ReSolution: Incrementally maintaining knowledge graphs

Thesis Report

Peter Baile Chen
University of Pennsylvania
cbaile@seas.upenn.edu

Zachary G. Ives
University of Pennsylvania
zives@cis.upenn.edu

May 3, 2022

Abstract

Entity matching (EM) refers to the process of determining and conflating the same real-world entities. Lately, there has been a lot of efforts [1, 2, 3, 4, 5] that uses machine learning (ML) models to learn EM. These work have shown that ML models can greatly enhance the accuracy of EM. However, all of these work focus on one-time training and inference/ prediction which is hardly the case in real life. In a real-life setting, models are constantly retrained to obtain higher accuracy on test data. Then, to get the most updated results, re-inference need to be done. However, inference itself is a time-consuming process, not to mention frequent re-inference. To address this issue of prolonged re-inference time, we propose ReSolution, a novel EM re-inference solution that reuses results from previous iterations and performs efficient pruning to speed up model re-inference. We demonstrate that ReSolution outperforms complete re-inference by 16-20% with 100% accuracy guarantees.

Keywords knowledge graph · threshold algorithm · inference

1 INTRODUCTION

Knowledge graph is a widely-used model to capture real-world information where different entities are represented as nodes in the graph and edges are created between two nodes if there is a relationship between them. For instance, in a knowledge graph that represents scientific papers [6], a node could be a paper, an author, or a conference. A paper node could be connected with an author node if the author wrote the paper.

Entity matching (EM), which refers to the process of determining and conflating the same real-world entities, is a crucial component to maintain a knowledge graph as there could be multiple duplicated nodes that slightly differ from each other but essentially referring to the same entities. Lately, there has been a lot of efforts [1, 2, 3, 4, 5] that uses machine learning (ML) models to learn EM. These work have shown that ML models can greatly enhance the accuracy of EM. However, all of these work focus on one-time training and inference/ prediction which is hardly the case in real life. In a real-life setting, if incorrect matchings in a knowledge graph are identified, these matchings will be added to the training data to re-train the model. After which, re-inferencing need to be done in order to obtain the most updated results. Unfortunately, inference is an extremely time-consuming task, which can easily take several hours up to several days for huge real-life datasets. Whenever the model is retrained, another complete reinference need to be performed. This frequent reinference can lead to prolonged update time.

To address this issue of prolonged reinference time, we propose ReSolution, a novel EM re-inference solution that reuses results from previous iterations and performs effective pruning to speed up model re-inference. One key observation we make is that unlike any general ML inference tasks that need to compute predictions/ scores for every testing data, EM models only need to output the pairs of entities for which the similarity between them is above a certain threshold. This suggests possibilities of optimization as we can prune away a lot of unpromising entities at an early stage and only carry on computing scores for the promising candidates.

The challenge, however, is that pruning might lead to accuracy sacrifice if it is done heuristically. To tackle this problem, ReSolution proposes the novel iterative threshold algorithm that applies threshold algorithm to each layer of the EM model to achieve accuracy guarantees. Tested on three major EM benchmark datasets, ReSolution is able to outperform complete re-inference by 16-20% with 100% accuracy. We make the following contributions:

- We develop a novel algorithm that reuses results from previous iterations and perform efficient pruning of unpromising candidates to speed up EM inference while ensuring accuracy guarantee.
- We experimentally demonstrate the feasibility of our solution over three major EM benchmark datasets.

2 BACKGROUND AND RELATED WORK

2.1 EM models

Lately, several ML-based EM solutions [1, 2, 3, 4] are proposed. These solutions mainly follow the following pattern. The dataset consists of pairs of entities that include attributes from both entities (e.g. title, conference, year for a paper entity). A pair of entity is regarded as a data point. Trained models take in a pair of entity, compute a similarity score, and predict that this pair of entities is a match if the similarity score is greater than or equal to a pre-defined threshold and a non-match otherwise.

To arrive at the predictions, the model first convert each attribute to either a context-free (e.g. FastText [7]) or a contexualized embeddings (e.g. BERT [8]). Then, the model uses the embeddings to compute the similarity score between each attribute. This step is either done via simple arithemetic operations (e.g. cosine similarity) or neural networks. The model finally applies a classifier to all attribute similarities to obtain the overall similarity score. The classifier could be as simple as linear/ logistic regression or more complicated neural networks. In this paper, we only consider models for which attribute similarities are not learned via models (i.e. persists over retrainings). Specifically, we consider DEEPER as our running framework (illustrated in Figure 1). DEEPER uses FastText to generate context-free attribute embeddings and computes both the consine distance and the normalized absolute distance as the attribute similarity score. The classifier DEEPER uses is a 7-layer neural network that consists of 4 linear layers and 3 ReLU layers. We leave models that involve learned attribute similarities to future work.

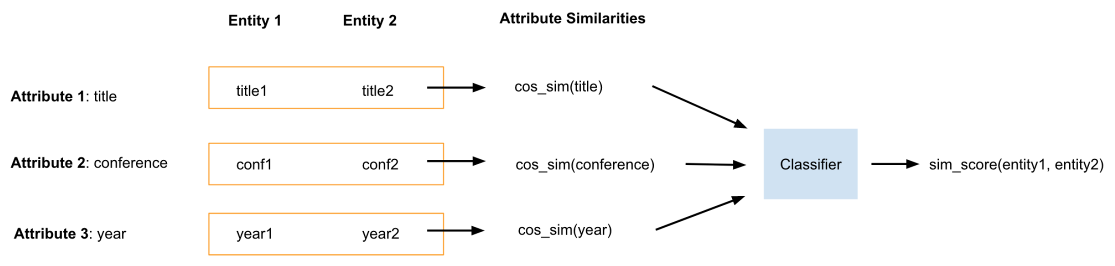

Figure 1: The flow of the DEEPER model. It first computes the attribute similarity score by converting each attribute into embeddings and use cosine similarity to calculate the distance. Then, all attribute similarities are passed into the classifier to obtain an overall similarity between the pair of entities.

2.2 Related work

ReSolution aims to reduce the re-inference time of EM ML models by reusing results from previous iterations and effectively **pruning** unpromising data points with 100% accuracy guarantees.

KRYPTON is a work that also aims to reduce the reinference time but in the context of explaining image prediction models using occlusion-based methods. KRYPTON observes that there are overlapping regions between the current occlusion and the previous ones. Thus, data associated with these overlapping regions could be memoized and reused in future reinference. Therefore, only the non-overlapping regions need to be recomputed upon reinference. However, KRYPTON performs no pruning and still needs to compute predictions for all test data. Besides memoizing results from previous iterations, ReSolution also reduce reinference time by pruning away unpromising candidates for the final predictions.

3 ARCHITECTURE

In this section, we present the main contribution of ReSolution: the iterative threshold algorithm (ITA). ITA adapts the original threshold algorithm (TA) to the setting of ML inference in order to prune away unpromising candidates. It applies TA to each basic unit (i.e. a layer) within a neural network and flips direction of stream to read from. We also introduce several techniques used to optimize ITA, including pruning of streams and usage of heaps.

3.1 Foundation

Recall in Section 2.1, the way a pair of entities is determined to be a match or not is by comparing the overall similarity score between the pair of entities with a predefined threshold. To efficiently compute the top-k entities above a certain threshold, Threshold Algorithm (TA) [9] could be employed. Essentially, given max stream (i.e. stream sorted in descending order) of each feature of a set of entities and a scoring function, TA simultaneously traverses each stream. For each seen entity, TA uses random access to fetch the values of the entity in other streams in order to compute a score (using the scoring function). At each access a, TA also uses the aggregate of values seen at time a to set a threshold. TA stops if there are k entities with scores greater than or equal to the threshold. Importantly, TA works only if the scoring function f obeys the monotonicity property. That is,

$$f(x_1, ..., x_m) \leq f(x'_1, ..., x'_m)$$

if for all i, $x_i \leq x'_i$

More generally, TA works because the entities with the highest scores are at the beginning of the streams. This ensures that as TA traverses down the sorted streams, threshold and scores can only decrease (or at least as large) in a monotone direction. This is the property P that we need to maintain.

General TA	TA in EM
entities	test data points/ a pair of entities
features	features of each layer
scoring function	weights of each layer

Table 1: Analogy between general TA and TA in EM

Under our context of ML-based EM solutions, we could treat each data point (pair of entities) as an entity and the feature of a layer as the feature of the entities. See Figure 2 for the detailed illustration.

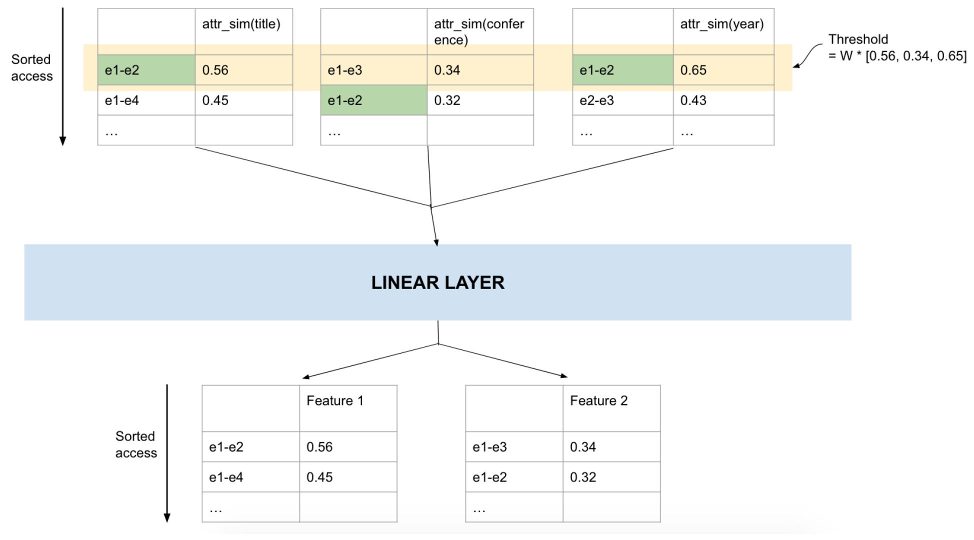

Figure 2: TA in a ML model. We read each feature in the sorted access. The orange region forms the value that are used to compute the threshold. The green boxes refer to using random access to obtain values of e1-e2 in different streams. We stop when there are k entities with scores larger than the threshold. We then compute the next set of max/ min streams using the weights in the linear layer as the scoring function. If the shape of of the weight in the linear layer is (3, 2), then the linear layer takes in inputs of 3 features and transform them to outputs of 2 features.

The challenge is that we cannot treat the entire model (i.e. the composition of all layers) as a scoring function as it may not necessarily respect property P. In our example of DEEPER, even though ReLU, matrix multiplication, and matrix addition are monotone, the fact that weights could be negative violate property P (because with negative weights, larger values translate to smaller scores).

To solve this, we propose the iterative threshold algorithm (ITA) that applies TA to each basic unit (i.e. a layer) within a neural network and flips direction of stream to read from so as to ensure that property P is hold. More details will be discussed in the following section.

3.2 Iterative Threshold Algorithm

3.2.1 Direction of streams

Because weights of a ML model is not necessarily positive, operations involving weights (e.g. matrix multiplication) can violate property P if we are always reading the max streams. For instance, if multiplied with a negative weight, the smaller the value, the larger the output. Therefore, we cannot simply use the entire model as a scoring function when computing top-k. Note that, however, we have control over the reading direction of the stream to ensure that property P holds. In other words, we can decide to read the stream in either descending order (max stream) or ascending order (min stream) based on the sign of weights. If we are trying to compute the top-k and the weight is positive, we read from max streams and check if threshold is smaller than or equal to the scores of seen entities. If the weight is negative, we instead read from min streams. This ensures that as we traverse down the stream (values gets larger), the scores are smaller. Thus, by adjusting which stream to read, property P is held.

3.2.2 Iterative application of TA

As noted above, the decision to read from max stream or min stream is based on the signs of weights. Therefore, we need to break down the model and traverse each layer in order to obtain the weights. This gives rise to an iterative nature of the problem where the max/ min streams are passed to one layer, producing max/ min streams using TA which are then fed to the next layer. We note that ITA guarantees correctness of the final outputs. This is because at each layer we decide to read from max stream or min stream according to the sign of weights. Thus, the output of each layer is correct. By induction, if the output at each layer is correct, then the output at the final layer is also correct.

In order to output the min stream, we need to compute the bottom-k. Therefore, we need to extend the traditional TA which only computes top-k. We also need to extend property P which now becomes: if computing top-k, as TA traverses down the sorted streams, threshold and scores can only decrease (or at least as large) in a monotone direction; if computing bottom-k, as TA traverses down the sorted streams, threshold and scores can only increase (or at least as small) in a monotone direction. To fulfill this (in the bottom-k context), if weight is positive, we read from min streams (so that entities with smaller scores are at the beginning of the stream). If weight is negative, we read from max streams (so that entities with larger scores at at the beginning to the stream). Then, we check if threshold is greater than or equal to the scores of seen entities.

Note that, however, TA is not by itself guaranteed to work iteratively because each output stream is produced independently, and so there is no guarantee that an entity's value is present in each stream. This is crucial because we need to obtain the scores of an entity by random accessing each stream. Therefore, at the end of each layer after all max/ min streams are produced, the set of entities present in each stream is computed and the missing entity at each stream is filled.

We summarize the points above using the following code. The algorithm ITA takes in a parameter k which specifies the maximum number of entities to find with scores above a threshold t. We first run the model on the test features until the specified layer index (more discussion in 4.3.2), pre-process the max streams and min streams by sorting the outputs of the model. Note that we can load test features because we memoized it in previous iterations (before the re-training occurs). Then, we run ITA on the max/ min streams iteratively. When there is a missing entity in a stream, we fill it in. Finally, we use the output of the last layer to compute the top-k entities above the threshold t.

```
def ita(k, t, layer_idx):
    test_features = load('storage_dir')

    for i in range(layer_idx):
        test_features = model[i](test_features)

    max_streams, min_streams = process(test_features)
```

```
# iterate through all layers
while layer_idx < len(model) do
    max_streams = ta_max(model[layer_idx].weight, k, max_streams, min_streams)
    min_streams = ta_min(model[layer_idx].weight, k, max_streams, min_streams)
    layer_idx++

    # fill in the missing values
    s = set(max_streams.entities, min_streams.entities)
    for entity in s:
        if entity is not in max_streams.entities:
            max_streams.add(entity, entity_val)
        if entity is not in min_streams.entities:
            min_streams.add(entity, entity_val)

# do a final TA that stops when threshold < t
max_streams = ta_max(score_fn, t, k, max_streams, min_streams)

return max_streams
```

3.2.3 Pruning streams

As seen in the pseudocode above, we need to do a final TA on the outputs of the model to obtain the top-k entities. However, for this last step to be done, we do not always need to compute both max and min streams for each feature in previous steps. This is because typically for the last layer of a EM ML model, some max and min streams are simply not used. Then, we can recursively backtrack to prior layers and remove the streams that will not be used in future layers, thereby saving computations and achieve further optimizations. We explain this idea using the following example. In the example, the last layer is a linear layer that outputs a matrix with two features. The first feature (denote using x_0) represents the score of being a non-match and the second feature (denote using x_1) represents the score of being a match.

In the DEEPER model, to predict whether a pair is a match or not, the maximum of x_0 and x_1 is considered. If x_1 is larger, then the pair is predicted as a match and non-match otherwise. Note that we only want to inference/ output the matches, so this problem can be converted to a threshold setting as follows. Consider the max stream of x_1 and the min stream of x_0, set the scoring function to be $x_1 - x_0$. Apply TA and stop when the threshold is below 0. This conversion follows two things: (1) property P is held because as we go down the streams, x_1 becomes smaller and x_0 becomes larger. Therefore, the scores/ threshold can only decrease. (2) we can stop at threshold = 0 because if $x_1 \geq x_0$, then $x_1 - x_0 \geq 0$. So if we have reached below a threshold of 0 we know that $x_1 < x_0$ which is a non-match. Therefore, we only need the last linear layer to output a max stream for x_1 and a min stream for x_0.

3.2.4 Heap

When using TA, we constantly need to keep track of the largest/ smallest k entities. A naive approach is to store a list of all entities seen. As TA traverses down the stream, new entities are seen and added to the list. Then, we sort the entire list in descending/ ascending order to obtain the top-k/ bottom-k entities.

Note that since we are only interested in the top-k/ bottom-k entities, a more efficient approach is to make use of a min heap/ max heap of size k. We make the following observation: the score of a seen entity remains unchanged after computed. This is because we are using random access to obtain the actual values of the entity and use those to compute the true scores instead of approximating the scores. Therefore, in a top-k setting for example, if an entity e is smaller than the current top-k entities, then it will be always be smaller than the top-k entities and thus can be ignored. In particular, e only need to be compared with the root of the min heap. If e has a smaller score, we can ignore it. Otherwise, we remove the root node and add e to the heap. Similarly, in a bottom-k setting, if an entity e is smaller than the root of the max heap, we remove the root node and add e to the heap.

Using a heap is also easier for TA to determine if the stop condition is met. Recall that TA terminates when there are k entities with score higher/ lower than the threshold in a top-k/ bottom-k setting. Therefore, we just need to compare the threshold with the root node of the min heap and see if root node is larger than the threshold. Similarly, for bottom-k, we check if the threshold is greater than the root node of the max heap.

We summarize the above points with the following pseudocode of TA (top-k version)

```
def ta_max(weight, k, max_streams, min_streams):
```

```
streams = []

if weight[i] < 0:
    streams.append(min_streams[i])
else:
    streams.append(max_streams[i])

min_heap = []

for seen_entities, threshold in traverse(streams):
    for entity in seen_entities:
        score = weight * entity

        if len(min_heap) < k:
            min_heap.add(entity)
        elif score > min_heap.root:
            min_heap.remove(root)
            min_heap.add(entity)

    if len(min_heap) >= k and threshold <= min_heap.root:
        return min_heap.sort()
```

4 EVALUATION

4.1 Experimental setup

Environment ReSolution and baselines are implemented in Python 3.7.3 and run on a MacBook Pro with a 2.7GHz Dual-Core CPU and 8GB Memory. To ensure fairness and that we are comparing performance differences on the algorithm level, highly-optimized open-source ML packages (e.g. PyTorch, numpy) are not used. Relevant matrix operations (e.g. matrix multiplcation) are instead re-coded using Python and used in both ReSolution and baselines.

Model DEEPER is used as the model for conducting the experiments. There are two version of DEEPER: the complete DEEPER that uses learned models to generate attribute similarities and the lite version that uses simple arithmetic operations to compute attribute similarities. In our setting, we use the lite version. The code can be found here.

Datasets Three widely-used benchmark datasets [10] in the field of EM are used: namely DBLP-ACM, DBLP-Scholar in the domain of scientific papers and Fodors-Zagat in the domain of restaurants. The different statistics of the datasets are summarized below.

Dataset Name	test data size	#predicted matches	#attributes
Fodors-Zagat	1460	57	12
DBLP-ACM	23230	1127	8
DBLP-Scholar	67615	2421	8

Table 2: Dataset sizes

4.2 Baselines and measurements

We study three relevant statistics to verify the accuracy and performance of ReSolution. Under our context of speeding up re-inference, for accuracy, we should compare the output of ReSolution (and baseline) with the predictions made by the model on the test data. For performance, we should compare the inference time of ReSolution (and baseline) on the test data.

Accuracy We want to justify that the use of ITA instead of the direct application of TA achieves accuracy guarantees. For the baseline, we implement the direct application of TA which simply use the entire model as the scoring function. We measure the F-1 scores of both ReSolution and the baseline with respect to the predictions on test data.

Percentage of entities seen To verify that ITA is pruning away unpromising candidates at each layer, we measure how many entities ITA computes for each stream at each layer and take the average across all streams. By dividing over the size of test data, we get the average percentage of entities seen at each layer. Consider the following baseline (BL) that runs the entire model on the test data and linearly scans through the test predictions to obtain the matches. Since this baseline always traverses through all entities, percentage of entities seen is always 100%.

Runtime We also want to know how the ITA performs in runtime compared to the baseline BL. For ReSolution, we measure the time it needs to run the model on the test data until the specified layer, prepare the max and min streams of all features, as well the time to run ITA. For BL, we measure the time it needs to run the entire model on the test data and the time to linearly scan through the test predictions to obtain the matches.

4.3 Results

4.3.1 Accuracy

As seen in the following table, using the entire model as the scoring function is hardly accurate. It verifies that we need to consider the signs of weights to ensure the accuracy of predictions.

	F-1 (ITA)	F-1 (Naive TA)
Fodors-Zagat	1	0.053
DBLP-ACM	1	0.013
DBLP-Scholar	1	0.008

Table 3: Accuracy of ReSolution compared with the baseline of directly applying TA to the entire ML model. ReSolution achieves 100% accuracy guarantee.

4.3.2 Percentage of entities seen

As seen in the following table, ReSoluion significantly reduces the number of entities that need to be traversed. This demonstrates that our solution is effectively pruning away unpromising candidates at early layers so that they won't get carried over to future layers for redundant computations. The reason for the effectiveness of ITA is that within the test predictions, the matches only constitute a relatively small portion of the entire dataset so it is easy to set apart these matching candidates from the obviously non-matching ones.

Even though the predicted matches is a small portion of the test data, we observe that at early layers, ITA is unable to easily distinguish between entities with higher scores and lower scores without having to traverse through the majority of the test dataset. That's why we set layer index to be 4 because we observe empirically that at this point the model is able to let TA easily separate apart the top-k from the rest. Setting layer any higher makes ITA having to traverse less entities, but it does not bring as significant speed boost.

	layer index = 4	layer index = 6	percentage of predicted matches
Fodors-Zagat	18.64%	8.5%	3.9%
DBLP-ACM	11.56%	7%	4.85%
DBLP-Scholar	8.5%	6%	3.58%

Table 4: Average percentage of entities computed across all streams

4.3.3 Runtime

As seen in Figure 3, for each of the three datasets, ReSolution outperforms the baseline by 16.01%, 17.99%, and 21.25% respectively. This is because with ITA, ReSolution only need to look at a much smaller portion of the dataset.

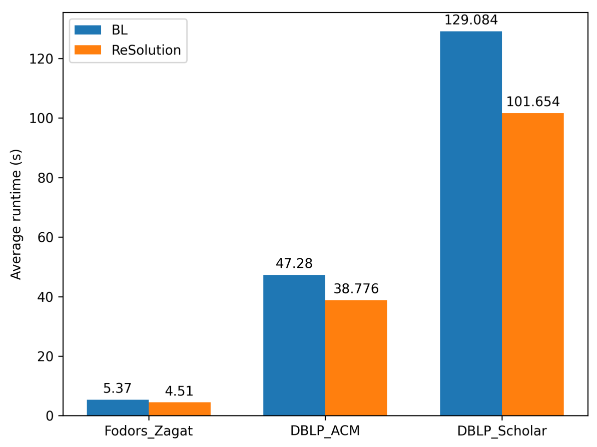

Figure 3: time needed to perform inference on the test data in seconds (ReSolution compared with the baseline approach of complete re-inference).

5 CONCLUSION AND FUTURE WORK

In this paper, we present ReSolution which is the first system that attempts at using TA to prune away unpromising candidates to speed up re-inference by 16-21%, but still with 100% accuracy guarantee. We experimentally verified the accuracy and performance of our solution compared to complete re-inference. This work as well as KRYPTON [11] demonstrate that traditional data management and data integration techniques could be employed in the settings of machine learning to achieve non-trivial performance gain.

As future work, we wish to further make use of incrementality so that we can avoid running the entire ITA to obtian the top matches. As of right now, we only use incrementality to memoize the attribute similarity scores of different pairs of entities. We also wish to explore other more efficient implementations so that we can exploit a lot of the optimizations that open-source packages (e.g. PyTorch, Numpy) are using to gain additional speed up. Finally, we also hope to extend our work to EM models with learned attribute similarities.

References

[1] Pradap Konda, Sanjib Das, AnHai Doan, Adel Ardalan, Jeffrey R Ballard, Han Li, Fatemah Panahi, Haojun Zhang, Jeff Naughton, Shishir Prasad, et al. Magellan: toward building entity matching management systems over data science stacks. *Proceedings of the VLDB Endowment*, 9(13):1581–1584, 2016.

[2] Muhammad Ebraheem, Saravanan Thirumuruganathan, Shafiq Joty, Mourad Ouzzani, and Nan Tang. Distributed representations of tuples for entity resolution. *Proceedings of the VLDB Endowment*, 11(11):1454–1467, 2018.

[3] Sidharth Mudgal, Han Li, Theodoros Rekatsinas, AnHai Doan, Youngchoon Park, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, and Vijay Raghavendra. Deep learning for entity matching: A design space exploration. In *Proceedings of the 2018 International Conference on Management of Data*, pages 19–34, 2018.

[4] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. Deep entity matching with pre-trained language models. *arXiv preprint arXiv:2004.00584*, 2020.

[5] Cheng Fu, Xianpei Han, Le Sun, Bo Chen, Wei Zhang, Suhui Wu, and Hao Kong. End-to-end multi-perspective matching for entity resolution. In *IJCAI*, 2019.

[6] Waleed Ammar, Dirk Groeneveld, Chandra Bhagavatula, Iz Beltagy, Miles Crawford, Doug Downey, Jason Dunkelberger, Ahmed Elgohary, Sergey Feldman, Vu Ha, et al. Construction of the literature graph in semantic scholar. *arXiv preprint arXiv:1805.02262*, 2018.

[7] Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hérve Jégou, and Tomas Mikolov. Fasttext. zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651*, 2016.

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[9] Ronald Fagin. Combining fuzzy information: an overview. *ACM SIGMOD Record*, 31(2):109–118, 2002.

[10] Hanna Köpcke, Andreas Thor, and Erhard Rahm. Evaluation of entity resolution approaches on real-world match problems. *Proceedings of the VLDB Endowment*, 3(1-2):484–493, 2010.

[11] Supun Nakandala, Arun Kumar, and Yannis Papakonstantinou. Incremental and approximate inference for faster occlusion-based deep cnn explanations. In *Proceedings of the 2019 International Conference on Management of Data*, pages 1589–1606, 2019.