# GPU-Based Graph Matching for Accelerating Similarity Assessment in Process-Oriented Case-Based Reasoning*

Maximilian Hoffmann[1,2], Lukas Malburg[1,2], Nico Bach[1], and Ralph Bergmann[1,2]

[1] Artificial Intelligence and Intelligent Information Systems, University of Trier, 54296 Trier, Germany, http://www.wi2.uni-trier.de
[2] German Research Center for Artificial Intelligence (DFKI) Branch University of Trier, Behringstraße 21, 54296 Trier, Germany
{hoffmannm,malburgl,s4nibach,bergmann}@uni-trier.de; {maximilian.hoffmann,lukas.malburg,ralph.bergmann}@dfki.de

**Abstract** In *Process-Oriented Case-Based Reasoning (POCBR)*, determining the similarity between cases represented as semantic graphs often requires some kind of inexact graph matching, which generally is an NP-hard problem. Heuristic search algorithms such as A* search have been successfully applied for this task, but the computational performance is still a limiting factor for large case bases. As related work shows a great potential for accelerating A* search by using GPUs, we propose a novel approach called *AMonG* for efficiently computing graph similarities with an A* graph matching process involving GPU computing. The three-phased matching process distributes the search process over multiple search instances running in parallel on the GPU. We develop and examine different strategies within these phases that allow to customize the matching process adjusted to the problem situation to be solved. The experimental evaluation compares the proposed GPU-based approach with a pure CPU-based one. The results clearly demonstrate that the GPU-based approach significantly outperforms the CPU-based approach in a retrieval scenario, leading to an average speedup factor of 16.

**Keywords:** GPU Graph Matching · A* Search · Similarity-Based Retrieval · Semantic Workflow Graphs · Process-Oriented Case-Based Reasoning

## 1 Introduction

During the past 15 years, the application of workflows, initially proposed to automate business processes, has significantly expanded to new areas such as scientific workflows [31], medical guideline support [22], cooking recipes [25],

* The final authenticated publication is available online at https://doi.org/10.1007/978-3-031-14923-8_16

robotic process automation [1], or IoT environments [4] such as manufacturing [13, 24, 28, 29]. Workflow management, which involves creating, adapting, optimizing, executing, and monitoring them, is getting increasingly challenging in these areas and provides ample opportunities for Artificial Intelligence (AI) support. During the past decade, *Case-Based Reasoning (CBR)* [2, 27] demonstrated various benefits in the workflow domain, which led to the subfield of *Process-Oriented Case-Based Reasoning (POCBR)* [5] addressing the integration of process-oriented information systems with CBR. Among other things, POCBR enables the experience-based creation of workflows by reusing successful workflows from the past, stored as cases in a case base. Respective research conducted so far addresses the similarity-based retrieval of reusable workflows as well as their adaptation towards new problems [7]. In both CBR-phases, the similarity assessment between a query and case from the case base is an essential operation, applied frequently during the problem-solving process. The fact that in POCBR query and case are workflows typically represented as semantically annotated graphs, turns the similarity assessment problem into a graph similarity problem based on some domain-specific model of similarity. For addressing this problem, graph matching [10,26] approaches have demonstrated significant benefits, as in addition to the similarity value also a related (best) mapping between query and case graph is computed, which is quite useful, e. g., during adaptation [7]. The involved search process required to compute the mapping can be implemented with A* search [5] but as the problem is inherently NP-hard [11,26] large graphs can still lead to unacceptable computation times for practical applications. Particularly for case retrieval, in which a query has to be compared with each case in the case base, several POCBR approaches have been proposed to either speed up the computation of a single similarity assessment [30, 33] or to speed up the overall retrieval using MAC/FAC approaches [14, 19] based on faster but approximative similarity measures for preselection [8,18,20]. However, up to now, only moderate speed improvements could be demonstrated.

To address this problem, this paper explores the opportunity to achieve larger performance benefits based on a highly parallelized A*-based graph matching approach for execution on *Graphic Processing Units (GPUs)*, which is inspired by impressive performance improvements obtained for A* search for path-finding [9,35] based on GPUs. Therefore, we describe and analyze a novel approach called AMonG (A*-Based Graph Matching on GPUs) and thereby contribute to other recent endeavors to exploit GPU computing for CBR [3,23]. The following section presents foundations on the used semantic graph representation and similarity assessment via A* graph matching. Next, we describe the AMonG approach in detail, before Sect. 4 demonstrates an experimental evaluation comparing AMonG with the currently based CPU-based approach. Finally, Sect. 5 concludes with a summary of the results and a discussion of future work.

## 2 Foundations and Related Work

Our approach addresses similarity assessment between pairs of semantic workflow graphs via A*-based graph matching. Although the method is generic regarding the semantic graph representation, we use NEST graphs [5] for the presentation and evaluation. In the following, the NEST graph representation (see Sect. 2.1) as well as the similarity assessment between NEST graphs using A* search (see Sect. 2.2) is introduced. Additionally, we present related work that covers accelerating the similarity assessment between semantic graphs and generic A* search approaches on GPUs (see Sect. 2.3).

### 2.1 Semantic Workflow Graph Representation

In our approach, we use semantically annotated graphs named *NEST* graphs introduced by Bergmann and Gil [5] to represent workflows. A *NEST* graph consists of four elements: *N*odes, *E*dges, *S*emantic descriptions, and *T*ypes. Figure 1 illustrates a fragment of a cooking recipe. Task nodes represent cooking steps (e. g., `cut`) and data nodes specify the corresponding ingredients that are processed or produced during the execution of the cooking step (e. g., `tomato`). In addition, workflow nodes are used to represent general information about the cooking recipe, e. g., calories or the needed preparation time. Each task and data node is connected with their corresponding workflow node by a part-of edge. Control-flow edges define the execution order in a workflow and data-flow edges indicate that a task consumes or produces this data node. For example, the task `dice` processes `cucumber` in one piece and produces it again, but as diced cucumber. Semantic descriptions can be used to attach specific information to nodes and edges. They are based on a semantic metadata language (e. g., an ontology) that expresses domain-independent and domain-dependent knowledge. In the *NEST* graph illustrated in Fig. 1, semantic descriptions are used to express properties of the data nodes or tasks (e. g., the duration or auxiliaries of a cooking step).

### 2.2 State-Space Search by Using A*

By using the presented semantic graph format, it is possible to perform a graph matching between a *query graph* and a *case graph* to get the similarity value. We determine this similarity value with the approach proposed by Bergmann and Gil [5] that aggregates local similarities of mapped nodes and edges to a global similarity value according to the well-known local-global principle by Richter [27]. The local similarities are computed with a model that defines similarities based on the semantic descriptions of nodes and edges. The graph matching process is restricted w. r. t. *type equality*, which means that only nodes and edges of the same type (see Sect. 2.1) can be mapped to each other. Edge mappings are further restricted such that two edges can only be mapped to each other if the nodes linked by the edge are already mapped. However, several valid mappings are possible for two graphs and an algorithm has to be used for finding the

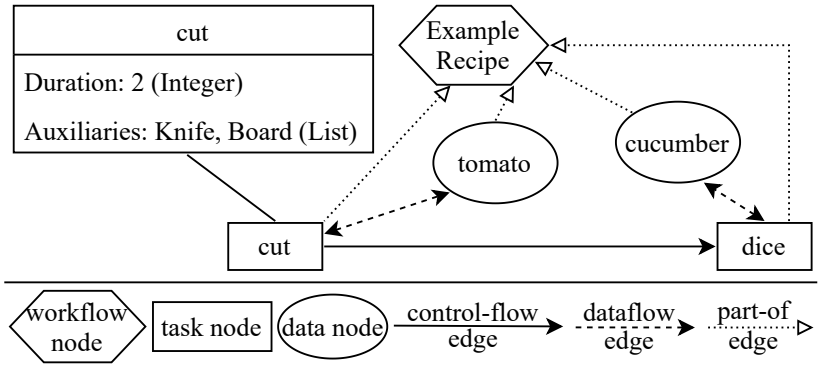


**Fig. 1.** A Cooking Recipe represented as a Semantic Workflow Graph.

best overall injective partial mapping $M$ that maximizes the global similarity $sim(QW, CW)$ between query workflow $QW$ and case workflow $CW$:

$$sim(QW, CW) = max\{sim_M(QW, CW) \mid \text{mapping } M\}$$

The algorithm that is used for this purpose is the A* search algorithm (originally proposed by Hart et al. [15]) that provides a complete and in particular optimal search. The way that A* is used in a graph matching environment is very similar to the original scenario of path search: The search goal is to find a mapping (solution) that maximizes the similarity between two semantic graphs. Starting with a set of unmapped nodes and edges of both graphs, the partial solutions are iteratively build up by adding possible legal mappings. This is done until all nodes and edges of the query graph are mapped. The process is guided by the following equation that evaluates a mapping solution: $f(M) = g(M) + h(M)$. The estimation of the similarity $f(M)$ of a mapping $M$ is given by the sum of the approved similarities of already mapped nodes and edges $g(M)$ and the estimation of the similarities of all unmapped nodes and edges computed by an admissible heuristic $h(M)$. Thereby, $g(M)$ represents the aggregated similarities of the nodes and edges that are already mapped. The function $h(M)$ is a heuristic estimation of the maximum similarity that can be achieved by mapping the currently not mapped nodes and edges. This means, that $f(M) = g(M)$ after the mapping process is finished and $f(M) = h(M)$ before the mapping process is started. Figure 2 depicts two possible mappings ($M1$ and $M2$) between the nodes of the query graph to the nodes of the case graph with their similarities. Please consider that, for simplicity reasons, only the node mappings are illustrated. In addition, it is important to note that the similarity between two graphs is usually not symmetric: Thus, it makes a difference whether the nodes and edges of the query graph are mapped to the case graph or the node and edges of the case graph are mapped to the query graph. The illustrated graph is a typical example for a query that is mostly only a partial graph that is a subgraph of the case workflow graph. We take a closer look at the data node `cheddar` from the

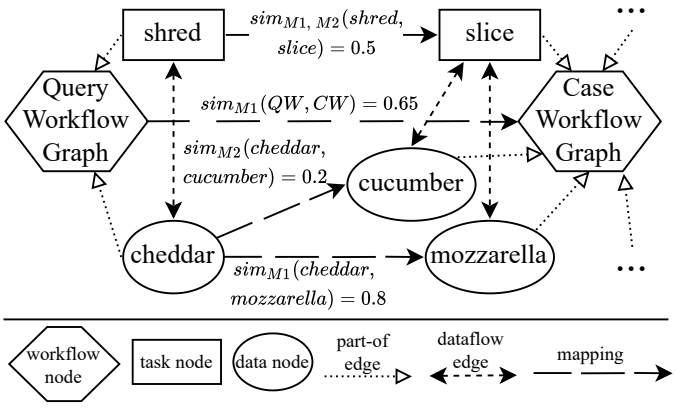


**Fig. 2.** Exemplary Mappings and Similarity Calculations between Two Semantic Workflow Graphs

query graph and its two possible mappings $M1$ and $M2$: The data node `cheddar` can due to the necessary *type equality* only be mapped to the other data nodes, i. e., `mozzarella` or `cucumber` in the case graph. If the data node `cheddar` is mapped to the data node `mozzarella` a similarity value of 0.8 can be reached. Whereas, a mapping from `cheddar` to `cucumber` results also in a legal mapping but with a lower similarity of 0.2. Thus, the first mapping increases the overall similarity ($sim_{M1} = 0.65$) between the two graphs more than the other mapping ($sim_{M2} = 0.35$).

### 2.3 Related Work

In this section, we present related work for our proposed approach. Thereby, we distinguish between: 1) related work from POCBR for accelerating case retrieval and similarity assessment by using deep learning or other techniques and 2) related work that uses GPU capabilities in CBR or for performing a state-space search with A*.

To accelerate the similarity assessment between two workflow graphs, several approaches have been proposed, mostly as a similarity measure to be used in the MAC-phase of a MAC/FAC retrieval approach [14]: Bergmann and Stromer [8] present an approach that uses a manually-modeled feature-vector representation of semantic graphs. By using this simplified representation, it is possible to efficiently determine an approximate similarity between two graphs. However, the manual modeling and knowledge acquisition effort for generating appropriate features is high. Klein et al. [20] and Hoffmann et al. [18] try to reduce the manual effort by using neural networks for similarity assessment that can automatically learn feature vectors for graphs. These approaches have good quality and performance characteristics. However, they are also only an approximation

of the mapping-based similarity, thus using them for case retrieval may lead to retrieval errors. They also require an offline training phase with high computational cost. In contrast to these approaches, Zeyen and Bergmann [33] present an enhanced version of the A* graph matching process introduced by Bergmann and Gil [5] to improve the search performance while still computing the exact similarity. The improvements consist of four aspects: 1) an improved initialization of the solutions also w. r. t. *type equality*, 2) a *case-oriented mapping* to swap the mapping direction if the case workflow has fewer nodes than the query workflow, 3) a *more-informed heuristic* to reduce the degree of overestimation and to make the $h$-values more precise, and 4) a *revised selection function* that decides when to map which nodes and edges during the process. However, even with these improvements of the matching process, it tends to be not possible to find optimal solutions in reasonable time for larger graphs.

A possible solution to this problem is GPU computing. To the best of our knowledge, CBR research dealing with GPU-based case retrieval is only rarely investigated [3, 23]. The motivation for this work are our previous findings [23] where GPU computing is used for case retrieval of feature-vector cases and achieves a speedup of 37 when compared to CPU-based retrieval. Agorgianitis et al. [3] also work with graph similarities in conjunction with GPU computation but they do not cover semantic annotations as part of the graph similarity computation and they do not use the approach in a retrieval context. Other related work performs a state-space search using A* on the GPU. Bleiweiss [9] presents an approach for accelerating the navigation planning and pathfinding in games. This work focuses on finding the shortest path of several agents, achieving a speedup of up to 24 times. Caggianese and Erra [12] propose a similar A* variant for multi-agent parallel pathfinding in a single graph. Individual threads represent different agents and search different paths in a shared graph by using a hierarchical graph abstraction. Zhou and Zeng [35] propose an A* algorithm that is able to run on a GPU in a massively-parallel fashion, reporting a speedup of up to 45 times. They explore the possibility of not only extracting a single state from the priority queue in each iteration, but rather extracting and processing multiple of the best states at once. All presented approaches are similar to our approach but they are not used for similarity assessment via graph matching. In addition, the gained speedups also promise a great potential for our use case.

## 3 AMonG: A*-Based Graph Matching on Graphic Processing Units

This section introduces our approach AMonG (A*-Based Graph Matching on GPUs) for performing an A*-based graph matching between semantic workflow graphs by exploiting the highly-parallel computing architecture of GPUs[3]. Our approach integrates the key concepts and the heuristic of the CPU-based A*

[3] Nico Bach formally described the approach in his bachelor thesis "Using Graphic Processing Units for A*-Based Similarity Assessment in POCBR" submitted 2021 at Trier University.

similarity search presented by Zeyen and Bergmann [33] into a modified version of the GPU-based A* search by Zhou and Zeng [35] where we reuse the structure of parallel search instances (see Sect. 2.3). We first introduce an overview of the approach with all associated components (see Sect. 3.1). Since the most important component of this architecture is the parallel graph matching algorithm, we describe its design in detail afterwards (see Sect. 3.2).

### 3.1 Overview and Components

The graph matching algorithm computes the maximum similarity for a *query graph* and a *case graph*. Both graphs are semantic NEST graphs, as introduced in Sect. 2.1. Due to the fact that NEST graphs have several types of nodes and edges, some mappings during the A* search are not legal, e. g., the mapping of a task node from the query graph to a data node in the case graph. For this reason, we compute all legal mappings between the individual nodes and edges of both graphs and their similarities (also called isolated mappings resp. *isolated similarities*) on the CPU and transfer them to the GPU. This way, the GPU-computation focuses on the pure graph matching process and not on the computation of similarities between nodes and edges. In addition, this also reduces the amount of data stored on the GPU since the semantic information of nodes and edges remain in the CPU memory. In order to parallelize the matching process, we introduce $n$ separate *search instances* that operate individually and in parallel for certain tasks (similar to [35]). Each of these instances operates on its own *priority queue* that stores *state descriptions* containing partially matched solutions. The search instances can be considered as separate A* search runs that run in parallel as often as possible but share their search progress in form of expanded states occasionally. The parameter $n$ has a high influence on the parallelism potential of the matching procedure. A higher value of $n$ corresponds to a higher degree of parallelism and possibly increased performance but comes with the risk of generating more overhead, e. g., for communication and exchange of solutions between the individual priority queues of the instances. The partially matched solution represented by a state description is denoted as $s$ and composed of a set of already *applied mappings* $A(s)$, i. e., all pairs of nodes and edges that are already mapped between query and case graph [5], and a set of *possible mappings* $P(s)$. $P(s)$ is computed by deleting all mappings from the isolated similarities in which a node or an edge is already mapped in $A(s)$ such that it cannot be mapped anymore. Prior to the iterative graph matching procedure, we initialize the priority queues by putting an empty solution, i. e., $A(s_i)$ is empty and $P(s_i)$ is equal to the isolated similarities, into one of the priority queues and leave all other priority queues empty. This is motivated by the findings of related work [35] to prevent the need for computationally-intensive deduplication. By starting with a single solution and deterministically adding new node or edge mappings, it is not possible to generate the same solution twice by different search instances.

### 3.2 Parallel Graph Matching

The matching procedure is depicted in Fig. 3. The process consists of three distinct phases that are executed iteratively for all search instances. These components are explained in more detail in the following.

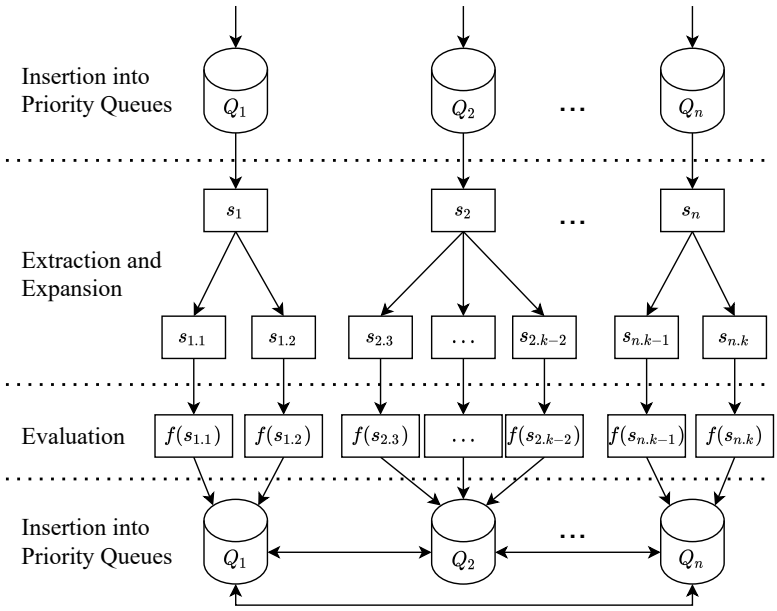


**Fig. 3.** Execution Phases of the GPU-Based Graph Matching Algorithm (Based on: [35])

The initial phase in the procedure consists of an *extraction* of the best partial solution $s_i$ by each search instance from its priority queue. The priority queues evaluate and return the best solutions according to their $f$-value, which is composed of the $g$-value and the $h$-value. Please note that although the extracted solutions are all optimal within the scope of the search instances, they are not guaranteed to be optimal in the global scope of all search instances. This might lead to non-optimal solutions being expanded, which can increase matching time. This also motivates a careful parameterization of the approach, in particular with the value of $n$. Each extracted solution $s_i$ with $i \in [1, n]$ is *expanded* by selecting the next node or edge to match in the query graph. The selected node or edge is then mapped to all legal nodes or edges of the case graph by using $P(s_i)$ that provides information which nodes and edges can be legally mapped according to their types (*type equality*). This leads to $k$ new solutions, denoted as $s_{i.j}$ with $j \in [1, k]$. Each $s_{i.j}$ is stored in a list specific to its search instance.

In the *evaluation* phase, the function $f$ of all previously expanded solutions is computed. Updating the $f$-value means updating the $g$- and $h$-values and is done with $A(s_{i.j})$ and $P(s_{i.j})$. The $g$-value can be computed by summing-up the similarities of all applied mappings $A(s_{i.j})$. The $h$-value is denoted as the estimated maximum similarity of all possible remaining mappings $P(s_{i.j})$. However, $P(s_{i.j})$ has to be updated after the expansion by removing mappings with already mapped nodes and edges before computing the correct value of $h$. In order to parallelize the computation of the $h$-value, we propose three different strategies: The first naive strategy ($\mathsf{Eval}_1$) only semi-parallelizes the phase by letting each search instance compute the $h$-value and the updated set of $P(s_{i.j})$ for its expanded mappings in *sequence*. This strategy is semi-parallel as it parallelizes across the search instances but the evaluation of each mapping is computed in serial within each instance. The second strategy ($\mathsf{Eval}_2$) aims at an improved load distribution by collecting the expanded mappings from all search instances and then invoking individual computations of $h$-value and $P(s_{i.j})$ for each expanded mapping $s_{i.j}$ in *parallel*. The third strategy ($\mathsf{Eval}_3$) follows $\mathsf{Eval}_2$ but additionally splits up the computation of the $h$-value and the update of $P(s_{i.j})$ into separate parallel subphases by evaluating each node and edge separately. The last two strategies aim at reducing the computations into small, parallelizable parts (*divide and conquer*) in order to improve load distribution. All strategies contribute to the trade-off between parallelism and communication overhead generated by the parallel execution.

In the *insertion* phase, the previously evaluated mappings are inserted into the priority queues of the individual search instances. The evaluated mappings should be distributed between the search instances in order to also distribute the workload for subsequent iterations of the search. An even distribution of partially matched solutions is crucial for the performance of this algorithm and has to be well coordinated. Although there are typically more insertions than extractions [9], priority queues might become empty or are not able to hold more solutions if the distribution is not well coordinated. We propose two different insertion strategies: The first strategy ($\mathsf{Ins}_1$) distributes the expanded mappings of one search instance into the priority queue of only one randomly chosen search instance. Thus, only fixed pairs of search instances share data with each other for one iteration. Consequently, less synchronization is needed but the solutions are not evenly distributed to all other search instances. In contrast, the second strategy ($\mathsf{Ins}_2$) distributes the expanded mappings of a single search instance evenly to the priority queues of *all other* instances. This achieves the maximum degree of distribution and aims to keep the similarities of the solutions in the priority queues uniformly distributed. However, accessing the priority queues of all instances simultaneously, requires a nearly-equal runtime of the instances in the previous expansion phase to avoid waiting for some instances. The insertion phase also has the purpose of checking the termination criteria of the algorithm. It terminates if one of the solutions from all priority queues is fully matched, i. e., all nodes and edges of the query graph are part of a mapping, and this

solution also features the maximum similarity value among all solutions, i. e., it satisfies the goal function.

## 4 Experimental Evaluation

In the experimental evaluation, we compare our AMonG approach with the CPU-based graph matching method [33] in the context of similarity-based retrieval. Our central claim states that using GPUs for similarity assessment via graph matching increases the performance compared to the CPU-based approach (see Sect. 2.3). We investigate the following hypotheses in our experiment:

**H1** Distributing all expanded mappings of one search instance into the priority queues of *all other* instances ($\mathsf{Ins}_2$) leads to a lower runtime in comparison to limiting the distribution by only inserting the mappings into the priority queue of another *random* instance ($\mathsf{Ins}_1$).

**H2** Evaluating the expanded mappings in parallel and parallelizing the computation of the $h$-value and the update of the possible mappings in strategy $\mathsf{Eval}_3$ increases the speedup compared to the other *evaluation* strategies $\mathsf{Eval}_1$ and $\mathsf{Eval}_2$.

In Hypothesis H1, we validate the proposed *insertion* strategies in our experiments. The hypothesis is derived from the assumption that the wider distribution by sharing the expanded mappings with all other priority queues ($\mathsf{Ins}_2$) leads to overall lower runtimes compared to less parallelization by inserting the expanded mappings only to *random* instances in strategy $\mathsf{Ins}_1$. Similar to this, we assume in Hypothesis H2 that the selected strategy in the *evaluation* phase should parallelize the workload as much as possible. For this reason, we assume that $\mathsf{Eval}_3$ increases the speedup compared to the other strategies. Thus, we expect that the experiments benefit more from an increased parallelization than the associated communication and synchronization overhead harms performance. In the following, we first describe our experimental setup (see Sect. 4.1) and subsequently the results of our experiments in Sect. 4.2 w. r. t. the hypotheses. Finally, a discussion and further considerations are presented in Sect. 4.3.

### 4.1 Experimental Setup

For our experiments, we implemented the previously presented approach in the open-source POCBR framework ProCAKE[4] [6], which is suitable for representing semantic graphs and enabling A*-based graph matching between them (see [33]). We use the bindings of JCUDA [32] for GPU-acceleration, since ProCAKE is implemented in Java. The case base for the experiments is an exemplary case base available in ProCAKE, containing 40 manually-modeled cooking recipes represented as NEST graphs [6]. These recipes are similar to the recipe that is illustrated in Fig. 1. To highlight the mapping complexity of this scenario, we present important properties of the case base in Tab. 1. The numbers

**Table 1.** Graph Size and Mapping Complexity of the Case Base

| | Graph Nodes | Graph Edges | Graph Node Mappings | Graph Edge Mappings |
|---|---|---|---|---|
| min | 9 | 21 | 70 | $9.8 \cdot 10^4$ |
| avg | 21.15 | 55.63 | $2 \cdot 10^8$ | $1 \cdot 10^{28}$ |
| max | 38 | 110 | $7 \cdot 10^{28}$ | $1.2 \cdot 10^{31}$ |

show the enormous search space of the mapping problem. The evaluation is restricted to similarity assessments between graphs from this case base that do not reach the memory limitations of the system (GPU and CPU memory). As already described in Sect. 2.2, query graphs are typically partial graphs with fewer nodes and edges than the case graphs. For this reason, we also restrict the experiments to similarities where this condition holds. The final number of evaluated similarity assessments is 704 where all other assessments either reach the memory limitations or do not meet the criteria of a query smaller than the case. Our central evaluation metric is the *speedup* in computation time that can be achieved by applying AMonG in addition to the CPU-based similarity assessment. AMonG is introduced such that it always computes the similarity in parallel to the pure CPU-based A* computation and considers the one that first finishes the computation (the slower computation can be stopped and discarded). Thus, the speedup is defined by the full computation time on the CPU divided by the minimum of the time of AMonG and the full computation time on the CPU. All experiments are repeated five times and averaged afterwards. The computation results of AMonG are verified by comparing them with the results of the CPU-based A* computation. The experiments are conducted on a computer running Windows 10 64-bit with an Intel i7 6700 CPU and 48 GB RAM. For the GPU, an NVIDIA GeForce GTX 1080 with compute capability 6.1 and 8 GB graphics RAM is used. In preliminary experiments, we determined that a number of $n = 2048$ priority queues is sufficient for our experiments.

### 4.2 Experimental Results

In this section, we present and discuss the results of our experiments. With Hypothesis H1, we want to examine which *insertion* strategy is best suited. Figure 4 depicts the runtime of all examined similarity computations on the GPU for both insertion strategies in relation to the number of expanded mappings. Both strategies are comparable w. r. t. their runtime for relatively small numbers of expansions. However, as the required number of expansions to find a solution increases, $\mathsf{Ins}_2$ provides a better runtime overall. Further, we investigate the total number of expanded mappings for both strategies: $\mathsf{Ins}_1$ expands fewer mappings than $\mathsf{Ins}_2$. This could be attributed to having more idle priority queues in the beginning of the search, as mappings are not spread out efficiently over the

[4] http://procake.uni-trier.de

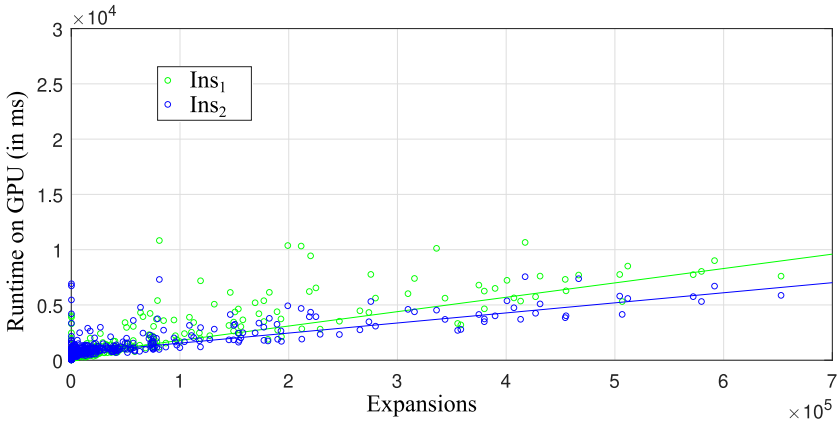


**Fig. 4.** Experimental Results of Different Insertion Strategies

priority queues, leading to fewer expansions in early phases. Due to the better distribution of the workload in early iterations, $\mathsf{Ins}_2$, by contrast, expands more mappings on average than $\mathsf{Ins}_1$. For our experiments, we confirm Hypothesis H1 as the runtime difference of $\mathsf{Ins}_2$ becomes larger as expansions grow and the runtime is overall lower than that of $\mathsf{Ins}_1$.

To check Hypothesis H2, we run the experiments by using the best *insertion* strategy from the first experiment $\mathsf{Ins}_2$ with all three proposed *evaluation* strategies ($\mathsf{Eval}_1$, $\mathsf{Eval}_2$, and $\mathsf{Eval}_3$). We measure the *average speedup* that is calculated with the speedups of all 704 similarity assessments. Additionally, the *average retrieval speedup* is given based on the mean of the conducted retrievals where the case base is queried with every graph. These values can be interpreted as the expected speedup for a single similarity assessment of a query-case pair and a typical retrieval, respectively. The results in Fig. 5 show that the strategy

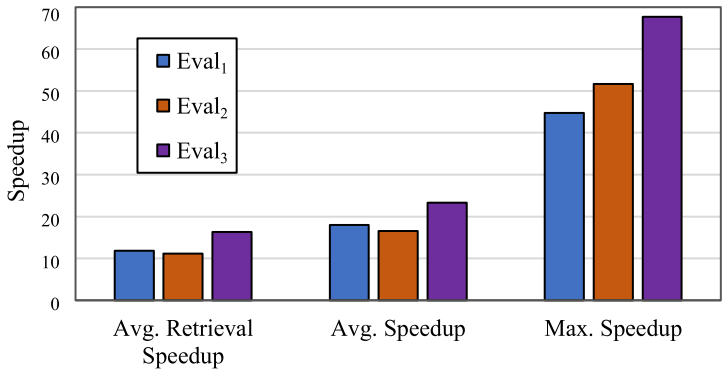


**Fig. 5.** Experimental Results of Different Evaluation Strategies

$\mathsf{Eval}_3$ achieves the highest values overall, with an average retrieval speedup of 16.29 and an average speedup of 23.33. This strategy also achieves a maximum speedup of a single similarity assessment of 67.72. For this reason, we clearly accept Hypothesis H2.

### 4.3 Discussion and Further Considerations

In our experimental evaluation, we have shown the performance gains that are possible by using the GPU for A* search and we support our central claim with the hypotheses. However, of the 704 similarity computations 379 were not accelerated using AMonG. Nevertheless, the similarity assessment on the GPU is faster in a typical retrieval scenario (see *retrieval speedup*), as 178 calculations can be performed with a speedup greater than 1 and less than 10, and 147 calculations can even be executed with a speedup greater than 10. Our approach thereby benefits from being used for larger problem spaces, where it clearly outperforms the CPU-based method. In our experiments, we determine that a similarity assessment, in which more than 2500 expansions on the CPU are needed, should be performed on the GPU. The required expansions on the CPU vary between 29 and 1,765,595 with an average of 44920.

## 5 Conclusion and Future Work

We presented the AMonG approach to accelerate A*-based similarity assessment between semantic graphs by using GPUs in POCBR. AMonG exploits the architecture of GPUs to evaluate multiple mappings in parallel during the search process. To factor in the trade-off between parallelism and synchronization overhead, we also discuss several strategies that contribute to either of these two aspects, making the approach customizable w. r. t. multiple scenarios. The evaluation investigates the potential of the proposed approach in speeding up similarity assessment between semantic workflow graphs. It demonstrates that our GPU-based approach significantly outperforms the CPU-based approach, leading to speedup factors of up to 67. GPU retrieval acceleration is novel to POCBR and is only little investigated in CBR (e. g., [3, 23]. The approach can serve as the basis for future applications dealing with a wide variety of CBR tasks. It can be especially useful in scenarios that deal with many similarity assessments such as the learning process of adaptation knowledge [7]. Further, it enables to process large case bases in complex domains that could be inefficient or even infeasible using only CPU-based methods (e. g., [33]).

To further enhance the performance of the overall system, we propose to use GPU-based and CPU-based matching in parallel. This could be achieved with a suitable method for distributing graph pairs to both components a-priori, e. g., by using a heuristic or a learning approach. It can lead to both approaches fully utilizing their potential. This can also be important in the context of a dependency-guided retrieval [21], where the search space can be particularly large due to consideration of dependencies between cases during the retrieval

phase. Furthermore, we aim to provide more flexibility with the algorithm's hyperparameters to better fit the computation for different domains of semantic graphs. This also emphasizes an automated method for hyperparameter optimization [16]. To validate the findings from this paper, future work should also cover more extensive evaluations with further POCBR domains (e. g., scientific workflows [34] or cyber-physical manufacturing workflows [24, 29]). In addition, the usage of the approach in combination with other techniques for accelerating similarity-based retrieval (e. g., [17, 18, 20, 23]) promises further benefits.

**Acknowledgments.** This work is funded by the Federal Ministry for Economic Affairs and Climate Action under grant No. 22973 SPELL.

## References

1. van der Aalst, W.M.P., Bichler, M., Heinzl, A.: Robotic Process Automation. BISE **60**(4), 269–272 (2018)
2. Aamodt, A., Plaza, E.: Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. AI Commun. **7**(1), 39–59 (1994)
3. Agorgianitis, I., et al.: Business Process Workflow Monitoring Using Distributed CBR with GPU Computing. In: 30th FLAIRS. pp. 495–498. AAAI Press (2017)
4. Janiesch et al., C.: The Internet of Things Meets Business Process Management: A Manifesto. IEEE Syst. Man Cybern. Mag. **6**(4), 34–44 (2020)
5. Bergmann, R., Gil, Y.: Similarity assessment and efficient retrieval of semantic workflows. Inf. Syst. **40**, 115–127 (2014)
6. Bergmann, R., Grumbach, L., Malburg, L., Zeyen, C.: ProCAKE: A Process-Oriented Case-Based Reasoning Framework. In: ICCBR Workshops. vol. 2567, pp. 156–161. CEUR-WS.org (2019)
7. Bergmann, R., Müller, G.: Similarity-based Retrieval and Automatic Adaptation of Semantic Workflows. In: Synergies Between Knowledge Engineering and Software Engineering, pp. 31–54. Adv. in Int. Syst. and Comp., Springer (2017)
8. Bergmann, R., Stromer, A.: MAC/FAC Retrieval of Semantic Workflows. In: 26th FLAIRS. AAAI Press (2013)
9. Bleiweiss, A.: GPU Accelerated Pathfinding. In: 23th EUROGRAPHICS/ACM SIGGRAPH. pp. 65–74. Eurographics Association (2008)
10. Bunke, H.: Recent developments in graph matching. In: 15th ICPR. pp. 117–124 (2000)
11. Bunke, H., Messmer, B.T.: Similarity measures for structured representations. In: 1st EWCBR. LNCS, vol. 837, pp. 106–118. Springer (1994)
12. Caggianese, G., Erra, U.: Parallel Hierarchical A* for Multi Agent-Based Simulation on the GPU. In: 19th Euro-Par Workshops. LNCS, vol. 8374, pp. 513–522. Springer (2013)
13. Erasmus, J., Vanderfeesten, I., Traganos, K., Grefen, P.W.P.J.: Using business process models for the specification of manufacturing operations. Comput. Ind. **123** (2020)
14. Forbus, K.D., Gentner, D., Law, K.: MAC/FAC: A Model of Similarity-Based Retrieval. Cogn. Sci. **19**(2), 141–205 (1995)
15. Hart, P.E., Nilsson, N.J., Raphael, B.: A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Trans. Syst. Sci. Cybern. **4**(2), 100–107 (1968)

16. Hoffmann, M., Bergmann, R.: Improving Automated Hyperparameter Optimization with Case-Based Reasoning. In: 30th ICCBR. LNCS, Springer (2022)
17. Hoffmann, M., Bergmann, R.: Using Graph Embedding Techniques in Process-Oriented Case-Based Reasoning. Algorithms **15**(2), 27 (2022)
18. Hoffmann, M., Malburg, L., Klein, P., Bergmann, R.: Using Siamese Graph Neural Networks for Similarity-Based Retrieval in Process-Oriented Case-Based Reasoning. In: 28th ICCBR. LNCS, vol. 12311, pp. 229–244. Springer. (2020)
19. Kendall-Morwick, J., Leake, D.: A Study of Two-Phase Retrieval for Process-Oriented Case-Based Reasoning. In: Successful Case-based Reasoning Applications-2, vol. 494, pp. 7–27. Springer (2014)
20. Klein, P., Malburg, L., Bergmann, R.: Learning Workflow Embeddings to Improve the Performance of Similarity-Based Retrieval for Process-Oriented Case-Based Reasoning. In: 27th ICCBR. pp. 188–203. Springer. (2019)
21. Kumar, R., Schultheis, A., Malburg, L., Hoffmann, M., Bergmann, R.: Considering Inter-Case Dependencies During Similarity-Based Retrieval in Process-Oriented Case-Based Reasoning. In: 35th FLAIRS. FloridaOJ (2022)
22. Lyng, K.M., Hildebrandt, T.T., Mukkamala, R.R.: From Paper Based Clinical Practice Guidelines to Declarative Workflow Management. In: BPM Workshops, LNCS, vol. 17, pp. 336–347. Springer (2008)
23. Malburg, L., Hoffmann, M., Trumm, S., Bergmann, R.: Improving Similarity-Based Retrieval Efficiency by Using Graphic Processing Units in Case-Based Reasoning. In: 34th FLAIRS. FloridaOJ (2021)
24. Malburg, L., Seiger, R., Bergmann, R., Weber, B.: Using Physical Factory Simulation Models for Business Process Management Research. In: BPM Workshops, LNBIP, vol. 397, pp. 95–107. Springer (2020)
25. Minor, M., Bergmann, R., Görg, S., Walter, K.: Adaptation of Cooking Instructions Following the Workflow Paradigm. In: ICCBR Workshops. pp. 199–208 (2010)
26. Ontañón, S.: An overview of distance and similarity functions for structured data. Artif. Intell. Rev. **53**(7), 5309–5351 (2020)
27. Richter, M.M., Weber, R.O.: Case-Based Reasoning: A Textbook. Springer (2013)
28. Rinderle-Ma, S., Mangler, J.: Process Automation and Process Mining in Manufacturing. In: BPM, vol. 12875, pp. 3–14. Springer (2021)
29. Seiger, R., Malburg, L., Weber, B., Bergmann, R.: Integrating process management and event processing in smart factories: A systems architecture and use cases. J. Manuf. Syst. **63**, 575–592 (2022)
30. Serina, I.: Kernel functions for case-based planning. Artif. Intell. **174**(16-17), 1369–1406 (2010)
31. Taylor, I.J., Deelman, E., Gannon, D.B., Shields, M.S. (eds.): Workflows for e-Science, Scientific Workflows for Grids. Springer (2007)
32. Yan, Y., Grossman, M., Sarkar, V.: JCUDA: A Programmer-Friendly Interface for Accelerating Java Programs with CUDA. In: 15th Euro-Par, LNCS, vol. 5704, pp. 887–899. Springer (2009)
33. Zeyen, C., Bergmann, R.: A*-based Similarity Assessment of Semantic Graphs. In: 28th ICCBR. LNCS, vol. 12311, pp. 17–32. Springer (2020)
34. Zeyen, C., Malburg, L., Bergmann, R.: Adaptation of Scientific Workflows by Means of Process-Oriented Case-Based Reasoning. In: 27th ICCBR. LNCS, vol. 11680, pp. 388–403. Springer (2019)
35. Zhou, Y., Zeng, J.: Massively Parallel A* Search on a GPU. In: 29th AAAI. pp. 1248–1255. AAAI Press (2015)