Integration of Named Entity Extraction based on deep learning for Neo4j graph database

Lea Roj
l.roj@um.si
Faculty of Electrical
Engineering and Computer Science,
University of Maribor
Koroška cesta 46
SI-2000 Maribor, Slovenia

Štefan Kohek
stefan.kohek@um.si
Faculty of Electrical
Engineering and Computer Science,
University of Maribor
Koroška cesta 46
SI-2000 Maribor, Slovenia

Aleksander Pur
pur.aleksander@gmail.com
Ministry of the Interior,
Štefanova ulica 2
SI-1000 Ljubljana, Slovenia

Niko Lukač
niko.lukac@um.si
Faculty of Electrical
Engineering and Computer Science,
University of Maribor
Koroška cesta 46
SI-2000 Maribor, Slovenia

Abstract

The increase in unstructured textual data has created a pressing demand for effective information extraction techniques. This paper explores the integration of Named Entity Extraction (NEE) using deep learning within the Neo4j graph database. Utilizing the Rebel Large Model, we converted raw text into structured knowledge graphs. The primary objective is to evaluate the efficacy of this integration by examining performance metrics, such as processing time, graph growth, and entity representation. The findings highlight how the structure and complexity of graphs vary with different text lengths, offering insights into the potential of combining deep learning-based NEE with graph databases for improved data analysis and decision-making.

Keywords

Named Entity Extraction, Deep Learning, Neo4j, Graph Database

1 Introduction

The rise of digital news and social media has significantly increased the importance of NEE. As more information is generated online, extracting this information became critical for various applications, such as search engines and recommendation systems [1]. NEE, along with Relation Extraction (RE), is essential for transforming unstructured text into structured data, enabling more effective data analysis and decision-making. Building on the findings of a previous work [2] that evaluated various hyper-parameters and analyzed sensitivity performance, this paper takes a step further by exploring the integration of NEE and RE within the Neo4j graph database.

Neo4j is a graph database that provides a powerful way to store and query complex relationships between entities. This makes it well-suited for applications involving interconnected data, such as social networks, recommendation systems, and fraud detection. In Neo4j, data is stored as nodes and relationships. Nodes represent entities, while relationships represent the connections between these entities. Both can have properties (key-value pairs) to store additional information. Neo4j uses Cypher, a declarative query language specifically designed for querying graph databases [3].

Integrating Named Entity Recognition (NER) based on deep learning within Neo4j graph databases has been an area of active research and development. Ni et al. [4] addresses the challenge of translating natural language queries into graph database queries for intelligent medical consultation systems. The authors developed a Text-to-GraphQL model that utilizes a language model with a pre-trained Adapter, enhancing the semantic parsing capabilities by linking GraphQL schemas with corresponding natural language utterances.

Fan et al. [5] wrote about geological hazards, a deep learning-based NER model that was used to construct a knowledge graph from literature. This model addresses challenges such as diverse entity forms, semantic ambiguity, and contextual uncertainty. The resulting knowledge graph, stored in Neo4j, enhances the usability of geological research data.

Chaudhary et al. [6] propose a system that converts raw text into a knowledge graph using Neo4j, addressing inefficiencies in traditional tools like Spacy, NLTK, and Flair. The method combines entity linkage and relation extraction to convert unstructured data into a knowledge graph. It leverages graph-based NER and Linking for a contextual understanding of data. The implementation uses the REBEL [7] model for relation extraction and the BLINK [8] model for entity disambiguation, demonstrating significant improvements in handling large, untagged datasets compared to these traditional tools.

The objective of this paper is to demonstrate the implementation process of integrating NEE into the Neo4j graph database. It aims to evaluate the effectiveness of this integration and analyze various performance metrics. Specifically, the paper will measure the processing time required to extract named entities from text and represent them in Neo4j, analyze graph growth in relation to text length, evaluate average total neighbors score based on text

length, and analyze how many entities are actually shown in graph and how many are filtered out.

The next section details the workflow from text pre-processing to graph visualization in Neo4j. The Results showcases the findings, including charts that visualize the performance metrics. Finally, the Conclusion summarizes the benefits and purpose of the integration, highlights key findings from the study, and suggests potential areas for future research and development.

2 Methodology

The workflow from text pre-processing till graph construction in Neo4j is represented in figure 1. The entire process consists of multiple crucial steps including text pre-processing, NEE, RE, entity normalization and filtration, and finally generation and visualization of the knowledge graph in Neo4j. The details of these steps have been in depth discussed in our previous paper [2].

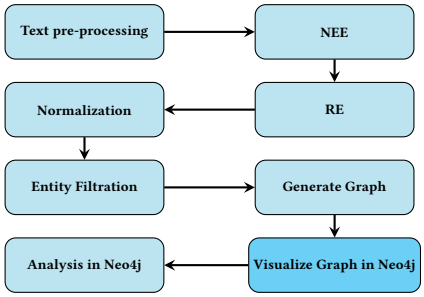

Figure 1: Workflow

Text pre-processing involves segmenting the text into manageable spans, with span length defining the number of words in each segment and overlap length ensuring coherence between consecutive spans. The length penalty manages the impact of longer sequences, while the number of beams allows simultaneous exploration of multiple sequences to find the best one. The number of returns specifies how many sequences are returned after the beam search.

NEE identifies and classifies entities by tokenizing the text, with each token corresponding to a unique word ID. The Rebel Large Model [7] then generates triplets (head, tail, and relationship type) from the text. For RE, the tokenized input generates predictions, which are decoded into text representations. Extracted relations are validated against the original text and, once confirmed, are added to the knowledge base.

Based on the previous paper, this paper proposes improvements in normalization and entity filtration, as follows. Entity names are first standardized by converting text to lowercase and removing common prefixes, followed by verification via Wikipedia's API. Non-contributive entities, such as dates or overly generic terms, are identified and excluded using pattern recognition and categorization techniques. The system checks for duplicates or highly similar entities to prevent clutter, merging or discarding them as needed. Cosine similarity measures are used to assess and reinforce thematic links between entities, enhancing the overall coherence of the knowledge base.

2.1 Knowledge graph within Neo4j

A knowledge graph is generated from the extracted and filtered entities, and relations. This structured representation helps in visualizing the connections and relationships within the text. Finally, the knowledge graph is stored and visualized in Neo4j.

In the integration process, the data obtained using NER is saved to a graph database through the Neo4j driver. Afterwards, the method iterates over entities in the knowledge base to determine category for each entity. Using the 'MERGE' Cypher command, as shown in Figure 2, the method either finds an existing node (based on the name) or creates a new node if none exists. Attributes such as 'url,' 'summary,' and 'category' are then added to each node.

```
MERGE (e:Entity {name: $entity})
ON CREATE SET e.url = $url, e.summary = $summary
SET e.category = coalesce(e.category, $category)
```

Figure 2: Cypher query to add entities.

After adding the entities, we processed each relationship defined in the knowledge base. We ensured that both entities involved in the relationship were present in the database and then created a relationship between the entities using the 'MERGE' command, as shown in figure 3, if it didn't already exist.

```
MATCH   (head:Entity {name: 'EntityName1'}),
(tail:Entity {name: 'EntityName2'})
MERGE (head)-[r:RELATIONSHIP_TYPE]->(tail)
```

Figure 3: Cypher query to add relations.

The knowledge graph is visualized in Neo4j to provide an intuitive and interactive representation of the extracted knowledge. Using Cypher queries, users can explore the graph, examine relationships, and derive insights from the interconnected data. To display the graph in the Neo4j application, we use the Cypher query in figure 4.

```
MATCH (n)-[r]->(m) RETURN n, r, m
```

Figure 4: Cypher query for graph visualization.

This query retrieves all nodes (n, m) and the relationships (r) between them, displaying the graph structure in the Neo4j interface. The directed edges in the graph illustrate the relationships, providing a clear visual representation of the underlying knowledge.

2.2 Knowledge Graph Analysis in Neo4j

After constructing the knowledge graph in Neo4j, various metrics and analyses were applied to explore the structure within the graph. One such metric is the Total Neighbors score, which measures the closeness of nodes by counting their unique neighbors. It is based on the idea that a highly connected node is more likely to gain new links. The Total Neighbors metric is calculated using the following formula:

$$TN(x, y) = |N(x) \cup N(y)|, \tag{1}$$

where $N(x)$ and $N(y)$ represent the sets of nodes adjacent to x and y, respectively. The Total Neighbors score measures the closeness of two nodes based on the number of unique neighbors they have. If a score is equal to 0 it indicates no closeness between the nodes, while higher scores indicate greater closeness [9].

The `gds.alpha.linkprediction.totalNeighbors` function from the Neo4j Graph Data Science (GDS) library calculates the total neighbors score between the two matched nodes (p1 and p2).

3 Results

The analysis was conducted using the text about Pablo Escobar from Wikipedia. For this paper, the original text was divided into sections of varying lengths to examine how text length influences the analysis results. Figure 5 displays a generated graph with a text length of 304 words. The graph was created using specific parameters that influence its structure and content. These parameters were heuristically determined to be span length = 30, length penalty = 0, number of beams = 5, number of returns = 2, and overlap length = 10. On the same set of parameters we measured processing time, similarity score based on total neighbors, and analyzed graph growth.

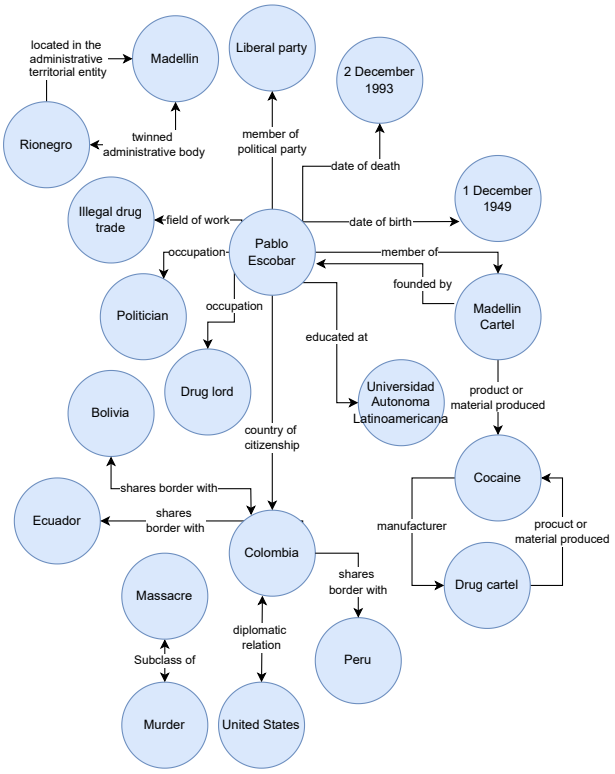

Figure 5: Generated graph with text length = 304 words

In Figure 6, the time required for graph generation is shown to increase linearly with the number of words in the text. This linearity is confirmed by a regression analysis, which yields an R^2 value of 0.9984, indicating an almost perfect fit.

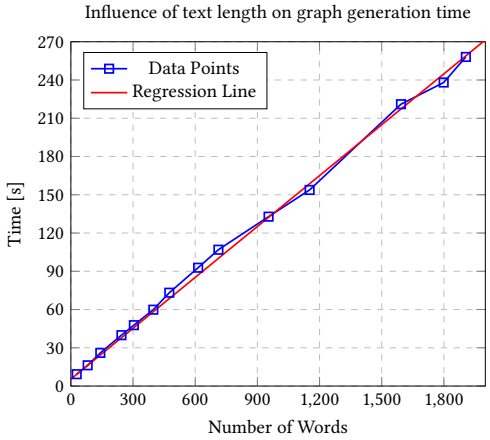

Figure 6: Influence of text length on execution time.

Figure 7 demonstrates the influence of text length on the number of recognized entities. As the number of words in the text increases, there is a corresponding increase in both the number of nodes shown and the number of entities that are recognized but not displayed. This pattern indicates that longer texts result in the recognition of more entities, although not all are displayed. The decision to display or exclude entities is determined by several processes designed to maintain the clarity and relevance of the graph. These processes include the combination and unification of similar entities, the removal of isolated entities, and the filtering out of date-related entities. These processes are essential for maintaining the graph's relevance and clarity, preventing clutter from redundant or less significant entities.

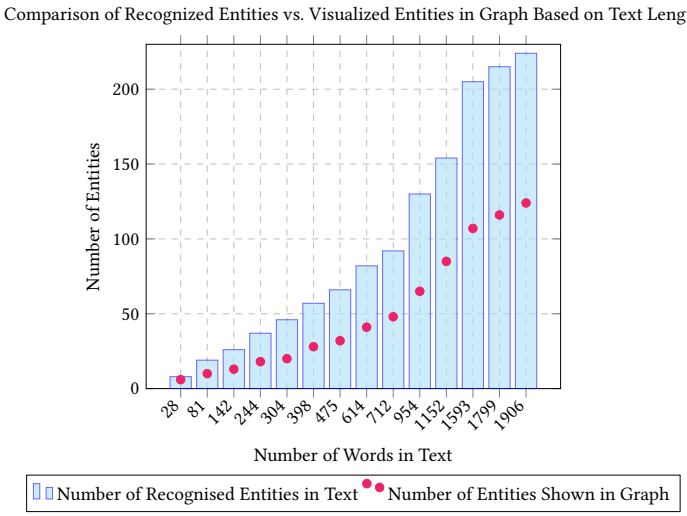

Figure 7: Comparison of recognized and visualized entities based on text length.

The following analyses represent the average similarity score for all possible pairs of entities (n1, n2) in the graph. For that is used a link prediction algorithm (totalNeighbors) to assess how connected two entities are based on their shared neighbors. The

purpose of this is to measure how interconnected the entities are throughout the entire graph. High scores generally appear between nodes directly related through historical, contextual, or thematic associations. On the other hand dates provide low similarity scores with entities, likely indicating less direct connection or relevance to these specific dates in the dataset.

In Figure 8, the average Total Neighbors score between all nodes is represented. The values are relatively stable, mostly ranging between 1.5 and 2.0. This suggests a moderate level of similarity between entities across different text lengths, without extreme variation. This stability suggests that the entities within each text maintain a consistent level of connectivity, regardless of text length.

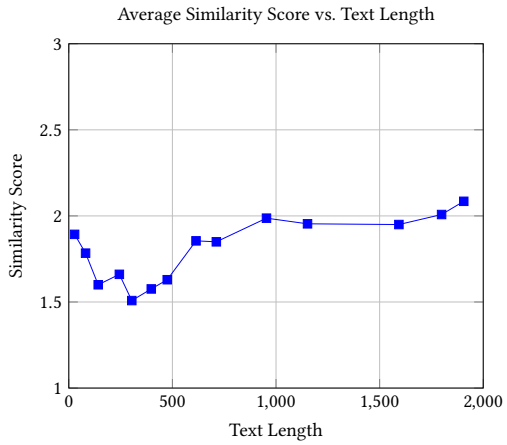

Figure 8: Average Total Neighbors Score between all nodes

Figure 9 highlights only ten strongest connections in the graph, which is particularly useful for identifying the most significant or central entities. Score increases with text length, particularly noticeable in texts longer than 900 characters. This indicates that longer texts tend to have more instances of highly interconnected nodes. This is due to the increased probability of recurring entities in longer texts, which leads to more common neighbors.
The text with the shortest length (28 characters) has the lowest similarity score (3.2). This suggest that very short texts lack sufficient content to establish strong connections between entities.
The highest scores for both average and top ten similarities occur in the longest texts (1593, 1799, 1906 characters). This supports the idea that more extensive content provides more opportunities for entities to connect or relate.

4 Conclusion

The results demonstrate that text length significantly impacts the performance and outcomes of NEE within Neo4j using deep learning techniques. As text length increases, so does the processing time for graph visualization, due to the need to extract and manage a larger number of entities and relationships. Moreover, the analysis of graph structure revealed that longer texts tend to produce more nodes, both displayed and recognized but not shown. This suggests that while longer texts provide more data, they also introduce challenges in managing graph complexity, which complicates graph management and requires the consolidation of similar entities and

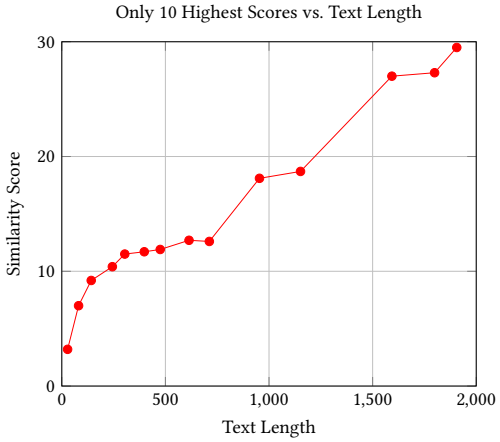

Figure 9: Average of ten highest Total Neighbors Scores

filtering of less relevant ones to maintain clarity. Furthermore the stability of the average similarity scores across various text lengths suggests a consistent level of connectivity among entities, with a noticeable increase in the strength of connections in longer texts. This supports the hypothesis that longer texts offer more opportunities for entity interconnections, which is crucial for tasks requiring comprehensive data analysis and decision-making.

In conclusion, integrating NEE with graph databases presents a promising approach for transforming unstructured text into structured knowledge. However, the complexity introduced by varying text lengths must be carefully managed to optimize both the performance and the utility of the resulting knowledge graphs. Future work could focus on exploring other NEE methods to further enhance the efficacy of this integration.

Acknowledgments

The authors acknowledge the support of the STALITA project, financed by Ministry of the Interior, Slovenia.

References

[1] Ing Michal Konkol. Named entity recognition. *Pilsen: PhD thesis, University of West Bohemia*, 2015.
[2] Lea Roj, Štefan Kohek, Aleksander Pur, and Niko Lukač. Sensitivity analysis of named entity extraction based on deep learning.
[3] Renzo Angles, Marcelo Arenas, Pablo Barceló, Aidan Hogan, Juan Reutter, and Domagoj Vrgoč. Foundations of modern query languages for graph databases. *ACM Comput. Surv.*, 50(5), sep 2017.
[4] Pin Ni, Ramin Okhrati, Steven Guan, and Victor Chang. Knowledge graph and deep learning-based text-to-graphql model for intelligent medical consultation chatbot. *Information Systems Frontiers*, 26(1):137–156, 2024.
[5] Runyu Fan, Lizhe Wang, Jining Yan, Weijing Song, Yingqian Zhu, and Xiaodao Chen. Deep learning-based named entity recognition and knowledge graph construction for geological hazards. *ISPRS International Journal of Geo-Information*, 9(1), 2020.
[6] Shikha Chaudhary, Hirenkumar Vyas, Naveen Arora, and Sejal D'Mello. Graph-based named entity information retrieval from news articles using neo4j. In *2024 11th International Conference on Computing for Sustainable Global Development (INDIACom)*, pages 320–324, 2024.
[7] Pere-Lluís Huguet Cabot and Roberto Navigli. Rebel: Relation extraction by end-to-end language generation. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 2370–2381, 2021.
[8] Martin Josifoski Sebastian Riedel Luke Zettlemoyer Ledell Wu, Fabio Petroni. Zero-shot entity linking with dense entity retrieval. In *EMNLP*, 2020.
[9] Neo4j. *Total Neighbors Algorithm*, 2024. Accessed: 2024-06-25.