Model-Based Knowledge Searching

Maxim Bragilovski(✉), Yifat Makias, Moran Shamshila, Roni Stern, and Arnon Sturm

Ben-Gurion University of the Negev, Beer Sheva, Israel
{maximbr,makias,moranshi}@post.bgu.ac.il, {sternron,sturm}@bgu.ac.il

Abstract. As knowledge increases tremendously each and every day, there is a need for means to manage and organize it, so as to utilize it when needed. For example, for finding solutions to technical/engineering problems. An alternative for achieving this goal is through knowledge mapping that aims at indexing the knowledge. Nevertheless, searching for knowledge in such maps is still a challenge. In this paper, we propose an algorithm for knowledge searching over maps created by ME-MAP, a mapping approach we developed. The algorithm is a greedy one that aims at maximizing the similarity between a query and existing knowledge encapsulated in ME-maps. We evaluate the efficiency of the algorithm in comparison to an expert judgment. The evaluation indicates that the algorithm achieved high performance within a bounded time. Though additional examination is required, the sought algorithm can be easily adapted to other modeling languages for searching models.

Keywords: Conceptual modeling · Matching · Searching

1 Introduction

Knowledge, especially in technology and engineering domains, is developing at a tremendous pace [5]. In such domains, we are especially concerned with know-how, the kind of knowledge that guides action towards practical objectives, prescribing solutions to technical problems, and evaluating alternative solutions [4,15]. Know-how management can provide various benefits such as: domain review, trade-off analysis, and support for decision making. While there is almost instant access to published know-how online, to the best of our knowledge, there has been little advance in how a body of know-how is organized for easier access, in particular, for searching the knowledge. Nowadays, searching for knowledge is mostly done textually using search engines. Yet, the outcomes of such search results are of limited accuracy, that is, many non-relevant answers are being retrieved leading to low efficiency of the search. To address this limitation, we suggest taking advantage of the underlying structure of the knowledge under examination. In the case of know-how, the underlying structure resembles the means-end relationship. For that purpose, we devised a knowledge scheme called ME-MAP [17] that uses the means-end relationship to map out know-how. Using

A. Ghose et al. (Eds.): ER 2021, LNCS 13011, pp. 242–256, 2021.
https://doi.org/10.1007/978-3-030-89022-3_20

that scheme, we are structuring and organizing the knowledge and are able to query it more precisely. The population of the scheme can be regarded as a graph that can now be searched using various techniques. In this paper, we adapt an algorithm for searching ME-maps and explore its effectiveness [2]. We also evaluated the performance of the algorithm by various information retrieval metrics and found the result promising.

The paper is structured as follows. Section 2 provides a background of the ME-MAP approach. Section 3 refers to search problems in graphs. Section 4 introduces the sought algorithm and Sect. 5 presents its evaluation. Finally, Sect. 6 concludes and sets plans for future research.

2 ME-MAP in a Nutshell

The ME-MAP approach draws its core concepts from the goal-oriented requirement engineering (GORE) paradigm [20]. It further leverages on concept maps [11] and specializes its nodes and link types. A main objective of the ME-MAP is to take advantage of the inherent means-ends relationship that characterizes know-how, where the identification of problems and development of feasible solutions is a key concern. The approach aims at presenting problems identified in a domain, properties of problems that are particularly relevant in the domain, and offer a systematic review of proposed solutions. In the following, we briefly introduce the ME-MAP language elements.

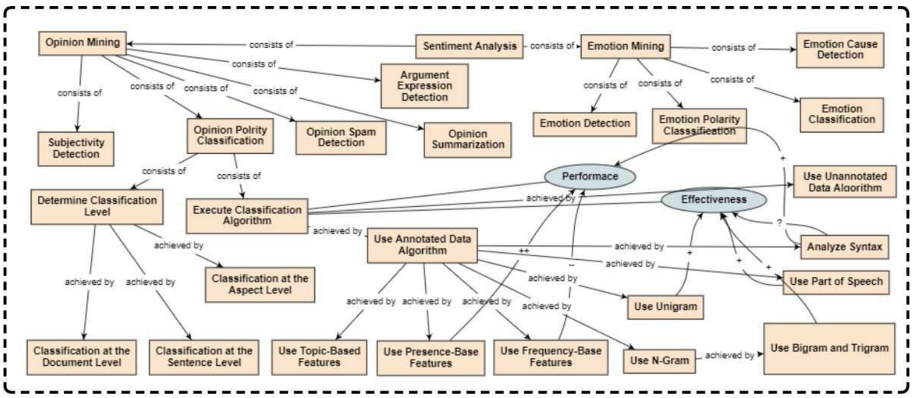


Fig. 1. A partial ME-MAP of the Sentiment Analysis domain

– A **Task** is a key element in the means-ends hierarchy. A Task can be interpreted either as a problem or as a solution depending on its location within the hierarchy. Tasks are graphically depicted as rectangular shapes. In Fig. 1, the task *Execute Classification Algorithm* is a problem that should be addressed,

whereas the task *Use Annotated Data Algorithm* is one alternative solution for that problem. When referring to the latter it is now a problem to be addressed. One alternative solution is the *Use Topic-Based Features*. Tasks are usually named as an action (verbal sentence), other alternatives exist such as placing a known technique (i.e., noun) with its name, implying the usage of the technique.
- A **Quality** expresses an attribute or a property that is desired for a task. In the ME-MAP a quality is depicted as an ellipse. For example, in Fig. 1, *Performance* and *Effectiveness* are qualities, which further characterize how a solution that supports the *Execute Classification Algorithm* task should be designed. A quality is therefore associated with a task. A quality can also be linked to another quality independently of a task.

Link elements connect tasks and qualities. In the following we elaborate on the link types included in ME-MAP:

- The **achieved by** link (a line arrow labeled by "achieved by") represents the means-ends relationship. The arrow points from the "end" to the "means". For example, in Fig. 1, *Use Topic-based Feature* is a means to accomplish Use *Annotated Data Algorithm*. The link indicates an alternative for achieving an end.
- The **consists of** link (a line arrow labeled by "consists of") indicates that a task has several sub-parts, all of which should be accomplished for the parent task to be accomplished. Note that this link does not refer to the ordering or dependencies of performing the task. For example, in Fig. 1, in order to use *Opinion Polarity Classification*, there is a need for both *Determine Classification Level* and to *Execute Classification Algorithm*.
- The **association link** (an unlabeled and non-directional link) indicates the desirable qualities for a given task.
- The **contribution link** (a curved arrow labeled with a contribution level) indicates a contribution towards a quality. Contributions can originate from a task or a quality and are always directed to a quality. The contribution ranges from strong negative (−−) to strong positive (++). For example, in Fig. 1, *Use Part of Speech* positively contributes to the *Effectiveness* quality.

For further explanations on the ME-MAP we refer the reader to [17].

3 Related Work

One of the challenges in searching knowledge refers to how to store the mined knowledge. Different studies suggest various structural databases that store knowledge from natural language text that might be ambiguous, contextual, and implicit [3]. In our research we use knowledge graph structure, yet, the structure also refers to the actual knowledge semantics. As we explicitly refer to know-how, we use the means-ends notion to store the knowledge [17].

Query knowledge graphs can be classified into four main categories: (1) graph-based query [18], (2) keyword-based query [6], (3) natural-language-based query

[7,23], and (4) path query language queries [13]. Most of these methods (2–4) transform the query to graph and then perform graph searching [18]. Searching the graph is based on checking the alignment between the query and the graph based on the node similarity and structure similarity.

Three different main path query languages described in [1] that are SPARQL, Cypher, and Gremlin for querying a graph. SPARQL standard structural query language to access Resource Description Framework (RDF) data and Cypher a declarative language for querying property graphs are based on SQL language while Gremlin, a query language of the Apache TinkerPop3 graph Framework, is more similar to functional language. These query languages are based on the assumption that users have prior knowledge about the graph that they search in. However, it is impractical for users to write statements in these languages due to the complexity of syntax and the lack of prior knowledge [6]. In contrast to these languages, keywords search [6] and natural language questions provide users with an interface for querying RDF data. However, they are still facing problems. Keyword-based search lack due to the ambiguity presented either by the keywords or their orders. Natural languages based search lacks due to the challenge of transforming these into formal queries (which is an NP-hard problem) [23].

The similarity flooding algorithm [9] also helps to find pairs of elements that are similar in two data schemes or two data instances. Yet, in this work we are interested in paths considering also the labels of the vertices and edges.

Similar to our work, Wang et al. [18] present an algorithm that finds subgraphs in knowledge graphs given a query graph. They divided the task into several phases: knowledge graph embedding, semantic graph generation, and A* semantic search that is based on a defined path semantic similarity. They further attempt to optimize response time. They experiment with the effectiveness and efficiency and got a recall of 0.69 and the answering time took 136.81 ms for top 200 answers in DPpedia[1]. In our work, the proposed algorithm is inspired by their semantic search algorithm. What differentiates it, is our use of a semantic search that also includes graph semantics in addition to using words semantics, and that we adopted the greedy approach.

4 The Search Algorithm

The problem we are aiming to address is the search within ME-maps. For that purpose, we devised a Greedy Search in Means-End Maps (GSME) algorithm that addresses the concerns of complexity and semantics. GSME adopts the similarity considerations appears in [14] and refers to label matching (exact match among labels), structure matching (in terms of links), semantic matching (in terms of labels semantic similarity and links' semantics, e.g., synonyms and related concepts), and type matching. In the following, we first set the ground for the algorithm and then elaborate on its design and execution.

[1] https://wiki.dbpedia.org/.

Definition 1. *A **ME-MAP** is a graph (G) consists of vertices (V) and edges (E). G = (V, E).*

$$\boldsymbol{V = Task \cup Quality},$$
$$\boldsymbol{E = AchievedBy \cup ConsistsOf \cup Association \cup Contribution}$$

Definition 2. *sim_w refers to the similarity of two words. This can be calculated using for example WordNet [10], or Sematch [22].*

Definition 3. *sim_l refers to the similarity of two vertices labels. We suggest achieving this task by two alternatives:*

1. Using words similarity:

$$sim_l(list_1, list_2) = \frac{\Sigma_{w_1 \epsilon list_1}(\arg max_{w_2 \epsilon list_2}(sim_w(\mathbf{w}_1, \mathbf{w}_2)))}{\arg\max(length(list_1), length(list_2))} \quad (1)$$

   Where the lists order the words from the vertex's label.
2. Using sentence similarity: *We transform the labels into vectors and measure the similarity using sbert*[2].

Definition 4. *sim_t refers to a predefined similarity among vertex types. In the case of the ME-MAP, based on our exploration we set sim_t(Task, Quality) = 0.5.*

Definition 5. *sim_v refers to the similarity of two vertices.*

$$sim_v(v_1, v_2) = \frac{\mathrm{sim}_l(v_1, v_2) \ + sim_t(v_1, \ v_2)}{2} \quad (2)$$

sim_l can be either defined by Definition 3(1) or by Definition 3(2).

Definition 6. *sim_e refers to a predefined similarity among edge types. Table 1 presents the similarity among ME-MAP edges. These similarities were determined by ME-MAP experts and represent the semantic similarity between types of edges.*

Table 1. Edge similarity

	Achieved by	Consists of	Association	Contribution
Achieved by	1	0.5	0.25	0.25
Consists of	0.5	1	0.25	0.25
Association	0.25	0.25	1	0.25
Contribution	0.25	0.25	0.25	1

Definition 7. *Node-edge similarity is calculated as follow:*

$$sim_{ne}(v_1, e_1, v_2, e_2) = \frac{2 * \mathrm{sim}_v(v_1, v_2) \ + sim_e(e_1, \ e_2)}{3} \quad (3)$$

[2] https://www.sbert.net/.

This definition captures the similarity of a vertex along with its incoming edge. We considered the vertex similarity of double importance as it is used as the basis for the similarity.

Definition 8. *A query Q is represented as a ME-map.* ***Single path*** *query has a single path of vertices and edges (see Fig. 2a).* ***Multi paths*** *query has multiple paths (see Fig. 2b).*

(a) Single path (b) Multi paths

Fig. 2. Multi paths and Single path queries

Definition 9. *To answer a query, we actually compare the similarity of the paths of the query with those of the map. The following formula is used for that purpose (NodeEdgeSim is a set of* sim_{ne}*).*

$$Path\ Sim\left(G^1, G^2, NodeEdgeSim\right) = sim_v\left(v_1^{G^1}, v_1^{G^2}\right) + \sum_{i=1}^{|NodeEdgeSim|} NodeEdgeSim_i$$

This formula actually sums all the node-edge similarities between paths and therefore we used it as an estimator for path similarity measurement.

In the following, we elaborate the algorithm for single path queries. We differentiate between two types of node similarity. The first relies solely on the node information, we call this Node-based similarity. The second also refers to the edge connecting the node from the previous one, we call this node-edge based similarity.

- The function of **Node-based similarity** returns the most similar node in the ME-map to a node within the query as calculated by Definition 5.
  Input: $G = (V, E)$, query node, similarity function. The similarity function is the one appears in Definition 3.
  Output: a node from G.
- The function of **Node-edge based similarity** is responsible for retrieving up to K most similar neighbors' nodes that exceed a predefined threshold as determined by Definition 7. If there is no single node that its similarity exceeds the threshold, the function returns all the neighbor's nodes with similarity of the threshold.
  Input: G, query node, parent node, similarity function, threshold, K
  Output: node list and their similarities from G.

Algorithm 1. GSME

Input $similarNodes, knowledgeGraph, queryGraph, K, threshold, queryID, isVisited$
Output Set of sub graphs with their similarities

1: $results \leftarrow \{\theta\}$
2: **if** $similarNodes \equiv \{\theta\}$ || $v_{queryID} \notin V_{queryGraph}$ **then**
3: $return\ results$
4: **end if**
5: **for** $node$ in $similarNodes$ **do**
6: **if** $node \in isVisited$ **then**
7: $continue$
8: **end if**
9: $isVisited \leftarrow isVisited \cup node$
10: $similarNodes \leftarrow Node - edge\_based\_function(knowledgeGraph, queryGraph[queryID], node, K, threshold)$
11: **if** $similarNodes \equiv \{\theta\}$ || $arg\ max_{sim \in similarNodes}(sim) \leq threshold$ **then** $results^{'} \leftarrow GSME(similarNodes, knowledgeGraph, queryGraph, K, th, queryID, isVisite)$
12: **else**
13: $results^{'} \leftarrow GSME(similarNodes, knowledgeGraph, queryGraph, K, threshold, queryID++, isVisite)$
14: **end if**
15: $results \leftarrow results^{'} \cup results$
16: **end for**
17: $return\ results$

To start the execution of the GSME, the number of nodes (K) the algorithm handles in each round (that refers to the next step of the query) and the threshold for the similarity need to be determined. The algorithm uses the Node-edge based similarity function to find the appropriate list of nodes for its execution in each iteration. We also have to determine the knowledge graph(s) in which the search should take place and *similarNodes* where the search should start (this task can be achieved by the Node-based similarity function). For the query, we have the *queryGraph* that needs to be found in the (set of) *knowledgeGraph*. Additional required parameters include the *queryID* that refers to the id of the node in a *queryGraph* and is initialized with '−1' that represents the first node of the query where we want to start the search from. *isVisited* is a list that contains all the nodes that GSME already iterates over in the *knowledgeGraph* to allow GSME to handle graphs with cycles. The output of the algorithm is a set of sub-graphs from the *knowledgeGraph* that are the most similar to the *queryGraph*. The algorithm is described in Algorithm 1.

The GSME algorithm steps are as following: (line 1) Initialize results with an empty set; (lines 2–4) Stop GSME if there are no nodes in the *similarNodes* or there are no nodes with a given ID in the *queryGraph*; (lines 6–8) skip already visited nodes; (line 9) For each node in *similarNodes* we add it to *isVisited* list; (line 10) Get up to K most *similarNodes* by the Next-edge based function

described above; (lines 11–12) If there are no nodes in $similarNodes$ or the similarity of the nodes in the $similarNodes$ is below the threshold the algorithm will run recursively for the selected $similarNodes$ and $queryID$, without moving on the next node; (lines 12–14) If there are nodes in $similarNodes$ and the similarity of the nodes in the $similarNodes$ are above the threshold - the algorithm will run recursively for the selected $similarNodes$ (up to K nodes) and $queryID$, each time moving on to the next node – in other words, in order to allow the return of results that are transitive, the node in the query is promoted only when the similarity obtained for the node is higher than the threshold; (line 15) Merge the results that the algorithm retrieved with the results within the function arguments. The algorithm continues until it passes all the nodes and edges of the query. At the end of the algorithm, sub-graphs from the ME-maps that answer the query will be obtained. The sub-graphs will be sorted by their similarity ranking, as determined by Definition 9 and normalized by the length. The ranking is a weighing of the similarity of the edges and nodes and is represented by a number 0-1.

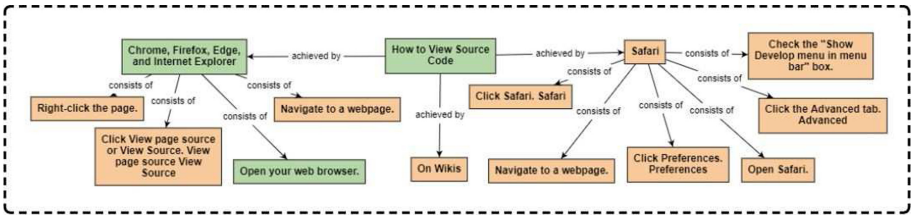


Fig. 3. GSME execution example

In the following, we demonstrate the execution of GSME over the ME-map shown in Fig. 3 whereas the expected output is marked in green with the requested query shown in Fig. 2a. We set the threshold to 0.65 and the number of lookup nodes (K) to 2. These parameters were determined based on a pre-execution evaluation. We also did a sensitivity analysis on the threshold ('th') and found that the value of 0.65 leads to the most accurate results.

1. In the first stage, the Node-based similarity function is executed. The following results demonstrate the output for the node "View Source Code" in the query graph:

   ```
   For the query node: 'View Source Code'
   [-1]: sim('How to View Source Code','View Source Code') = 0.526
   [-3]: sim('Open your web browser.','View Source Code') = 0.513
   ```

   (For all other nodes in the graph, the similarity was below 0.513)
2. Following the highest similarity of 0.526, GSME starts from the node 'How to View Source Code' (id: '−1'). In this stage, the algorithm keeps trying to

find the K (K = 2 in our case) most similar adjacent nodes to the selected node in the knowledge graph to the next node of the query graph, that is, 'web browser'. The results were the following:

```
For the query node: ['web browser']
[-2]: sim('Chrome, Firefox, Edge, and Internet Explorer',
'web browser') = 0.512
[-7]: sim('Safari','web browser') = 0.513
[-14]: sim('On Wikis','web browser') = 0.511
The most similar node is: [(0.6, -2), (0.6, -7), (0.6, -14)]
```

3. Since all the nodes sim_{ne} value (Definition 7) is below the threshold, all the nodes advance recursively to the next step without advancing to the next query node. We will only demonstrate the iteration over the node with id '−2' in this step, and the iteration over node '−14' in step number 4(b), despite the fact that the algorithm does a similar iteration also over the node with id '−7'.

```
For the query node: ['web browser']
[-3]: sim('Open your web browser','web browser') = 0.682
[-4]: sim('Navigate to a webpage.','web browser') = 0.678
(other neighbors nodes  below the 0.678 similarity)
The most similar nodes are: [(0.682, -3), (0.678, -4)]
```

4. The GSME algorithm terminates only when one of the following scenarios occur:
   (a) GSME finishes iterating over the nodes in the *queryGraph* therefore, GSME finds at least one sub-graph among all the potential sub-graphs like in step 3.
   (b) There are no more nodes to iterate in the *knowledgeGraph*. For example, in one of the steps of the GSME, it reaches node '−14':

   ```
   For the query node: ['web browser']
   The neighbors of vertex: -14 are: []
   The most similar node is: []
   ```

   therefore, all the sub-graphs that end with node '-14' are irrelevant answers.
5. Finally, the results paths are calculated for their similarity based on Definition 9 and are sorted accordingly.

```
1 : (path: '-1,-2,-3', Path-Sim: 0.603, map id: 4796)
2 : (path: '-1,-7,-4', Path-Sim: 0.601, map id: 4796)
```

Indeed, the first row in the results appears in Fig. 3. Its path similarity is calculated by Definition 9 as follow: $\frac{0.526+0.6+0.682}{3} = 0.603$.

5 Evaluation

To evaluate GSME, we conducted three experiments to check the algorithm performance in various settings. This section will be focusing on the technical aspects of GSME rather than the benefits of using GSME.

5.1 Settings

We considered several maps from the Human Know-How Dataset [12] for the first experiment. For the second experiment, we considered three domains for which we developed ME-maps: (1) a simple map referring to the search domain. (2) a DPSL map that (partially) maps out [8]. (3) a Sentiment Analysis map that (partially) maps out [19]. For the domains from the Human Know-How Dataset, we performed transformations from their graph format to the ME-MAP representation. These maps contain only tasks and two link types: 'Achieved By' and 'Consists of' unlike the maps in the second experiment that have all the types of edges and vertices. The transformations rules were the following rules:

1. 'MainTask', 'Steps', 'Methods' and 'Parts' nodes were transformed to a 'Task' node.
2. The label of the nodes was set according to the name of the attribute of the JSON object.
3. An edge from 'MainTask' to 'Methods' is labeled with 'Achieved By' label.
4. All the other edges were labeled with 'Consists of' label.

The purpose of the first experiment was to examine the algorithm's ability to handle maps with long labels and to check its accuracy, in terms of finding the relevant map. The goal of the second experiment is to examine how well GSME performs against maps that have short labels, especially in finding the relevant sub-graphs. The third experiment was executed on a synthetic map to examine GSME's performance on a large map.

We set several queries for each experiment that were classified into three categories of complexity. The complexity of the categories was determined based on two parameters: (1) **Length** the number of vertices in the graph; and (2) **N-*hop*** supporting *hop* edge in the map according to the expected result. In other words, the difference between the length of the query graph and the expected graph. That is, the number of skips on the graph edges required to get to the desired result.

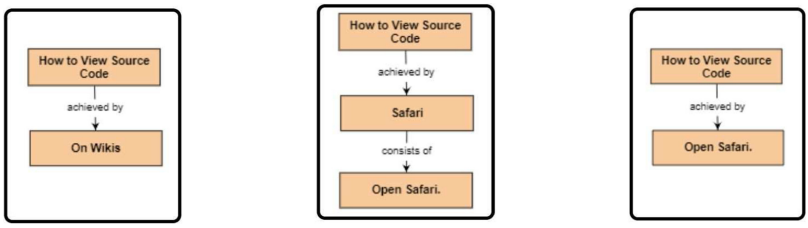
The categories were as following: (1) **Simple** the query length is less then 3 and n-*hop* equals to 0. (2) **Medium** the query length is more then 2 and n-*hop* equals to 0. (3) **Complex** the query has more then 1 n-*hop*.

Examples of simple, medium, and complex queries are shown in Fig. 4. The labels for each node of the queries are matched to the node labels in Fig. 3 to simplify the demonstration of how the complexity parameters influence the difficulty level of each category. Figure 4a is a simple query because the expected result is $'How\ to\ View\ Source\ Code' \rightarrow 'On\ Wikis'$ therefore, N-hop $= 0$

and $Length = 2$. Figure 4b is a medium query because the expected output is $'How\ to\ View\ Source\ Code' \rightarrow 'Safari' \rightarrow 'Open\ Safari'$ and as a result N-hop $= 0$ and $Length = 3$. Finally Fig. 4c is a complex query seeing as $'How\ to\ View\ SourceCode' \rightarrow 'Safari' \rightarrow 'Open\ Safari'$ results in N-hop $= 1$.

Table 2. Graph statistics

Experiment	Nodes	Edges	Number of neighbors	Label length	Number of queries	Number of maps
1	12.452(4.17)	11.452(4.17)	5.359(2.33)	6.169(2.37)	15	200
2	21.333(2.35)	37.333(11.26)	2.770(0.12)	2.355(0.68)	18 each map (56)	3
3	65,599	131,951	3.964	1	9	1



(a) Simple (b) Medium (c) Complex

Fig. 4. Three complexity query levels

Table 2 shows the statistical information of the maps and queries in each experiment. The numbers indicate the average and the standard deviation in brackets. The experiment material and results can be found in[3].

To evaluate the performance of the GMSE algorithm we used the following metrics. (1) **Exact and Domain Mean Reciprocal rank (MRR)** indicates whether the right path and map were retrieved, respectively; (2) **Graph Similarity (G-sim)** measures the similarity between a query graph and a sub-graph from the knowledge graph as defined in Definition 9. The G-sim takes into the account the highest G-sim score of each query and ignores cases in which no answer was found; (3) **Recall@5** that determines whether the expected result appears within the top five results in term of domain match; (4) The **time** it took the algorithm to return a result;

We executed GSME with K = 2, threshold = 0.65, and with two different label similarity (sim_l) functions. The first is Definition 3(1) and the second is Definition 3(2). We also checked whether sim_e and sim_t contributed to improving the search results by running the algorithm using sim_l as the only semantic

[3] https://tinyurl.com/y4ar8bhb.

similarity (LSO) using both Definition 3(1) and Definition 3(2) and comparing the results to a GSME that uses all similarity functions (sim_l, sim_e, sim_t) as defined in Definition 9. We measured the execution time on an Intel® Core™ i-7 6700HQ CPU @ 2.600 GHz processor with 16 GM of RAM.

5.2 Results

Table 3 presents the results of the first two experiments. The complexity column refers to the query complexity. The similarity column refers to the label similarity function used in the experiment: word2vec is used for sim_l as defined in Definition 3(1), sent2vec is used for sim_l as described in Definition 3(2), and LSO as defined above. Next, the various metrics are presented. Each metric column was split into two sub-columns. In the first experiment, it appears that using sent2vec leads to better results in terms of E-MRR, D-MRR and Recall@5. In the second experiment, it appears that the mean results achieved by executing GSME on the three different maps using 18 queries using sim_l as defined in Definition 3(1), led to more accurate results in terms of E-MRR.

With respect to performance, executing the queries on small maps takes a fraction of a second. In the third experiment, in which we checked the performance of GSME on large maps, we found out that the time it took to get the response was on average 2.23 s with a standard deviation of 0.104 s. The measured runtime of GSME was achieved without the use of sim_l. The decision not to use sim_l is due to the fact that the use of the said function increases the runtime during the initialization of GSME due to a lack of indexing.

In addition, we checked which value of the threshold yields a better result in certain groups of complexity. We found a positive correlation between the complexity and threshold values.

Table 3. Experiments Results

Comp.	Sim.	E-MRR		D-MRR	Recall@5	Time		G-sim	
		FE	SE	FE	FE	FE	SE	FE	SE
Simp.	word2vec	0.222	**0.944(0.471)**	0.49	0.6	12.39s(3.41)	0.047s(0.028)	0.803	0.897(0.012)
	sen2vec	0.8	0.388(0.360)	**0.802**	0.8	0.430s(0.07)	0.005s(0.024)	0.6	0.664(0.153)
	LSO	0.002	0.665(0.209)	0.602	0.6	0.003s(0.0)	0.064s(0.031)	0.561	0.913(0.043)
Med.	word2vec	0.3	**0.638(0.274)**	0.47	0.6	8.049s(2.74)	0.041s(0.017)	0.732	0.861(0.076)
	sen2vec	0.4	0.333(0.272)	**0.801**	0.8	0.450s(0.06)	0.021s(0.019)	0.6	0.446(0.377)
	LSO	0.281	0.221(0.314)	0.610	0.4	0.394s(0.04)	0.055s(0.016)	0.57	0.935(0.068)
Complex	word2vec	0.072	**0.666(0.360)**	0.487	0.6	9.74s(3.47)	0.064s(0.028)	0.6	0.8(0.103)
	sen2vec	0.25	0.611(0.437)	**0.850**	1	0.401s(0.04)	0.064s(0.028)	0.601	0.412(0.352)
	LSO	0.281	0.666(0.417)	0.803	0.4	0.369s(0.06)	0.059s(0.016)	0.575	0.792(0.072)

E-MRR = Exact-MRR;LSO_{se} = LSO using sematch; FE = First experiment; SE = Second experiment; Comp. = Complexity; Sim = Similarity; Simp. = Simple; Med. = Medium

5.3 Discussion

In this section, we discuss the results with respect to the parameters of GSME and the input characteristics.

In the experiment, we set the number of neighbors (K) to 2. It might be beneficial to increase K in correlation to the number of neighbors of each node. This might increase the accuracy of the search and allow the exploration of additional paths. On the other hand, it might increase the execution time.

The threshold parameter also affects the results by determining the relevant nodes from which the algorithm moves forward. It's possible that the threshold should be determined based on the domain and the sim_l function. The semantic of two nodes (sim_l) depends on two parameters: the similarity of two labels (Definition 3(1) or Definition 3(2)) and the length of a node's label in terms of how many words the vertex contains. In the case of long labels, the similarity that is defined in Definition 3(2) leads to better accuracy. For short labels, using the similarity in Definition 3(1) achieves better results.

Recent path algorithms showed similar results in terms of recall, and in the second experiment also in terms of time [18,21]. Still, there are differences between recent works to ours. The results of [21] are dependent on the quality of prior knowledge while queries in our experiments are constructed by the assumption that the user does not has this knowledge. [18] assumes that each edge in the target path needs to be similar to at least one edge in the query graph. We speculate that this assumption leads to better performances in terms of time than ours in the third experiment. In addition, these path algorithms are focused only on the similarity between paths rather than the similarity of the path to the domain that the path comes from. This domain similarity is expressed in our experiment by the D-MRR metric.

Alternatives path algorithms appear in [16]. These algorithms look for a specific pattern in an unknown graph. Therefore they search for isomorphic graph. In our problem, isomorphic graph may not be the optimal result for a query graph because we also consider the semantic similarities of the labels. Using such algorithms, for complex queries will not retrieve relevant result.

5.4 Threats to Validity

The initial evaluation we performed should be taken with caution and should consider the following threats to validity:

- **Small maps**: The maps in the experiment are of limited size. Wang et al. [18] used DBPedia, Freebase, Yago, and a synthetic set for experimental validation. These knowledge graphs are more convincing but they are not based on know-how mapping. We should explore the algorithm with much larger know-how maps.
- **Self-authoring**: We as the authors of the paper, developed the queries and the domain maps in the second experiment, so some biases might exist. Nevertheless, our aim in this evaluation was to challenge the algorithm, so the queries we devised accordingly.
- **Simple queries**: As the domain maps are small, so are the queries. There is a need to incorporate more complex queries and multi-paths ones as well.

– **Comparing to other works**: Indeed, the results should be compared to other alternatives. Yet, such alternatives need to be adjusted for searching Know-How maps.

6 Summary

In this paper, we propose GSME, an algorithm for searching ME-maps. The algorithm is a greedy one that takes into account structure, semantic, and type similarity. The initial evaluation we performed shows promising results.

Nevertheless, we want to test how well the algorithm performs with other domains and examine alternatives for calculating the similarity of the vertices and edges. This includes the tuning of the algorithm's parameters, either a-priory or during its execution. We also plan to test the GSME performance with respect to other adjusted alternatives (i.e., datasets and algorithms).

Acknowledgment. This research was partially supported by the Data Science Research Center at Ben-Gurion University of the Negev (DSRC@BGU).

References

1. Angles, R., Arenas, M., Barceló, P., Hogan, A., Reutter, J., Vrgoč, D.: Foundations of modern query languages for graph databases. ACM Comput. Surv. (CSUR) **50**(5), 1–40 (2017)
2. Bragilovski, M., Makias, Y., Shamshila, M., Stern, R., Sturm, A.: Searching for class models. In: Augusto, A., Gill, A., Nurcan, S., Reinhartz-Berger, I., Schmidt, R., Zdravkovic, J. (eds.) BPMDS/EMMSAD -2021. LNBIP, vol. 421, pp. 277–292. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-79186-5_18
3. Ferrucci, D., et al.: Building Watson: an overview of the DeepQA project. AI Mag. **31**(3), 59–79 (2010)
4. Garud, R.: On the distinction between know-how, know-why, and know-what. Adv. Strateg. Manag. **14**, 81–101 (1997)
5. Greenstein, L.: Assessing 21st Century Skills: A Guide to Evaluating Mastery and Authentic Learning. SAGE Publications, Thousand Oaks (2012)
6. Han, S., Zou, L., Yu, J.X., Zhao, D.: Keyword search on RDF graphs-a query graph assembly approach. In: Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, pp. 227–236 (2017)
7. Hu, S., Zou, L., Zhang, X.: A state-transition framework to answer complex questions over knowledge base. In: Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, pp. 2098–2108 (2018)
8. Khwaja, S., Alshayeb, M.: Survey on software design-pattern specification languages. ACM Comput. Surv. **49**(1), 1–35 (2016)
9. Melnik, S., Garcia-Molina, H., Rahm, E.: Similarity flooding: a versatile graph matching algorithm and its application to schema matching. In: Proceedings 18th International Conference on Data Engineering, pp. 117–128. IEEE (2002)
10. Miller, G.A.: Wordnet: a lexical database for English. Commun. ACM **38**(11), 39–41 (1995)

11. Novak, J., Cañas, A.: The theory underlying concept maps and how to construct them (2006)
12. Pareti, E.H., Klein, P.: The human know-how dataset (2014). https://doi.org/10.7488/ds/1394
13. Pérez, J., Arenas, M., Gutierrez, C.: Semantics and complexity of SPARQL. ACM Trans. Database Syst. (TODS) **34**(3), 1–45 (2009)
14. Reinhartz-Berger, I.: Towards automatization of domain modeling. Data Knowl. Eng. **69**(5), 491–515 (2010)
15. Sarewitz, D., Nelson, R.R.: Progress in know-how: its origins and limits. Innov. Technol. Gov. Global. **3**(1), 101–117 (2008)
16. Stern, R., Kalech, M., Felner, A.: Finding patterns in an unknown graph. AI Commun. **25**(3), 229–256 (2012)
17. Sturm, A., Gross, D., Wang, J., Yu, E.: Means-ends based know-how mapping. J. Knowl. Manag. **21**, 454–473 (2017)
18. Wang, Y., Khan, A., Wu, T., Jin, J., Yan, H.: Semantic guided and response times bounded top-k similarity search over knowledge graphs. In: 36th International Conference on Data Engineering (ICDE), pp. 445–456. IEEE (2020)
19. Yadollahi, A., Shahraki, A.G., Zaiane, O.R.: Current state of text sentiment analysis from opinion to emotion mining. ACM Comput. Surv. **50**(2), 1–33 (2017)
20. Yu, E., Giorgini, P., Maiden, N., Mylopoulos, J.: Social Modeling for Requirements Engineering. The MIT Press, Cambridge (2011)
21. Zheng, W., Zou, L., Peng, W., Yan, X., Song, S., Zhao, D.: Semantic SPARQL similarity search over RDF knowledge graphs. Proc. VLDB Endow. **9**(11), 840–851 (2016)
22. Zhu, G., Iglesias, C.A.: Sematch: semantic similarity framework for knowledge graphs. Knowl.-Based Syst. **130**, 30–32 (2017)
23. Zou, L., Huang, R., Wang, H., Yu, J.X., He, W., Zhao, D.: Natural language question answering over RDF: a graph data driven approach. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp 313–324 (2014)