A*-based Similarity Assessment of Semantic Graphs

Christian Zeyen[1] and Ralph Bergmann[1,2]

[1] Business Information Systems II, University of Trier, 54286 Trier, Germany
{zeyen,bergmann}@uni-trier.de
http://www.wi2.uni-trier.de
[2] German Research Center for Artificial Intelligence (DFKI), Branch University of Trier, Behringstraße 21, 54296 Trier, Germany
ralph.bergmann@dfki.de

Abstract The similarity assessment of graphs is a fundamental problem that is particularly challenging if efficiency is of core importance. In this paper, we focus on a similarity measure for semantically labeled graphs whose labels are composed in an object-oriented manner. The measure is based on A* search and is particularly suited for case-based reasoning as it can be combined with knowledge-intensive local similarity measures and outputs similarities and corresponding mappings usable for explanation and adaptation. However, particularly for large graphs, the search space must be pruned to improve efficiency of A* search at the cost of sacrificing global optimality. We address this issue and present complementary improvements of the measure, which we systematically evaluate for the similarity assessment of semantic workflow graphs. The experimental results demonstrate that the new measure considerably reduces the computation time and memory consumption while increasing the accuracy.

Keywords: Semantic Graphs · Graph Matching · Graph Similarity

1 Introduction

Graph-based case representations with semantically labeled nodes and/or edges are significantly gaining importance in case-based reasoning (CBR). They allow to represent arbitrary relational structures and thus considerably increase expressiveness compared to attribute-value or pure object-oriented representations. However, the gain in expressiveness comes with the cost of increased complexity in the similarity assessment during the retrieval phase. In the literature, various similarity measures for graph-based representations have been proposed [12]. However, assessing the similarity in an efficient way is a fundamental problem due to the computational complexity of the approximate graph matching involved. It is particularly challenging for semantically enriched graphs since their similarity is affected by structure and semantics.

In this paper, we consider a specific form of semantically enriched graphs, which we refer to as semantic graphs. A semantic graph is a generic directed

graph whose nodes and edges have different types and are associated with semantic descriptions, which can be composed in an object-oriented manner. Semantic graphs are particularly used as case representation in process-oriented case-based reasoning (POCBR) for representing semantic workflows [2] but also for representing arguments [4] in case-based argumentation.

To assess the similarity of such graphs, Bergmann and Gil [2] proposed a generic semantic similarity measure following the well-known local-global principle [1,6] that is based on finding the best possible mapping of similar nodes and edges between the graphs to be compared. In particular, this measure outputs the similarity values along with the corresponding mappings, which can be the basis for adaptation [5] and which can be also used for explanation purposes. The involved optimization problem can be solved in principle by applying A* search, which is theoretically able to find the optimal mapping. However, in practical applications with large graphs, the search space can become so large that it must be limited, thus trading optimality against efficiency. To overcome this performance issue during retrieval, several two-step MAC/FAC retrieval approaches [9,5,10] have been proposed, which reduce the number of expensive similarity computations by using an efficient pre-selection of cases. A recent approach by Hoffmann et al. [8] shows promising results in approximating graph similarities with siamese graph neural networks. In this paper, however, we follow a different route of research as we aim at improving the efficiency of each single similarity computation by speeding-up the A* search. We do so by reorganizing the search space and by proposing a better-informed heuristic that guides the search. In total, four complementary improvements are described and evaluated systematically.

The following section 2 introduces the graph representation and briefly surveys approaches to graph similarity before we describe the A*-based similarity measure to be improved. Section 3 presents the improvements for the measure while section 4 investigates the performance impact in an experimental evaluation. Finally, we summarize the paper and discuss future work.

2 Background

2.1 Semantic Graphs

Based on the definition of semantic workflow graphs [2], we consider a semantic graph as a quadruple $G = (N, E, S, T)$. The graph elements are defined by a set of nodes N and a set of edges $E \subseteq N \times N$. $S : N \cup E \to \Sigma$ associates to each graph element a semantic description from a semantic metadata language Σ while $T : N \cup E \to \Omega$ associates to each graph element a type from Ω. While types are assumed to be disjoint, semantic descriptions can be organized in a hierarchy. Figure 1 gives an example of a semantic workflow graph representing a sandwich recipe. The graph consists of three different types of nodes and edges, which can be distinguished by the different shapes and lines in the figure. The workflow node (diamond) represents general information about the recipe, task nodes (rectangles) represent preparation steps, and data nodes (ovals) represent

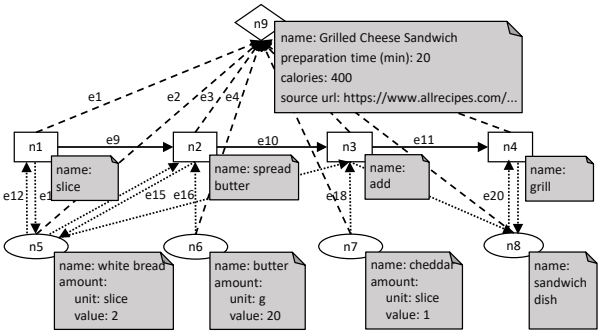

Fig. 1. Example of a semantic workflow graph

ingredients. Furthermore, control-flow edges (solid lines) define the execution order of preparation steps, data-flow edges (dotted lines) specify the consumption and production of ingredients, and part-of edges (dashed lines) link all nodes to the workflow node. Semantic descriptions of the nodes are written in grey boxes. In this implementation, we treat the semantic descriptions in an object-oriented fashion and use the local-global principle [1,6] for assessing sim_Σ.

Following the CBR paradigm, the underlying retrieval problem that requires computing graph similarities is the following: For a given query graph G_q, the best-matching graph $G_c \in CB$ is being searched in a repository of graphs CB, which is referred to as case base. For this purpose, G_q is compared with each case graph G_c and rated with a similarity $sim(G_q, G_c) \in [0, 1]$.

2.2 Approaches to Similarity Assessment of Multi-Labeled Graphs

In general, numerous similarity measures for graph-based representations have been proposed in the literature [12]. For semantically labeled graphs and particularly for graphs with multiple labels (also referred to as multi-attributed graphs), considerably less approaches exist. For such graphs, approaches based on greedy search [7] and Tabu search [14] have been proposed. However, due to the incomplete search only local optima are found and the similarity error can be hardly controlled. Particularly tailored for large graphs, Zhu et al. [15] presented an index-based approach combined with a greedy algorithm and Shemshadi et al. [13] presented an approach based on graph simulation. The focus is put on graphs with textual labels instead of composed semantic descriptions. More recently, Li et al. [11] proposed an embedding approach with graph neural networks, which was extended by Hoffmann et al. [8] to support composed semantic descriptions. Like any other supervised learning approach, a sufficiently large number of training data is required. To reduce the manual effort, further unsupervised approaches to graph similarity are required that allow for assessing graph similarity values in an appropriate and efficient manner.

Most approaches have in common that they restrict the semantic annotations of graph elements to attribute-value representation. More importantly, they do

not allow for knowledge-intensive similarity assessments of semantic descriptions. Ontañón [12] recently identified, among others, scalability and interpretability as open research questions. While graph embeddings using neural networks is a promising research direction to address scalability, interpretability remains a major challenge.

2.3 Semantic Graph Similarity following the Local-Global Principle

To assess the similarity between two graphs $G_q = (N_q, E_q, S_q, T_q)$ and $G_c = (N_c, E_c, S_c, T_c)$, we follow the approach proposed by Bergmann and Gil [2] which allows to consider the structure as well as the semantics of the graphs. The proposed similarity measure applies the local-global principle [1,6] to allow for assessing the similarity in a flexible manner. It enables the comparison of the graph elements w.r.t. their semantic descriptions by knowledge-intensive local similarity measures. For this purpose, a similarity function $sim_\Sigma : \Sigma \times \Sigma \to [0,1]$ is modeled as part of the knowledge-engineering process such that semantic descriptions of nodes and edges from query and case can be compared w.r.t. their similarity. Depending on the choice of Σ, this similarity can itself be assessed following the local-global principle. This is particularly useful for semantic descriptions represented in an object-oriented fashion. Thus, available similarity knowledge for the graph elements can be considered in a flexible manner and the computed similarity values are transparent and interpretable.

Following the local-global principle, the global similarity during graph comparison is obtained by an aggregation function combining the local similarities of related graph elements from G_q and G_c. Based on this principle, the node similarity $sim_N(n_q, n_c)$ for $n_q \in N_q$ and $n_c \in N_c$ is defined as follows:

$$sim_N(n_q, n_c) = \begin{cases} sim_\Sigma(S_q(n_q), S_c(n_c)) & \text{if} \quad T_q(n_q) = T_c(n_c) \\ 0 & \text{otherwise} \end{cases}$$

Edge similarity does not only consider the semantic description of the edges being compared, but also the nodes linked by the edges. We define edge similarity $sim_E(e_q, e_c)$ for $e_q \in E_q$ and $e_c \in E_c$ as follows:

$$sim_E(e_q, e_c) = \begin{cases} F_E \begin{pmatrix} sim_\Sigma(S_q(e_q), S_c(e_c)), \\ sim_N(e_q.l, e_c.l), \\ sim_N(e_q.r, e_c.r) \end{pmatrix} & \text{if} \quad T_q(e_q) = T_c(e_c) \\ 0 & \text{otherwise} \end{cases}$$

$e.l$ denotes the left node of an edge e and $e.r$ denotes its right node. The function F_E is an *aggregation function* that combines the semantic similarity between the edges and the similarities of the connected nodes to the overall similarity value. In our implementations we define F_E as follows: $F_E(s_e, s_l, s_r) = s_e \cdot (s_l + s_r)/2$.

The similarity $sim_m(G_q, G_c)$ between G_q and G_c is defined with respect to a *legal mapping* $m : N_q \cup E_q \to N_c \cup E_c$ that satisfies the following five constraints:

$$\begin{array}{lll} T_q(n_q) = T_c(m(n_q)) & T_q(e_q) = T_c(m(e_q)) & \\ m(e_q.l) = m(e_q).l & m(e_q.r) = m(e_q).r & \forall x, y \; m(x) = m(y) \to x = y \end{array}$$

Please note that such a legal mapping m is *type-preserving*, i.e., only nodes and edges of the same type can be mapped. The mapping is *injective*, which means that a case node or edge can only be the target of one query node or edge, respectively. Moreover, the mapping can be *partial*, i.e., not all nodes and edges of the query must be mapped to case elements and can be mapped to null instead. For instance, null mappings are required if more query elements exist than case elements of a certain type. An edge can only be mapped if the nodes that the edge connects are also mapped to the respective nodes that the mapped edge connects. For a given mapping m, a second aggregation function F_G is defined that combines the individual similarity values for mapped elements:

$$sim_m(G_g, G_c) = F_G \left(\begin{array}{l} (sim_N(n, m(n)) | n \in N_q \cap \mathrm{Dom}(m)), \\ (sim_E(e, m(e)) | e \in E_q \cap \mathrm{Dom}(m)), |N_q|, |E_q| \end{array} \right)$$

$\mathrm{Dom}(m)$ denotes the domain of m. The parameters $|N_q|$ and $|E_q|$ enable F_G to consider partial mappings, i.e., nodes and edges not mapped should not contribute to the similarity. In our implementation we define F_G as follows:

$$F_W((sn_1, \ldots, sn_i), (se_1, \ldots, se_j), n_N, n_E) = \frac{sn_1 + \cdots + sn_i + se_1 + \cdots + se_j}{n_N + n_E}$$

The overall similarity $sim(G_q, G_c)$ is determined by the mapping with the highest similarity:

$$sim(G_q, G_c) = \max\{sim_m(G_q, G_c) | \, m \text{ is legal mapping}\}$$

2.4 A*-based Similarity Search

To find the best mapping m, Bergmann and Gil [2] proposed to apply an A* search. The A* algorithm maintains a priority queue Q of partial solutions for this optimization problem. In such a solution Sol, $Sol.m$ represents the current mapping and $Sol.N$ and $Sol.E$ are the not yet mapped nodes and edges of the query graph. In each step, the first (best) solution in the queue $first(Q)$ is removed. If it represents a completely expanded solution, A* terminates. Otherwise, the solution is expanded, i.e., the next query graph element x_q is selected and all legal mappings to the case graph elements are determined. For each such mapping, a new solution extended by this additional mapping is created and inserted into the priority queue. If more query elements exist than case elements of a certain type, an additional *null mapping* must be added as a new solution to allow for partial mapping. The total amount of required null mappings corresponds to the difference between the query and case elements. The order in which the solutions are inserted into the priority queue is essential for A*. Therefore, each solution Sol is evaluated by a function $f(Sol) = g(Sol) + h(Sol)$ and the value is stored in the solution as $Sol.f$. In the traditional formulation, A* aims at minimizing cost, hence $g(Sol)$ are the cost already occurred and $h(Sol)$ is a heuristic estimation function for the remaining cost to the solution. As we apply A* for maximizing the similarity value, $g(Sol)$ is the similarity of the

current mapping $Sol.m$, while $h(Sol)$ is a heuristic estimation of the additional similarity that can be achieved through the mapping of the remaining nodes and edges. Solutions are inserted into the priority queue in decreasing order of $f(Sol)$. Consequently, the solution with the highest f-value is expanded first. To achieve an admissible heuristic estimation function, which ensures that the optimal solution is found, $h(Sol)$ must be an upper bound of the similarity.

```
A*Search(G_q = (N_q, E_q, S_q, T_q), G_c = (N_c, E_c, S_c, T_c))
  | Q = insert(initSolution(), Q);
  | while first(Q).N ≠ ∅ ∧ first(Q).E ≠ ∅ do
  |   | Q = expand(Q);
  | end
return first(Q).f;

initSolution()
  | Sol_0.N = N_q;  Sol_0.E = E_q;  Sol_0.m = ∅;  Sol_0.f = 1;
return Sol_0;

expand(Q)
  | Sol = first(Q);  Q = rest(Q);  x_q = select(Sol);
  | forall x_c ∈ N_c ∪ E_c such that the mapping Sol.m ∪ (x_q, x_c) is legal do
  |   | Q = insert(newSolution(Sol, x_q, x_c), Q);
  | end
  | if T_q(x_q) requires null mapping w.r.t. Sol.m then
  |   | Q = insert(newSolution(Sol, x_q, ⌀), Q);
  | end
return Q;

newSolution(Sol, x_q, x_c)
  | Sol'.N = Sol.N \ {x_q};  Sol'.E = Sol.E \ {x_q};
  | Sol'.m = Sol.m ∪ (x_q, x_c);
  | Sol'.f = sim_{Sol'.m}(G_q, G_c) + h(Sol');
return Sol';
```

The overall A* search algorithm is sketched above. The function *first*(Q) returns the first solution in the priority queue, *rest*(Q) removes the first solution and returns the rest and *select* determines the next graph element x_q to be mapped, which can be either a query node or edge. The function *insert*(Sol, Q) inserts a solution Sol into Q according to the f-value. During insert, the maximum size of the queue can be restricted to prune the search space for improving the performance on the risk of losing global optimality.

Bergmann and Gil [2] presented two A* variants named *A*I* and *A*II* with different estimation and select functions. In this paper, we build up upon *A*II*:

$$h_{II}(Sol) = \sum_{x \in Sol.N \cup Sol.E} \left(\max_{y \in N_c \cup E_c} \{sim_{N/E}(x, y)\} \right) \cdot \frac{1}{|N_q|+|E_q|}$$

$$select_{II}(Sol) = \begin{cases} e_q \in Sol.E & \text{if } e_q.l \notin Sol.N \wedge e_q.r \notin Sol.N \\ n_q \in Sol.N & otherwise \end{cases}$$

$sim_{N/E}(x, y)$ refers to the corresponding similarity function sim_N or sim_E. It was shown that *A*II* clearly outperforms *A*I* since it uses a more informed admissible heuristic. While h_I uses the maximum similarity of 1.0 as estimation for each not mapped query graph element, h_{II} determines the best possible similarity a mapping can achieve independent of the mapping of the other elements.

This can be computed in advance and cached. The select function chooses elements randomly according to an internal id and selects edges as soon as possible. As mapping edges requires the linked nodes being mapped already, it requires only a low number of new solutions to be added to the priority queue. Only one solution is created, if between nodes there is at most one edge per type. Hence, the size of the queue does not increase while the accuracy of $f(Sol')$ increases.

3 Improving the A*-based Similarity Search

We now present four complementary improvements for enhancing the performance of the A*-based similarity search. For illustration purposes, we refer to the simple example graphs G_q and G_c depicted in Fig. 2. The graphs consist of two different types of nodes and edges (depicted by different shapes and lines). The semantic descriptions consist of symbolic values with $\Sigma = \{a, b, c\}$ and we assume sim_Σ to be defined as a binary measure returning 1 if symbols are equal and 0 otherwise. Moreover, edges and diamond shaped nodes do not have semantic descriptions and thus match with a similarity of 1. Please note that the similarity assessment is asymmetric. Hence, $sim(G_q, G_c) = \frac{6}{9}$ although G_c is sub-graph isomorphic with G_q, i.e., $sim(G_c, G_q) = 1$.

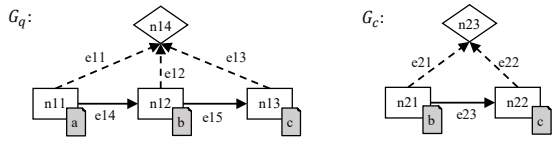

Fig. 2. Example of a query and case graph

3.1 Search Space Reduction

According to the definition of semantic graphs, a query $G_q = (N_q, E_q, S_q, T_q)$ and a case graph $G_c = (N_c, E_c, S_c, T_c)$ consist of different types $T_q \subseteq \Omega$ and $T_c \subseteq \Omega$ of nodes and edges. A legal mapping of graph elements is type-preserving. Consequently, we can add element pairs (x_q, x_c) to the initial solution for which only one legal mapping exists. By this means, the search space can be reduced prior to the A* search. Regarding the graph representation, it is advisable to assign different types to graph elements whenever their semantic descriptions are disjoint. To implement this improvement, we redefine *initSolution* as follows:

initSolution2()
  $Sol_0 = initSolution()$;
  forall $(x_q, x_c) \in N_q \times N_c$ *such that the mapping is legal* $\wedge\ (\nexists x'_q : x'_q \neq x_q \wedge T_q(x'_q) = T_q(x_q)) \wedge (\nexists x'_c : x'_c \neq x_c \wedge T_c(x'_c) = T_c(x_c))$ **do**
    $Sol_0 = newSolution(Sol_0, x_q, x_c)$;
  end
return Sol_0;

In the given example, types $T_q(n14) = T_c(n23)$ are equal and the mapping of the nodes is the only possible legal mapping. Consequently, it can be added to the initial solution. This improvement also comes into effect for the semantic workflow graph representation depicted in Fig. 1 since such graphs have a single workflow node linked to all other nodes via part-of edges. During the mapping process, the already mapped workflow node enables that a part-of edge can be always mapped subsequent to the mapping of another node.

3.2 Adaptive Mapping Orientation

If the query graph is larger than the case graph, we assume that the mapping process can be made more efficient by orienting the mapping towards the case elements. This is referred to as *case-oriented mapping*. Please note that the direction of the mappings and the similarity assessment is unaffected, i.e., mapping and similarity are still oriented from the query elements to the case elements. The mapping mode is selected prior to the search according to the following rule:

$$mapping\_mode = \begin{cases} case\text{-}oriented & \text{if } |N_q \cup E_q| > |N_c \cup E_c| \\ query\text{-}oriented & otherwise \end{cases}$$

To implement case-oriented mapping, the algorithm is modified as follows: The collections of not mapped elements are initialized with the case graph elements $Sol_0.N = N_c$ and $Sol_0.E = E_c$ instead of the query elements. In each expansion step (invocation of *expand*), the *select*-function identifies the next case element to which all query elements are mapped:

$$select_{II_C}(Sol) = \begin{cases} e_c \in Sol.E & \text{if } e_c.l \notin Sol.N \wedge e_c.r \notin Sol.N \\ n_c \in Sol.N & otherwise \end{cases}$$

New solutions are created for each legal mapping and the f-value is updated using the following modified heuristic estimation function:

$$h_{II_C}(Sol) = \sum_{y \in Sol.N \cup Sol.E} \left(\max_{x \in N_q \cup E_q} \{sim_{N/E}(x, y)\} \right) \cdot \frac{1}{|N_q|+|E_q|}$$

When creating new solutions in case-oriented mapping mode, in contrast to the query-oriented mapping mode, null mappings $(\varnothing, x_c)$ are not meaningful to the final mapping and thus are not added to the solution. Instead, it is checked whether all case elements have been considered by the search. In this event, a function *completeSolution* is invoked that adds a null mapping $(x_q, \varnothing)$ to the solution for each not mapped query element x_q. Following this approach, all required null mappings are added in a single step to the same solution. Consequently, fewer solutions are added to the priority queue and we expect that the computation time as well as the memory consumption is reduced. The effect of the adaptive mapping orientation can be demonstrated with the given example. Here, the required null mappings $(n11, \varnothing)$, $(e11, \varnothing)$, and $(e14, \varnothing)$ are postponed in case-oriented mapping mode. The modified functions are as follows:

```
initSolution_C()
  |  Sol_0.N = N_c;  Sol_0.E = E_c;  Sol_0.m = ∅;  Sol_0.f = 1;
return Sol_0;

expand_C(Q)
  |  Sol = first(Q);  Q = rest(Q);  x_c = select_C(Sol);
  |  forall x_q ∈ N_q ∪ E_q such that the mapping Sol.m ∪ (x_q, x_c) is legal do
  |    |  Q = insert(newSolution_C(Sol, x_q, x_c), Q);
  |  end
  |  if T_c(x_c) requires null mapping wrt. Sol.m then
  |    |  Q = insert(newSolution_C(Sol, ∅, x_c), Q);
  |  end
return Q;

newSolution_C(Sol, x_q, x_c)
  |  Sol'.N = Sol.N \ {x_c};  Sol'.E = Sol.E \ {x_c};
  |  if x_q ≠ ∅ then
  |    |  Sol'.m = Sol.m ∪ (x_q, x_c);
  |  end
  |  if Sol'.N = ∅ ∧ Sol'.E = ∅ then
  |    |  completeSolution(Sol');
  |  end
  |  Sol'.f = sim_{Sol'.m}(G_q, G_c) + h(Sol');
return Sol';
```

3.3 More Informed Heuristic

A well-informed admissible heuristic $h(Sol)$ is crucial for the efficiency of the A* search. Since $h(Sol)$ must be an upper bound of the estimated similarity, a more informed heuristic overestimates the similarity to a lower degree. Higher accuracy is beneficial since it decreases the possibility that less expanded solutions are ranked higher in the priority queue. Consequently, the search becomes more like a depth-first search and expanding the same element is less often required.

We propose a novel heuristic $h_{III}(Sol)$ that, in contrast to heuristic $h_{II}(Sol)$, considers the current mapping $Sol.m$ for determining the maximum possible similarity a new mapping can achieve. It excludes mappings from the estimation that do not lead to a legal mapping when added to $Sol.m$. Hence, this heuristic is computationally more expensive since all independent mappings must be computed in advance and the heuristic must be updated with respect to the current mapping $Sol.m$. However, since the heuristic is more accurate, we expect that partial solutions with non-optimal (but legal) mappings are ranked lower in the priority queue and hence the overall performance of the search is improved. We define the heuristic estimation function as follows:

$$
\begin{aligned}
h_{III}(Sol) = \tfrac{1}{|N_q|+|E_q|} \Big(& \sum_{x \in Sol.N} \max_{y \in N_c} \{ sim_N(x,y) \mid \nexists n \in Sol.N : (n,y) \in Sol.m \} \\
& + \sum_{x \in Sol.E} \max_{y \in E_c} \{ sim_E(x,y) \mid \nexists e \in Sol.E : (e,y) \in Sol.m \\
& \qquad \wedge \nexists (x.l, n) \in Sol.m : y.l \neq n \\
& \qquad \wedge \nexists (x.r, n) \in Sol.m : y.r \neq n \\
& \qquad \wedge \nexists (n, y.l) \in Sol.m : x.l \neq n \\
& \qquad \wedge \nexists (n, y.r) \in Sol.m : x.r \neq n \} \Big)
\end{aligned}
$$

An isolated mapping (x, y) between nodes/edges is invalid and thus not considered for estimating the similarity of x, if another node n/edge e was already mapped to y. With respect to isolated edge mappings, it is required that if the

left/right node of the edge was already mapped to another node n, the mapping must correspond with the node mapping that is prerequisite for the edge mapping. For the given example graphs, a mapping $(n11, n21) \in Sol.m$ invalidates e.g. the isolated mappings $(n12, n21)$ and $(e15, e23)$. Regarding case-oriented mapping mode, $h_{III}(Sol)$ is slightly modified analogous to $h_{II_C}(Sol)$.

3.4 Heuristic-based Element Selection

Besides the heuristic, the selection of the next element to be mapped is essential for the A* search and has a significant impact on the performance. If the next element is mapped with a high similarity, the expanded solution is ranked in front of the priority queue. For this reason, we propose to use the estimated similarities from the heuristic. The new select function first selects the element whose best-possible mappings are rated with the highest similarity. If several of such elements exist, the element is chosen with the smallest number of best-possible mappings. If still several elements remain, the element is chosen randomly according to an internal id analogous to $select_{II}(Sol)$. In contrast to $select_{II}(Sol)$, edges are not preferred over nodes. However, if an edge x is selected with respect to the new criteria but the linked nodes have not been mapped yet ($x.l \in Sol.N \vee x.r \in Sol.N$), such nodes are mapped first. The *select*-function is formalized as follows:

```
select_III(Sol)
    x = select_{h_III}(Sol.N ∪ Sol.E);
    if x ∈ Sol.E ∧ x.l ∈ Sol.N then
        return x.l;
    else if x ∈ Sol.E ∧ x.r ∈ Sol.N then
        return x.r;
    end
return x;
```

Here, $select_{h_{III}}(Sol.N \cup Sol.E)$ determines the best graph element (node or edge) to be mapped with heuristic h_{III} regarding the criteria mentioned above.

4 Experimental Evaluation

The evaluation is structured in two experiments. In the first experiment, we evaluate the performance of the A* variants without pruning the solution space. Consequently, the measures ensure that the obtained graph similarity is the global optimum. We compare the A* variants regarding the computation time and the maximum number of solutions in the priority queue as an indicator of memory consumption. In the second experiment, we enable pruning and compare the best performing A* variant from the first experiment with the baseline approach *A*II*. We compare the computation time and the similarity error for different size limits of the priority queue.

The experiments are conducted with a set of 40 sandwich recipes represented as semantic workflow graphs[3] such as the graph depicted in Fig. 1 showing the

[3] For implementation details, please refer to [3].

smallest workflow graph from the case base. In both experiments, all pairwise similarities are computed between the graphs (i.e., each graph is used once as query) resulting in a total of 1600 computations. The similarity values of the graph pairs range from 0.0758 to 1.0 with an average of 0.4331. Table 1 shows the quantities of graph elements in the case base. Even though the workflow graphs have a particular structure such as a single workflow node, we note that the improvements are largely independent of specific graph characteristics.

Table 1. Workflow graph elements in the case base

	size	workflow nodes	task nodes	data nodes	part-of edges	control-flow edges	data-flow edges
min	30	1	4	4	8	3	10
median	72	1	8	12	20	7	24
max	148	1	20	17	37	19	54
avg	77	1	10	10	20	9	27

We implemented all A* variants in Java for the ProCAKE framework [3]. Each similarity computation is run on a new Java Virtual Machine (JVM) instance to minimize the effects of JVM runtime optimizations. The experiments are run on a computer with a 2.1 GHz processor and each JVM may use a maximum of 80 GB of memory, which does not constitute a restriction.

4.1 Similarity Computation with Ensured Optimality

In the first experiment, we investigate the impact of each single A* improvement and their combination on the performance of the similarity computation. Performance is assessed with respect to the computation time and the maximum number of solutions in the priority queue as an indicator of memory consumption. We test the following hypotheses:

H1a *Search Space Reduction* improves the avg. performance of A* variants.
H1b *Adaptive Mapping Orient.* improves the avg. performance of A* variants.
H1c The avg. performance of *A*III* variants (using the more informed heuristic) is better than that of *A*II* variants.
H1d *Heuristic-based Element Selection* improves the avg. performance of *A*III* variants.

In this experiment, the solution space is not pruned by limiting the priority queue size, which ensures that a global optimum is found. However, a timeout of 120 seconds is set for each computation. We tested all combinations of the various improvements, resulting in a total number of 12 A* variants. *A*II* is used as baseline and can be extended with *Search Space Reduction (R)* and *Adaptive Mapping Orientation (M)*. The new A* variant *A*III* that uses the more informed heuristic can be combined with improvements *R*, *M*, and also with *Heuristic-based Element Selection (S)*.

Figure 3 shows the number of computations (line chart) completed before the specified timeout of 120 seconds was reached. It also shows the maximum

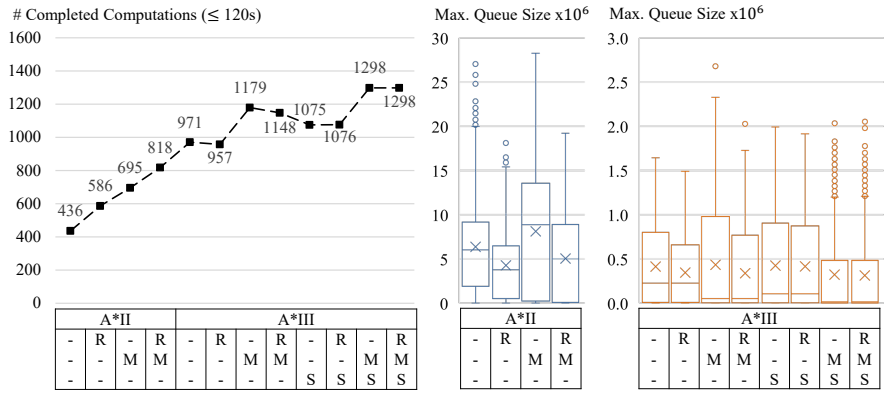

Fig. 3. Performance of A* variants

size of the priority queue (box plots) recorded in that time span for each of the 1600 computations. With no A* variant all computations could be finished. Comparing the baseline *A*II* with the fully supplemented variant *A*III-RMS*, three times more computations completed and the maximum size of the priority queue is about ten times smaller. The numbers indicate that *A*III* variants expand considerably less solutions resulting in much smaller priority queues and hence in a lower memory consumption than that of the *A*II* variants. The 436 computations completed with *A*II* took 13.07 seconds and required a maximum queue size of 1,175,633 in average while the 1298 computations completed with *A*III-RMS* took 8.78 seconds and a maximum queue size of 96,714 in average. Consequently, the baseline measure is clearly outperformed.

The results also indicate that each improvement positively affects the performance of A* variants. *Search Space Reduction (R)* considerably increased the number of completed computations for *A*II* variants but does not affect or slightly reduces that of *A*III* variants. However, it reduces the maximum queue size for every A* variant. *Adaptive Mapping Orientation (M)* has a higher positive impact on both *A*II* and *A*III* variants regarding the completed computations at the cost of an increased queue size. However, in combination with *Heuristic-based Element Selection (S)* it reduces the queue size. *Heuristic-based Element Selection (S)* itself results in an increased number of completed computations and reduced maximum queue size for the *A*III* variants. All in all, we see hypotheses H1a to H1d confirmed.

4.2 Similarity Computation without Ensured Optimality

The second experiment addresses the similarity computation with a pruned solution space. Pruning becomes particularly necessary for large graphs due to the high memory consumption of the A* search. As pruning may cause a similarity error, we investigate the impact of different queue size limits on the accuracy in

this experiment. We expect that the error decreases for higher limits of the priority queue. We compare the best A* variant from the previous experiment, i.e., *A*III-RMS*, with the baseline *A*II* regarding the computation time and similarity error for different queue size limits. We formulate the following hypotheses for the experiment:

H2a The accuracy of *A*III-RMS* is higher than that of *A*II* for a similar avg. computation time.

H2b The memory consumption of *A*III-RMS* is lower than that of *A*II* for a similar avg. computation time.

For each query graph, we store the highest similarity value obtained from all computations as the global optimum for assessing the similarity error. We do not set a timeout since computation time is restricted by the queue size limit.

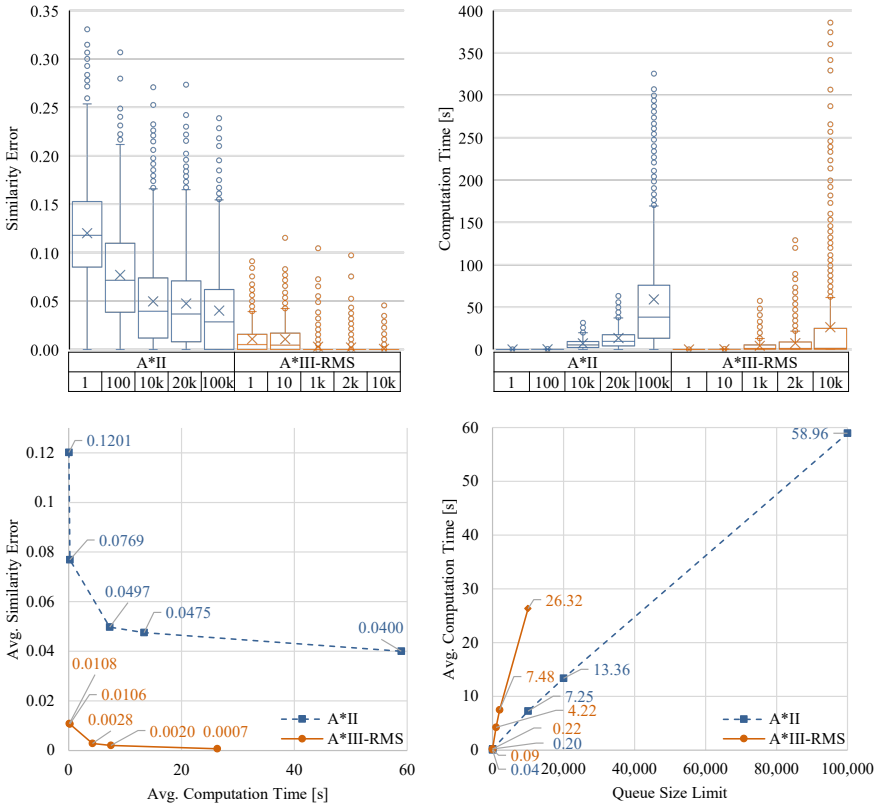

Fig. 4. Similarity error and computation time for different queue size limits

Figure 4 depicts the results for selected queue size limits. The box plots show the similarity errors (top left) and the computation times (top right) for different

queue size limits. For *A*III-RMS* with queue size limit of 10.000 four extreme outliers (with a maximum value of 642) are cut off. The graphs below plot the similarity error over the computation time (bottom left) and the computation time over the queue size limit (bottom right).

It can clearly be seen, that *A*III-RMS* outperforms *A*II* regarding the similarity error in terms of accuracy for similar average computation times. Consequently, H2a is confirmed. It is apparent that the similarity errors of *A*II* deviate more strongly than that of *A*III-RMS*, independent of the tested queue size limits. For a queue size limit of one, *A*III-RMS* has an average similarity error of 0.01 and a maximum error of 0.9 which seems acceptable for some use cases particularly in consideration of the computation times and errors with larger queue size limits. We observed that the queue size limit of *A*III-RMS* must be chosen about five to ten times smaller than that of *A*II* for achieving a similar computation time in average. Consequently, *A*III* has a considerably lower memory consumption than *A*II*, which confirms H2b. The computation time increases proportionally to the queue size limit. For *A*III-RMS*, the computation time increases more strongly for larger queue sizes in contrast to *A*II*.

5 Conclusion and Future Work

In this paper, we presented an improved similarity measure for assessing the similarity of semantic graphs whose labels are composed in an object-oriented manner. For such graphs, the efficient similarity computation is particularly challenging since their similarity is affected by structure and semantics. The measure discussed in this paper is based on A* search and is particularly suited for case-based reasoning as it can be combined with knowledge-intensive local similarity measures and outputs similarities and corresponding mappings usable for explanation and adaptation. We presented four complementary improvements that are suitable for enhancing the computation time and memory consumption of the similarity computation. We also demonstrated that the improvements considerably increase the accuracy of computations with pruned solution space.

In a next step, we plan to add an additional parameter to the measure for completing the A* search within a certain period or with limited memory consumption. Based on the given limits, the A* search is performed with minimal pruning that can be intensified dynamically if required. In future work, we also plan to integrate this measure with different retrieval approaches. For instance, Bergmann and Gil [2] presented a parallelized A*-based retrieval approach that enables to compute the top k graphs from the case base without fully computing the similarity for all graphs. To this end, the search process is parallelized for all graphs and the search terminates, when at least k searches have terminated and when the similarity of the k-best graphs is higher than all f-values of the remaining computations. The integration of the improved measure with MAC/FAC retrieval approaches seems also promising for performing knowledge-intensive similarity computations in an efficient manner.

Acknowledgements. This work is funded by the German Research Foundation (DFG) under grant No. BE 1373/3-3 and grant No. 375342983.

References

1. Bergmann, R.: Experience Management: Foundations, Development Methodology, and Internet-Based Applications, LNCS, vol. 2432. Springer (2002)
2. Bergmann, R., Gil, Y.: Similarity assessment and efficient retrieval of semantic workflows. Information Systems **40**, 115–127 (2014)
3. Bergmann, R., Grumbach, L., Malburg, L., Zeyen, C.: ProCAKE: A process-oriented case-based reasoning framework. In: Case-Based Reasoning Research and Development: 27th ICCBR 2019, Workshop Proceedings (2019)
4. Bergmann, R., Lenz, M., Ollinger, S., Pfister, M.: Similarity measures for case-based retrieval of natural language argument graphs in argumentation machines. In: Barták, R., Brawner, K.W. (eds.) Proceedings of the Thirty-Second FLAIRS Conference, 2019. pp. 329–334. AAAI Press (2019)
5. Bergmann, R., Müller, G.: Similarity-based retrieval and automatic adaptation of semantic workflows. In: Nalepa, G.J., Baumeister, J. (eds.) Synergies Between Knowledge Engineering and Software Engineering, Advances in Intelligent Systems and Computing, vol. 626, pp. 31–54. Springer (2018)
6. Burkhard, H.D., Richter, M.M.: On the notion of similarity in case-based reasoning and fuzzy theory. In: Pal, S.K., Dillon, T.S., Yeung, D.S. (eds.) Soft Computing in Case-Based Reasoning, pp. 29–45. Springer (2001)
7. Champin, P.A., Solnon, C.: Measuring the similarity of labeled graphs. In: Ashley, K.D., Bridge, D.G. (eds.) Case-Based Reasoning Research and Development: 5th ICCBR 2003. LNCS, vol. 2689, pp. 80–95. Springer (2003)
8. Hoffmann, M., Malburg, L., Klein, P., Bergmann, R.: Using siamese graph neural networks for similarity-based retrieval in process-oriented case-based reasoning. In: Case-Based Reasoning Research and Development: 28th ICCBR 2020. LNCS, Springer (2020), Accepted for publication.
9. Kendall-Morwick, J., Leake, D.: A study of two-phase retrieval for process-oriented case-based reasoning. Studies in Computational Intelligence **494**, 7–27 (2014)
10. Klein, P., Malburg, L., Bergmann, R.: Learning workflow embeddings to improve the performance of similarity-based retrieval for process-oriented case-based reasoning. In: Bach, K., Marling, C. (eds.) Case-Based Reasoning Research and Development: 27th ICCBR 2019. LNCS, vol. 11680, pp. 188–203. Springer (2019)
11. Li, Y., Gu, C., Dullien, T., Vinyals, O., Kohli, P.: Graph matching networks for learning the similarity of graph structured objects. In: Chaudhuri, K., Salakhutdinov, R. (eds.) Proc. of the 36th Int. Conf. on Machine Learning, ICML 2019. Proceedings of Machine Learning Research, vol. 97, pp. 3835–3845. PMLR (2019)
12. Ontañón, S.: An overview of distance and similarity functions for structured data. CoRR **abs/2002.07420** (2020), http://arxiv.org/abs/2002.07420
13. Shemshadi, A., Sheng, Q.Z., Qin, Y.: Efficient pattern matching for graphs with multi-labeled nodes. Knowl.-Based Syst. **109**, 256–265 (2016)
14. Sorlin, S., Solnon, C.: Reactive tabu search for measuring graph similarity. In: Brun, L., Vento, M. (eds.) Graph-Based Representations in Pattern Recognition, 5th IAPR International Workshop. LNCS, vol. 3434, pp. 172–182. Springer (2005)
15. Zhu, L., Ng, W.K., Cheng, J.: Structure and attribute index for approximate graph matching in large graphs. Information Systems **36**(6), 958–972 (2011)