctly found correspondences over total number of expected correspondences as shown in Equation (2). It is used to check the degree of completeness of an ontology matching algorithm:

$$Recall = \frac{\text{correct correspondences}}{\text{expected correspondences}}. \tag{2}$$

The recall of the alignment A_1 when A is the reference alignment is

$$Recall = \frac{2}{4} = 0.5.$$

Even though precision and recall are widely used and accepted measures, in some occasions it may be preferable having a single measure to compare different systems or algorithms. Moreover, systems are often not comparable based solely on precision or recall, because the one that has a higher recall may have a lower precision and vice versa [31]. Therefore, F-measure was introduced according to Equation (3):

$$F - measure = \frac{2 \times recall \times precision}{precision + recall}. \tag{3}$$

F-measure of the alignment A_1 is

$$F - measure = \frac{2 \times 0.6667 \times 0.5}{0.6667 + 0.5} = 0.5714.$$

In each subsequent section of this article, we (1) describe a specific technique, (2) compare tools using the technique, (3) give a review of the tools using the technique, and (4) give some insights on the technique being discussed. It is also important to note that tools have progressively changed from the original developed version, so for each technique we discuss the version of the tool that implements the technique.

2 SCALABILITY TECHNIQUES

In this section, we review techniques and tools that are geared toward achieving scalability in ontology matching. We use the categorization proposed in Reference [86]:

(1) Reduction of search space techniques.
(2) Parallel matching.

2.1 Reduction of Search Space

Most ontology mapping tasks involve producing 1:1 cardinality mappings. This, therefore, means that given an entity e_i in the source ontology, the vast majority of entities in the target ontology need not to be directly compared to the entity e_i through similarity computations. To reduce the quadratic complexity n^2 of computing similarity values between entities of source and the target ontologies in an all-against-all fashion, for each entity e_i of the source ontology, the number of potential entities from the target ontology that it can be matched with should be kept at a minimum [34]. The entities of the target ontology that are eliminated are those that will yield zero or low similarity values in case of similarity computations. Techniques that are currently being used by ontology matching tools to reduce search space include:

(1) Ontology partitioning.
(2) Use of data structures.
(3) Use of ontology structure.

2.1.1 Ontology Partitioning. The current forms of ontology partitioning that ontology matching tools are implementing to support ontology matching process include:

(1) Module extraction.
(2) Complete ontology partitioning.

Module Extraction. The ontology matching process relies heavily on reasoning over the relationships that exist between entities of an ontology to disambiguate entities' meanings. However, the reasoning process is costly in terms of its memory demands and execution time, especially for large ontologies; therefore, ontology matching tools are implementing strategies that avoid reasoning over an entire ontology if only a fragment of it can provide the required answer. Modularization of an ontology is a strategy that views an ontology as an artifact similar to a program that can get quite large and complex, hence extracting a module from it and reasoning over the module as opposed to the whole ontology, which can significantly speed up the reasoning process and optimize memory utilization. The extracted module must guarantee coverage, i.e., it must capture all information contained in an ontology related to a set of entities to enable effective reasoning (see also the definition of justification in Definition 8). Research on module extraction has been explored in References [21], [37], and [39]. Module extraction has been used in ontology matching mostly in the alignment repair process as described in Section 3.2 to speed up the reasoning process of detecting coherence violations in the alignment produced by a tool. Tools that implement modularization include LogMap2 [59] and AML [89].

Complete Ontology Partitioning. Here, a large ontology is broken down into sub-ontologies (partitions) based on some selected criteria. Ideally, an ontology should be partitioned into blocks representing stand-alone topics that exist in it; however, the goal for which ontology partitioning is sought dictates how it is partitioned. Apart from ontology matching purposes, an ontology can also be partitioned for reuse and maintenance purposes. The standard procedure for matching ontologies based on ontology partitioning involves four key steps:

(1) Partition the source (O_S) and the target ontologies (O_T) based on a given partitioning algorithm.
(2) For each partition (P_i) $\in O_S$, find its likely match (P_j) $\in O_T$ where $|P_j| \geq 1$ with an ideal case being one to one match.
(3) Match entities between the matched partitions.
(4) Establish the final alignment.

Other methods that can be considered are extensively discussed in Reference [103]. In addition to reducing the search space, complete ontology partitioning plays three more key roles in advancing scalability:

(1) Maintaining effectiveness of a matching tool.
(2) Supporting parallelization.
(3) Reducing time complexity.

Reducing Search Space. To match entities of two ontologies, a naive non-scalable approach would be to compare each entity of the source ontology against all entities of the target ontology. This makes the number of comparison computations between the pairs of entities of source and the target ontologies to increase quadratically with the number of entities in the two ontologies. To avoid the Cartesian product of the entities of the source and the target ontologies, partitioning the input ontologies is usually performed. Partitioning of an ontology is a divide and conquer technique that divides a large ontology into k blocks or modules. By partitioning the input ontologies, assuming

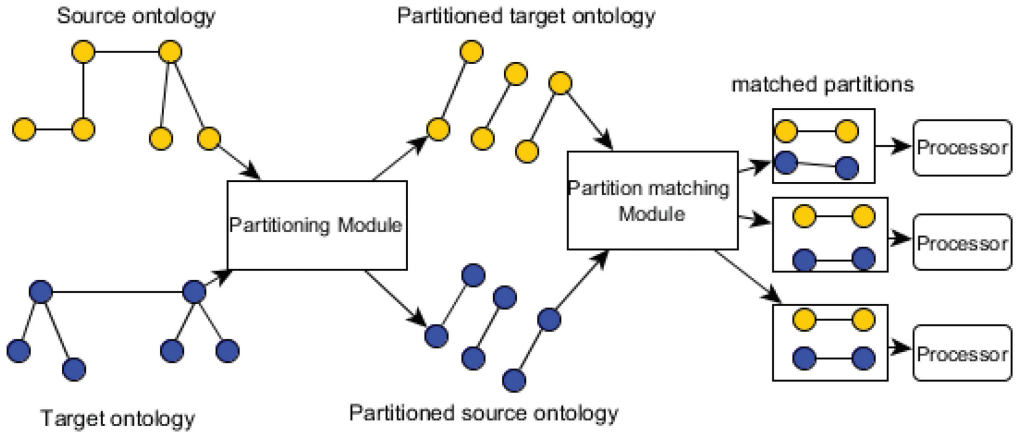

Fig. 2. Demonstrating how ontology partitioning supports parallelization.

each of the ontology has n entities to be matched, and the partitioning method results in k blocks (all of the same size containing $\frac{n}{k}$ entities), the resulting number of entity pair comparisons reduces the space complexity to O $(\frac{n^2}{k})$ [86] [103]. This has the double effect of speeding up the matching process and better utilization of the memory.

Supporting Parallelization. To reduce the execution time of the large-scale ontology matching process, the matching process task can be divided into multiple smaller processes and each process executed in its processor in parallel with other processes. Partitioning of input ontologies supports parallelization by enabling the matched blocks (partitions) of the source and the target ontologies to be executed in different processors or nodes in parallel as shown in Figure 2. Exploiting partitioning to support parallelism has specifically been used in VDoc+ [111] and [107], where they use MapReduce framework [19] to implement the matching process in parallel. In MapReduce, input data is split into key-value pairs. These pairs are then partitioned into different reducers based on shared keys. All values that share the same keys are partitioned together. Different partitions can then be executed in parallel in different computing nodes. In VDoc+, for each entity e_i, they identify important words from the entity's annotations and local name, which can be used to discriminate (classify) the entity. TF-IDF is used to rank the important words of an entity. Each important word of an entity acts as the key while the entity is the value. Entities that share the same words as their keys are partitioned together. Matching of entities is restricted only to entities that share the same key (i.e., only entities that are in the same partition can be matched). The matching process in each partition is then executed in different nodes in parallel. In Reference [107], they first independently partition the input ontologies using the partitioning algorithm proposed in Reference [45]. They then align the two most similar partitions of the source and target ontologies. Two aligned partitions are assigned a unique key, which is used to identify them. These partitions are then fed into the reducer with their respective keys. Matching of entities in restricted to partitions with the same key. The matching tasks are executed in parallel in different nodes. SPHeRe [5] also proposes creation of subsets of datasets from the input ontologies such that the different datasets will be executed at different processing units (PUs) in parallel.

Maintaining Effectiveness of a Matching Tool. An ontology matching tool should be able to maintain the quality of mappings it produces with regard to standard measures of precision and recall, regardless of the type and sizes of input ontologies being matched. However, as the sizes of the input ontologies increase, so does the number of axioms that a tool has to reason over for it to

establish mappings between entities of the input ontologies. This demand for increased reasoning when matching large ontologies increases the level of complexity of the matching process resulting in a considerable decrease in efficiency (precision, recall) of the mappings that an automatic alignment tool produces [88]. One key strategy of mitigating this problem is by partitioning the source and the target ontologies. After partitioning the input ontologies, the most similar sub-ontologies of the source and target ontologies are then matched. Different matched sub-ontologies of the input ontologies can then be fed into the ontology matching tool for their entities to be aligned. In this case, matching of entities is restricted to only those entities within matched sub-ontologies. Consequently, reasoning is restricted to only the axioms that exist in the matched sub-ontologies, hence reducing matching complexity. The final alignment between the two input ontologies is the union of all mappings from different mapped sub-ontologies. Through partitioning, the size of an ontology matching load can be controlled to that which a tool can handle effectively without compromising its efficiency. Here, ontology partitioning is seen as a means of reducing the complexity of the matching process.

Reducing Time Complexity. Given the source and the target ontologies O_s and O_t, respectively, if each ontology has n entities and each pairwise matching task between an entity $e_i \in O_s$ and $e_j \in O_t$ takes time t, then the entire matching process of the two ontologies takes $(n^2 \times t)$ for a case of Cartesian product matching. However, if each of the ontologies is partitioned into k partitions, and entities' matching is restricted to matching entities between the matched partitions of the source and the target ontologies, then the time complexity is reduced to $(\frac{n^2}{k} \times t)$.

Requirements of an Ontology Partitioning Algorithm. Despite these benefits of ontology partitioning to the ontology matching process, if the partitioning is done poorly, it can be risky and can easily alter the original structure of ontology, therefore resulting in a low quality alignment [86]. Matching of the source and the target ontologies' partitions can also introduce new time and space complexities if the partitions are many due to small-sized partitions produced, hence eroding the benefits of ontology partitioning to the matching process. Consequently, an ontology partitioning algorithm that produces partitions geared toward ontology matching should:

(1) Produce partitions that are optimally small. An optimum size should be set to avoid the partitions either being too small, hence introducing new time and space complexities during matching the source and the target ontologies' partitions or being large hence not reducing the search space significantly.
(2) Allow some level of redundancy between partitions of a given ontology [86] [94], i.e., allowing some axioms to overlap from one partition to another. This increases the reasoning power and robustness within a partition, which is important in ontology matching.
(3) Minimize the distance between the entities within a partition. Distance in this context is in terms of similarity of entities. Similar entities should be partitioned together.
(4) Since one of the goals of partitioning is to reduce time complexity, the partitioning algorithm should be able to partition an ontology such that

$$T(P) < T_m(O) - \{T_m(P) + T_m(E)\}\,. \tag{4}$$

Where T(P) is the ontology partitioning time, $T_m(O)$ is the time that an algorithm would have taken to match entities of two ontologies without partitioning, $T_m(P)$ is the time taken to match partitions of the source and the target ontologies and $T_m(E)$ is the time for matching entities of all matched partitions. This is to ensure that partitioning provides benefits in reduction of time complexity.
(5) Produce locally correct and complete partitions, i.e., the relationships involving a given atomic concept, role, or individual entailed in the partitions should also be entailed in the

original ontology. Properties that ensure a partition is locally correct and complete are introduced in Reference [39].

Definition 1. Let O be an OWL-ontology to be partitioned with signature $S(O) = (C, R, I)$, where C,R,I represent sets of atomic concepts, roles, and individuals. A partition $P_i(O) \subseteq O$ is locally correct and complete for an atomic concept $A \in C$ in O if it meets the following conditions:

(1) $P_i(O) \models A \sqsubseteq B$ iff $O \models A \sqsubseteq B \; \forall B \in C$.
(2) $P_i(O) \models A(a)$ iff $O \models A(a) \; \forall a \in I$.

The two conditions ensure that the partitions do not introduce new relationships between entities that do not exist in the original ontologies. The same properties can be extended for atomic roles and individuals as demonstrated in Reference [39]. There are many ontology partitioning algorithms that exist in literature. A review can be found in Reference [80]. However, not all ontology partitioning algorithms produce partitions tailored for ontology matching. Therefore, there is a need to evaluate an algorithm against the properties enumerated above to gauge if it is fit for partitioning an ontology for matching purposes.

Ontology Partitioning Techniques. According to Reference [1] ontology partitioning techniques can be categorized into two groups:

(1) Graph-based approach.
(2) Logic-based approach.

Graph-Based Approach. This approach applies graph-based algorithms to traverse the ontology hierarchy to extract partitions. The technique is scalable since it avoids the reasoning approach, which for a large ontology results in high space and time complexities [1], however it may produce incomplete partitions since it ignores the semantics modeled in the underlying ontology language. Research works in References [16] [27] [45] [66] [77], [93], and [96] propose ontology partitioning algorithms by applying graph-based approaches.

Logic-Based Approach. This approach uses description logic to partition an ontology. It reasons over the relationships modeled in the ontology to generate more complete partitions as compared to the graph-based approach. Due to its dependence on reasoning, it is less scalable than the graph-based approach. This technique is implemented in research such as References [36], [37], and [38].

Comparing Tools That Use Partitioning in Ontology Matching. In this section, we compare ontology matching tools that use partitioning to achieve scalability. We specifically answer the following questions for each tool:

(1) Does it use module extraction or complete partitioning?
(2) What is the key purpose for partitioning, i.e., to achieve parallelization or search space reduction?
(3) Does it use a graph-based or logic-based technique for ontology partitioning?
(4) Does it achieve scalability? To gauge scalability of a tool, we check if a tool was able to complete all tasks in a large biomedical track at the OAEI conference during its participation—or rely on the results reported in the article reporting the tool's performance.
(5) Which part of the ontology matching process is the partitioning applied repair or matching entities?

The summary of the analysis is given in Table 2. From the analysis, the graph-based technique is the most popular way of partitioning ontology among the ontology matching tools that adopt

Table 2. Comparing Tools That Use Partitioning in Ontology Matching

Tool	Modular extraction	Complete partitioning	Parallelization	Search space reduction	Logic based?	Graph based?	Scalable?
LogMap2	Yes	No	No	Yes	Yes	No	Yes
AML	Yes	No	No	Yes	Yes	No	Yes
DKP-AOM	No	Yes	No	Yes	Yes	No	No
COMA++	No	Yes	No	Yes	No	Yes	Yes
Falcon-AO	No	Yes	No	Yes	No	Yes	Yes
COGOM	No	Yes	No	Yes	No	Yes	Yes
Anchor-flood	No	Yes	No	Yes	No	Yes	Yes
Optima+	No	Yes	No	Yes	No	Yes	Yes
GOMMA	Yes	No	No	Yes	No	Yes	Yes
VDoc+	No	Yes	Yes	Yes	No	No	Yes
Reference [107]	No	Yes	Yes	Yes	No	Yes	Yes

partitioning in the matching process. This may be attributed to its scalability nature. In most tools, partitioning is mainly geared toward search space reduction. Apart from DKP-AOM, all other tools achieve scalability making ontology partitioning an effective way of attaining scalability in ontology matching process. DKP-AOM does not achieve scalability since it relies on $owl:disjointWith$ axiom to perform partitioning. In most cases, ontologies are modeled with very few explicit $owl:disjointWith$ axioms, hence generating partitions based on disjoint axiom may still generate large partitions, hence no significant reduction in the search space. It should be noted, however, that LogMap and AML only use modularization in the ontology alignment repair process, hence partitioning is not their key way of achieving scalability.

In this section, we review some of the ontology matching tools that use partitioning as way of reducing time complexity.

DKP-AOM [32] partitions an ontology along $owl:disjointWith$ axioms that are modeled in the ontology. Despite being the ideal case of partitioning, most ontologies are modeled without explicitly including $owl:disjointWith$ axioms therefore limiting the effectiveness of partitioning an ontology using $owl:disjointWith$ axiom. This may explain why the tool fails to complete tasks in large biomedical ontologies track of OAEI.

COMA++ [3] [67] uses partitioning during ontology matching. It refers to each partition as a fragment. The fragment matching works in two stages. In the first phase, the fragments of a specified type (e.g., user-specified fragments or subschemas such as relational tables or message formats in large XML schemas) are determined and compared with each other to identify the most similar fragments between the two schemas (ontologies) being matched. The search for similar fragments is similar to partition matching. In the second phase, entities of the matched fragments are aligned.

Falcon-AO [52] operates in three phases to address large ontology matching. It first partitions entities of the input ontologies into a set of clusters. It then matches the clusters from the source and the target ontologies based on pre-calculated anchors. It finally performs matching of concepts. Matching of concepts is restricted to to the concepts that are within matched clusters.

COGOM [91] splits input ontologies into partitions using the notion of concept network. The partitions of an ontology are created such that semantically similar concepts are partitioned together. Similar partitions of the source and target ontologies are then matched. Entity matching is restricted to matched partitions hence significantly reducing search space.

Anchor-flood [20] addresses the scalability issue by partitioning the input ontologies. It first partitions an ontology by using the taxonomical and disjoint relations of the concepts in the ontology hierarchy. It then produces concept-level clusters by aggregating similar types of concepts and avoiding disjoint concepts. Finally, schema level matching algorithm is used to get concept-level cluster alignment.

Optima+ [50] partitions the source and the target ontologies by exploiting the structure of the ontologies. It then compares each block of the source ontology against those of the target ontology to find the matching partitions. Finally, it establishes an alignment by matching entities of the matched partitions.

GOMMA [41] is a tool that matches large ontologies in the life science domain. GOMMA identifies the relevant part of the broader ontology between the two input ontologies that are relevant (has more overlap) to be matched with the smaller ontology.

2.1.2 Use of Data Structures.

Indexing. The most common data structure that ontology matching tools are exploiting to reduce search space during ontology matching is indexing. Indexing is commonly used in information retrieval to speed up information search [7]. In ontology matching, indexing entails creating an inverted index for either one of the input ontologies or both them. There are two main types of indexes that can be created during ontology matching:

(1) Structural index.
(2) Lexical index.

Structural Indexing. During ontology matching, establishing hierarchical relationships that exists between entities of an ontology is crucial in disambiguating the meaning of an entity. The structural information of a given entity helps in performing semantic verification and similarity propagation [74]. However, to establish the hierarchical information of an entity in a large ontology is costly in terms of its memory requirements and execution time. Consequently, efficient access to hierarchical information of entities is key for a tool to achieve scalability. One way of ensuring efficient access to hierarchical information of an entity is through structural indexing of an ontology such that transitive closure of the "IS-A" and "PART-OF" relationships of an entity can be answered more efficiently. Structural indexing of an ontology can be implemented by representing an ontology as a direct acyclic graph (DAG), and then the state-of-the-art techniques that have been proposed to speed up reachability within DAGs can be used to index the DAG of the ontology. Some of the research that proposes techniques to speed up reachability within DAG are References [2], [51], [61] and [101]. Tools that implement structural indexing include LogMap [58], which uses interval labeling technique proposed in Reference [2], and YAM++ [74] which indexes the structure using bits.

Lexical Indexing. Here, indexation of an input ontology involves a named entity, its label name(s), and annotations. Indexes are created by linking an entity to the normalized words that appear in its local name, label(s), and annotation(s). Once the lexical indexes of the source and target ontologies have been created, they are intersected to establish candidate mappings that exist between their entities. By doing this, a Cartesian product comparison of input ontologies' entities is avoided hence reducing the search space. This kind of indexing has been applied in tools such as LogMap [58] and YAM++ [74], and Reference [22].

Use of Hashmaps (or Hashtable). Some tools such as AML [34] and Eff2Match [11] apply the use of hashmaps to reduce the search space. In this scheme, an entity is stored in the hashmap as the key and its corresponding information such as the normalized words of its local name, label(s),

Table 3. Comparing Tools That Use Data Structures to Reduce Search Space

Tool	Data structure used	Year of participation	Number of task	Number of tasks completed	Scalable?	Average Execution time for all tasks
LogMap	Indexes	2016	6	6	Yes	243
AML	HashMap	2016	6	6	Yes	214
ServOMBI	Indexes	2015	6	2	No	did not complete all tasks
IAMA	Indexes	2013	6	6	Yes	117
YAM++	Indexes	2013	6	6	Yes	344

and annotations are the values associated with the key. A key or value from the hashmap of one of the input ontologies is then used to query the hashmap of the other input ontology directly, hence reducing the search space from $O(n^2)$ to $O(n)$.

Evaluation of Tools That Use Data Structures to Reduce Search Space. In this section, we evaluate the effectiveness of data structures in achieving scalability in ontology matching. We use the information reported at the OAEI conference in the year a tool participated. For a tool that has participated in the OAEI conference in multiple years, we use the results reported in its most current year of participation. A tool is rated as having achieved scalability if it was able to complete all the tasks in the large Biomedical Ontology Track (BOT). The analysis is shown in Table 3. Out of the five tools shown in Table 3, four of the tools, i.e., AML, LogMap, IAMA, and YAM++, achieved scalability. Only ServOMBI failed to achieve scalability. This signals the effectiveness of the use of data structures as means of achieving scalability in ontology matching. Other tools that employ data structures such as SLINT+ and EXONA are instance matching tools, hence were not evaluated in BOT, while CLONA focuses on multilingual ontologies, hence was also not evaluated in BOT. Out of the eight tools discussed in this section, seven of them use indexes, making indexing the most popular data structure being implemented by tools to achieve scalability.

Tools That Use Data Structures. In this section, we review some of the ontology matching tools that use data structures technique as means of reducing search space

CLONA [110] is an ontology matching system that focuses on multilingual ontologies. It addresses quadratic complexity by using Lucene to develop an index of the concepts, relationships, data types, and instances in the input ontology. It then runs a cross search between the source and the target indexes of ontologies to establish initial candidate mappings of entities.

EXONA [15] is an ontology matching system solely for instance matching. It indexes instances of input ontologies using URI of the instance. It then does a cross index query of the source and the target ontologies indexes to establish initial candidate mappings.

LogMap Family [54] is a tool that uses logic reasoning to refine initial mappings. To generate initial candidate mappings, it creates an inverted index of an ontology by using the labels of entities together with their annotations. It then intersects the inverted indexes of the input ontologies to establish entities of source and target ontology with matching string. This becomes initial mapping candidates that need to be refined. LogMap currently has three variants: LogMaplt, LogMapC, and LogMapBio.

IAMA [112] indexes the lexical information of a larger ontology between the target and source ontology. The normalized entity names of the of the smaller ontology is then used to query the created index to create initial mapping pairs.

ServOMBI [64] uses indexing strategy to reduce the search space. It performs a terminological-based inverted index, which is built from ontology entities. It also uses parallelization to speed up indexing.

AgreementMaker Light (AML) [34] employs hashmaps cross-searches, i.e., searches where a key in a hashmap is used to query another hashmap directly and thus take O(n). This strategy avoids comparing all the elements of the source and the target ontologies.

SLINT+ [75] performs generation of candidates to be matched by using shared objects. If instances share objects, then they are paired as candidates for matching. This helps to trim search space and avoid all against all comparison.

YAM++ [73] [74]. To reduce the search space, YAM++ creates a context index for the larger ontology between the source and the target ontologies. The context index contains all entities and their respective annotations, all entities and their respective ancestors, and all entities and their respective descendants. Lucene is used in the indexing. The entities of the smaller ontology is used to query the index of the larger ontology to establish the initial candidate mappings.

2.1.3 Use of Ontology Structure. In OWL ontology, entities are linked through relationships that are modeled in the ontology in form of axioms. OWL has class, property, and individual axioms that model relationships between classes, properties, and individuals, respectively. Ontology matching tools exploit these relationships to restrict the matching space of two ontologies. Some of the techniques being used include: -

(1) The use of owl:disjointWith axiom. Given a source and target ontology O_s and O_t, respectively, if there exists a pair of entities $(X, X') \in O_s$ and $(Y, Y') \in O_t$ such that $O_s \models X \sqcap X' \sqsubseteq \bot$ and $O_t \models Y \sqsubseteq Y'$, if according to some similarity computation X is matched to Y, then the similarity computation between X' and its children and Y and its children can be avoided. Likewise similarity computation between Y' and its children and X and its children can be avoided.
(2) Use of similarity flooding as proposed in Reference [72]. Given a source and target ontology O_s and O_t, respectively, if there exists a pair of entities $(X, X)' \in O_s$ and $(Y, Y') \in O_t$ such that $O_s \models X \sqsubseteq X'$ and $O_t \models \neg(Y \sqsubseteq Y')$, if according to some similarity computation X is matched to Y, then the pairwise similarity computation between X' and its children and Y' and its children can be avoided. Likewise, pairwise similarity computation between Y' and its children and X and its children can be avoided.

Some of the tools that exploit this technique include:

CODI [46] is an ontology matching tool that uses Markov logic [26] to produce mappings. It employs the principle of stability and coherence in the two ontologies to be aligned as proposed in Reference [76] to restrict mapping space.

Lily [108] employs reduction anchors to address the challenge of scalability. Lily uses two key strategies. If $a_1 \in O_1$ matches $b_p \in O_2$, the similarity calculation between the super and sub-concepts of b_p and the entity a_1 can be skipped since a_1 is already matched to b_p, hence saving time. If $a_1 \in O_1$ does not match b_p, then the neighbor of a_1 do not match b_p as well, therefore the similarity computation are skipped saving time.

2.2 Parallel Composition

To improve the performance of computer systems, processors are being manufactured with an increasing number of cores. Likewise, to keep up with the ever-increasing computation power of processors, programming languages are also developing libraries that programmers can use to take advantage of the increased power of the processors. The libraries provide mechanisms of

distributing different tasks of a given process over the multiple cores so that they can be executed in parallel. This speeds up the overall execution time of the entire process. Parallelism is mostly exploited when executing processes with high time complexity. Ontology matching is one such process; consequently, some ontology matching tools have explored the use of parallelism as means of reducing the overall execution time of the process. Parallelization in ontology matching can be discussed based on various implementation techniques that exist in literature [8]. In this review article, we discuss parallelization in ontology matching tools under two key categories:

(1) Instruction vs data relationship.
(2) Matcher implementation.

2.2.1 Instruction vs Data Relationship. Under this categorization, the implementation of parallelization is differentiated based on whether a similar instruction is executed on different datasets or multiple instructions are executed on the same or different datasets. Here, parallelization is implemented in three main ways [106]:

(1) Single instruction multiple data.
(2) Multiple instruction single data.
(3) Multiple instruction multiple data.

Single Instruction Multiple Data (SIMD). In this model, all PUs execute the same instruction on different data elements [8]. In ontology matching, this model has been implemented by different tools at different levels of the ontology matching process. SPHeRe [5] exploits SIMD during the input ontologies loading stage, where the ontology load interface of SPHeRe executes the load instruction on both the source and the target ontologies for the input ontologies to be loaded into the memory in parallel. In the entity matching stage of the ontology matching process, SIMD implementation is achieved by partitioning the input ontologies. This therefore means that the application of SIMD in the entity matching stage of the ontology matching process also addresses the problem of space complexity (see Section 2.1.1 on how partitioning of input ontologies reduces space complexity). VDoc+ [111] implements SIMD during the matching of entities stage. It first uses MapReduce framework to partition the input ontologies into different independent clusters (see Section 2.1.1 for more details). The clusters are then executed in parallel on different processors. Here, all the clusters are processed by a single instruction, which is the use of cosine similarity metric to compute similarities between entities of a given cluster. ServOMBI [64] utilizes SIMD during the pre-processing stage of the ontology matching process where the terminology indexing command is executed on both the source and the target ontologies. This allows the entities of the source and the target ontologies to be indexed in parallel.

Multiple Instruction Single Data (MISD). In this implementation of parallelism, each PU executes a different instruction on the same dataset [8]. This is the most common model currently being exploited by ontology matching tools to achieve parallelism. It has been implemented at the entity matching stage of the ontology matching process. Different matchers are executed in parallel in different PUs to process the same input ontologies. For instance, if an ontology matching tool integrates three matchers, i.e., string, structural, and semantic-based matchers, each matcher running on a given PU is executed on the same input ontologies. The final alignment is established through the aggregation of results from the the three matchers. Tools that apply MISD include XMAP++ [24], MaasMatch [92], OMReasoner [97], InsMT [65], XMapGen [23], and XMapSig [23]. All these tools implement this model of parallellization at the entity matching level of the ontology matching process. MISD only reduces time complexity and not space complexity.

Multiple Instruction Multiple Data (MIMD). This is an implementation of parallelism where each PU is executing a different instruction on a different independent dataset [8]. In ontology matching, this model has been implemented at the entity matching stage of the ontology matching process. The implementation is achieved by first partitioning the input ontologies into different independent clusters. The clusters can then be processed at different PUs by different matchers. SPHeRe [5] implements MIMD inform of data parallelism [6] at the entity matching level of the ontology matching process. SPHeRe forms independent clusters of entities of the source and the target ontologies based on type of the available matchers. For instance, if there are only two types of matchers, i.e., string-based matcher and structure-based matcher, both the source and target ontologies will be serialized into two subsets, i.e., both the source and the target ontologies will have one subset containing an ontology's entities and the entities' respective names for purposes of string-based matcher while the other subset will contain an ontology's entities and structural information that exist between entities in the ontology, this is to be exploited by structural matcher. A pair of entities from the source and target ontologies within the same pair of serialized subset constitute a matching job to be processed by a matching task (a single independent execution of a matching algorithm over a resource from source and target ontologies). The matching task is replicated such that the total number of matching tasks MT_{total} is equal or greater than the Cartesian product of the number of entities in source and target ontologies, i.e., $M_{total} \geq m \times n$ with m and n being the number of entities in source and target ontologies, respectively. The matching tasks are distributed evenly over the available computing cores and are the basis for achieving parallelism in SPHeRe. The matching jobs are distributed among the relevant matching tasks to be processed in parallel.

2.2.2 Matcher Implementation. In ontology matching, it is generally an accepted principle that multiple features of an ontology must be exploited for a tool to generate quality mappings. Consequently, most tools are combining multiple matchers with each matcher exploiting a given feature of the ontology to compute similarity between entities of two ontologies. The multiple matchers can be composed in parallel, sequential or hybrid fashion [86]. Consider a case where each input ontology has n entities and each pairwise similarity computation between the entities of source and target ontologies takes a time t, then the total time complexity needed to match the two ontologies is $O(n^2 \times t)$. If an ontology matching tool integrates n matchers and executes all the matchers in a sequential manner with the subsequent matcher refining the results of the previous matcher, then the total time complexity needed to complete the matching process is $n(n^2 \times t)$ (assuming all matchers have equal complexity). This kind of implementation is not scalable to large ontology matching tasks. Based on this realization, some ontology matching tools are executing their matchers in parallel with a goal of reducing the execution time. According to Reference [40], parallel matcher composition can be implemented in two ways:

(1) Inter-matcher parallelization,
(2) Intra-matcher parallelization.

Inter-matcher Parallelization. In this implementation of parallelization, multiple independent matchers are executed in parallel to match entities of input ontologies by utilizing multiple available processors [40]. Each matcher generates a similarity matrix, which is then aggregated with similarity matrices from other matchers resulting in a single final similarity matrix. The final alignment between the input ontologies is established from the final similarity matrix through the application of different selection approaches [70]. Consider the earlier example of matching two ontologies where each ontology has n entities. If the n matchers of an ontology matching tool are executed in parallel in an inter-matcher fashion, then the time complexity is $O(n^2 \times t)$,

this therefore means that parallel execution of matchers using inter-matcher parallelization scales down the execution time by a factor of n as compared to the sequential execution. If the matchers are of different time complexities, the slowest matcher determines the overall execution time. Inter-matcher parallelization, however, does not address the challenge of high space complexity that exists during the matching of large ontologies. This technique therefore is not appropriate if space complexity reduction is desired, otherwise it has to be augmented by a space complexity reduction technique. Examples of tools that use this parallelization technique include MaasMatch [92], XMap++ [24], and CroMatcher [42].

Intra-matcher Parallelization. In this type of parallelization, apart from executing the multiple matchers in parallel, an individual matcher is also decomposed into several matching tasks it is composed of, then the tasks are executed in parallel within a matcher [40]. Intra-matcher parallelization therefore involves two level of results aggregation. First at the intra-matcher level and at inter-matcher level. This technique has been implemented by Reference [41].

2.3 Multiple Matcher Combination

A major challenge of executing matchers in parallel is how to combine the results of the multiple matchers into a final single similarity matrix. The choice of a combination technique can affect:

(1) The quality of the final alignment [85].
(2) The overall execution time of a matching tool.

Some of the techniques that exist in literature for aggregating mapping results from different matchers are defined below. Given the source and the target ontologies O_s and O_t, respectively, let $e_i \in O_s$ and $e_j \in O_t$. If a tool executes n matchers in parallel, i.e., $\{M_1, M_1, \ldots, M_n\}$ and each matcher generates a similarity value S_i $1 \leq i \leq n$ between e_i and e_j, then the weighted sum aggregation W_s is defined according to Equation (5):

$$W_s = \sum_{i=1}^{i=n} W_i S_i, \tag{5}$$

where W_i is the weight assigned to similarity value from matcher M_i.

The average weighted aggregation A_w is defined according to Equation (6):

$$A_w = \frac{\sum_{i=1}^{i=n} W_i S_i}{n}. \tag{6}$$

The average sum aggregation A_s is defined according to Equation (7):

$$A_s = \frac{\sum_{i=1}^{i=n} S_i}{n}. \tag{7}$$

The average sum aggregation assumes similar weights for all matchers. The MAX aggregation Max is defined as S_i such that there is no other $S_j > S_i$ while MIN aggregation Min is defined as S_i such that there is no other $S_j < S_i$.

2.3.1 Comparing Tools Using Parallel Matching. In this section, we compared ontology matching tools that use parallelization to achieve scalability. We specifically evaluated the tools on the following parameters:

(1) At which stage of the ontology matching process does the tool apply parallelism?
(2) Does the tool achieve scalability?
(3) Which aggregation technique(s) does the tool use to combine the multiple similarity matrices?

Table 4. Comparing Tools That Employ Parallelization During Ontology Matching Process

Tool	Stage of parallization in the ontology process	Parallelization technique	Additional scalability Technique?	Aggregation method	Scalable?
OMReasoner	Entity matching stage	Inter-matcher/ MISD	No	weighted summarizing algorithm	No
XMap++	Entity matching stage	Inter-matcher/ MISD	No	Weighted average using neural network	Yes
MaasMatch	Entity matching stage	Inter-matcher/ MISD	No	Dempster-Shafer theory	No
GOMMA	entity matching stage	Intra-matcher	Yes (Blocking)	Union and average	Yes
XMapGen	Entity matching stage	Inter-matcher/ MISD	No	Machine learning	No
XMapSig	Entity matching stage	Inter-matcher/ MISD	No	Machine learning	No
SPHeRe	Loading and Entity matching stage	SIMD and MIMD	Yes (Partitioning)	Union	Yes
CroMatcher	Entity matching stage	Inter-matcher/ MISD	No	Weighted sum	No
InsMT+	Instance matching stage	Inter-matcher/ MISD	No	average aggregation	No
ServOMBI	Pre-processing stage	SIMD	Yes (Indexing)	Not needed	Yes

(4) Which type of parallelization does the tool use?
(5) Does the tool use additional scalability technique?

The summary is presented in Table 4. Out of the 10 tools presented in Table 4, the most popular models of parallelization that they employ is inter-matcher and MISD model of parallelization. Seven of the tools use these models at the entity matching stage of the ontology matching process. Only XMAP++ among the seven tools that use inter-matcher and MISD is scalable. This may be attributed to the fact that these techniques of parallelization do not address the challenge of high space complexity associated with matching large ontologies. Except for ServOMBI where parallelization has been exclusively implemented at the pre-processing stage, all other tools at least implement parallelization at the entity matching stage of the ontology matching process. Therefore, the entity matching stage of the ontology matching process is currently the most popular stage of implementing parallelism among the ontology matching tools. SPHeRe is currently the only tool that implements parallelism at the different stages (loading and entity matching stage) of the ontology matching process.

2.3.2 Tools That Implement Parallelization. *CroMatcher* [42] [43] is an ontology matching tool based on two set of matchers, i.e., the terminological and structural matchers. It first executes terminological matchers in parallel, then the structural matchers in sequential fashion. The results from all the terminological matchers are aggregated using weighted aggregation. The resultant output from the aggregation act as the input of structural matchers.

OMReasoner [97] employs multiple matchers that are executed in parallel. The matchers use Edit Distance and WordNet to compute similarities between entities of the input ontologies. Each correspondence generated by a matcher has a respective confidence value. The normalized results of each matcher are then combined using weighted summarizing algorithm to obtain final alignment.

InSMT [65] is an ontology matching system dedicated to matching instances automatically. It executes multiple string-based matchers in parallel to establish similarities between entities of input

ontologies. It uses local filter to refine the similarity matrix of each matcher. The final alignment is obtained by average aggregation of all similarity matrices generated by the matchers.

XMap++ [24] employs three key matchers, i.e., string, linguistic, and structural-based matcher, which are executed in parallel. The three similarity matrices are combined into a single similarity metric using weights that are assigned using neural networks as proposed in Reference [109].

MaasMatch [92] utilizes the information located in the concept names, labels, and descriptions to produce a mapping of ontologies' similarities. Computations are executed in parallel using a dynamic number of threads, depending on the hardware. This facilitates the computation of alignments of ontologies through a more effective usage of all available computing power. The different similarity values are combined using Dempster-Shafer theory.

GOMMA [41] employs multiple parallel matchers on multiple computing nodes and CPU cores. It performs fragment level match tasks in parallel. This kind of parallelization is possible since for all used matchers all information used for matching is directly associated to the concepts. It uses union and average weighting to combine the different similarity values from the multiple matchers.

XMapGen [23] is a variation of XMAP++. It uses three matchers which are executed in parallel. Like XMAP++ the matchers include string matcher, linguistic matcher and structural. It then uses aggregation, selection, and combination to select final alignment.

SPHeRe [5], [4], [63] is an ontology matching tool that creates subset of input ontologies, then executes the different matching processes of the subsets of the input ontologies on multicore that are distributed in different computational nodes for parallel execution of the matching process.

3 IMPROVING QUALITY OF MAPPINGS IN LARGE ONTOLOGY MATCHING

Due to the complexity of the matching process of large ontologies, an ontology matching tool is likely to generate a significant number of wrong mappings between the entities of two input ontologies, hence degrading the quality of final alignment generated by the tool. The main focus in this section is to answer question: How do ontology matching tools tackle the problem of decreasing quality of mappings when matching large ontologies? Since quality techniques integrated on a tool degrade its scalability, we also investigated how tools add scalability attributes in the quality maintenance techniques. The review in this section is mainly focused on two key techniques:

(1) Interactive mapping repair.
(2) Automatic repair.

3.1 Pay As You Go Ontology Matching Technique

As ontologies become huge, automatic matchers become an inevitable solution for mapping the concepts of the source and the target ontologies. However, the quality of mappings produced by automatic matchers drop with an increase of the sizes of the ontologies being matched [25]. To address the challenge of degraded efficiency, some tools such as LogMap [55] and ALCOMO [69] have repairing capabilities to automatically repair inconsistent mappings. However, a fully automatic matching process places an upper limit on quality of alignment that a tool can be produce [81], therefore, to scale this quality barrier, human intervention is vital. One key strategy that ontology matching tools are implementing to scale the quality barrier and improve the quality of mappings is the pay as you go technique, which is a technique where the initial matching is performed automatically and the results improved incrementally by expert users of the domain of the two ontologies being matched [78]. Automatic matchers, therefore, can be viewed as providing partial or untrusted mappings, which users can be engaged to improve. Users improve the mappings by iteratively evaluating them through annotations. For example, a mapping m selected

for user evaluation can be annotated as "correct" or "wrong" depending on a user's judgment. If the mapping m is annotated as wrong, it is removed from the final alignment, otherwise it is maintained in the final alignment. The pay as you go technique is implemented as summarized in Algorithm 1.

ALGORITHM 1: Summary of the pay as you go technique

1: **INPUT** Source ontology(O_s), target ontologies(O_t) and automatic matching tool M.
2: **OUPUT** Quality alignment A.
3: Generate initial alignment A^I between O_s and O_t using automatic mapping tool M.
4: Select a set of mappings L from A^I that are considered low quality.
5: User or users query the set of low quality mappings L for validation.
6: Improve the quality of A^I based on the users' feedback.
7: Iterate the process from step 4 until mappings quality of A^I attains the highest possible quality (i.e., highest possible precision and recall).
8: output the final alignment A.

The general issues in the pay as you go technique are:

(1) How is an expert user selected to participate in mapping evaluation?
(2) What are the design features that should be included in the user interface?
(3) How do tools select candidate mappings for validation?
(4) How do tools accept feedback from the user(s)?
(5) In case many users are involved in the validation process, how is consensus reached?
(6) How is a user's feedback utilized?
(7) How does a system deal with user fatigue?

The first two issues are discussed in References [28] and [49]. Our review is mainly focused on the last five issues by extending the discussion in Reference [28].

3.1.1 Selecting Candidate Mappings for Validation. Manual curation of mappings such as pay as you go technique are time-consuming, especially if the input ontologies are large since they generate a large number of candidate mappings [58]. To make the process of user validation feasible, i.e., appealing to the user(s) and scalable, not all mappings generated by automatic matcher(s) can be queried by the user for evaluation since this is tedious and will slow down the matching process. Therefore, ontology matching tools performing interactive matching should implement techniques of selecting the best minimum sample mappings which, when evaluated by dedicated user(s), the user's feedback will yield the final quality mappings within a short time (i.e., few iterations). It is, therefore, crucial to select the minimum most informative candidate mappings that after users' feedback, the feedback will maximize the improvement of the previous iteration's matching performance. Work in Reference [98] proposes the use of an erroneous mapping as the most informative mapping since its correction and the propagation of the correction in the similarity matrix guarantees further improvement of the previous mapping results. An erroneous mapping in this context is defined as a mapping between two entities from the source and the target ontologies where an automatic matcher establishes a certain relationship, e.g., equivalence but according to some defined (possibly unknown) reference alignment, it is wrong. The problem of selecting candidate mapping for evaluation can be defined as:

Given an initial mapping set M, generated by automatic matcher(s), find a minimum set of mapping $M^u \subseteq M$ such that for each iteration $M \setminus M^u \rightarrow M^f$, where M^f is the final alignment.

This definition assumes that all mappings in the set M^u are validated as wrong mappings. Three key techniques currently exist in literature for candidate mapping selection for user validation:

(1) Similarity matrix-based techniques.
(2) Probabilistic-based techniques.
(3) Ontology structure-based techniques.

Similarity MatrixBased Techniques. These techniques have been used in tools that produce 1:1 mappings.

Selection Based on Individual Matcher Disagreement. Most matching tools integrate multiple matchers to perform the matching task. The key matchers being used include structural matcher, which exploits the relationship that exists between entities of an ontology; lexical matcher, which matches entities based on the textual labels of the entities together with their annotations; and semantic matcher, which exploits background knowledge to establish semantic similarities between entities of two ontologies [30]. During the matching process, each matcher generates a similarity matrix $S = (s_{ij})_{i=1,..n,j=1,..m}$ with each (s_{ij}) being the confidence value that shows how close entity e_i and e_j of the source and the target ontologies are related. To pick low-quality mapping, disagreement in the results of a given matcher can be a signal that one of the mappings involved is wrong.

Definition 2 (Disagreement Mappings of a Matcher in 1:1 Mapping). Given a source and a target ontology O_S and O_T, respectively, a matcher M and entities $e_i \in O_S$ and $e_j, e_j'' \in O_T$ where $e_j \neq e_j''$, If the matcher M generates a similarity matrix S such that $s(e_i, e_j, r_1, k)$ and $s(e_i, e_j'', r_2, w)$ then there is a disagreement if

(1) $r_1 = r_2$ where r_i can be $\equiv$,$\sqsubseteq$ or $\sqsupseteq$,
(2) k and w ranging [0,1] are greater than set selection threshold T, and
(3) $|k - w| \leq \epsilon$, i.e., the difference between the similarity values of the two mappings is very small. The mappings participating in the disagreement are all flagged and become candidates for user evaluation. This technique is implemented in References [13], [17], and [98] with some variations.

Selection Based on Multiple Matcher Disagreement. This occurs when two or more matchers generate conflicting mappings.

Definition 3 (Disagreement Mappings of Multiple Matchers in 1:1 Mapping). If the matchers M_1 and M_1 are executed in parallel to match entities of source and target ontologies O_S and O_T, then there is disagreement between the matchers if:

(1) Matcher M_1 establishes the mapping $M_1(e_i, e_j, r_1)$ and matcher M_2 establishes the mapping $M_2(e_i, e_j'', r_2)$ where $r_1 = r_2$ with the entities $e_i \in O_S$ and $e_j, e_j'' \in O_S$ and $e_j \neq e_j''$, then the two marchers are conflicting with regard to the mappings.
(2) Matcher M_1 establishes the mapping $M_1(e_i, e_j, r_1)$and matcher M_2 establishes the mapping $M_2(e_i, e_j, r_2)$, where $r_1 \neq r_2$ with the entities $e_i \in O_S$ and $e_j \in O_S$ then the two matchers are conflicting with regard to the mappings.
(3) Matcher M_1 establishes the mapping $M_1(e_i, e_j, r_1)$ and matcher M_2 rejects the mappings. This may result, for example, if two entities are lexically similar but their neighbors are different, hence lexical and structural matchers disagree.

The mappings that generate disagreements become candidates for user evaluation. The variations of this definition is implemented in References [13] and [98]. The similarity matrix-based

techniques have the advantage that they pay more attention to selecting wrong informative mappings, which is desirable for interactive mapping [98]; however, since matchers investigate different properties of entities in an ontology, disagreement among them can be many, especially in a large ontology matching task; therefore, these techniques may generate many candidate mappings for evaluation which may overwhelm the user.

Probabilistic-Based Techniques. These techniques apply statistical methods to pick candidate mappings for evaluation. After establishing an initial alignment using automatic matching tool, statistical methods are then applied to the alignment to pick sample mappings that user feedback will be sought. This is applied in Reference [78] where they apply simple random sampling as proposed in Reference [18] to pick candidate mappings for user evaluation. A clear disadvantage of these techniques is that they do not pay attention to selecting wrong mappings (informative mappings), and therefore may result in users not providing a significant improvement to the quality of mappings. However, they have the advantage that the sample size of the selected mappings for user evaluation can be controlled to fit the user's needs.

We motivate this part of discussion by giving an example of how simple random sampling can be used to determine the size of the candidate mappings for user evaluation. If a matcher generates an initial alignment containing 4,203 mappings, using simple random sampling with a confidence of 95% and a confidence value of 5%, the required sample size is computed by first computing the sample size for infinite population (SSI) as shown in Equation (8), then the sample size for finite population (SSF) as shown in Equation (9):

$$SSI = \frac{Z^2 \times p \times (1-p)}{c^2}, \tag{8}$$

where $Z = Z$ value (e.g., 1.96 for 95% confidence level), p = percentage of picking a mapping (0.5) and c = confidence level (0.05).

$$SSF = \frac{SSI}{1 + \frac{SSI-1}{Population}}, \tag{9}$$

$$SSI = \frac{1.96^2 \times 0.5 \times 0.5}{0.05^2} = 384.16,$$

$$SSF = \frac{384.16}{1 + \frac{384.16-1}{4203}} = 352.$$

Therefore, for an initial alignment of size 4,203, using the parameters above a sample mapping set of 352 mappings should be selected for user evaluation.

Ontology Structure-Based Techniques. These techniques exploit the relationships that exists in the ontologies being matched to flag low-quality mappings. It is majorly based on the discussion first introduced by Reference [70]. The techniques are anchored on the condition that the mappings generated by automatic matching tools should respect (i) consistency principle, i.e., they should not lead to unsatisfiable concepts in the merge, (ii) conservativity principle, i.e., they should not introduce new semantic relationships between concepts from one of the input ontologies, and (iii) locality principle, the mappings should link entities that have similar neighborhoods [53] [56].

Definition 4 (Merged Ontology). Given an alignment A between two ontologies O_1 and O_2, the merged ontology $O^M_{O_1,O_2}$ is defined as $O_1 \cup O_2 \cup A$.

Unstable Mappings. The notion of stability first introduced by Reference [70] asserts that mapping between two ontologies should not introduce new structural relationship that were not originally modeled in the two ontologies.

Definition 5 (Unstable Alignment). An alignment A between ontology O_1 and O_2 is regarded as unstable if there exists a pair of concepts $B, C \in O_i$ with $i \in \{1, 2\}$ such that $O_i \not\models B \sqsubseteq C$ and $O^M_{O_1,O_2} \models B \sqsubseteq C$.

If a correspondence introduces a new subsumption relationship between the concepts of a given ontology in the merged ontology that did not exist in the original ontology, the correspondence is flagged as unstable (unreliable) mapping. The correspondences that introduce instability in the merged ontology should be displayed to the user for evaluation. Examples of mappings that should be flagged based on instability include:

(1) If $O_1 \models X \sqsubseteq X'$, and $O_2 \models \neg(Y \sqsubseteq Y')$, and any of the following mapping combination $\langle X, Y, \equiv\rangle \cap \langle X', Y', \equiv\rangle$ and $\langle X, Y', \equiv\rangle \cap \langle X', Y, \equiv\rangle$ exists in the final alignment, then flag the mappings as unstable.
(2) If $O_1 \models \exists R_1.\top \sqsubseteq X$, and $O_2 \models \neg(\exists R_2.\top \sqsubseteq Y)$, and the mapping combination $\langle R_1, R_2, \equiv\rangle \cap \langle X, Y, \equiv\rangle$ exist in the final alignment, flag the mappings as unstable.

Incoherent Mappings. Incoherence arises if the merged ontology has contradicting axioms.

Definition 6 (Incoherent Alignment). An alignment A between ontology O_1 and O_2 is regarded as incoherent if there exists a concept $C \in O_i$ with i $\in \{1, 2\}$ such that $O_i \not\models C \sqsubseteq \bot$ and $O^M_{O_1,O_2} \models C \sqsubseteq \bot$.

This problem can arise either due to incompatibility between ontologies O_1 and O_2 or as a result of wrong mappings. Therefore, mappings that result in consistency violations become candidates for user evaluation. Some example mappings that are flagged as incoherent include:

(1) If $O_1 \models X \sqcap X' \sqsubseteq \bot$, and $O_2 \models Y \sqsubseteq Y'$, and any of the mapping combinations $\langle X, Y, \equiv\rangle \cap \langle X', Y', \equiv\rangle$ and $\langle X, Y', \equiv\rangle \cap \langle X', Y, \equiv\rangle$ exists in the final alignment, flag the mappings as incoherent.
(2) If $O_1 \models \exists R_1.\top \sqsubseteq X$, and $O_2 \models \exists R_2.\top \sqsubseteq \neg Y$, and the mapping combination $M(R_1, R_2, \equiv) \cap M(X, Y, \equiv)$ exist in the final alignment, flag the mappings as incoherent.

These techniques have the disadvantages that:

(1) They require extensive reasoning for violations to be detected. This impacts negatively on the scalabilty of a tool. Tools such as AML [89] and LogMap [59] employ ontology modularization to speed up violation detection.
(2) A significant amount of violations are contributed by the incompatibility of the input ontologies, therefore flagging mappings based on these violations may result in flagging a significant number of mappings that are inconsistent but are not wrong mapping, hence displaying them to the user for evaluation will not yield maximum benefits of interactive mappings.

3.1.2 User Feedback. To get user feedback, systems implement two key dimensions

(1) Single user vs multiple users
(2) Controlled annotation vs uncontrolled annotation.

Controlled Annotation vs Uncontrolled Annotation. In pay as you go, user(s) provide feedback in the form of annotations. They annotate mappings based on the technique being applied by a system to verify correctness of a mapping. Formally, feedback can be viewed as a tuple $\langle M, a, u, t\rangle$ with M specifying the mapping, a the annotation provided by the user u, and t indicates the type of feedback [9]. Depending on the system, the set of annotations provided by the user can either be controlled or uncontrolled.

Controlled Model. In this setup, a user is allowed only to choose annotations from a fixed set of options such as evaluating if a mapping is correct or incorrect. Compared to the uncontrolled scheme, this model is relatively fast since a user has no extra cognitive burden of coming up with his or her own tags, a process which may take some time. This scheme is implemented in References [13], [17], [29], [59], [89], and [98].

Uncontrolled Model. Here, users provide the annotations in free form nature. Despite being slower, it is supported by work in Reference [12], who found out that users prefer tagging of concepts to assign them their own meaning rather than being restricted to narrow range of mapping terminology such as to answer questions like "corresponds to" and "similar to."

Single User vs Multiple Users: Multiple User Evaluation. This is where more than one user provides feedback. With multiple users, conflicts in annotations of a given mapping are likely to arise. A conflicting feedback is a feedback where a mapping M is assigned two or more annotations with conflicting meaning [9]. Tools that employ this scheme have an extra task of establishing consensus among the conflicting users in a given mapping. Such tools, therefore, should integrate mechanism of addressing conflicts, a solution which may impact negatively on its scalability. This scheme is implemented in Reference [13], where they use simple majority vote to come to the final evaluation of a mapping from different answers provided by different users. Reference [90] uses an alignment API to assess mappings provided by the crowd.

Single User Evaluation. This is where only a single user is allowed to validate a mapping. This has the advantage that it avoids dealing with diverse and sometimes disagreeing answers that are common when multiple users are engaged, hence relatively faster. However, it faces downsize that, in case a user makes an error in evaluation, his or her evaluation is adopted as right since there are no other users to dispute. Some research such as Reference [78] evaluates the reliability of a single user using Intra Observer Reliability (IaOR) as proposed in Reference [47]. The single user technique is applied in References [14], [17], [29], [59], [89], and [98].

3.1.3 Improving Mappings Using User Feedback. In pay as you go technique, user feedback is used to improve the mappings generated by the automatic mapping tool. The existing techniques in literature utilize user feedback by either updating the similarity matrix or directly correcting the mappings. In Reference [98], once users confirm a mapping to be either correct or wrong, they use similarity flooding proposed in Reference [72] to correct other related mapping, therefore maximizing the effect of user's feedback. In Reference [13], they update similarity matrix based on the users' responses, then execute the automatic selector algorithm to select a new improved alignment from the adjusted similarity matrix. In Reference [59], they directly remove mapping rejected by users from the final set of mappings.

3.1.4 User Fatigue. To address the problem of user fatigue, tools implement different strategies, which include:

(1) Allowing users to terminate the process of validation at any time [57], [58], [59].
(2) Placing an upper bound on the number of validation requests that a user can handle [33], [89].
(3) Asking a user to validate a given mapping only once [33], [89], [100].

3.1.5 Tools Evaluation. In this section, we compare tools that participated in the OAEI 2015 and 2016 conference in the interactive track against the parameters discussed in this section. We specifically compare the tools by answering the following questions for each tool.

Table 5. Comparing Tools Performing Interactive Matching

Tool	Mapping selection	Number of users	Type of user feedback	Correction Technique	Technique to handle fatigue	Scalability Technique
LogMap	Structural heuristics	Single	Controlled	Directly remove mappings	user initiated stop	modular reasoning
AML	Structure and Similarity matrix (confidence value)	Single	Controlled	Directly remove mappings	upper bound limit	reasoning over a partitioning
XMAP	Threshold filter	Single	Controlled	Directly remove mappings	Not included	Not included
ALIN	Not included	Single	Controlled	Directly remove mappings	Not included	Not included
ServOMBI	Threshold filter	Single	Controlled	Directly remove mappings	Not included	Not included
JarviSOM	Disagreement of classifiers	Single	Controlled	Directly remove mappings	Not included	Not included

Table 6. Comparing Tools Performing Interactive Matching

Tool	Size of initial mappings	Total request	Correct requests	Wrong request	non Interactive precision	Interactive precision	Non Interactive recall	Interactive recall	F-measure increase
ALIN	1142.8	803	626	594	0.984	0.993	0.335	0.749	0.3541
AML	1484	241	51	189	0.95	0.965	0.936	0.948	0.0268
LogMap	1306	590	287	303	0.911	0.982	0.846	0.846	0.0316
XMAP	1486	35	5	30	0.928	0.929	0.865	0.867	0.0015

(1) Which candidate mapping selection technique does it apply?
(2) It is single- or multiple-user based?
(3) Is the feedback controlled or uncontrolled?
(4) How does it utilize the users' feedback?
(5) How does it deal with user fatigue?
(6) Does the tool include any scalability technique in candidate mapping selection?

Table 5 gives a comparison of the tools based on the questions listed above.

We further evaluated the tools using OAEI 2016 data on interactive track on the following questions:

(1) Does interactive process result in improvement of precision and recall of a tool?
(2) What percentage of initial mappings does a tool select for evaluation?
(3) How accurate is a tool in selecting wrong mappings?

We evaluated the tools using data where the user provides perfect feedback, i.e., has 0% error rate. Table 6 provides data on different parameters measured.

From Table 6, it is clear that tools that present more wrong requests (i.e., correct mappings that do not need user evaluation) to the user benefit the least from the user's feedback—the biggest culprit tool being XMAP, which, out of 35 mappings, it presents to the user only 14.28% correct requests (wrong mappings). AML presents the second-lowest correct request, i.e., 21.26%, hence the second-least beneficiary from the user's feedback. Therefore, there is a direct link between type of mappings presented to the user and the magnitude of improvement in the quality of mappings. Linking Tables 5 and 6, we observe that XMAP presents the lowest request to the user for evaluation only 2.26% of the total initial correspondences. This is expected since it relies on "high

threshold" to select mappings for evaluation, this high threshold restrict the number mappings selected. Despite this being desirable from a user's perspective, XMAP records the lowest benefit (F-measure increase) from user's involvement. LogMap and AML, which use structural incoherence to generate mappings for evaluation, still have a high number of wrong requests. Out of the total requests to the user, LogMap presents 51.35% wrong requests while AML presents 78.42%. This may be a signal that a significant amount of the structural incoherence flagged is due to incompatibility between input ontologies as opposed to wrong mappings. ALIN makes the most requests to the user and benefits the most from the interactive matching process; this may be attributed to the fact that it is specifically a tool for interactive matching, hence it may not be strict on how it computes initial mappings. It also presents more than one distinct mapping per user request. From the results in Table 6, there is still a lack of a clear standout method for wrong mapping selection and this may still be an active area of research.

3.1.6 Interactive Matching Tools. We finally conclude this section by giving a review of some of the active tools that implement interactive matching.

LogMap2 [57], [58], [59]. After establishing initial mappings M by exploiting inverted lexical indexes of concepts and labels, it uses a combination of both semantic indexes and lexical indexes to prune non reliable mappings from M. Mappings M^u in M, which are still not "clear cut" after discarding non-reliable mappings, are the candidate mappings for user verification. LogMap2 creates a partial order of mappings in the set M^u using their similarity value. Users are then asked questions to either approve or reject the mappings according to the order created. Since LogMap implements a single user model, it anticipates user fatigue by allowing the user to terminate the process at any time. Mappings that are still unresolved after user termination are solved automatically.

XMAP [29] uses a number of terminological matchers and structural matchers to generate initial candidate mapping set M. It then applies two thresholds to filter the set M into two, first is the set M^l which contains final mappings and the second set M^u, which contains mappings for user evaluation. The threshold for user evaluation is set to be high to minimize the number of mappings in the set M^u. The mappings in M^u accepted by the user are moved to the set M^l. However, XMAP does not register any significant benefit from the user involvement as compared to the non-interactive mode.

AML [89] exploits incoherence techniques between the merged ontology and the input ontologies to flag mappings that are likely to be low quality. For a large ontology matching task, it employs a partitioned-based approach to avoid reasoning over the large ontologies, hence it is able to flag low-quality mapping by reasoning over a partition [89], therefore achieving scalability. The low-quality mappings are then displayed to the user for validation. AML also does not ask a similar question twice to the user by keeping track of already asked questions. It also uses a query limit to handle user fatigue. It directly removes the validated mapping from the low quality mapping set.

ALIN [100] uses linguistic metrics to compute similarities of the entities of the input ontologies. It then applies stable marriage algorithm with incomplete preference list [48] to pick an initial set of candidate mappings. The candidate mappings are further trimmed using WordNet, i.e., correspondences whose entities are not in the same synset of WordNet are removed. The remaining correspondences are sorted according to the sum of similarity metric values with the greatest sum being the first in the list. The correspondences are then displayed to the user in the order of the list for evaluation. To reduce the number of correspondences, only class entities are allowed in the initial set of correspondences. After a user's feedback, wrong correspondences are removed from the correspondence set and correct correspondences are removed from the set to be added to the final alignment. Due to the list of candidate mapping created, ALIN does not ask the user to verify a mapping more than once.

ServOMBI [64] employs terminological-based inverted index to compute initial mappings set M of input ontologies. It then employs users in a two stage process to help in refining the mapping M into the final mapping M^f. In the first stage, the mappings in M are presented to the user for validation, then machine learning techniques are used to refine the mappings M based on the user's feedback. The user is again involved to refine the modified mapping set M into the final set M^f.

JarviSOM [1] employs a number of classifiers to determine the mapping set M^u to be verified by the user. The set M^u contains the mappings where the classifiers disagree the most. After user validation, the classifiers are used to generate the final alignment M^f.

3.2 Automatic Mapping Repair

Automatic ontology matching tools generate mappings that are composed of a set of correct and wrong mappings when compared to a reference alignment of the two ontologies [71]. To flag potentially wrong mappings automatically, ontology matching tools are designing techniques that reason over the mappings generated and input ontologies to identify the set of mappings that are wrong and diagnose them. The key issues in automatic mapping repair include:

(1) How to identify wrong mappings.
(2) How to diagnose the inconsistent alignment.
(3) How to incorporate scalability in mapping repair process.

3.2.1 Identifying Wrong Mappings. To identify potentially wrong mapping, Reference [71] introduces the concept of consistency as a means of identifying wrong mappings. Inconsistency is evaluated according to Definition 6. Therefore, the problem of identifying wrong mappings involves identifying a set of mappings that introduce consistency violations in the input ontologies when compared to the ontology generated by merging the input ontologies via the mappings established between them. Tools with repair functionality, therefore, integrate reasoning techniques that are able to detect these mappings automatically. As demonstrated in Reference [71], not all mappings that are flagged as causing inconsistency in the mappings should participate in the given matching repair; they propose the use of a minimal inconsistent set to repair the mappings into consistency in a process known as diagnosis.

Definition 7 (Minimal Inconsistent Set). Let M be a set of mappings between input ontologies O_1 and O_2. A set $C \subseteq M$ is inconsistent set for a concept $A \in O_i$ with $i \in (1, 2)$ iff $O^M_{O_1,O_2} \models A \sqsubseteq \bot$ and $O_i \not\models A \sqsubseteq \bot$. C is a minimal inconsistent set for $A \in O_i$ iff C is an inconsistent set for $A \in O_i$ and there is no $C' \subset C$ that is also an inconsistent set for $A \in O_i$.

Therefore, the task for repair changes to only identifying the minimal inconsistent set that will repair the inconsistent alignment. Two key strategies exist for identifying minimal inconsistent set:

(1) Local reasoning strategy.
(2) Global reasoning strategy.

Local Reasoning Process. A local reasoning is where the minimal inconsistent set is determined by considering a justification for a particular entailment.

Definition 8 (Justification). Let $O \models \alpha$, a fragment $O' \subseteq O$ is a justification $JUST(\alpha, O)$ for α in O if $O' \models \alpha$ and there is no other O'' such that $O'' \models \alpha$ for every $O'' \in O$ [62]

With local repair reasoning strategy, reasoning over the whole input ontologies is avoided, hence making the process scalable. LogMap2 [59] detects unsatisfiable mappings by performing

[1] http://oaei.ontologymatching.org/2015/results/interactive/index.html.

modularization of input ontologies. Given the input ontologies O_1 and O_2 and initial mapping M between them, LogMap2 computes fragments $O_1' \subseteq O_1$ and $O_2' \subseteq O_2$. These fragments should contain all the information in the input ontologies relevant to given concepts to detect unsatisfiability of mappings between the concepts. AML [89] also employs modularization technique by defining a set of conditions that a module must meet to fully represent the information in the input ontologies sufficient to detect incoherence violations of a mapping. Some research such as References [44], [62], and [105] explore means of extracting justification. Despite being scalable, local reasoning process is incomplete, i.e., may be scalable but not does not guarantee that all members of the minimal inconsistent set will be flagged.

Global Reasoning Process. A global reasoning determines the minimum inconsistent set by considering all the classes and relationships modeled in the input ontologies. Despite producing better results that local reasoning process, it is not scalable for large ontologies. ALCOMO [60] [69] can be configured to perform complete repair process.

3.2.2 *Mapping Repair.*

Definition 9 (Mapping Repair). Let M be a set of inconsistent mappings between input ontologies O_S and O_T, a set of mappings $R \subseteq M$ is a mapping repair for M if $M \setminus R$ is consistent. Ontology matching tools therefore perform repair by removing mappings that introduce inconsistencies in the alignment.

Aggression Problem. Automatic repair techniques have been criticized as being too aggressive since they remove many correct mappings [84]. To mitigate aggression problems, LogMap splits an equivalence mapping flagged as inconsistent into two subsumption mappings and keeps the one that does not violate any logical constraints. For instance, if there exist an equivalence relation $M(A \equiv B)$ that is inconsistent, it breaks it into $M(A \sqsubseteq B)$ and $M(B \sqsubseteq A)$ and removes the one that is inconsistent in the merged ontology. This, however, can create a relationship that is consistent but does not represent a real relationship entailed between the input ontologies. AML tackles aggression problems by partitioning the initial detected conflicting sets into disjoint clusters, i.e., it divides the initial set of conflicting mappings into sets that have at least one mapping in common. By doing this, AML is able to determine the minimal mappings to be removed from each of these clusters independently. ALCOMO has no mitigation for aggression problems.

3.2.3 *Mapping Repair Tools.*

ALCOMO [60] [69] is a tool that was specifically developed to repair inconsistent mappings. It implements a two-tier optional reasoning to flag mapping repair. The first is an incomplete reasoning that uses OWL2 reasoner to detect mappings that lead to inconsistencies by exploiting disjoint axioms in the input ontologies. The second reasoning is complete and can either be used to refine the results of the incomplete reasoning or can flag mappings by reasoning over the entire input ontologies.

AML [89]. It first partitions the input ontologies into fragments by the use of four key conditions that are based on disjoint axiom and superclass-subclass relationships modeled in the input ontologies. The conditions guarantee that reasoning over fragments will flag incoherent mappings the same way the non-partitioned ontology would. The partitioning is to ensure scalability, which is key for repairing mappings in large ontology mapping.

LogMap [55] converts the axioms in the input ontologies into propositional theories and applies horn rules to detect unsatisfiable clauses. To enable scalability, it employs local repair, which is performed by establishing the minimal effect in small subset of matched ontology. The complete process is described in Reference [60].

3.2.4 Criticism of Automatic Mapping Repair. Based on the discussion in Reference [84], the existing automatic mapping repair techniques face the following criticisms, which still need to be addressed:

(1) Since they put emphasis on coherence, they sometimes generate mappings that are coherent but are wrong.
(2) They remove large number of correct mappings to achieve coherence.
(3) Different techniques results in different repaired alignments, i.e., there is not consistency on the results of the alignment repairing techniques.

4 PROGRESS SO FAR

In this section, we evaluate the progress made by ontology matching tools in matching large ontologies. We use the results provided by the OAEI conference in its large biomedical track for the of period 2012–2016. We specifically exploit the results provided to answer the following questions:

(1) Is there a significant increase in the quality (precision, recall, and F-measure) of the mappings generated by the ontology matching tools over 5 years?
(2) Have the tools managed to register a significant reduction in execution time of the ontology matching processes over 5 years?
(3) What are the most popular techniques that the scalable tools are using to achieve scalability?
(4) Are the number of tools that can handle large matching tasks increasing?

We only use the results of the tools that completed all nine (2012) and six tasks (2013–2016) of the large biomedical track. The summary of results is provided in Table 7.

From the summary in Table 7, ontology matching tools have consistently registered a high precision from the inception of large biomedical track in 2012. The average best precision values in all 5 years are above 90% (see Table 8) with YAM++ recording the highest average precision of 94.2% in 2013. This is still the average upper limit that no tool has been able to scale. Over 5 years, the average best precision has been fluctuating between 90.3% and 94% with the years 2013 and 2015 recording the highest increase of 0.41 and 0.23 from the previous year, respectively. On the overall, tools have been successful in achieving high precision in large ontology matching tasks. The overall best performing tools in precision over 5 years include ServOMapL (2012), YAM++ (2013), LogMap-C (2014), RSDLWB (2015), and AML (2016). Apart from GOMMA and its variant $GOMMA_{BK}$, all other tools have generated mappings with an average precision of above 80%. When it comes to recall, the highest attained average value was 79.1% in 2012 by YAM++. No tool has been able to exceed this average in large ontology matching tasks. The average best recall has been fluctuating between 72.8% and 79.1% over five years. Compared to precision, tools are still lagging behind in producing mappings thath have high recall. The highest average recall 79.1% has a margin of 11.2% from the lowest best-recorded precision, 90.3%, over 5 years. The leading tools in recall over the 5 years are $GOMMA_{BK}$(2012), YAM++ (2013), AML (2014), $XMAP_{BK}$ (2015), and AML (2016). Some tools such as IAMA (2013) and RSDLWB (2015) produced mappings with remarkably low recall average values of 38.6% and 23.6%, respectively. F-measure has been marginally increasing over the years, as shown in Table 8, with the highest average value recorded being 82.4%, which was attained in 2016 by AML. The consistent improvement in F-measure over the years shows that tools are making progress in improving the quality of mappings they generate. AML has generated mappings with the best F-measure for the last 3 years, making it one of the most stable tools for matching large ontologies. When it comes to execution time, LogMaplite has consistently provided lowest time. It took an average of only 10 seconds to complete all six

Table 7. Performance of Ontology Matching Tools in Large Biomedical Track (2012–2016)

Tool	Scalability Technique	Average precision	Average recall	Average F-measure	Average Time(s)
2012					
YAM++	Indexes	0.876	0.781	**0.782**	7535
ServOMapL	Indexes	**0.890**	0.699	0.780	264
LogMap-noe	Indexes	0.869	0.695	0.770	440
$GOMMA_{BK}$	Blocking Parallelizaion	0.767	**0.791**	0.768	647
LogMap	Indexes	0.869	0.684	0.762	341
ServOMap	Indexes	0.903	0.657	0.758	256
GOMMA	Blocking Parallelizaion	0.746	0.553	0.625	593
LogMaplite	Indexes	0.831	0.515	0.586	**85**
2013					
YAM++	Indexes	**0.942**	**0.728**	**0.817**	344
AML-BK	HashMap	0.908	0.709	0.792	302
LogMap-BK	Indexes	0.904	0.700	0.785	399
AML-BK-R	HashMap	0.921	0.692	0.785	331
AML	HashMap	0.926	0.683	0.783	**280**
LogMap	Indexes	0.910	0.689	0.762	341
AML-R	HashMap	0.939	0.666	0.776	303
$GOMMA_{2012}$	Blocking Parallelization	0.813	0.567	0.654	320
LogMaplite	Indexes	0.874	0.517	0.598	61
ServOMap	Indexes	0.903	0.657	0.758	256
GOMMA	Blocking Parallelization	0.746	0.553	0.625	593
LogMaplite	Indexes	0.831	0.515	0.586	85
SPHeRe	Parallelization	0.857	0.464	0.569	7006
IAMA	Indexes	0.912	0.386	0.517	117
2014					
AML	HashMap	0.906	**0.752**	**0.819**	307
LogMap	Indexes	0.890	0.719	0.792	291
LogMap-Bio	Indexes	0.843	0.744	0.784	1439
XMAP	Parallelization	0.813	0.702	0.750	210
LogMap-C	Indexing	**0.907**	0.559	0.688	1055
LogMaplite	Indexing	0.868	0.539	0.613	52
2015					
AML	HashMap	0.905	0.754	**0.819**	323
$XMAP_{BK}$	Parallelization	0.904	**0.764**	**0.819**	420
LogMap	Indexes	0.903	0.714	0.794	435
LogMap-Bio	Indexes	0.867	0.733	0.789	2285
XMAP	Parallelization	0.892	0.654	0.751	395
LogMap-C	Indexes	**0.907**	0.551	0.613	1436
LogMaplite	Indexes	0.868	0.532	0.613	221
RSDLWB	Early pruning	0.923	0.236	0.0.367	222
2016					
AML	HashMap	**0.907**	**0.7588**	**0.824**	214
LogMap-Bio	Indexes	0.8738	0.7322	0.7931	2386
LogMap	Indexes	0.897	0.7143	0.7918	243
LogMaplite	Indexes	0.8581	0.5318	0.789	**10**

tasks in the year 2016. However, this comes at the expense of the quality of mappings it generates since LogMaplite also posts the lowest F-measure of 0.789 in the year 2016. If we consider the execution times of the leading tools in F-measure in 5 years, we notice that the execution time has been decreasing. In 2012 YAM++, which generates mappings with the best average F-measure, takes an average time of 7,535 seconds to complete a task. It manages to reduce this time to 344 seconds in 2013. This represents a 95.4% reduction in execution time. The reduction in execution time is attributed to the fact that the pre-processing and indexing algorithm was revised such that its complexity $O(n^2)$ in 2012 was reduced to $O(||n|| + ||v||)$ in 2013 [73], where n and v are entities

Table 8. Leading Precision, Recall, and F-measure Values in the Last 5 Years (2012-2016)

Top average precision	Top average recall	Top average F-measure
2012		
0.903	0.791	0.782
2013		
0.942	0.728	0.817
2014		
0.907	0.752	0.819
2015		
0.923	0.764	0.819
2016		
0.907	0.7588	0.824

contained in the source and target ontologies. In 2014, AML generates mappings with the best average F-measure with each task taking an average time of 307 seconds. This increases slightly to 323 seconds in 2015. In 2016, AML reduces the average execution time of a matching task is reduced to 214 seconds. This overall shows that tools are achieving success in reducing time complexity of the matching process. Notable exceptions are LogMapBio, LogMap-C, and XMAP. Out the nine distinct tools presented in Table 7 (i.e., not considering the different variants of a tool), five of them use data structures (HashMap and indexes) to achieve scalability. This represents 56% of the tools, hence making the use of data structures the most popular technique used by the tools that have been able achieve scalability in matching large ontologies at OAEI conference. Two tools (IAMA and SPHeRe) use parallelization to achieve scalability. While GOMMA combines partitioning and parallelization to achieve scalability. Over 5 years, only nine distinct tools have been able to complete all the large biomedical track matching tasks. This shows that scalability is still a major challenge to most ontology matching tools.

5 OPEN CHALLENGES

From this review, we identified the following challenges that still remain in the matching of large ontologies.

(1) *Mapping Cardinality.*
From the review, all the scalable tools only establish 1:1 cardinality mappings. However, not all mapping scenarios are restricted to 1:1 mappings. There are instances where one-to-many or many-to-many mappings may be desired. In such cases, the space and time complexity challenges become more exacerbated as compared to 1:1 mappings. Therefore, tools need to design scalability techniques that, in addition to handling 1:1 cardinality mappings, can also scale when one-to-many or many-to-many cardinality mappings are desired.

(2) *Parallel Matching.*
- —There is need for a further evaluation on the impact of the different multiple matcher combination techniques on the execution time of an ontology matching tool while using parallel execution notwithstanding some work done on this by Reference [85].
- —The performance gain of combining different aggregation techniques over employing only one technique during parallel composition of matchers needs to be evaluated.
- —A discussion on workload balancing among the PUs during parallel ontology matching is still lacking.

(3) *Automatic Repair.*
   —During an ontology mapping repair process, how can a tool separate coherence violations caused due to wrong mappings generated and those caused due to incompatibility between the input ontologies. Some recent works such as Reference [102] have proposed heuristics of minimizing conservativity violations in ontology matching.
   —During the ontology mapping repair process, how can the tradeoff between scalability and accuracy of mapping repair be improved?

(4) *Ontology Partitioning.*
   —What percentage of execution time should be allocated to the ontology partitioning algorithm for the partitioning of the input ontology to yield maximum reduction in execution time?
   —What is the effect of number of partitions on the quality of mappings produced by a matching tool?

(5) *Recall.*
   —From Table 7, it is clear that scalable ontology matching tools achieve scalability at the expense of recall. Therefore, the key question here is, What new techniques can be integrated into the ontology matching tools to achieve higher recall in large ontology matching tasks?

6 CONCLUSION

We have provided a discussion on quality assurance and scalability techniques used by ontology matching tools in large ontology matching tasks. The techniques are mainly geared toward addressing space complexity, time complexity, and improving quality of the final alignment. We have also provided a state-of-the-art discussion with regard to the progress tools have made in matching large-scale ontologies as well as some open challenges that we identified, which we hope will help in progressing the state of the art.

REFERENCES

[1] M. J. S. Abadi and Kamran Zamanifar. 2011. Producing complete modules in ontology partitioning. *Proceedings of the 2011 International Conference on Semantic Technology and Information Retrieval (STAIR'11)*. 137–143.

[2] Rakesh Agrawal, Alexander Borgida, and H. V. Jagadish. 1989. Efficient management of transitive relationships in large data and knowledge bases. *SIGMOD'89: Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data* (1989). 253–262.

[3] Alsayed Algergawy, Sabine Massmann, and Erhard Rahm. 2011. A clustering-based approach for large-scale ontology matching. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 415–428.

[4] M. B. Amin, W. Ali Khan, S. Hussain, D.-M. Bui, O. Banos, B. H. Kang, and S. Lee. 2016. Evaluating large-scale biomedical ontology matching over parallel platforms. *IETE Technical Review (Institution of Electronics and Telecommunication Engineers, India)* 33, 4 (2016), 415–427.

[5] Muhammad Bilal Amin, Rabia Batool, Wajahat Ali Khan, Sungyoung Lee, and Eui-Nam Huh. 2014. SPHeRe. *The Journal of Supercomputing* 68, 1 (2014), 274–301.

[6] Diego Andrade, Basilio B. Fraguela, James Brodman, and David Padua. 2009. Task-parallel versus data-parallel library-based programming in multicore systems. *Proceedings of the 17th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP'09)*, 101–110.

[7] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. 2007. *Modern Information Retrieval.* ACM Press/Addison-Wesley (1999).

[8] Blaise Barney. 2016. Introduction to parallel computing. Retrieved June 14, 2016 from https://computing.llnl.gov/tutorials/parallel_comp.

[9] K. Belhajjame, Norman W. Paton, Alvaro A. A. Fernandes, Cornelia Hedeler, and Suzanne M. Embury. 2011. User feedback as a first class citizen in information integration systems. *Proceedings of the Conference on Innovative Data Systems Research (CIDR'11)*. 175–183.

[10] J. A. Blake et al. 2015. Gene ontology consortium: Going forward. *Nucleic Acids Research* (2015), D1049–D1056.
[11] Watson Wei Khong Chua and Jung Jae Kim. 2010. Eff2Match results for OAEI 2010. *CEUR Workshop Proceedings* 666, 1 (2010), 150–157.
[12] Colm Conroy, Declan O'Sullivan, and Dave Lewis. 2007. A "tagging" approach to ontology mapping. *CEUR Workshop Proceedings* (2007). 1–5.
[13] I. F. Cruz, Francesco Loprete, and Matteo Palmonari. 2014. Pay-as-you-go multi-user feedback model for ontology matching. In *Proceedings of the International Conference on Knowledge Engineering and Knowledge Management*. 80–96.
[14] Isabel F. Cruz, Cosmin Stroe, and Matteo Palmonari. 2012. Interactive user feedback in ontology matching using signature vectors. *Proceedings of the International Conference on Data Engineering* (2012). 1321–1324.
[15] Syrine Damak, Hazem Souid, Marouen Kachroudi, and Sami Zghal. 2015. EXONA results for OAEI 2015 and Sami Zghal. *The 10th International Workshop on Ontology Matching - Ontology Alignment Evaluation Initiative* (2015). 5.
[16] Mathieu D'Aquin, Marta Sabou, and Enrico Motta. 2006. Modularization: A key for the dynamic selection of relevant knowledge components. *CEUR Workshop Proceedings* 232 (2006).
[17] V. Jirkovský and R. Ichise. 2013. Mapsom: User involvement in ontology matching. In *Proceedings of the 3rd JIST Conference,* volume 2. Springer.
[18] D. de Vaus. 2002. Surveys in social research. *Routledge* (2002).
[19] J. Dean and S. Ghemawat. 2004. Simplified data processing on large clusters. *Sixth Symp. Oper. Syst. Des. Implement.* 1 (2004), 107–113.
[20] Rudra Pratap Deb Nath, Hanif Seddiqui, and Masaki Aono. 2012. Resolving scalability issue to ontology instance matching in semantic web. *Proceedings of the 15th International Conference on Computer and Information Technology, (ICCIT'12)*. 396–404.
[21] Chiara Del Vescovo, Bijan Parsia, Uli Sattler, and Thomas Schneider. 2011. The modular structure of an ontology: Atomic decomposition. *IJCAI International Joint Conference on Artificial Intelligence* (2011), 2232–2237.
[22] Gayo Diallo. 2014. An effective method of large scale ontology matching. *Journal of Biomedical Semantics* 5 (2014), 44.
[23] Warith Eddine Djeddi and Mohamed Tarek Khadir. 2013. XMapGen and XMapSiG results for OAEI 2013. *CEUR Workshop Proceedings* (2013). 203–210.
[24] Warith Eddine Djeddi and Mohamed Tarek Khadir. 2014. XMap++: Results for OAEI 2014. *CEUR Workshop Proceedings* (2014). 163–169.
[25] Hong Hai Do and Erhard Rahm. 2007. Matching large schemas: Approaches and evaluation. *Information Systems* (2007), 857–885.
[26] Pedro Domingos, Daniel Lowd, Stanley Kok, Hoifung Poon, Matthew Richardson, and Parag Singla. 2008. Just add weights: Markov logic for the semantic web. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (2008). 1–25.
[27] P. Doran, V. Tamma, and L. Iannone. 2007. Ontology module extraction for ontology reuse: An ontology engineering perspective. *Proceedings of the 16th ACM Conference in Information and Knowledge Management* (2007). 61–70.
[28] Zlatan Dragisic, Valentina Ivanova, Patrick Lambrix, Daniel Faria, Ernesto Jiménez-Ruiz, and Catia Pesquita. 2016. User validation in ontology alignment. In *Proceedings of the International Semantic Web Conference* 9981, 1 (2016), 200–217.
[29] Warith Eddine, Mohamed Tarek, and Sadok Ben. 2016. XMap: Results for OAEI 2016. *Proceedings of the 11th International Workshop on Ontology Matching* 1766, 5 (2016), 122–127.
[30] P. Euzenat and J. Shvaiko. 2005. A survey of schema-based matching approaches. *Journal on Data Semantics* (2005).
[31] P. Euzenat and J. Shvaiko. 2007. *Ontology Matching*. Springer, Heidelberg.
[32] Muhammad Fahad. 2015. Initial results for ontology matching workshop 2015 DKP-AOM: Results for OAEI 2015. In *CEUR Workshop Proceedings* 1766, 5 (2015), 82–96.
[33] Daniel Faria, Catarina Martins, Amruta Nanavaty, Daniela Oliveira, Booma S. Balasubramani, Aynaz Taheri, Catia Pesquita, Francisco M. Couto, and Isabel F. Cruz. 2015. AML results for OAEI 2015. In *CEUR Workshop Proceedings*. 116–123.
[34] D. Faria, C. Pesquita, E. Santos, M. Palmonari, I. F. Cruz, and F. M. Couto. 2013. *The AgreementMakerLight Ontology Matching System*. Springer-Verlag, Berlin, 527–541.
[35] Jennifer Golbeck, Gilberto Fragoso, Frank Hartel, Jim Hendler, Jim Oberthaler, and Bijan Parsia. 2001. The national cancer institute's thésaurus and ontology. *Web Semantics: Science, Services and Agents on the World Wide Web* (2001).
[36] Bernardo Cuenca Grau, Ian Horrocks, Yevgeny Kazakov, and Ulrike Sattler. 2007. A logical framework for modularity of ontologies. *IJCAI International Joint Conference on Artificial Intelligence* (2007), 298–303.
[37] Bernardo Cuenca Grau, Ian Horrocks, Yevgeny Kazakov, and Ulrike Sattler. 2008. Modular reuse of ontologies: Theory and practice. *Journal of Artificial Intelligence Research* (2008), 273–318.

[38] Bernardo Cuenca Grau, Bijan Parsia, Evren Sirin, and Aditya Kalyanpur. 2005. Automatic partitioning of OWL ontologies using e-connections. *CEUR Workshop Proceedings* 946, 1 (2005), 160–167.
[39] Bernardo Cuenca Grau, Bijan Parsia, Evren Sirin, and Aditya Kalyanpur. 2006. Modularity and web ontologies. *Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning* (2006). 198–209.
[40] Anika Gross, Michael Hartung, Toralf Kirsten, and Erhard Rahm. 2010. On matching large life science ontologies in parallel. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 6254 LNBI (2010). 35–49.
[41] Anika Groß, Michael Hartung, Toralf Kirsten, and Erhard Rahm. 2012. GOMMA results for OAEI 2012. *Proceedings of the International Semantic Web Conference* 946, 2 (2012), 160–167.
[42] Marko Gulić and Boris Vrdoljak. 2013. CroMatcher - Results for OAEI 2013. In *Proceedings of the 10th International Workshop on Ontology Matching*, volume 1545. Bethlehem, PA, USA, 117–122.
[43] Marko Gulic, Boris Vrdoljak, and Marko Banek. 2016. CroMatcher: An ontology matching system based on automated weighted aggregation and iterative final alignment. *Journal of Web Semantics* 41 (2016), 50–71.
[44] Matthew Horridge, Bijan Parsia, and Ulrike Sattler. 2008. Laconic and precise justifications in OWL. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (2008). 323–338.
[45] Wei Hu, Yuanyuan Zhao, and Yuzhong Qu. 2006. Partition-based block matching of large class hierarchies. *Proceedings of the 1st Asian Semantic Web Conference (ASWC'06)*. 72–83.
[46] Jakob Huber, Timo Sztyler, Jan Noessner, and Christian Meilicke. 2011. CODI: Combinatorial optimization for data integration—Results for OAEI 2011. *Proceedings of the 6th International Workshop on Ontology Matching* 814, 1 (2011), 134–141.
[47] Panagiotis G. Ipeirotis, Foster Provost, and Jing Wang. 2010. Quality management on Amazon mechanical turk. *Proceedings of the ACM SIGKDD Workshop on Human Computation (HCOMP'10)*. 64.
[48] Robert W. Irving, David F. Manlove, and Gregg O'Malley. 2009. Stable marriage with ties and bounded length preference lists. *Journal of Discrete Algorithms* (2009), 213–219.
[49] Valentina Ivanova, Patrick Lambrix, and Johan Å. Berg. 2015. Requirements for and evaluation of user support for large-scale ontology alignment. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 3–20.
[50] Arkadiusz Jachnik, Andrzej Szwabe, Pawel Misiorek, and Przemyslaw Walkowiak. 2012. TOAST results for OAEI 2012. *CEUR Workshop Proceedings* (2012). 205–212.
[51] H. V. Jagadish. 1990. A compression technique to materialize transitive closure. *TODS* (1990). 558–598. DOI: https://doi.org/10.1145/99935.99944
[52] Fatsuma Jauro, S. B. Junaidu, and S. E Abdullahi. 2014. Falcon-AO++ - An improved ontology alignment system. *International Journal of Computer Applications* 94, 2 (2014), 1–7.
[53] Yves R. Jean-Mary, E. Patrick Shironoshita, and Mansur R. Kabuka. 2009. Ontology matching with semantic verification. *Journal of Web Semantics* 7, 3 (2009), 235–251. arxiv:NIHMS150003
[54] E. Jim. 2014. OAEI 2014—LogMap family results for OAEI 2014. In *CEUR Workshop Proceedings*. 3–7.
[55] Ernesto Jiménez-Ruiz and Bernardo Cuenca Grau. 2011. LogMap: Logic-based and scalable ontology matching. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 273–288.
[56] E. Jiménez-Ruiz, B. Cuenca Grau, R. Berlanga, and I. Horrocks. 2009. Towards a logic-based assessment of the compatibility of UMLS sources. *CEUR Workshop Proceedings* (2009). S2.
[57] E. Jiménez-Ruiz, B. Cuenca Grau, A. Solimando, and V. Cross. 2015. LogMap family results for OAEI 2015. *CEUR Workshop Proceedings* (2015). 171–175.
[58] Ernesto Jiménez-Ruiz, Bernardo Cuenca Grau, and Yujiao Zhou. 2011. LogMap 2.0: Towards logic-based, scalable and interactive ontology matching. In *SWAT4LS* 66, 2 (2011), 45–46.
[59] Ernesto Jiménez-Ruiz, Bernardo Cuenca Grau, Yujiao Zhou, and Ian Horrocks. 2012. Large-scale interactive ontology matching: Algorithms and implementation. *Frontiers in Artificial Intelligence and Applications*, 444–449.
[60] Ernesto Jiménez-Ruiz, Christian Meilicke, Bernardo Cuenca Grau, and Ian Horrocks. 2013. Evaluating mapping repair systems with large biomedical ontologies. *CEUR Workshop Proceedings* (2013). 1000–1010.
[61] Ruoming Jin, Yang Xiang, Ning Ruan, and Haixun Wang. 2008. Efficiently answering reachability queries on very large directed graphs. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data - (SIGMOD'08)*. 595.
[62] Aditya Kalyanpur, Bijan Parsia, Matthew Horridge, and Evren Sirin. 2007. Finding all justifications of OWL DL entailments. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (2007). 267–280.

[63] Wajahat Ali Khan, Muhammad Bilal Amin, Asad Masood Khattak, Maqbool Hussain, and Sungyoung Lee. 2013. System for parallel heterogeneity resolution (SPHeRe) results for OAEI 2013. In *CEUR Workshop Proceedings*. 184–189.

[64] Nouha Kheder and Gayo Diallo. 2015. ServOMBI at OAEI 2015. *Proceedings of the 10th International Workshop on Ontology Matching—Ontology Alignment Evaluation Initiative* (2015). 1–6.

[65] Abderrahmane Khiat and Moussa Benaissa. 2014. InsMT/InsMTL results for OAEI 2014 instance matching. *CEUR Workshop Proceedings* (2014). 120–125.

[66] Wacław Kuśnierczyk. 2008. Taxonomy-based partitioning of the Gene Ontology. *Journal of Biomedical Informatics* 41, 2 (2008), 282–292.

[67] Sabine Massmann, Salvatore Raunich, David Aumüller, Patrick Arnold, and Erhard Rahm. 2011. Evolution of the COMA match system. *CEUR Workshop Proceedings* (2011). 49–60.

[68] Cynthia Matuszek, John Cabral, Michael Witbrock, and John Deoliveira. 2006. An introduction to the syntax and content of Cyc. *Proceedings of the 2006 AAAI Spring Symposium on Formalizing and Compiling Background Knowledge and Its Applications to Knowledge Representation and Question Answering* (2006). 44–49.

[69] Christian Meilicke. 2011. *Alignment Incoherence in Ontology Matching - dissertation - meilicke*. Ph.D. Dissertation. Universitat Mannheim.

[70] C. Meilicke and H. Stuckenschmidt. 2007. Analyzing mapping extraction approaches. *CEUR Workshop Proceedings* (2007).

[71] C. Meilicke, H. Stuckenschmidt, and A. Tamilin. 2007. Repairing ontology mappings. In *Proceedings of AAAI* 67, 3 (2007), 1408–1423.

[72] Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm. 2002. Similarity flooding: A versatile graph matching algorithm. *Data Engineering* (2002), 117–128.

[73] DuyHoa Ngo and Zohra Bellahsene. 2013. YAM++ - Results for OAEI 2013. *CEUR Workshop Proceedings* (2013). 211–218.

[74] Duy Hoa Ngo and Zohra Bellahsene. 2016. Overview of YAM++ - (not) yet another matcher for ontology alignment task. *Journal of Web Semantics* 41 (2016), 30–49.

[75] Khai Nguyen and Ryutaro Ichise. 2013. SLINT+ results for OAEI 2013 instance matching. *CEUR Workshop Proceedings* (2013). 177–183.

[76] Mathias Niepert, Christian Meilicke, and Heiner Stuckenschmidt. 2010. A probabilistic-logical framework for ontology matching. In *Proc. of AAAI* (2010), 1413–1418.

[77] Natalya F. Noy and Mark A. Musen. 2004. Specifying ontology views by traversal. *The Semantic Web ISWC 2004* (2004). 713–725.

[78] Fernando Osorno-Gutierrez, Norman W. Paton, and Alvaro A. A. Fernandes. 2013. Crowdsourcing feedback for pay-as-you-go data integration. *CEUR Workshop Proceedings* (2013). 32–37.

[79] Lorena Otero-Cerdeira, Francisco J. Rodríguez-Martínez, and Alma Gómez-Rodríguez. 2015. Ontology matching: A literature review. *Expert Systems with Applications* (2015), 949–971.

[80] Jyotishman Pathak, Thomas M. Johnson, and Christopher G. Chute. 2009. Survey of modular ontology techniques and their applications in the biomedical domain. *Integrated Computer-Aided Engineering* 3 (2009), 225–242.

[81] Heiko Paulheim, Sven Hertling, and Dominique Ritze. 2013. Towards evaluating interactive ontology matching tools. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (2013). 31–45.

[82] Pavel Shvaiko and Jerome Euzenat. 2008. Ten challenges of ontology matching. In *Proceedings of the 7th International Conference on Ontologies, DataBases, and Applications of Semantics (ODBASE'08)*. 1163–1181.

[83] Adam Pease, Ian Niles, and John Li. 2002. The suggested upper merged ontology: A large ontology for the semantic web and its applications. *Imagine* (2002), 7–10.

[84] Catia Pesquita, Daniel Faria, Emanuel Santos, and Francisco M. Couto. 2013. To repair or not to repair: Reconciling correctness and coherence in ontology reference alignments. *CEUR Workshop Proceedings* (2013). 13–24.

[85] Eric Peukert and S. Massmann. 2010. Comparing similarity combination methods for schema matching. *Journal of GI Jahrestagung* (2010), 692–701.

[86] Erhard Rahm. 2011. Towards large-scale schema and ontology matching. *Schema Matching and Mapping* (2011), 3–27.

[87] Cornelius Rosse and José L. V. Mejino. 2003. A reference ontology for biomedical informatics: The foundational model of anatomy. *Journal of Biomedical Informatics* 36, 6 (2003), 478–500.

[88] F. Hamdi, B. Safar, C. Reynaud, and H. Zargayouna. 2009. Alignment-based partitioning of large-scale ontologies. *Advances in Knowledge Discovery and Management. Studies in Computational Intelligence Series*. Springer, Heidelberg.

[89] Emanuel Santos, Daniel Faria, Catia Pesquita, and Francisco M. Couto. 2015. Ontology alignment repair through modularization and confidence-based heuristics. *PLoS ONE* (2015), 1–17. arxiv:1307.5322

[90] Cristina Sarasua, Elena Simperl, and Natalya F. Noy. 2012. CrowdMap: Crowdsourcing ontology alignment with microtasks. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 7649 LNCS (2012). 525–541.
[91] K. Saruladha and S. Ranjini. 2016. COGOM: COgnitive theory based ontology matching system. *Procedia - Procedia Computer Science* (2016), 301–308.
[92] Frederik C. Schadd and Nico Roos. 2014. Alignment evaluation of MaasMatch for the OAEI 2014 campaign. *CEUR Workshop Proceedings* 1317 (2014). 135–141.
[93] Anne Schlicht and Heiner Stuckenschmidt. 2007. Criteria-based partitioning of large ontologies. *Proceedings of the 4th International Conference on Knowledge Capture* (2007). 171–172.
[94] Anne Schlicht and Heiner Stuckenschmidt. 2008. A flexible partitioning tool for large ontologies. *Proceedings - 2008 IEEE/WIC/ACM International Conference on Web Intelligence, WI 2008* (2008). 482–488.
[95] Stefan Schulz, Ronald Cornet, and Kent Spackman. 2011. Consolidating SNOMED CT's ontological commitment. *Applied Ontology* 6, 1 (2011), 1–11.
[96] Julian Seidenberg, Julian Seidenberg, Alan Rector, and Alan Rector. 2006. Web ontology segmentation: Analysis, classification and use. *WWW'06: Proceedings of the 15th International Conference on World Wide Web*. 13–22.
[97] Guohua Shen, Yinling Liu, Fei Wang, Jia Si, Zi Wang, Zhiqiu Huang, and Dazhou Kang. 2014. OMReasoner: Combination of multi-matchers for ontology matching: Results for OAEI 2014. *CEUR Workshop Proceedings* (2014). 142–148.
[98] Feng Shi, Juanzi Li, Jie Tang, Guotong Xie, and Hanyu Li. 2009. Actively learning ontology matching via user interaction. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (2009). 585–600.
[99] P. Shvaiko and J. Euzenat. 2013. Ontology matching: State of the art and future challenges. *Knowledge and Data Engineering, IEEE* (2013). 158–176.
[100] Jomar Silva, Fernanda Araujo Baião, and Kate Revoredo. 2016. ALIN results for OAEI 2016. *International Semantic Web Conference (ISWC 2016)* (2016).
[101] Klaus Simon. 1988. An improved algorithm for transitive closure on acyclic digraphs. *Theoretical Computer Science* 58, 3 (1988), 325–346.
[102] Alessandro Solimando, Ernesto Jimenez-Ruiz, and Giovanna Guerrini. 2017. Minimizing conservativity violations in ontology alignments: Algorithms and evaluation. *Knowl. Inf. Syst.* 51, 3 (2017), 775–819. DOI: https://doi.org/10.1007/s10115-016-0983-3
[103] Rebecca Steorts, Samuel Ventura, Mauricio Sadinle, and Stephen Fienberg. 2014. A comparison of blocking methods for record linkage. *Proceedings of the International Conference on Privacy in Statistical Databases* 70, 3 (2014), 253–268. arxiv:1407.3191
[104] Giorgos Stoilos, Giorgos Stamou, and Stefanos Kollias. 2005. A string metric for ontology alignment. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (2005). 624–637.
[105] Boontawee Suntisrivaraporn, Guilin Qi, Qiu Ji, and Peter Haase. 2008. A modularization-based approach to finding all justifications for OWL DL entailments. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (2008). 1–15.
[106] Axel Tenschert, Matthias Assel, Alexey Cheptsov, Georgina Gallizo, Emanuele Della Valle, and Irene Celino. 2009. Parallelization and distribution techniques for ontology matching in urban computing environments. *CEUR Workshop Proceedings* 551 (2009). 248–249.
[107] Uthayasanker Thayasivam and Prashant Doshi. 2013. Speeding up batch alignment of large ontologies using MapReduce. *Proceedings—2013 IEEE 7th International Conference on Semantic Computing (ICSC'13)*. 110–113.
[108] Peng Wang. 2010. Lily-LOM: An efficient system for matching large ontologies with non-partitioned method. In *CEUR Workshop Proceedings*, Vol. 658. 69–72.
[109] Mohamed T. Warith E. 2013. Introducing artificial neural network in ontology alignment process. *Springer-Verlag, Berlin*, 176–186.
[110] M. El-Abdi, H. Souid, M. Kachroudi, and S. Ben-Yahia. 2015. CLONA results for OAEI 2015. In *Proceedings of the 10th Workshop on Ontology Matching ISWC 2015, USA* 72, 1 (2015).
[111] Hang Zhang, Wei Hu, and Yu-zhong Qu. 2012. VDoc+: A virtual document based approach for matching large ontologies using MapReduce. *Journal of Zhejiang University-SCIENCE C (Computers & Electronics)* 13, 4 (2012), 257–267.
[112] Yuanzhe Zhang, Xuepeng Wang, and Shizhu He. 2013. IAMA results for OAEI 2013. In *Proceedings of the 8th International Workshop on Ontology Matching* 1111 (2013), 45–47.

Received October 2016; revised January 2018; accepted April 2018