Chapter 9

Conclusion

The graph-based analysis of dynamic systems promises great insights into their working principles, key properties, and the time-dependent change thereof. It allows us to monitor, evaluate, and improve systems from various fields, including biology, chemistry, computer networks, transportation networks, social sciences, and online social networks. This enables us to understand how a system's characteristics change over time and what impact parameters have on their performance. This knowledge helps us to identify important components of a system, react to anomalies, and, thereby, guarantee its correct functionality.

The interpretation of analysis results, obtained from the graph-based analysis of a dynamic system, depends on the system itself and the model used to represent it as a graph. The selection of the model is, therefore, crucial. It determines the graph properties that should be computed in order to deduce the system's characteristics of interest.

We identified three system-specific tasks that must be solved for each system to realize its graph-based analysis. The system must be translated into a graph using an appropriate model (*T1*). Graph properties that are relevant to the modeled system and the characteristics of interest must be computed via the analysis of the resulting dynamic graph (*T2*). The obtained results must be interpreted to reflect the characteristics of the system and its components (*T3*).

The main challenge that arises in the process of graph-based analysis of dynamic systems is the performance of the analysis itself. It is influenced by many factors, including the algorithms used for graph analysis, the representation of the graph in memory, the frequency of the analysis, as well as the size of the graph. Due to all these influences, it is not straight forward to predict which algorithm or graph representation performs best for the analysis of a dynamic graph.

It is desirable to understand the performance of algorithms and data structures for graph analysis, develop new algorithms and approaches to speedup the analysis, and find ways to identify efficient graph representations. We formulated five research questions based on these general problems for the efficient analysis of dynamic graphs.

It is a challenging problem for an analyst to decide which approach or algorithms to use for the computation of certain properties for a given graph. It is crucial to understand the influence of different factors on the performance of these algorithms. An analyst can only make an informed decision if such knowledge is available. Therefore, we asked how algorithms for the analysis of dynamic graphs can be benchmarked and compared (*Q1*).

A vast number of snapshot-based algorithms exists for the computation of various graph properties. Stream-based algorithms promise great speedups for the analysis of dynamic graphs, especially for an analysis at high frequencies. For various graph properties, no stream-based algorithm has been developed yet. This provides the opportunity to investigate stream-based approaches for such algorithms to speed up the analysis of dynamic graphs. As a result, we asked how various graph measures can be computed efficiently for dynamic graphs (*Q2*).

While the performance of stream-based analysis scales well with higher frequencies, it does not scale well with the size of the graph, especially for complex measures. A general possibility to speedup graph analysis is the distribution of computational load among multiple workers. Hence, asked how the analysis of dynamic graphs can be speed up using distributed processing (*Q3*).

The use of different data structures for the representation of a dynamic graph in memory has a high impact on the overall performance. It influences the time required by the read operations of the metric computation as well as the write access during graph maintenance. It is not always easy to foresee which data structures perform best in a given scenario. Hence, it is crucial for the selection of efficient data structures to benchmark and compare different data structures for the representation of dynamic graphs. Therefore, we asked how different graph representations for dynamic graphs can be benchmarked and compared (*Q4*).

The selection of the most efficient data structures can be supported by their benchmark and comparison. It is not straight-forward to predict which data structure performs best in a given scenario, even with this knowledge. Hence, we asked how the most efficient representation of a dynamic graph can be determined (*Q5*).

In this thesis, we investigated all five research questions and presented four related contributions:

1. A benchmarking framework for dynamic graph analysis,
2. three novel algorithms that enable the efficient analysis of dynamic graphs,
3. an approach for the parallelization of dynamic graph analysis, and
4. a novel paradigm to select and adapt the data structures for dynamic graph analysis.

Furthermore, we presented use cases for the graph-based analysis of dynamic systems from three different fields: social, computer, and biological networks. They served as examples for demonstrating the three system-specific tasks *T1*, *T2*, and *T3*. We summarize our contributions in the following.

Benchmarking Framework We require means to benchmark and compare graph analysis algorithms (*Q1*) as well as graph representations (*Q4*) to understand their differences and judge their usefulness for different scenarios. A benchmarking framework should also support the analysis of dynamic graphs in general, the development of new algorithms (*Q2*), and the design of graph models.

We deduced twelve requirements from these high-level features and discussed the degree to which existing frameworks achieve them. While they provide means to represent dynamic graphs, none allows for the exchange of data structures used to represent them in memory, a prerequisite for their benchmark and comparison. All considered frameworks provide the analysis using snapshot-based approaches. Only two of them enable the analysis using stream-based approaches. This excludes most of them as a benchmarking framework for the comparison of multiple approaches.

We developed a benchmarking framework for the analysis of dynamic graphs, called *Dynamic Network Analyzer* (DNA), and presented it in Chapter 4. It provides the generation of dynamic graphs of various types and arbitrary sizes to aid in the benchmarking and comparison of algorithms and data structures. DNA also supports the development of new algorithms of different types with verifications for their correctness and the automatic determination of the precision of their results. Its extensive visualization components for dynamic graphs and analysis results enables us to monitor them during the analysis.

Overall, we presented a new benchmarking framework for the analysis of dynamic graphs. It provides means to benchmark and compare the performance of algorithms for the computation of graph properties for different approaches (*Q1*). The data structures used to represent a dynamic graph in memory during analysis can be freely exchanged, benchmarked, and compared (*Q4*). The framework supports the development of new algorithms for the analysis of dynamic graphs using different approaches. DNA thereby provides the foundation for investigating and developing new efficient algorithms (*Q2*).

Algorithms The performance of dynamic graph analysis highly depends on the algorithms used to compute the graph measures of interest for a dynamic graph. In Chapter 5, we presented and evaluated new stream-based algorithms to speed up the analysis of dynamic graphs and thereby provide means to analyze them at high frequencies (*Q2*). We introduced novel stream-based algorithms for the computation of degree distribution, rich-club coefficient, and the frequencies of k-vertex motifs.

We presented the snapshot-based algorithm DD_S and developed DD_U, its stream-based counterpart, for the computation of the degree distribution.. The runtime of DD_S depends on the number of vertices in the graph and is independent of the batch size. The runtime of DD_U has a complexity of $O(1)$ for processing edge removals and edge additions while processing vertex removals requires $O(d_{max})$ time. Hence, the performance of DD_U depends on the batch size and is independent of the graph size. DD_S performs best for the analysis of dynamic graphs where many changes occur between snapshots while DD_U should be used in case the batch size is small relative to the graph size.

We developed the stream-based algorithm RCC_U^k for the computation of the rich-club coefficient and presented its snapshot-based counterpart RCC_S^k. The performance of RCC_S^k only depends on the current graph state and has a complexity of $O(|V| + |RC^k| \cdot d_{max})$. With a complexity of $O(k)$ and $O(k \cdot d_{max})$ for processing an edge addition and removal, RCC_U^k outperforms RCC_S^k in most cases, especially for an analysis at high frequencies.

For the computation of k-vertex motif frequencies in dynamic graphs, only snapshot-based algorithms have been developed. They are not capable of analyzing the motif frequencies in dynamic graphs at high frequencies. We developed the first stream-based algorithm, called StreaM$_k$. It is based on the insight that the addition or removal of edges in a dynamic graph can lead to the composition of a new motif, the decomposition of an existing one, or the transformation of one motif into another. We compared its runtime for the analysis of dynamic graphs to four existing snapshot-based algorithms: Fanmod, G-Tries, Kavosh, and ACC. On synthetic dynamic graphs, StreaM$_k$ achieves speedups between 6.25× and 19,043×. We also evaluated the performance on dynamic graphs generated from four MD trajectories with different distance thresholds. Here, StreaM$_k$ outperforms the existing approaches by up to 2882×. As the first stream-based algorithm for counting motif frequencies, StreaM$_k$ allows for the analysis of dynamic graphs at a high frequency. Thereby, it enables the analysis of MD trajectories modeled as dynamic graphs and provides a great tool for the in-depth investigation of molecular dynamics.

Overall, we showed the great potential of stream-based algorithms to speed up the analysis of dynamic graphs compared to snapshot-based approaches. We developed three new stream-based algorithms: DD_U, RCC_U^k, and StreaM$_k$. We evaluated them in different scenarios and compared their performance to existing snapshot-based algorithms.

Parallelization of Dynamic Graph Analysis The performance of dynamic graph analysis highly depends on the graph's size and the complexity of considered graph measures. The analysis does not scale well with an increase in graph size, especially for complex graph measures. Existing approaches for the parallelization of dynamic graph analysis distribute the computation among multiple workers. They all operate on the same state of the graph and thereby do not allow the computation using stream-based algorithms. Therefore, we investigated the problem how the analysis of dynamic graphs could be speed up using parallelization (*Q3*).

We developed and presented a novel approach for the distributed processing of dynamic graphs, called *parallel Dynamic Graph Analysis* (pDNA) in Chapter 6. The computational workload is distributed among workers based on a partition of the vertex set. Corresponding subgraphs are assigned to each worker, which computes the respective measure on its local graph view. The changes to the main graph are propagated accordingly to the workers. The results from all workers are then aggregated into the measures for the whole graph for each point in time in a collation step.

We identified six problems that should be investigated in order to implement the distributed analysis using the conceptual design of pDNA. So far, we investigated three problems in detail: the creation of different subgraph types (*P2*), the maintenance of subgraphs over time (*P3*), and the collation of the results of all workers into the results for the main graph (*P6*). We proposed three subgraph types and developed a partitioner capable of maintaining them over time. We developed six algorithms for the collation of betweenness centrality, all-pairs shortest paths, clustering coefficient, weakly connected components, and degree distribution. For the remaining problems, we used the same solutions employed by existing approaches. We applied hash-based partitioning to partition the set of vertices (*P1*) and assign new vertices to workers (*P4*). We used snapshot-based algorithms for the computation of graph measures at each worker (*P5*).

We evaluated the performance of pDNA for the analysis of five metrics: betweenness centrality, all-pairs shortest paths, clustering coefficient, weakly connected components, and degree distribution. We analyzed three datasets: a growing graph of a social network and two instances of a constantly changing biological network. We distributed graphs and computational work among up to 32 workers. Our results show that pDNA achieves great speedups when increasing the number of workers for complex metrics like betweenness centrality, all-pairs shortest paths, and clustering coefficient. For simple measures like the degree distribution, the overhead of the actual distribution exceeds the gains achieved for faster computations.

With the introduction of pDNA and our first implementation on-top of DNA, we provided a great starting point to investigate the remaining problems in the future. We already implemented many partitioning strategies that can be used as a basis to investigate their impact on the performance of pDNA and test new ideas (*P1*). In addition, we also added different approaches for the assignment of vertices to workers over time (*P4*). Furthermore, DNA provides the computation of a large number of metrics using snapshot-, batch-, and stream-based algorithms. Hence, it provides all necessary tools to investigate the impact of their use in different scenarios (*P5*). Furthermore, DNA includes a large number of generators for dynamic graphs of various types and sizes. Hence, it is straight-forward to test the performance for larger graphs. In addition, the relation between graph type, algorithms for metric computation, partitioning strategies, vertex assignments, and collation algorithms can be investigated.

Data Structure Selection The selection of data structures, used to represent a dynamic graph in memory, has a high impact on the performance of an analysis. They influence the runtime of algorithm execution and the maintenance of the dynamic graph over time. The performance for the execution of single operations on data structures is well understood and can be investigated using benchmarks. The sizes of data structures and the frequencies of operations executed on them are not easy to foresee in the context of dynamic graph analysis. This makes it hard to predict which data structures perform best for the analysis of certain graph measures on a specific dynamic graph. Therefore, we developed and presented a novel paradigm for the selection and adaptation of the data structures for dynamic graph analysis in Chapter 7. It determines efficient data structures to represent a dynamic graph in memory during analysis (*Q5*).

We proposed a compile-time approach for optimizing these data structures. As a case study, we performed a measurement study of seven data structures, fitted estimation functions from the results, implemented our approach on top of DNA, and evaluated it using real-world datasets. Our results show that our optimization achieves speedups of up to 5.4× over basic configurations on real-world datasets.

The data structure configuration proposed by our approach outperformed all seven default configurations for the computation of all metrics for a constant workload. For non-constant workloads, we achieved speedups in many but not all cases. Thereby, our approach is well-suited for improving the analysis of dynamic graphs with a constant workload but not capable of adapting to the drastic changes of list sizes that can occur in non-constant workloads.

To close this gap, we developed a new run-time approach for the adaptation of graph data structures during the execution of an application. It ranks data structure configurations based on their expected performance and exchanges them during runtime if a performance gain is expected. We analyzed the performance of our approach using a synthetic workload designed to capture most operations and generate a non-constant workload. In this scenario, our approach performed as expected and achieved speedups over basic configuration of up to 7.3×.

Overall, we presented a novel paradigm for the selection and adaptation of data structures for dynamic graph analysis. The compile-time approach achieves speedups up to 5.4× over basic configurations on real-world datasets in the case of constant workloads. The run-time approach achieves speedups up to 7.3× on synthetic, non-constant workloads.

Use Cases Three system-specific tasks must be solved for each dynamic system to perform its graph-based analysis: modeling the system as a graph (*T1*), computing graph properties relevant to the system (*T2*), and interpreting the results to deduce characteristics of the underlying system (*T3*). We presented three examples of dynamic systems from the fields of social, computer, and biological networks in Chapter 8. Their graph-based analysis served as examples to demonstrate the three system-specific tasks and the expressiveness of the graph-based analysis of dynamic systems.

As the first use case, we analyzed an instance of the *PGP Web-of-Trust* (WoT), a social trust graph. We interpreted the properties of a single snapshot and deduced generative principles that explain how these properties can evolve in a graph that is grown over time. We developed two models using these principles as design guidelines. The growth model WoT_{gr} allows us to grow any input graph with the characteristic principles identified in the analysis by adding vertices and edges one after the other. The community-based model WoT_{com} enables us to create WoT graphs without an initial graph by creating a set of separate communities first and interconnecting them afterwards. We evaluated our models and compared them to existing graph models. Even the most promising candidate, a modification of the Forest Fire model, did not match the properties of a real-world WoT as good as our models. Therefore, we believe that WoT_{gr} is the first model to actually reflect the user behavior that contributes to the growth of a WoT.

Existing approaches for intrusion detection analyze network traffic on a packet or netflow level. This prohibits them from the observation of complex communications patterns that can occur during an attack. We proposed a new approach for intrusion detection based on dynamic graphs, called *graph-based Intrusion Detection System* (gIDS) as our second use case. It models netflows as dynamic graphs and uses the analysis results as features to train classifiers for the detection of intrusions. We evaluated the capability of gIDS to detect intrusions using the DARPA'98 dataset. We compared the results using a naive feature generator as baseline that extracts features based solely on netflows. In our evaluation, gIDS achieves a detection rate of up to 98.62 % compared to a maximum detection rate of 53.33 % for the baseline feature generator. These results indicate that modeling network communication as a graph is more expressive than naive approaches that only consider local measures based on the properties of single netflows.

We analyzed biological networks originating from *Molecular Dynamics* (MD) as the third use case. We investigated MD trajectories from the simulations of two systems: an enzyme and SPC/E water. In the first example, we analyzed the frequencies of 4-vertex motifs. We showed that they are able to illustrate detailed properties of the investigated system in clear contrast to the commonly used root-mean-square deviation. For the MD trajectories of SPC/E water, we analyzed the frequencies of 3-vertex motifs and used them to approximate the number of interactions between the water molecules over time. We showed that there is a high correlation between these graph properties and experimental entropy values. This implies that the graph-based analysis of MD trajectories is capable of reflecting the thermodynamics of water accurately. Our results from both examples show that the representation of MD trajectories as dynamic graphs and their subsequent analysis are capable of expressing many desirable properties and are a promising way of analyzing them.

Overall, we investigated three use cases for the application of analyzing dynamic systems based on graph models. For all cases, we showed that the results obtained from such an analysis are expressive and reveal various characteristics of the analyzed system.

In this thesis, we investigated the graph-based analysis of dynamic systems. We advanced the efficient analysis of dynamic graphs with four contributions: a benchmarking framework, three stream-based algorithms, a parallelization approach, and a paradigm for selecting and adapting graph data structures. The benchmarking framework DNA provides great means to better understand the impact of various factors on the performance of dynamic graph analysis and supports the development of new algorithms. The novel stream-based algorithm StreaM$_k$ achieves remarkable speedups for the computation of k-vertex motifs in dynamic graphs of up to 19,043× for synthetic and 2882× for real-world datasets. Our novel compile-time approach for the selection of efficient graph data structures achieves great speedups of up to 5.4× over baseline representations. The novel run-time approach for the adaptation of graph data structures speeds up the analysis up to 7.3× compared to baseline configurations. Our approach pDNA for the distribution of dynamic graph analysis achieves great speedups when increasing the number of workers, especially for the computation of complex metrics in dynamic graphs.