LogMap-P: On matching ontologies in parallel

Muhammad Sadiq
Ubiquitous Computing Lab, Kyung Hee University
Yongin-si, Gyeonggi-do, South Korea
sadiq@oslab.khu.ac.kr

Muhammad Bilal Amin
Ubiquitous Computing Lab, Kyung Hee University
Yongin-si, Gyeonggi-do, South Korea
m.b.amin@ieee.org

Hafiz Syed Muhammad Bilal
Ubiquitous Computing Lab, Kyung Hee University
Yongin-si, Gyeonggi-do, South Korea
bilalrizvi@oslab.khu.ac.kr

Musarrat Hussain
Ubiquitous Computing Lab, Kyung Hee University
Yongin-si, Gyeonggi-do, South Korea
musarrat.hussain@oslab.khu.ac.kr

Anees Ul Hassan
Ubiquitous Computing Lab, Kyung Hee University
Yongin-si, Gyeonggi-do, South Korea
anees@oslab.khu.ac.kr

Sungyoung Lee
Ubiquitous Computing Lab, Kyung Hee University
Yongin-si, Gyeonggi-do, South Korea
sylee@oslab.khu.ac.kr

ABSTRACT

An enormous amount of research have been published related to ontology matching. The core motivation behind these researches aim to develop matching techniques that result in highly accurate ontology matching systems. However the performance (in terms of execution time) of these matching techniques is predominantly unexplored and is equally important. Among the well established research implementations, LogMap an open source system, is considered as state-of-the-art in ontology matching due to its accuracy. This paper presents LogMap-P, an enhanced version of LogMap with motivation to boost performance while preserving the accuracy of the matching techniques.

CCS CONCEPTS

• **Information systems → Entity resolution**; • **Computing methodologies → Parallel algorithms**; *Concurrent computing methodologies*;

KEYWORDS

Ontology matching, Semantic web, Parallel matching

ACM Reference Format:
Muhammad Sadiq, Muhammad Bilal Amin, Hafiz Syed Muhammad Bilal, Musarrat Hussain, Anees Ul Hassan, and Sungyoung Lee. 2018. LogMap-P: On matching ontologies in parallel. In *IMCOM '18: The 12th International Conference on Ubiquitous Information Management and Communication, January 5–7, 2018, Langkawi, Malaysia.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3164541.3164589

1 INTRODUCTION

The excess of available knowledge over the ubiquitous platforms have resulted in semantic heterogeneity issues [2]. The resolution for this issue is ontology matching which derives alignment between semantically related ontologies representing knowledge resources [3]. However, ontology matching is a two-fold problem where challenges exist in two primary categories: (i) accuracy, which measures the effectiveness of the matching process; and (ii) performance, in terms of execution time taken by the matching process [8]. Although, substantial amount of effort has been placed by the semantic web researchers on the accuracy of the ontology matching systems, but, performance is still a challenge that has largely been unaddressed. Performance wise, ontology matching is a computationally intensive task with quadratic computational complexity [4]. It has been reported by researchers [2][1], that whole matching process to generate alignment on large-scale ontologies have taken from hours to days depending upon the complexity of the matching algorithms. Due to the excess of available knowledge in current years, ontologies have grown larger with increased amount of complexity. Traditionally, ontology matching has been an offline process; therefore, matching time was not considered as a challenge even with larger and complex ontologies; however, requirements of current dynamic knowledge base systems require in-time resolution. Thus, performance aspects of ontology matching needs to be addressed with scalability and resource utilization in perspective [1].

Ontology matching systems developed over the years have taken the execution time of the matching process in consideration and device possible resolutions; however, their implementations are accuracy bound, i.e., complexity of the matching algorithms. Authors of [2], have termed these implementations as effectiveness-dependent implementations where trade-off between accuracy (F-measure, precision, and recall) and execution time (performance) exists. For effectiveness-independent implementations there are only few attempts. For example, ontology matching system GOMMA [7] implements partial parallelism over its matchers to improve overall performance. On the other hand SPHeRe [1][6] takes a end to end parallelism approach that exploits the multicore and multinode platforms for ontology matching making it extremely effective for large scale ontology matching. However, reviewing the last five years of Ontology Alignment and Evaluation Initiative (OAEI)[1] evaluation, both of these promising systems failed to participate regularly, which indicates a gap in the expected evolution of ontology

IMCOM '18, January 5–7, 2018, Langkawi, Malaysia

ACM ISBN 978-1-4503-6385-3/18/01…$15.00
https://doi.org/10.1145/3164541.3164589

[1] http://oaei.ontologymatching.org

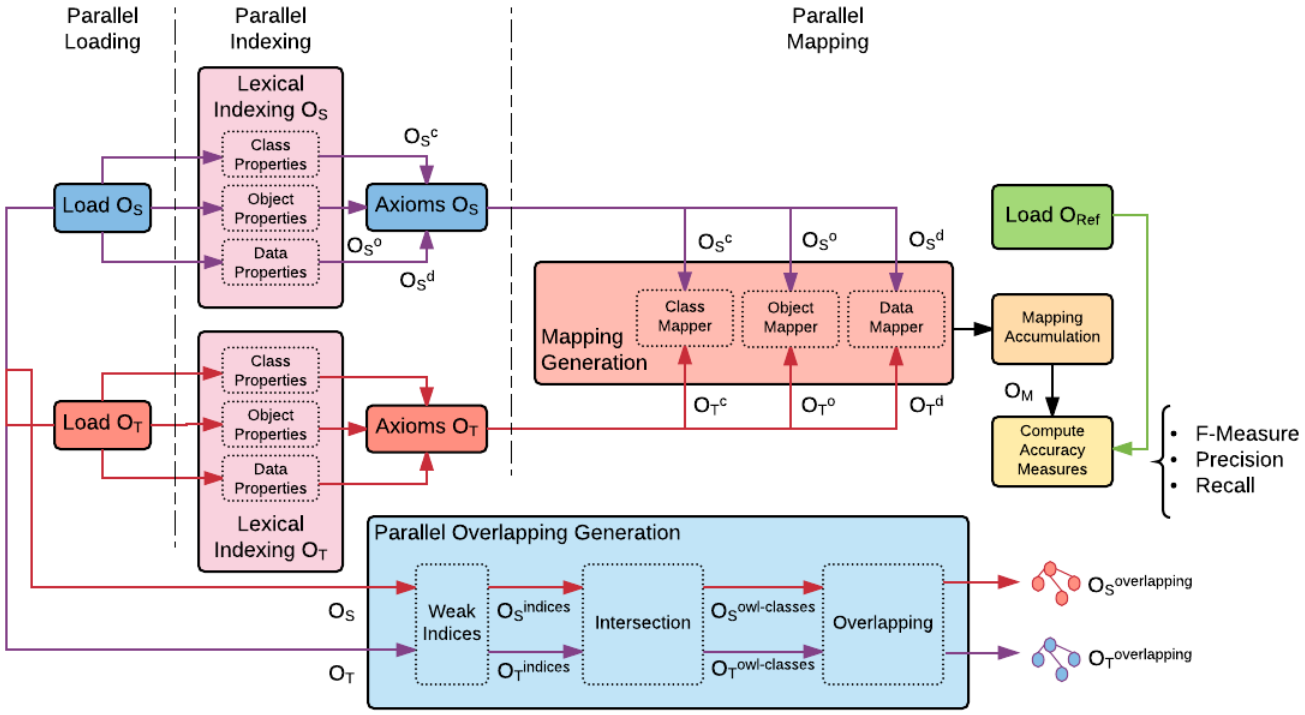

Figure 1: Logmap-p parallel execution flow diagram.

matching from the perspective of effectiveness-independent performance gain. Furthermore, among the OAEI Participants, LogMap is one such system that has been consistently participating in evaluation and is considered as a benchmark/state-of-the-art in ontology matching. Although LogMap is highly effective but is effectiveness-dependent performance wise. Being available as open source[2], LogMap brings an opportunity for other researchers to study its implementation and provide possible performance measures without losing its effectiveness. Considering this possibility as an opening, we present LogMap-P as our performance enhanced implementation of LogMap.

This paper is structured as follows: Section 2 describes the methodology adopted for the implementation of LogMap-P. Section 3 provides details on the evaluation of LogMap-P in contrast with original implementation of LogMap. Section 4 concludes this paper.

2 PROPOSED METHODOLOGY

LogMap[5] is a scalable and domain independent state-of-the-art ontology matching system which is developed to draw alignments between small and large-scale ontologies. It has three implementations, i.e.,LogMapLT (a lite version for lexical matching), LogMap-Full(a version for large-scale ontologies), and LogMapBK (a version for matching the ontologies that require background knowledge). For its lite verions, it uses inverted lexical and structural indexing for its matching process. In lexical indexing an inverted index is constructed by splitting each class label into components and computing its lexical variation. While structural indexing uses interval labeling schema to efficiently access hierarchical information from ontologies. Although its highly accurate; however, lacks performance enhancements. Upon detail code review, we have been able to establish numerous execution points that can be efficiently parallelize for performance reasons without effecting the overall accuracy of the system. This section of the paper provides details of our methodology concerning the performance efficency enhancment of LogMap; resulting in an upgraded version we call LogMap-P.

Table 1: Terminologies And Notations

Notation	Description
O_S, O_T	Source and Target ontology respectively
O_i	For simplicity O_i represents both O_S, O_T
$O_i^{indices}$	Represents O_S, O_T Inverted lexical indices
$O_i^{OWLClasses}$	Represents OWLClasses of O_S, O_T
$O_i^{Overlapping}$	Represents Overlapping of O_S, O_T
GS	Gold standard file
O_i^C, O_i^O, O_i^D	Represents set of class, Object and data property labels respectively for O_S, O_T
$\rightarrow$	Mapping symbol
$\leftarrow$	Assignment operator

[2]https://sourceforge.net/projects/logmap-matcher/

2.1 LogMap-P Execution Flow

LogMap-P performs indexing, mapping, and generates overlapping of ontologies in parallel. An extended version of LogMap[5], which focuses on parallelism is depicted (*Figure 1*). To support flexible and effective parallelization we decompose LogMap architecture into four major parts: (1) Parallel Loading, (2) Parallel lexical indexing, (3) Parallel Mappings, and (4) Parallel overlapping (*Figure 1*).

Parallel Loading deals with loading of O_S, O_T ontologies. These ontologies are independent of each Other, therefor can be safely loaded in parallel. As a consequences, there is a great time reduction in loading parallel ontologies.

In Parallel Lexical Indexing lexical indices for class, object and data property labels are generated for O_S, O_T in parallel. The lexical indexing process consists of two major parts $O_S^{indices}$, $O_T^{indices}$ and each of them is further decomposed into O_i^C, O_i^O and O_i^D respectively as shown in (*Figure 1*). Conventionally, LogMap creates *lexical indices* for O_i^C, O_i^O and O_i^D separately, because in mapping phase, it maps $O_S^C \rightarrow O_T^C$, $O_S^O \rightarrow O_T^O$, and $O_S^D \rightarrow O_T^D$. Therefor, lexical indexing time is reduced incredibly by parallelizing O_S, O_T in two separate parallel processes. Further, each parallel process creates O_i^C, O_i^O and O_i^D of O_S, O_T independently in parallel.

Parallel Mappings: In existing LogMap, to extract equivalent classes, each O_S^C, O_S^O, O_S^D are mapped with O_T^C, O_T^O and O_T^D respectively. In extended LogMap-P, the mapping phase consists of three independent parallel processes e.g. $p_1(O_S^C, O_T^C)$, $p_2(O_S^O, O_T^O)$, and $p_3(O_S^D, O_T^D)$ where each *p1*, *p2*, and *p3* represents an independent process. These concurrent processes add computed mappings to a data structure that supports concurrent operations. Once mappings are generated, then an alignment of O_S, and O_T is computed in parallel. The alteration of mapping phase into parallel processes resulted in a significant improvement in execution time.

Parallel Overlapping: LogMap-P extracts $O_S^{indices}$, and $O_T^{indices}$ in parallel and then intersects them to compute overlapping indices e.g. $Common^{indices} = O_S^{indices} \cap O_T^{indices}$. Then, the *OWLClass* for each of $Common^{indices}$ is extracted from O_S, O_T in parallel. Finally $O_S^{overlapping}$, and $O_T^{overlapping}$ ontologies are created.

The whole process, goes in parallel as shown in *(Figure 1)* which remarkably improves the performance of LogMap-P in terms of execution time as compared to LogMap.

2.2 LogMap-P Algorithm

LogMap-P algorithm match ontologies in parallel. It requires O_S, O_T and *GoldStandard* file as an input. As a result, it generate mappings for O_S, O_T *(see Algorithm 1)*.

To efficiently utilize the available processors and avoid load-imbalance which is mostly caused by non-uniform data distribution among the available cores. The algorithm maintains a balance among the cores in-hand and the number of parallel threads. It creates an *executor* service which is an asynchronous execution mechanism that has the ability to execute multiple tasks in parallel *(see line 1, 2)*. The algorithm loads O_S, O_T, and *GoldStandard* by using *executor*. To compute *mappings* of O_S, O_T algorithm defines a procedure *(see algorithm 2)* which takes O_S, O_T as an input and returns the desired *mappings* of O_S, O_T. The algorithm computes *precision, recall, and f-measure* of the *mappings w.r.t GoldStandardFile*.

Algorithm 1: LogMap-P LogMap Performance

Input : $O_S File, O_T File, GoldStandardFile$
Output: $AlignmentFileRDF, Overlapping$

```
1  CPUs ← Runtime.getAvailableProcessors();
2  pExec ← Executor.createFixedThreadPool(CPUs);
3  O_S ← pExec.load(O_S File);
4  O_T ← pExec.load(O_T File);
5  GoldStandard ← pExec.load(GoldStandardFile);
6  # see procedure 2;
7  Mappings ← generateMappings(O_S, O_T, pExec);
8  Precision ← computePrecision(Mappings, GoldStandard);
9  Recall ← computeRecall(Mappings, GoldStandard);
10 # see procedure 3;
11 O_S O_T Overlapping ← Overlappings(O_S, O_T, pExec);
12 AlignmentRDF ← generateAlignment(Mappings);
```

Finally the algorithm generates *Overlapping* ontologies of O_S, O_T by using overlapping procedure *(see algorithm 3)*.

Algorithm 2: Parallel Mappings Procedure

Input : $O_S, O_T, pExec$
Output: $Mappings$

```
1  concurrentStruct ← newConcurrentStruct();
2  O_S InvertedIndex ← newHashMap();
3  O_T InvertedIndex ← newHashMap();
4  LabelSet ← [Class^label, Object^label, Data^label];
5  for label ← in LabelSet do
6      O_S InvertedIndex.add(pExec.generateInvertedIndices(O_S, label));
7      O_T InvertedIndex.add(pExec.generateInvertedIndices(O_T, label));
8  for label ← in LabelSet do
9      O_S LabelIndx ← O_S InvertedIndex.getIndexOf(label);
10     O_T LabelIndx ← O_T InvertedIndex.getIndexOf(label);
11     concurrentStruct.add(
         pExec.computeMappingsFor(label, O_S LabelIndx, O_T LabelIndx)
       );
12 Mappings ← concurrentStruct.getAllMappings();
```

Mapping Procedure: Mapping is a computation intensive work which requires pairwise mappings. Therefor we proposed an efficient parallel version as shown in *algorithm 2*. It requires O_S, O_T ontologies as an input. At first, *inverted indices* of *class*, *object* and *data property labels* are created for O_S and O_T in parallel *line 5 to 7*. After that, set of pairs of *class*, *object* and *data property* mappings of O_S, O_T are generated and added to a data structure that support concurrent operations *line 8 to 11*. Finally all sets of pairs of mappings are fetched from concurrent data structure.

Overlapping Procedure: The overlapping algorithm consists of hierarchical steps and the computational complexity of the algorithm approaches to quadratic. In order to minimize execution time we

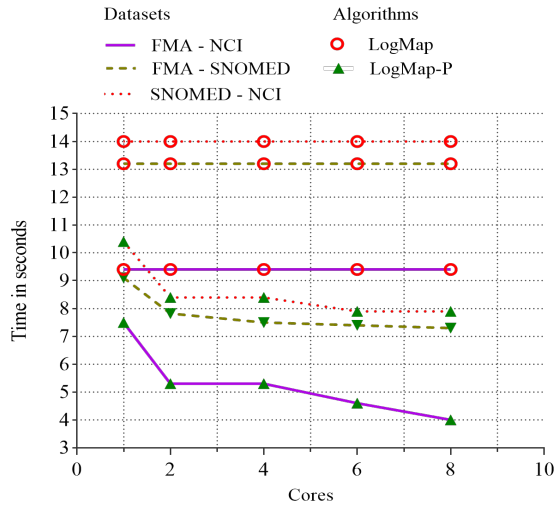

Figure 2: Time comparison between logmap and logmap-p.

parallelized overlapping algorithm without increasing its computational complexity (*see algorithm 3*). To obtain it, new data structures have been introduced that support concurrent operations (*see line 2, 3*). Futher, we compute *common-indices* in O_S, O_T (*line 4*). In the first iteration of the loop *(line 5 to 13)* set of weak inverted indices of *Class labels* are created for O_S and O_T and then concurrently added. These indices are associated with each *index* where *index* $\in$ *common-indices*. Similarly inverted indices of *Object* and *Data property labels* are created subsequently in the second and third iteration of the loop. Finally overlapped ontologies for O_S , O_T are computed in *line 14, 15* respectively.

3 EVALUATION

We performed extensive evaluation of the proposed LogMap-P algorithm against LogMap on multiple benchmark datasets. We have taken FMA, NCI and SNOMED ontologies with 73920, 64880, and 306160 classes respectively from Ontology Alignment Evaluation Initiative (OAEI) datasets. For experiment we combined ontologies into three different pairs e.g. *FMA-NCI, FMA-SNOMED, SNOMED-NCI*. We have evaluated the matching algorithm by drawing the comparison among these ontologies. The efficiency of proposed algorithm is shown in *Figure 2*. We have conducted the experiment for each pair in multiple of 100 and then draw the average time of mapping taken by LogMap and LogMap-P for each pair. All the experiments are conducted on a system with memory 16GB, Intel Core i7 with 4 physical cores of 3.60GHz processing speed. Our proposed algorithm LogMap-P performed around 38 % efficient than LogMap algorithm in-term of execution time as shown in *Figure 2*. The proposed algorithm concerned only the efficiency in terms of execution time through parallelizing the matching components so there is no degradation of matching accuracy of ontologies. Hence, our algorithm preserved the same accuracy as that of LogMap algorithm.

Algorithm 3: Parallel Overlapping Procedure

Input : $O_S, O_T, pExec$
Output: Overlapping Ontologies for O_S, O_T

```
1  LabelSet ← [Class^label, Object^label, Data^label];
2  O_S_overlappings ← newConcurrentHashSet();
3  O_T_overlappings ← newConcurrentHashSet();
4  Intersection ←
     O_S.weakInvertedInices ∩ O_T.weakInvertedInices
5  for labelSet ← in Intersection do
6      pExec.submit(
         for wii ← in O_S.weakInvertedIndices.get(labelSet)
         do
7            concurrentUpdate(O_S_overlappings, wii);
8      );
9      pExec.submit(
         for wii ← in O_T.weakInvertedIndices.get(labelSet)
         do
10           concurrentUpdate(O_T_overlappings, wii);
11     );
12 O_S_overlapping ←
     generateOverlappingOntology(O_S_overlappings);
13 O_T_overlapping ←
     generateOverlappingOntology(O_T_overlappings);
```

4 CONCLUSIONS

Our proposed LogMap-P algorithm is an extended version of open source LogMap algorithm which has enhanced the efficiency by 38 %, while preserving the same accuracy level. LogMap-P code is available as an open-source at Bitbucket (https://bitbucket.org/bit-whacker/logmap-p) for research enhancement and development. For evaluation purposes we have selected the well known data set from OAEI. In future we want to extend our work to parallelize the LogMapFull algorithm.

ACKNOWLEDGMENTS

This work was supported by the Korea Research Fellowship Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT (NRF-2016H1D3A1938039). This work was also supported by the Industrial Core Technology Development Program (10049079 , Develop of mining core technology exploiting personal big data) funded by the Ministry of Trade, Industry and Energy (MOTIE, Korea)

REFERENCES

[1] Muhammad Bilal Amin, Rabia Batool, Wajahat Ali Khan, Sungyoung Lee, and Eui-Nam Huh. 2014. SPHeRe. *The Journal of Supercomputing* 68, 1 (01 Apr 2014), 274–301. https://doi.org/10.1007/s11227-013-1037-1
[2] Muhammad Bilal Amin, Wajahat Ali Khan, Sungyoung Lee, and Byeong Ho Kang. 2015. Performance-based ontology matching. *Applied Intelligence* 43, 2 (01 Sep 2015), 356–385. https://doi.org/10.1007/s10489-015-0648-z
[3] Jrme Euzenat and Pavel Shvaiko. 2013. *Ontology Matching* (2nd ed.). Springer Publishing Company, Incorporated.
[4] Anika Gross, Michael Hartung, Toralf Kirsten, and Erhard Rahm. 2010. *On Matching Large Life Science Ontologies in Parallel.* Springer Berlin Heidelberg, Berlin, Heidelberg, 35–49. https://doi.org/10.1007/978-3-642-15120-0_4
[5] Ernesto Jiménez-Ruiz and Bernardo Cuenca Grau. 2011. Logmap: Logic-based and scalable ontology matching. In *International Semantic Web Conference*. Springer,

273–288.

[6] 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 *Proceedings of the 8th International Conference on Ontology Matching - Volume 1111 (OM'13)*. CEUR-WS.org, Aachen, Germany, Germany, 184–189. http://dl.acm.org/citation.cfm?id=2874493.2874511

[7] Toralf Kirsten, Anika Gross, Michael Hartung, and Erhard Rahm. 2011. GOMMA: a component-based infrastructure for managing and analyzing life science ontologies and their evolution. *Journal of Biomedical Semantics* 2, 1 (13 Sep 2011), 6. https://doi.org/10.1186/2041-1480-2-6

[8] P. Shvaiko and J. Euzenat. 2013. Ontology Matching: State of the Art and Future Challenges. *IEEE Transactions on Knowledge and Data Engineering* 25, 1 (Jan 2013), 158–176. https://doi.org/10.1109/TKDE.2011.253