TIM Results for OAEI 2025

Alexander Becker[1,*], Axel-Cyrille Ngonga Ngomo[1] and Mohamed Ahmed Sherif[1]

[1]Data Science Group (DICE), Heinz Nixdorf Institute, Paderborn University, Paderborn, Germany

Abstract

TIM is a knowledge graph matching system with an architecture of **T**iered **I**terative **M**atchers that matches ontologies and instances simultaneously. The main idea behind TIM is to create matches based on existing matches and their connections within the knowledge graph. This is the first year where TIM is participating in the OAEI knowledge graph track and it achieves the highest performance for class and property matching as well as the second best performance for instance matching. Furthermore, TIM is the fastest non-baseline system participating in the knowledge graph track.

Keywords

Link Discovery, Ontology Matching, Instance Matching, Tiered Iterative KG Matching, Knowledge Graph Matching, TIM OAEI 2025

1. Presentation of the System

1.1. State, Purpose, General Statement

TIM is a knowledge graph matching system which iteratively creates new matches based on already found matches. During this process, TIM relies on similar triples of instance pairs instead of creating matches based on lexical or phonetic similarities.

TIM has two main components: Iterative matchers and bootstrap matchers. *Iterative matchers* run multiple times during the approach and therefore require a low total runtime; optimally linear with respect to the number of triples in the input knowledge graphs. An iterative matcher gets the instance pairs, property pairs, and class pairs that were matched since the last time this matcher was started as an input (when a matcher creates a match, this matcher will be in the input set of the next iteration). An example for an iterative matcher is the following: *When a class pair gets matched, match all instance pairs that belong to the respective classes when the instances have the same label.*

Bootstrap matchers are used to create new seed matches when there are no more matches created by the iterative matchers. Each bootstrap matcher is only run once and therefore they do not have such strict runtime restrictions. An example for a bootstrap matcher is the `Match all classes with the same label` matcher.

1.2. Specific Techniques Used

Let S and T be the two input knowledge graphs that contain the classes S_C and T_C, the properties S_P and T_P, and the instances S_I and T_I. Furthermore, let $s \cong t$ for $s \in S_C \cup S_P \cup S_I, t \in T_C \cup T_P \cup T_I$ mean that s is matched to t. The $\cong$ relation is injective both ways, i.e., we only create 1-to-1 matches and reject any new matching attempt if the source component or target component of the match is already matched to something else.

For checking the equivalence of elements from the KGs based on the current state of the matching process, we define the $\textcircled{I}$ function that maps target elements to their IRI, source elements that are

OM 2025: The 20th International Workshop on Ontology Matching collocated with the 24th International Semantic Web Conference (ISWC 2025), November 2nd, 2025, Nara, Japan

*Corresponding author.

beckeral@campus.upb.de (A. Becker); axel.ngonga@upb.de (A. Ngonga Ngomo); mohamed.sherif@upb.de (M. A. Sherif)
https://dice-research.org/ (A. Becker); https://dice-research.org/ (A. Ngonga Ngomo); https://dice-research.org/ (M. A. Sherif)
0009-0003-8212-4647 (A. Becker); 0000-0001-7112-3516 (A. Ngonga Ngomo); 0000-0002-9927-2203 (M. A. Sherif)

© 2025 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073
published 2025-12-22
CEUR-WS.org/Vol-4144/om2025-oaei-paper5.pdf

matched to a target element to the target elements IRI, unmatched source elements to their own IRI, and literals to their lexical representation. Formally,

$$\textcircled{\scriptsize I}(x) = \begin{cases} \texttt{IRI}(x) & : \text{if } x \in (T_C \cup T_P \cup T_I) \\ \texttt{IRI}(t) & : \text{if } x \in (S_C \cup S_P \cup S_I) \wedge \exists x \cong t \\ \texttt{IRI}(x) & : \text{if } x \in (S_C \cup S_P \cup S_I) \wedge \nexists x \cong t \\ \texttt{lexicalRepresentation}(x) & : \text{if } x \text{ is a literal} \end{cases}$$

Based on this, we define triple representations that map triples consisting of $\langle$head, rel, tail$\rangle$ to a string representation with ψ being a symbol that is not present in either input knowledge graph and $+$ being string concatenation.

$$\textcircled{\scriptsize I}_{\text{HRT}}(\text{head,rel,tail}) = \textcircled{\scriptsize I}(\text{head}) + \psi + \textcircled{\scriptsize I}(\text{rel}) + \psi + \textcircled{\scriptsize I}(\text{tail}).$$

Two triple representations are only equal if all components of both components are matched to their respective counterpart in the other triple.

Sometimes not all parts of the triple should be included in the representation, for example if we only want to check if the head and rel component of two triples are equal given the existing matches. Hence, we define representations for parts of a triple $\textcircled{\scriptsize I}_{\text{HR}}$(head,rel,tail) based on the head and relation component, $\textcircled{\scriptsize I}_{\text{HT}}$(head,rel,tail) based on the head and tail component, and $\textcircled{\scriptsize I}_{\text{RT}}$(head,rel,tail) based on the relation and tail component:

$$\textcircled{\scriptsize I}_{\text{HR}}(\text{head,rel,tail}) = \textcircled{\scriptsize I}(\text{head}) + \psi + \textcircled{\scriptsize I}(\text{rel})$$

$$\textcircled{\scriptsize I}_{\text{HT}}(\text{head,rel,tail}) = \textcircled{\scriptsize I}(\text{head}) + \psi + \textcircled{\scriptsize I}(\text{tail})$$

$$\textcircled{\scriptsize I}_{\text{RT}}(\text{head,rel,tail}) = \textcircled{\scriptsize I}(\text{rel}) + \psi + \textcircled{\scriptsize I}(\text{tail})$$

1.2.1. Tim Architecture

The architecture of Tim based on iterative matchers and bootstrap matchers is visualized in Figure 1. The iterative matchers are tiered by precision with the first iterative matcher having the highest precision. When the first iterative matcher is run and finds a match, the matcher is run again to be able to find more matches based on this match. When the first iterative matcher does not find a new match, then the second iterative matcher is run. As long as no new match is found by any matcher, the next tier matcher is run. When a new match gets found, the process starts at the first iterative matcher again because lower tier matchers can find more accurate matches that could conflict with matches found by subsequent matchers.

After the last iterative matcher does not find any new match, the first bootstrap matcher that was not run yet is run. As long as the bootstrap matchers do not find any new match, the next bootstrap matchers is run as by definition no iterative matcher cannot find a new match. When a new match is found, the first iterative matcher is run again. When all bootstrap matchers were run already and the iterative matchers do not find any new match, the process ends.

1.2.2. Iterative Matchers

The iterative matchers build matches upon already existing matches. They are based mostly on the structure of the input knowledge graphs and match pairs that share similar connections to other instance pairs that were matched before. Properties are matched by unique occurrences and classes are matched based on the overlap of matched instance pairs with classes in the other KG. We summarize the iterative matchers of Tim in Table 1.

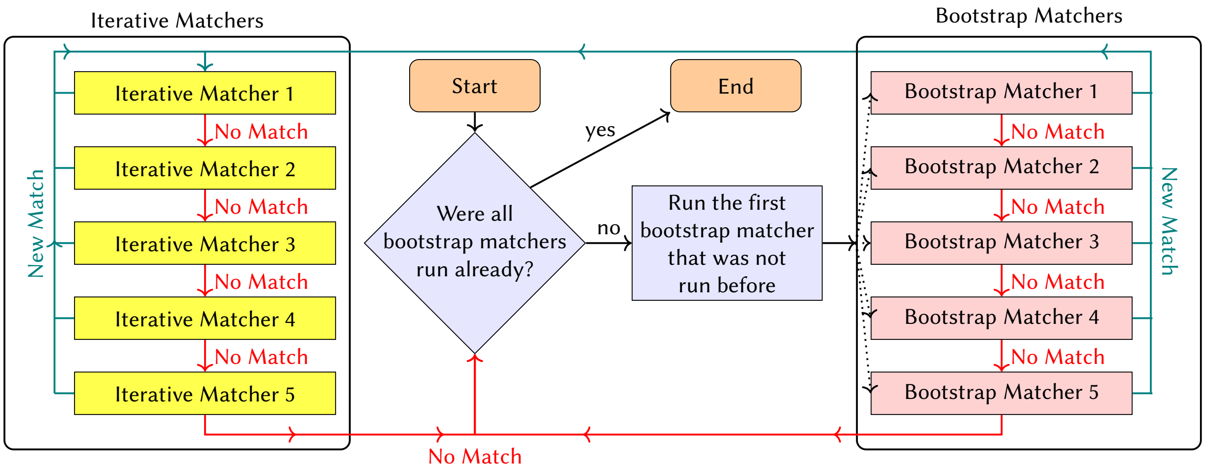

Figure 1: A visualization of the TIM architecture for 5 iterative matchers and 5 bootstrap matchers. Whenever a new match is found, the process starts again with the first iterative matcher. When no match is found, the next matcher is run.

Table 1
Summary of TIM's iterative matchers

Tier	Type	Description
1	Instance	Matches instances that have the same label and alternative label, and belong to matching classes.
2	Instance	Matches instances that have the same label and alternative label, and are both the tail component in a triple where the head components got matched since the last iteration of this matcher and relation components are matched.
3-7	Instance	Matches instances that have at least two common triple representations (excluding labels) based on relation and tail component where the instances are the head component, and depending on the tier have: an equivalent label and an equivalent altLabel (Tier 3); an equivalent label (Tier 4); an equivalent alternative label (Tier 5); a source instance label equivalent to a target instance altLabel (Tier 6); a source instance altLabel equivalent to a target instance label (Tier 7).
8-12	Instance	The same as Tier 3-7 except that one common triple representation is enough.
13	Property	Matches properties that have a unique triple representation based on head and tail component.
14	Instance	Matches instances that have two common unique triple representations based on the relation and tail component.
15	Class	Matches classes when the overlap between instances belonging to these classes is greater than the mean instance overlap for already matched classes (excluding class pairs with less than 20% overlap).

1.2.3. Bootstrap Matchers

The bootstrap matchers are used to create more seed matches in case the iterative matchers do not have any new matches available to build new matches upon. Most of the bootstrap matchers are simple and rely on matching classes, properties, or instances that have the same IRI, same label, similar words in the label, or unique attributes. We summarize the bootstrap matchers of TIM in Table 2.

Table 2
Summary of TIM's bootstrap matchers. String equivalence is used case insensitive by all the matchers. The bootstrap matchers 1a, 1b, and 1c are run after another without checking if new matches got created in between.

Tier	Type	Description
1a	Property	Matches properties with an equivalent IRI.
1b	Class	Matches classes that have an equivalent label and equivalent altLabel.
1c	Property	Matches properties that have an equivalent label and equivalent altLabel.
2	Class	Matches the classes whose labels have more than a 50% jaccard overlap (split by spaces) starting with the highest overlap pairs.
3	Property	Matches the properties whose labels have more than a 50% jaccard overlap (split by spaces) starting with the highest overlap pairs.
4	Instance	Matches instances that have an equivalent label and equivalent altLabel.
5	Instance	Matches instance pairs where the source and target instance have one unique common triple representation based on the relation and tail component.
6	Instance	Matches instance pairs that have a common word (separated by spaces) in any literal attribute that is unique to that pair.

2. Results

Since TIM is specifically designed to work on knowledge graphs that contain both ontology elements and instance information, TIM only produces significant results when the instances are present in the input KGs. While TIM is able to produce a mapping even when no instances are present in the input, only the very easy to find matches are included in the mapping; e.g., classes with equivalent labels. This is caused by all iterative matchers requiring instance information to function.

2.1. Knowledge Graph Track

As TIM's main focus is the knowledge graph track, we asses the results in the following with TIM's results for this track shown in Table 3. For the full results including other systems, see [1].

Table 3
F-measure results for TIM in the OAEI 2025 knowledge graph track. Cells with bold text indicate that TIM achieved the best score of all matchers for this dataset.

Type	mcu-mv	mma-mmb	mma-stx	sw-swg	sw-swtor
CLASS	**1.00**	**0.96**	**0.96**	0.89	0.93
PROPERTY	**0.95**	**0.97**	**0.98**	**1.00**	0.95
INSTANCE	**0.82**	0.92	**0.95**	0.81	0.91

TIM produces better results than all participants from the last years edition. The only competitor that was able to surpass TIM in parts of the results is the newly participating DOGMA system.

For the *class mappings*, TIM placed first for three out of five datasets and second for the remaining two. Additionally, TIM achieves the best overall class F-measure. The most outstanding result is generated for the datasets about Startrek where TIM surpasses all other systems by at least 0.14 F-measure. For the *property mappings*, TIM again achieves the best overall property F-measure with the first place for four datasets and the second place for one dataset. For the *instance mappings*, TIM places second behind DOGMA in the overall comparison. TIM only places at the top spot two times for the instance datasets and places second three times. While the difference to DOGMA is very small for most datasets, the difference amounts to 0.08 for the `starwars-swg` dataset.

Considering the *runtime*, overall Tim is the fastest non-baseline matcher[1] with a total runtime of only 34 minutes and 13 seconds while the second fastest matcher (LogMap) needed over 56 minutes to finish the alignment process. Tim's achieved time is only three times as much time as needed for the baseline matchers that only compare labels. Hence, we consider our matcher incredibly efficient.

3. General Comments

3.1. Comments on the Results

Throughout all results, Tim currently prioritizes precision over recall. The system provides better results than all systems that were participating in the last year and achieves best as well as second best rankings in this years version.

3.2. Discussions on the Way to Improve the Proposed System

Up until this point, we have not done a lot of optimizations on Tim. Some points we will improve in the future are the following:

For the *iterative matchers*, there are still a lot of possibilities. Currently, for the iterative matchers 3-12 (see Table 1), the process is only executed to match head instances based on common triple representations on the basis of relations and tail instances. This could easily be expanded to also work the other way around, i.e. matching tail instances that have common triple representations based on the head and relation component. Furthermore, the iterative matchers strongly rely on equivalent labels and alternative labels, which could be relaxed to similar labels to increase recall.

The *bootstrap matchers* possess significantly more potential capabilities than those presently realized in the current version of Tim. They do not consider lexical or phonetical similarities, which can be beneficial for finding additional matches.

Additionally, the architecture could be adjusted to enable certain iterative matchers to activate only after a specified number of matches have been identified or once a predefined set of bootstrap matchers has been executed. This adjustment would help prevent erroneous matches produced by iterative matchers when only a limited number of matches are available (in particular, iterative matcher 15 is prone to such errors under sparse matching conditions).

3.3. Comments on the OAEI Test Cases

We really enjoyed the test cases extracted from wikis using DBkWik[2]. However, when analyzing the errors made by our approach, we discovered that the test cases based on the starwars wiki contain several instance pairs where we do not believe that any matcher has a chance finding the correct match. This is caused by the starwars wiki having two pages for multiple instances: One for the canon based description of an entity and another one based on starwars stories outside of the canon called *Legends*. One example for this is https://starwars.fandom.com/wiki/Yoda and https://starwars.fandom.com/wiki/Yoda/Legends, which both describe *Yoda*. The other wikis often do not distinguish between the canon and legends version of a character and the external links often only exist to either the canon or legends version.

We believe that this can be easily fixed in the next years version, for example by removing all instances from `https://starwars.fandom.com` that end with `/Legends`, and changing the alignment file to let correspondences containing an instance from legends to point to the canon instance instead. Alternatively, the gold standard could be extended by including both the canon as well as the legends version for instances that currently are only linked to one of them.

[1]DogMa is excluded from this because it has no runtime information available as only it's alignment files were submitted

4. Conclusion

Within this system paper, we presented Tim, a knowledge graph matching system using an architecture of tiered iterative matchers that jointly align ontologies and instances. The approach leverages structural information about the knowledge graphs utilizing previously established matches to discover new matches. Throughout the evaluation on the knowledge graph track, Tim demonstrates a strong performance, achieving best or second-best results in this years edition, outperforming all previously established systems from the last years edition. Notably, the system showed a high degree of efficiency with a total runtime significantly below all non-baseline competitors.

In future work, we are going to add additional matchers into Tim further increasing the accuracy especially regarding instance matching. Furthermore, we are going to optimize the runtime even more to make the approach scalable to even bigger datasets.

Acknowledgments

This work has been supported by the Ministry of Culture and Science of North Rhine-Westphalia (MKW NRW) within the project SAIL under the grant no NW21-059D. This work has been supported within the project "WHALE" (LFN 1-04) funded under the Lamarr Fellow Network programme by the Ministry of Culture and Science of North Rhine-Westphalia (MKW NRW). This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101070305.

Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

[1] Results for oaei 2025 - knowledge graph track, 2025. URL: https://oaei.ontologymatching.org/2025/results/knowledgegraph/index.html, accessed: 2025-10-23.

[2] A. Hofmann, S. Perchani, J. Portisch, S. Hertling, H. Paulheim, Dbkwik: Towards knowledge graph creation from thousands of wikis., ISWC (Posters, Demos & Industry Tracks) 1 (2017) 2.