J Vis (2023) 26:477–492
https://doi.org/10.1007/s12650-022-00885-0

REGULAR PAPER

Michael Burch · Huub van de Wetering · Günter Wallner · Freek Rooks · Olof Morra

Exploring the dynamics of graph algorithms

Received: 7 February 2022 / Revised: 23 July 2022 / Accepted: 31 August 2022 / Published online: 14 October 2022

Abstract In this paper, we describe an interactive visualization tool for representing the dynamics of graph algorithms. To reach this goal, we designed a web-based framework which illustrates the dynamics as time-to-space mappings of dynamic graphs. Such static diagrams of dynamic data have the benefit of being able to display longer time spans in one view, hence supporting the observer with comparison tasks which is challenging or even impossible for graph algorithm animations. Our tool can show details about how an algorithm traverses a graph step-by-step in a static and animated fashion, for graph algorithm exploration as well as educational purposes. The animation together with the time-to-space mapping hence forms an overview-and-detail approach. We also allow changing of speed, replaying, stopping, storing intermediate stages with parameter configurations, as well as measuring and monitoring performance and memory consumption to eventually identify bottlenecks in a graph algorithm. By using flight carrier data from the United States Department of Transportation and a network of autonomous systems we demonstrate how we used the tool to explore two standard graph-theoretic algorithms. Finally, we discuss scalability issues and limitations.
Keywords Algorithm animation · Algorithm dynamics · Graph visualization

1 Introduction

Algorithms are useful mechanisms in nearly any discipline of computer science to transform data. Graph algorithms (Euler 1741) are a special type thereof that process relational data, for example, to find an efficient or suitable solution to a graph-theoretic problem that is typically linked to a real-world application

M. Burch (✉)
Center for Data Analytics, Visualization, and Simulation, University of Applied Sciences, Ringstrasse 40, 7000 Chur, Graubünden, Switzerland
E-mail: michael.burch@fhgr.ch

H. van de Wetering · F. Rooks · O. Morra
Department of Mathematics and Computer Science, Technische Universiteit Eindhoven, PO Box 513, 5600MB Eindhoven, The Netherlands
E-mail: h.v.d.wetering@tue.nl

F. Rooks
E-mail: f.rooks@tue.nl

O. Morra
E-mail: o.morra@tue.nl

G. Wallner
Institute of Computer Graphics, Johannes Kepler University Linz, Altenbergerstraße 69, 4040 Linz, Austria
E-mail: guenter.wallner@jku.at

(von Landesberger et al. 2011). Finding shortest paths in a road network (Greilich et al. 2009), detecting connected components in a social network (Bedi and Sharma 2016), or identifying call hierarchies in a software system (Burch et al. 2017) are different applications.

Typically, a graph algorithm is running for a certain amount of time until it provides a solution to a given graph-theoretic problem. For many purposes a visual outcome is sufficient. However, it might be insufficient if the result is not understandable, wrong, or even misleading, and someone wants to understand the intermediate steps to identify the problem. Moreover, exploring the graph algorithm under different circumstances such as adding or deleting vertices, edges, or even changing weights might lead to different outcomes that are worth investigating and comparing. Existing algorithm visualizations have been mostly used for educational purposes (Baecker 1998) and typically only small examples are involved. In this paper, we focus on larger graphs and provide a way for exploring the graph algorithm dynamics in a visual way.

For this reason, we have developed a visual analysis tool that accepts graph data (Eades and Klein 2018) as input and provides solutions to an (extendable) repertoire of graph algorithms. The tool does not only provide a solution in a visual form but also offers the possibility to show the intermediate steps, either as a static time-to-space mapping of a dynamic graph (Beck et al. 2017; Burch et al. 2017) or in an animated fashion (Diehl and Görg 2002; Frishman and Tal 2004) as a time-to-time mapping. Together, these two views form an overview-and-detail approach (Cockburn et al. 2008) where the time-to-space mapping serves as an overview and the animation offers a detailed view on the operations performed in each step. During execution, the tool permits changes to the graph to allow the user to observe the impact of them.

The user can choose one or both mappings, apply interactions (Yi et al. 2007) to both of them, and can view performance metrics such as memory consumption and running times of the applied algorithm for intermediate steps and time spans. With this combination, we not only provide an overview about graph algorithms but also additional information about the intermediate steps. At the same time, the user can navigate to relevant steps in the algorithm, for example, by playing an animation covering a (short) period of time for illustrative purposes.

At the same time, our tool can still be used for educational purposes (Burch and Melby 2019) due to the combination of algorithm animation and static visualizations for the dynamic aspects. In the following, after reviewing related work, we first describe our interactive web-based visualization tool and then apply it to graph datasets of varying sizes and several runs of Dijkstra's shortest path algorithm (Dijkstra 1959) as well as Prim's algorithm for computing minimum spanning trees (Prim 1957). Hereby, we identify formerly summarized tasks and how they can be answered by using our tool. Finally, we discuss limitations and scalability aspects and provide a perspective on future challenges and open gaps.

This article is an extension of a paper published at VINCI 2021 (Burch et al. 2021). Compared to Burch et al. (2021) this extended version has the following additions while at the same time increasing the number of figures:

- We extended the related work section by reviewing more papers on algorithm animation as well as dynamic graph visualization (Sect. 2).
- To better understand the mathematical concepts behind this line of research we included a data model (Sect. 3.)
- We also extended the *tasks and design decisions* section to better illustrate the usefulness of the proposed technique (Sect. 4.2).
- To make a clearer comparison between the two major visual concepts we added a more thorough perspective on and description of time-to-space and time-to-time mappings (Sects. 4.3 and 4.4).
- Insights on performance measures are now included to provide an additional perspective on the graph algorithms (Sect. 4.5).
- An additional application example (Prim's algorithm) showcases the value of the approach further (Sect. 5.2).

2 Related work

Showing the dynamics of a graph algorithm can be viewed as the problem of visualizing a sequence of graphs, either as an animation or as a static representation of the graphs side-by-side (Beck et al. 2017). A graph algorithm takes as input a graph with vertices and edges and processes it step-by-step following the

rules given by the algorithm. Each processing step involves a certain subset of the input graph. Taking all those processing steps into account generates a sequence of graphs, i.e., a dynamic graph.

A major goal of this line of research is to provide an overview about the evolution of the dynamic graph in a way that it becomes easy and intuitive to explore, be it for trends, countertrends, or anomalies. This field has received considerable attention in the past (see Beck et al. 2017) and can be split into time-to-time mappings (animations where the time in the data is mapped to physical time) and time-to-space mappings (static representations of the evolution, where the time in the data is mapped to the available display space), apart from some other less frequently used concepts. The visualization of the dynamics of graph algorithms can fall into both major categories, typically depending on the users and the tasks to be solved. Despite the advances in the area of dynamic graph visualization, many challenges remain such as the visual depiction of graph algorithm dynamics.

2.1 Algorithm animation

The animation of algorithms goes back to 1984 to a system developed for teaching and research in computer science and mathematics (Brown and Sedgewick 1984) which also focused on the interaction with dynamically changing graphical representations (Brown and Sedgewick 1985).

The execution of algorithms always leads to a time-dependent dataset since the applied algorithm transforms or processes the underlying data in a certain—typically user-defined—way and the intermediate steps describe the changes over time, step-by-step. The visual outcome of the individual steps has been the focus of research in algorithm animation (Brown 1988; Stasko 1990a, b; Brown 1991), a field that was developed as a means to educate people about algorithmic concepts (Baecker 1998).

The general idea has been applied mainly to sorting algorithms. For example, the *sorting out sorting* videos (Baecker 1998) were used to demonstrate the animations of several sorting algorithms. Recently, Végh and Stoffová (2017) described methods using algorithm animations for teaching sorting algorithms. However, also route finding problems, or strategic game playing when illustrating possible movements and choices of players, are interesting applications to be illustrated by an algorithm animation, either to explore the internal workings of an algorithm or to teach and educate people. On the downside, algorithm animation was typically restricted to non-interactive videos in which it was not possible to change intermediate states of the input data. Moreover, an overview about what happens in the animation video is difficult to obtain, and hence, a more static representation of the dynamics is desirable (Tversky et al. 2002), in particular for graph animations (Archambault and Purchase 2016). A static representation takes on a crucial role if the user cannot track all the changes simultaneously, either because there are too many, the areas of interest are not highlighted, or several runs of an animation have to be compared over time.

The major contribution of our work does not lie in teaching and education in particular, but rather in providing a temporal overview about the internal workings of graph algorithms. To reach this goal, we provide a combination of time-to-time and time-to-space mappings for dynamic graphs to create a visualization system for graph algorithm animation. This combination provides an overview-and-detail design for the time dimension in graph data (Cockburn et al. 2008). By providing extra views in form of performance charts, the user can quickly browse to relevant time points in a running algorithm and can start the exploration processes from critical points that might be difficult to find with only a graph animation.

2.2 Time-to-time and time-to-space mappings

Time-to-time mappings for dynamic graphs have been a major line of research for several years (Diehl and Görg 2002; Frishman and Tal 2004). One of the biggest issues is to maintain the preservation of the mental map (Purchase et al. 2007; Misue et al. 1995) which can, for instance, be achieved by keeping a high degree of dynamic stability (Görg et al. 2004). However, the generated graph layouts for showing the animations do not scale well and do not easily support comparison tasks for longer graph sequences. This is neither possible for comparison within a single long graph sequence nor for comparison of graph sequences that result from several runs of a graph algorithm under different parameter settings but on the same input graph.

User studies have explored whether graph animations are useful approaches or not. For example, Archambault et al. (2011) compared them to non-animated techniques such as small multiples that show the same information but in one static view. They found that the non-animated small multiple variants showed a faster performance than the animated counterparts, while animated ones might lead to lower error

rates. Since both approaches have their own benefits and drawbacks, there is a tendency to support both major concepts.

One downside of time-to-time mappings is that graph animations have limitations when it comes to comparing individual time steps. In a graph animation, only one graph is visible at a time, and then, it is (typically) smoothly transformed into the next one in the sequence (Diehl and Görg 2002). Without a smooth animation, the human observer would lose track of, for instance, vertices and no time-varying patterns and insights in the dynamic graph data can be gained. However, if the graph layout is stable and only a small change is shown in each animation step, the animation might be a good choice for educational purposes (Tversky et al. 2002). Nevertheless, for temporal data exploration over longer time spans or even for comparison tasks, time-to-time mappings might be suboptimal.

In order to improve this and to support comparison tasks with a visualization of a graph sequence, time-to-space mappings have been introduced (Burch et al. 2011). They represent as many graphs as possible next to each other (in a single view), depending on the available display space. Recently, the graphs forming the graph sequence of a dynamic graph have been represented as bipartite graphs to provide a vertex-aligned scalable variant for dynamic graph visualization (Burch et al. 2011), see Fig. 1.

Another scalable variant was developed by Burch et al. (2017), presenting an interleaved edge splatted version of such a dynamic graph visualization that scales to thousands of graphs. This scalar field is smoothed and color coded (Burch et al. 2011), an idea originally presented for a static approach, inspired by edge splatting for graphs (van Liere and de Leeuw 2003). Abdelaal et al. (2018) later improved the approach by Burch et al. (2017) by integrating a vertical 1D clustering to reduce the number of link crossings and to convey more details on vertex group patterns and their changes over time. In our work, we make use of the bipartite dynamic graph visualization, but we only splat and interleave the graphs if the temporal sequence becomes too long in time, i.e., the graph algorithm produces too many time steps.

Since graph algorithms typically produce a large number of graphs in different states (subsets of the original input graph), we require a scalable approach that supports an overview about the graph dynamics. Hence, interleaving the graphs or aggregating them over time is a good strategy (Abdelaal et al. 2018) to provide this overview and to support comparison tasks. The static overview is consequently important as a starting point for further explorations (Shneiderman 1996), as a contextual overview, as well as a temporal navigation and annotation tool, also in context of the performance measures aligned with the time-to-space visualization.

Besides node-link diagrams, adjacency matrices were utilized as well (Elmqvist et al. 2008; Henry and Fekete 2006). We, however, have opted for node-link diagrams as we can depict long graph sequences using a bipartite layout.

3 Data model

We model a graph $G = (V, E)$ as a pair consisting of vertices $V := \{v_1, \ldots, v_n\}$ and edges $E := \{e_1, \ldots, e_m\} \subset V \times V$ Gross and Yellen (2005). If weights are attached to the edges we denote that by a weight function $f : E \longrightarrow \mathbb{R}$. The interpretation of a graph as a directed or undirected graph is left to the algorithm that processes it.

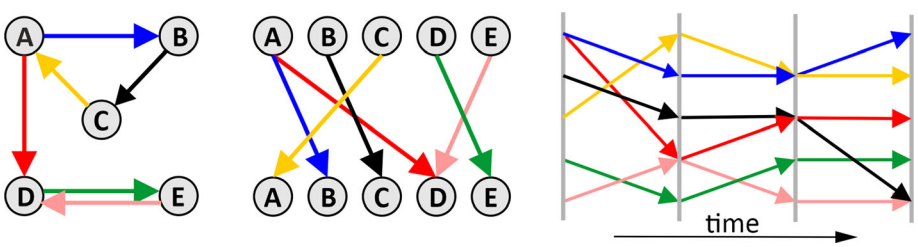


Fig. 1 An illustration showing how a sequence of graphs can be represented as a sequence of bipartite layouts (from left-to-right, adapted from Burch et al. (2011)). Left: A small graph G with five vertices and six edges represented using a traditional node-link layout. Center: The same graph G as bipartite layout in top-to-bottom reading direction. Right: Three graphs in a sequence using a left-to-right reading direction of which the leftmost graph is again graph G

3.1 Graph algorithms

A graph algorithm A is a finite computation process that not only takes a graph G as input but also some input parameters S, where S is an associative container that maps parameters to values. Applying an algorithm to a graph can then be denoted as $A(G, S)$.

If we consider Dijkstra's shortest path algorithm, denoted as $A_{\text{Dijkstra}}(G, S)$, it takes a (directed) graph G as input together with input parameters $S = \{(\text{ source }, v_s), (\text{ target }, v_t), (\text{ weights }, f)\}$, where $v_s = S[\text{ source}]$ is the source vertex, $v_t = S[\text{ target}]$ is the target vertex, and $f = S[\text{ weights}]$ is a weight function for the edges.

3.2 Dynamic graphs

Running graph algorithm $A(G, S)$ generates a sequence of state graphs $G_i = (V_i, E_i)$ from the input graph $G = (V, E)$. A state graph G_i is a sub-graph of G ($V_i \subset V$ and $E_i \subset E$) that typically contains all edges that have been processed in all time steps up to step $\sigma \cdot i$, where the number of steps $\sigma \in \mathbb{N}$ is a factor to control the sample frequency. The state graphs combined form a dynamic graph, which can be seen as a sequence of static graphs (Michail 2015):

$$\Gamma := (G_1, \ldots, G_N).$$

The algorithm also generates a sequence of associative containers with parameter and variable settings

$$\Sigma = (S_1, S_2, \ldots, S_N)$$

that represent the non-graphical state of the algorithm in each step. For instance, in our application examples, we collect time series with the total memory $S_i[\text{ memory}]$ in use at the end of step i as well as time series with the running time $S_i[\text{ dt}]$ of step i.

The pair (Γ, Σ) constitutes the input of our dynamic graph visualization tool. The dynamic visualization in the time-to-time mapping visualizes the state graphs directly, whereas the static visualization in the time-to-space mapping visualizes state graph updates, i.e., the changes between consecutive state graphs.

4 Approach

Our approach supports a time-to-time mapping (also for educational purposes) and a time-to-space mapping, mainly for getting an overview and as a means for navigating in the dynamic graph. Both approaches are interactively linked, and additional visual components conveying running times per step and memory consumption are also available. The combination of both concepts can be regarded as an overview-and-detail approach for the time dimension in dynamic graphs or, as in our case, for graph algorithm dynamics.

4.1 Graphical user interface

Figure 2 shows an illustration of the graphical user interface. It is divided into three major parts: the parameter and data input panel on the left ❶, the visual analysis panel ❷ in the center where the time-to-space mapping of the dynamic graph (for the animation steps) is shown, and the animation panel ❸, showing the input graph in a hierarchical layout. The time-to-space mapping for the dynamic graph is aligned with the running times and/or memory consumption charts ❹ to give an indication about when the graph algorithm might produce severe performance or memory issues. An animation step slider ❺ can be used to start an animation at a given point in time and to restrict the time interval for the time-to-space mapping.

Both approaches are linked as coordinated views, and there are various ways to interact with those views and visualizations (see Fig. 3). For example, selecting nodes or links in one view highlights those in the other as well. In the case of the time-to-space mappings, the time granularity, the vertical and horizontal display region (zoom in and out), and the color coding can be changed. For the time-to-time mappings, for example, the animation speed can be altered, node positions can be manually modified, or even the whole layout can be changed.

4.2 Tasks and design decisions

When starting this line of research we identified a certain repertoire of typical tasks for graph algorithm analysis (Lee et al. 2006). We discussed those tasks and grouped them together into five major categories:

- *Involved elements (T1):* Which vertices and edges are involved in an algorithm over time? Which vertices and edges are frequently or seldom visited?
- *Performance issues (T2):* Is there a running time or memory consumption peak which might indicate bottlenecks at a certain time point/period?
- *Comparisons (T3):* What are the differences in several runs of the same algorithm (for different parameter settings or different input graphs)? What are the differences for several algorithms on the same input graph?
- *Correlations (T4):* Are there correlations between the performances and different algorithmic processing strategies?
- *Temporal events (T5):* Are there time periods in which we find a higher activity than in others? Are there different stages in a graph algorithm? Are there temporal patterns or anomalies in the performance indicators?

Based on these tasks, we made the following design decisions.

- *Web-based:* The visual analysis tool should run in a web browser and should be accessible online. This is of great benefit if graph data, algorithms, and results have to be shared.
- *Graph algorithms:* We currently support two algorithms (Prim's and Dijkstra's), but additional algorithms can be added in future.
- *Parameter selection:* Each algorithm can be executed with a certain parameter configuration. Vertices, edges, and weights can be modified for each run.
- *Storing of steps:* Intermediate steps can be stored to explore them later or for showcasing the most important aspects about an algorithm run to an audience (e.g., for educational purposes).
- *Extra data generation:* Performance measures are important for analyzing the internal aspects of a graph algorithm, for instance, memory consumption, running times, or power consumption.

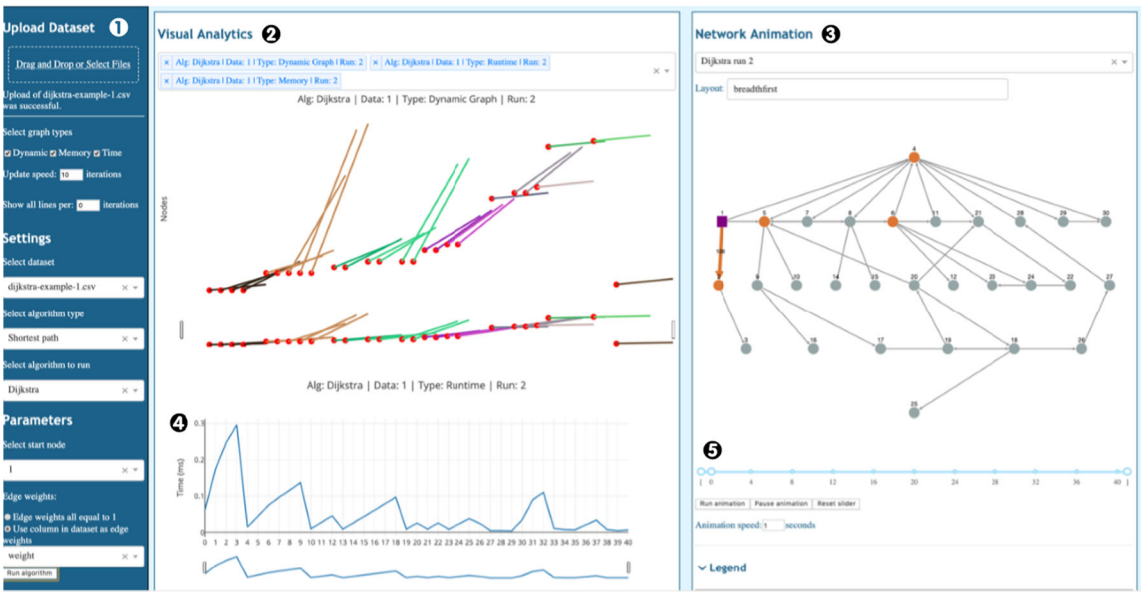


Fig. 2 The graphical user interface of the visualization tool with the parameter and data input panel ❶, the visual analysis panel ❷, and animation panel ❸ showing a dynamic graph visualization for Dijkstra's algorithm applied to a small artificially generated graph. Performance charts ❹ are temporally aligned with the time-to-space mapping, and an animation step slider ❺ can be used to start the animation at a certain point in time. The underlying graph consists of 29 vertices and 41 edges, and the graph algorithm produces 40 time steps

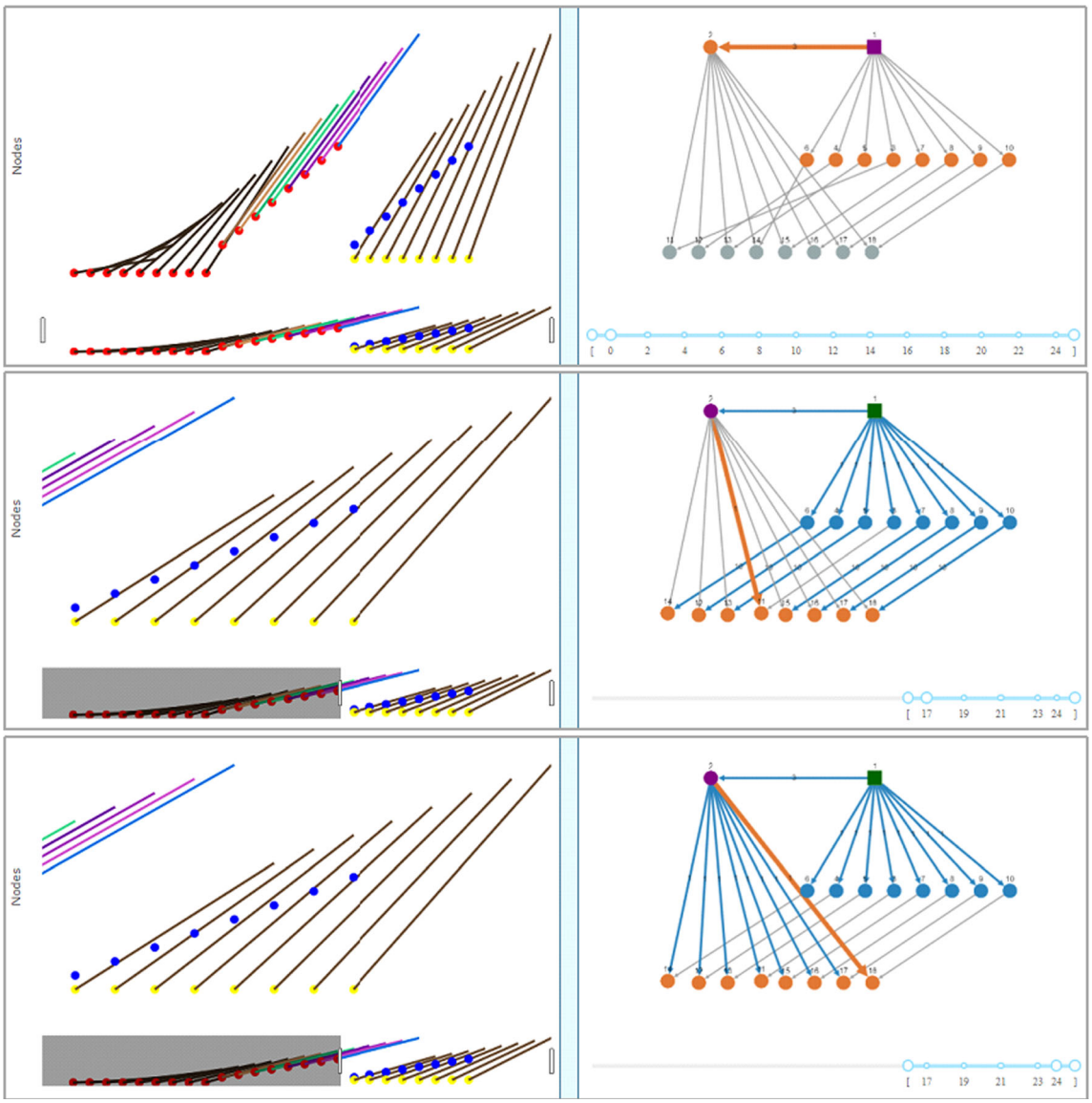


Fig. 3 The time-to-space mapping (left) and the time-to-time mapping (right) are linked. The animation step slider can be used to jump from one animation step to another and to restrict the time interval to be displayed in the time-to-space mapping. The first row shows the complete time interval on the left and the status after the first step on the right. In the bottom two rows, the left two images show the (same) time sub-interval, while the images on the right show a step at the beginning and one at the end of the sub-interval in which new shortest paths to the orange nodes are discovered via the purple node

- *Brushing and linking:* The provided time-to-space and time-to-time visualizations should be displayed next to each other to support interactive linking and to be able to observe the impact of an effect in one visualization directly in the other.
- *Animation features:* The animated diagram should support starting, stopping, and replaying. Moreover, selecting a specific time step and highlighting it in the overview-based time-to-space mapping are supported for contextual reasons.
- *Scalability:* Since a graph algorithm can quickly produce various iterations, i.e., long graph sequences, consisting of many vertices and edges, we require a visually and algorithmically scalable dynamic graph visualization for the time-to-space mapping.

4.3 Time-to-space mapping

For the visualization of a dynamic graph $\Gamma = (G_1, \ldots, G_N)$ that has been generated by a graph algorithm, we use a bipartite representation to give an overview of the state graph updates ΔG_i. The vertices are vertically aligned equidistantly, edges are pointing from left to right, and the time axis is leading from left to right. This is to conform with the approach introduced by Burch et al. (2011).

Figure 1 illustrates a small artificial example of a dynamic graph as a bipartite left-to-right time-to-space mapping, while Fig. 4 shows a time-to-space mapping for an application of Dijkstra's algorithm in which edges (u, v) that are used in a shortest path at some point of time are shown from left to right. In the color coding used for the time-to-space mapping if edge (u, v) has a red marker at node u, v is a newly discovered node; if there is a green marker at u, (u, v) replaces edge (w, v) in the shortest path toward v; node w then gets a blue marker. Hence, at each time step, a shorter path is detected, and a blue marker for the old node and a green marker for the new one are shown. The color of the edge is given by the numeric node ID.

Figure 5a shows an example of a time-to-space mapping generated with our visualization tool (in an interleaved fashion to make it more visually scalable along the time dimension). In Fig. 5b, we see a zoomed-in section, which is aimed at supporting the inspection and detection of details, for example, on a finer temporal granularity. Such a dynamic graph visualization has the benefit that we can show arbitrarily large graphs, since it can be interleaved and splatted in case the graph sequence gets very long (as has been illustrated by Burch et al. (2017)). Although such an interleaved dynamic graph visualization does not show details, it is still useful for rapidly detecting dynamic visual patterns in time-dependent graph data. In particular, for the dynamics of a graph algorithm, it provides a temporally scalable overview and can guide a viewer to the time points or time periods of interest, for instance, to play an animation starting from this point in time. Hence, the combination of time-to-space and time-to-time mappings serves as an overview-and-detail approach for algorithm dynamics.

Moreover, a time-to-space mapping has several advantages compared to a time-to-time mapping, for example, providing an overview, supporting navigation tasks (knowing where to start a replay), giving contextual information (with an animated time slider if the animation is running). Annotations over time can also be made to jump to intermediate, possibly important, steps in the algorithm animation. Finally, general interactions can be applied in an easier way in a static time-to-space mapping. An animation, in contrast, has to be stopped first to conveniently be interacted with.

4.4 Time-to-time mapping

If an algorithm and a graph are selected, we present the (static) input graph in a default node-link layout based on force-directed placement of the vertices. However, the user can select a certain graph layout which we support as a built-in JavaScript code. The selection of the layout can also be changed during the animation. Moreover, nodes can be selected and moved around to, for instance, achieve a better view on a certain part of the graph that the algorithm is currently processing.

Figure 6 illustrates the processing steps of Dijkstra's algorithm on top of a node-link diagram using color coding to indicate the states of the nodes and edges. In the algorithm, for the current purple node, each of its neighbors (orange) is visited, in no particular order, and added to the queue of nodes that need to be handled. Then, the most promising node from that queue is chosen as the next purple node. In the example, the start

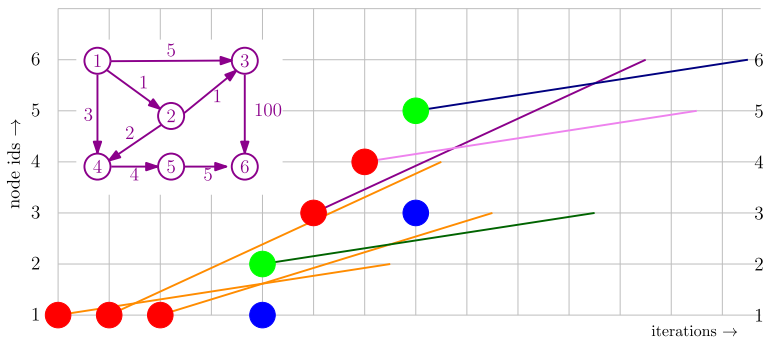
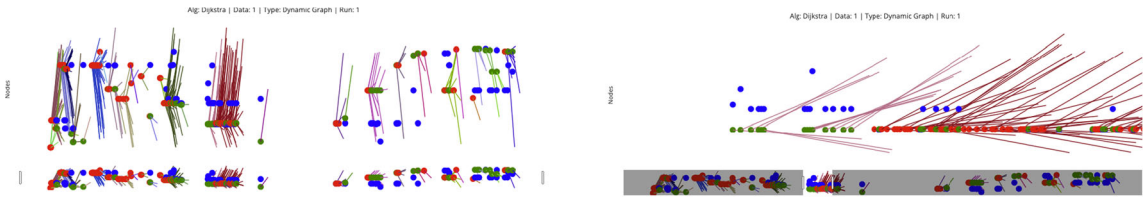


Fig. 4 Time-to-space mapping of Dijkstra's algorithm to find the shortest paths to start node 1 for a small network consisting of six vertices and eight weighted edges



(a) **(b)**

Fig. 5 **a** Time-to-space mapping of the dynamic graph generated by applying Dijkstra's algorithm to an input graph. **b** Zooming into a certain time interval helps to explore the algorithm dynamics on a temporally finer scale

node 1 is the first purple node, followed by the nodes, 2, 3, 4, again 3, and then 5. Node 3 is chosen twice because it has been added to the queue twice, which a more efficient implementation could have avoided.

4.5 Time-aligned performance visualizations

In order to show additional measures in the form of run times and memory consumption, we use line graphs. These are temporally aligned with the time-to-space mapping of the dynamic graph to detect correspondences between performance flaws and graph patterns. The algorithm animation can then give additional insights in the issues that happened there. Figure 7 shows examples for running times and a zoomed-in excerpt of a certain time period. For example, someone might be interested in the peak in the running times per step, hence zooming into that time period can provide more details.

From a general interaction perspective, the temporal granularity can be changed by using the time period sliders in the right line graph in Fig. 7. This interaction feature supports a viewer if the number of animation steps is very high and only the peaks of a performance chart can be observed. Temporally zooming into the peak region might uncover a certain issue in the dynamic graph that could not be seen by just looking at the time-to-space mapping. However, the performance charts cannot help if the algorithm runs smoothly without peaks but still produces unexpected results. In such a case, the time-to-space mapping can give visual support, and for further details, the linking to the animation might give the final insight into the detected or hypothesized issue.

5 Application examples

Our graph algorithm visualization tool can handle graph algorithms that are augmented with statements to log additional graphical and non-graphical data such as memory consumption; currently, Dijkstra's and Prim's algorithm is supported. To demonstrate the tool we use two data sets. On each, we apply one of the supported algorithms to illustrate how the tool facilitates the tasks T1 to T5.

5.1 Dijkstra's algorithm

For Dijkstra's shortest path algorithm (Dijkstra 1959), we use flight data obtained from the Bureau of Transportation Statistics [39] from which we generated two datasets with flights on May 17, 2018:

1. flight data of California and Nevada with $|V| = 113$ and $|E| = 893$;
2. dataset 1 plus flight data of New York, Pennsylvania, North Dakota, and Texas with in total $|V| = 228$ and $|E| = 2,273$.

Vertices represent the airports and edges the routes between them. The edge weights are the distances in miles between airports.

Figure 8 shows the overview after multiple runs of Dijkstra's algorithm applied to the two datasets. Based on Fig. 8, we can answer task *T1 [involved elements]* since the time-to-space mapping indicates which vertices and edges have been processed over time. Peaks in running time can be observed in the line chart—*T2 [performance issues]*. The visual comparison feature focuses on answering task *T3 [comparisons]*. Here, we compare two runs of Dijkstra's algorithm. Correlations between the performances and the algorithmic behavior can be observed as a solution to task *T4 [correlations]*. For example, approximately in

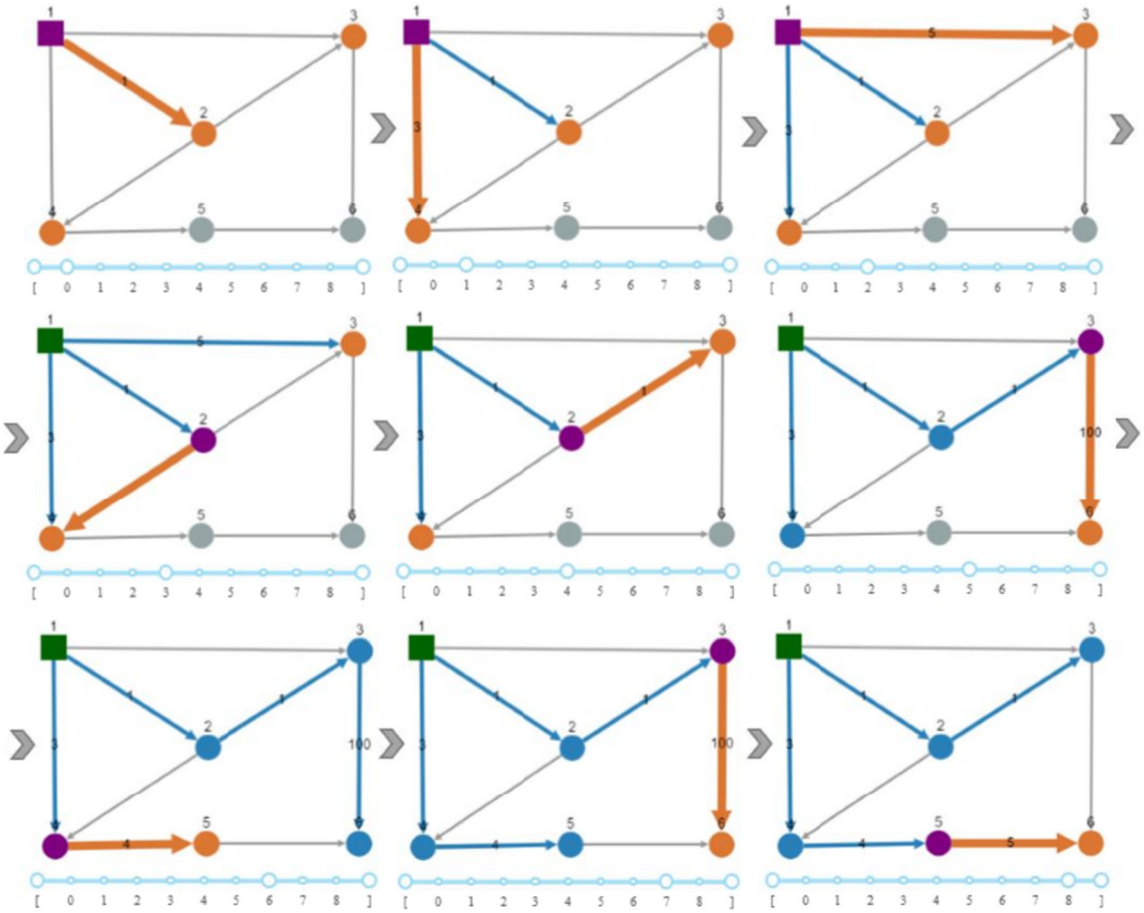


Fig. 6 Animation of Dijkstra's algorithm to find the shortest paths to start node 1 for the same network as in Fig. 4. The animation step slider at the bottom shows the progress of the algorithm animation. The start node is indicated by a green square, yet unvisited nodes are gray, and visited nodes are shown in blue. The node with the lowest distance from the start node is purple, while its neighbors that are considered for the next step are orange. Gray edges are not part of any shortest path or have not been looked at. Blue and thicker edges (v, w) are part of the minimum spanning tree and are on the shortest path from the start to node w

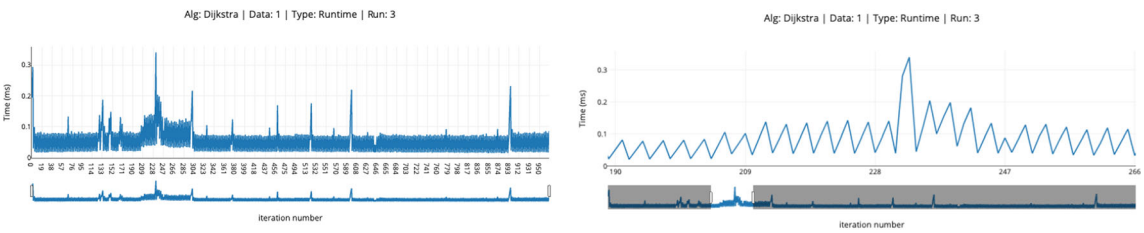


Fig. 7 Running time per time step (**a**). Zooming into a time period can show a temporally finer-grained view of the performance data (**b**)

the horizontal center of the timeline, we can see that the dynamic graph pattern changed after the last peak in time. The activity patterns for answering task *T5 [temporal events]* can also be explored by rapidly inspecting the dynamic graph patterns.

As the colors of the edges in the dynamic graphs are based on their starting node, similar edge groupings between the first and second dataset can easily be identified, as shown in Fig. 9. This helps to answer a variant of task *T3 [comparisons]* for inspecting different datasets for a run of the same algorithm. Moreover, groupings that are different such as the pink edges in the center in Fig. 8 become apparent. It may be noted

that this is only possible as the first dataset is a subset of the second one. Besides that, the markers not only give a good overview of nodes with outgoing edges but also indicate when a shorter path is found and from where.

From Fig. 8 (center), one can deduce that the starting node with the pink outgoing edges might be a hub, i.e., a strongly connected node. If we look more carefully into the data, it is evident that the hub node is McCarran International Airport, which is the main airport in the region of Las Vegas, Nevada. However, in the second dataset, shown in the upper graph of Fig. 8, this hub is not that important any more, as there are fewer pink edges.

Another phenomenon that can be observed is the replacement of a smaller hub with a better-connected node. In Fig. 10, a horizontal row of green and red markers represents a single airport a_0 that provides shorter routes to existing nodes and connections to new nodes, respectively. For the shorter routes, it tends to replace one particular airport a_1 shown as a horizontal row of blue markers. Apparently, a_0 has a more central location than a_1.

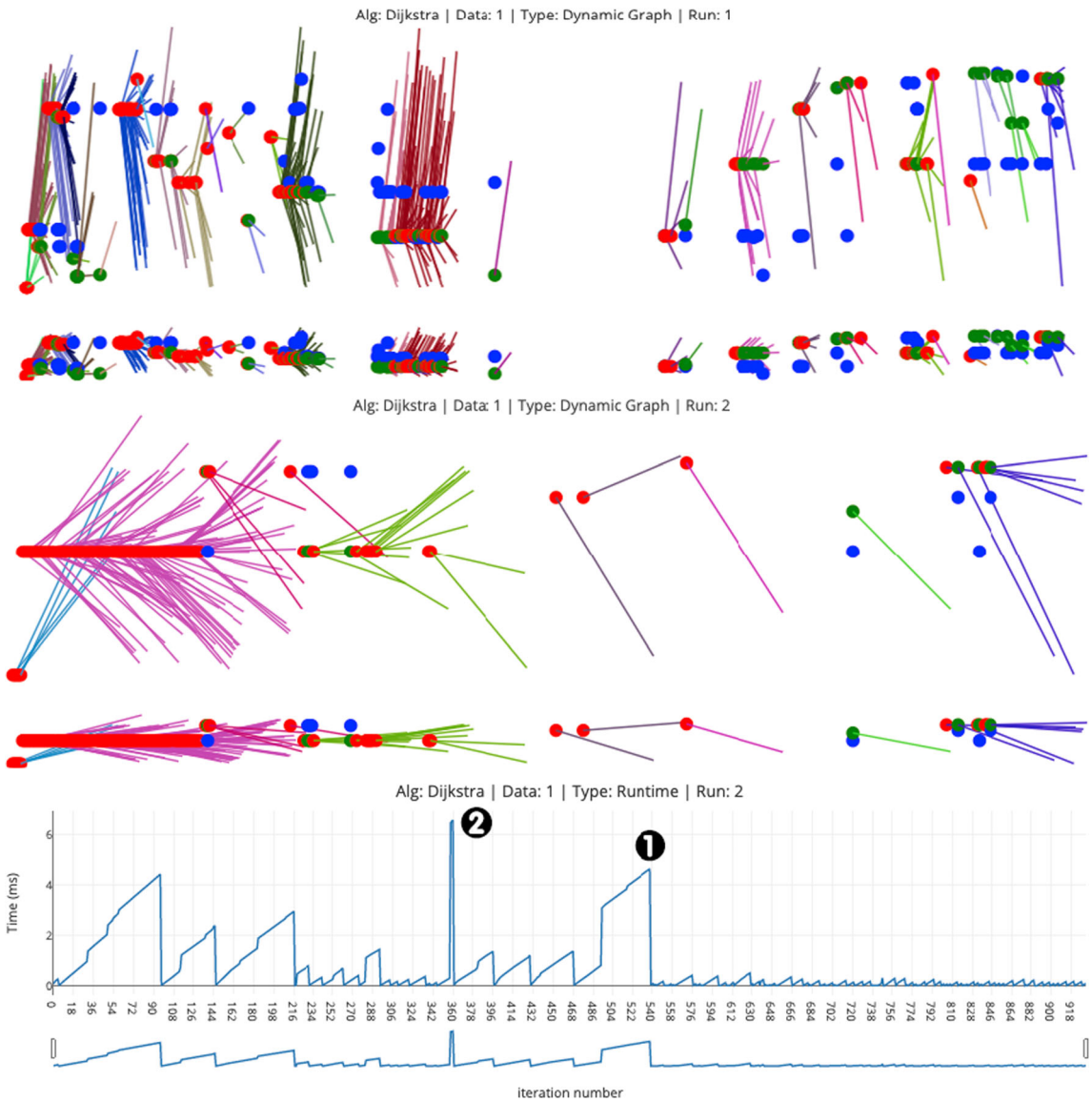


Fig. 8 Examples of two dynamic graphs. Dijkstra's algorithm applied to the second flight dataset resulting in 3968 time steps (top) and to the first flight dataset (subset) with 918 time steps (center), both using *distance* as weight. Bottom: Running times of the steps for the second dataset

In both dynamic graphs, but especially the top one, there are sequences of time steps where no changes appear, hence, there is only white space. This is because the algorithm iterates over each neighbor when a node is added to the Dijkstra tree and updates a path only if a shorter path is found. For that reason, such empty spaces appear and more detailed animations of the graphs could provide insights in the dynamics during such iterations.

On the other hand, the running times are easier to interpret using the bottom plot in Fig. 8. From this plot and Fig. 7, it can be noticed that the running time has a 45deg slope for some time and then drops to zero ❶. This phenomenon is caused by obtaining the current time at the moment a new node is popped from the queue and saving the difference in time for each iteration that is caused by the popped node. Hence, there is a peak when a new node is added to the Dijkstra graph. This insight can be used to answer aspects of task *T2 [performance issues]*.

The peak at iteration 360 ❷, although only lasting seven milliseconds, might be interesting as well since only one node extended the running time approximately twenty times. As the dynamic graph does not give any insight into why this peak emerged, more detailed inspection of the algorithm dynamics might be useful. Playing the algorithm animation linked to the annotated insights from the time-to-space mapping helps to inspect the phenomenon in more depth. Hence, a correlation task T4 is answered using the combination of performance charts and the corresponding graph animation.

Memory usage needed to store data, graphs, and intermediate data is measured as well. A plot of memory usage over time of Dijkstra's algorithm applied to the first dataset under investigation is shown in Fig. 11. While the amount and variation are quite low in our example, larger graphs will need more memory to store data and larger fluctuations might surface. Again, this constitutes a correlation task T4 that leads to the hypothesis that the performance chart and the size of a graph in the graph sequence are dependent.

5.2 Prim's algorithm

In this section, we apply Prim's minimum spanning tree algorithm (Prim 1957) on a network of autonomous systems. Autonomous systems are sub-graphs of Internet routers. The network that we use consists of 522 nodes and 1,244 edges and reflects the autonomous systems on December 30, 1998 (Leskovec et al. 2005). Prim's algorithm is a greedy algorithm for computing the minimum spanning tree of a weighted and undirected graph. The final output of the algorithm is a subset of edges including all vertices which form a tree with the minimal possible total edge weights. In each iteration, the algorithm processes one vertex by including the cheapest connection from the already existing tree to another vertex in the input graph.

Figure 12 shows two runs of Prim's algorithm using two different start nodes together with the temporally aligned performance and memory consumption charts. Because Prim's algorithm is greedy, there are no green nodes in these figures. Despite the different start nodes, we can hardly see a difference between the runs. However, in (a), we can observe that there is a higher running time peak approximately in the center of

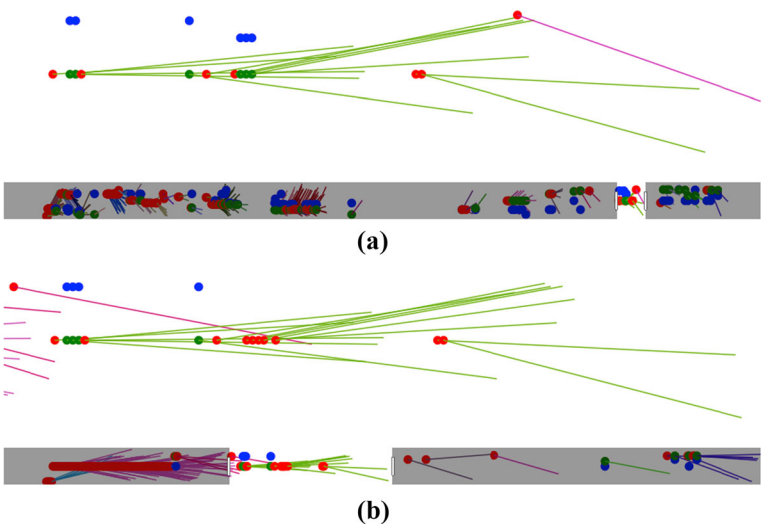


Fig. 9 Similar edge groupings: Zoomed-in excerpt of the graph displayed in Figure 8 top (**a**) and center (**b**)

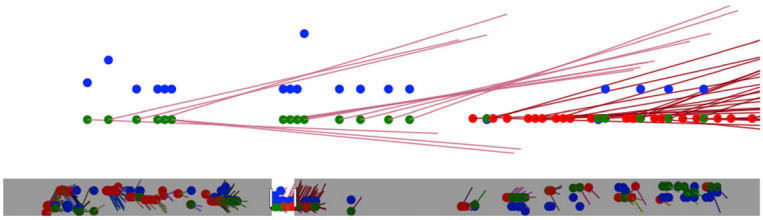


Fig. 10 A more central airport from a list of airports can be observed

the timeline. In (b), there is a deciding difference which comes in form of the maximum peak appearing a bit later, but also not persisting as long as in scenario (a). However, the memory consumption looks very similar for both scenarios. Although when looking carefully, differences — especially at the beginning — can be observed. However, for the dynamic graph patterns, it seems as if the start vertex does not have a big impact on the overall pattern over time. It might be hypothesized that Prim's algorithm runs more stable than Dijkstra's algorithm, at least for these two runs on the same input graph.

Displaying two runs of the same algorithm on the same input graph and using different start vertices indicates differences over time. This is only possible in the time-to-space mapping due to the fact that comparison tasks T3 can be solved by exploiting the perceptual abilities of the human's visual system to recognize visual patterns (Ware 2008). Again, the other tasks can also be answered by applying the supported interaction techniques in the coordinated views for finding correlations (task T4).

6 Discussion

In our application examples, we have shown that our tool can visualize the runs of two different graph algorithms: Dijkstra's shortest path algorithm and Prim's spanning tree algorithm. Furthermore, more algorithms such as Ford–Fulkerson for flow networks or HeapSort for trees can relatively easily be added but have to be implemented in a way to log suitable graphical and non-graphical data to be used by the visualizations. The static visualization of the runs can currently manage over 1,000 time steps. The dynamic visualization is easily scalable with the number of time steps but less with the number of nodes and edges which bears the risk of cluttered visualizations.

We have not created a web-based algorithm visualization tool that can fully manage all graphs containing up to a thousand nodes and a million edges. Implementations for such graphs may require different methods such as level-of-detail or level-of-abstraction approaches. Nevertheless, the foundation for such a tool is laid and dynamics of algorithms over such graphs might eventually be possible. This would then allow for comparisons of large-scale algorithm dynamics which are incomprehensible at the moment without further analysis and visualization tools. Leveraging the scalability of the dynamic graph visualization proposed by Burch et al. (2017) in the developed version of our dynamic graph visualization we see, however, potential for solving bigger graph algorithm challenges in future. Additionally, the vertex ordering added by Abdelaal et al. (2018) can also contribute to readability. These two additions might help avoiding

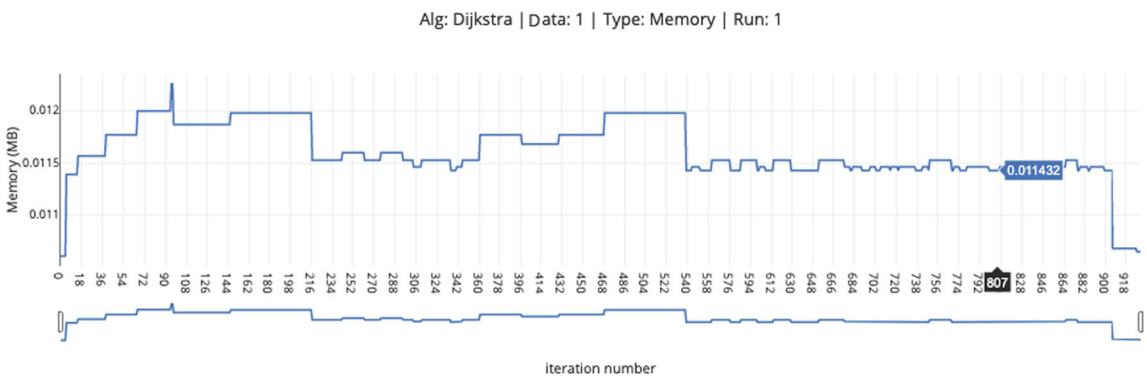


Fig. 11 A line chart for the memory usage in each time step

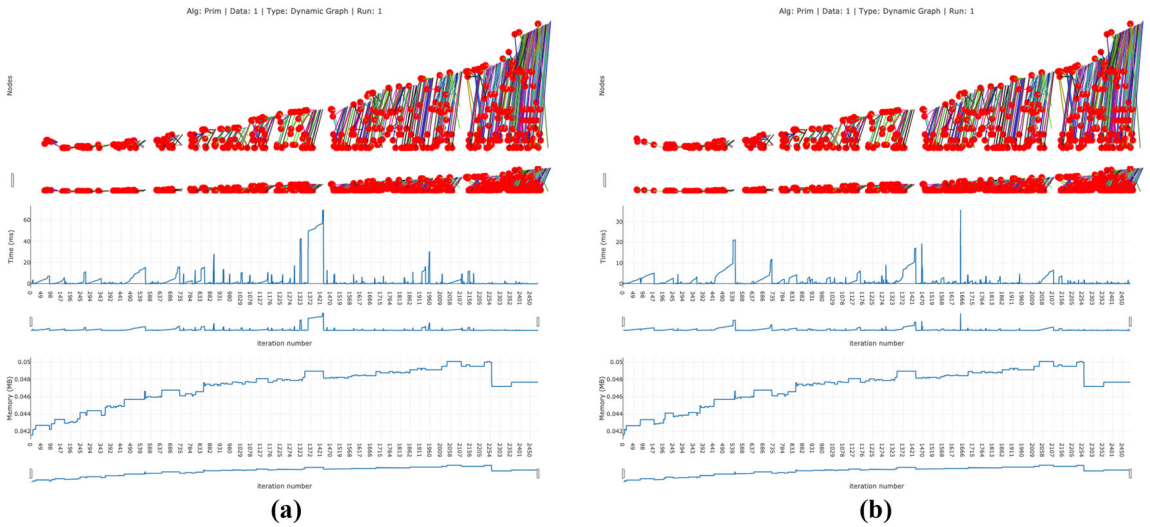


Fig. 12 Two runs of Prim's algorithm on the same input graph but using different start vertices. Although the graph algorithm used different input parameters, we hardly see any difference in the time-to-space mapping, however, the performance charts vary slightly. The variation is more apparent in the running time chart than in the memory consumption chart

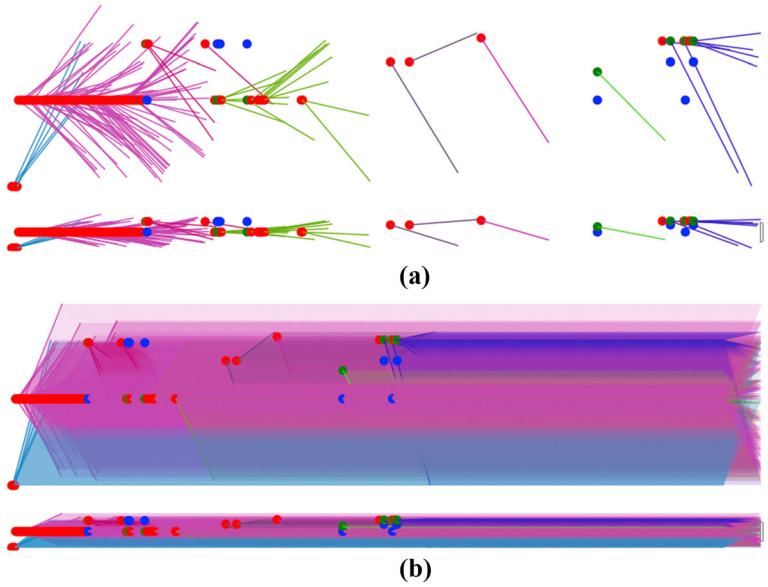


Fig. 13 The same dynamic graph visualization without (**a**) and with (**b**) displaying all edges visited per time step

that processed edges are depicted as in Fig. 13b where, except for some edges at the beginning and some changes halfway through the run of the algorithm, not much can be discerned.

It should also be noted that the running times are tainted by the overhead introduced by the logging and visualization. Logging overhead cannot be avoided, and however, the visualization overhead might be overcome by generating the visualization input (Γ, Σ) beforehand. On the other hand, alternative measures may be used to circumvent the inaccuracy in performance measurement. For instance, counting the number of expensive operations in a time step such as queuing or dequeuing of nodes is not influenced by any overhead in running the algorithm, while it still may give a sufficient performance indication. By setting the sampling factor σ larger than one, the visualization overhead might also be partially overcome, but this comes at the cost of a reduced temporal resolution.

7 Conclusion and future work

In this paper, we proposed a visual analysis tool for exploring a graph sequence generated from algorithm dynamics. For that purpose the tool makes use of time-to-space and time-to-time mappings, supporting an overview-and-detail strategy. The two views are linked with each other and are flexible in a way that one can interact with them to find temporal patterns and potential flaws in a graph algorithm. We also support a temporal overview of measures such as memory consumption and running times per time step. Moreover, the tool can serve as an educational asset to illustrate how algorithms work and how they unfold over time.

As part of future work, more extensive visualizations that further aid in the inspection and comparison of dynamics in large graphs could be integrated. Techniques such as edge splatting and contour lines may emphasize unrecognizable patterns and can contribute positively to the performance of the tool as well. Currently, the tool cannot fully manage graphs containing up to a thousand nodes also because of limitations arising from the web-based approach. Besides node-link diagrams, matrix representations can be an interesting avenue for future work, or even combinations of visual metaphors for dynamic graph visualizations. Also, further interaction concepts might be worth investigating. Finally, a user evaluation should be conducted to investigate the usefulness of our tool, either for data exploration tasks or as a means to teach and educate people interested in graph algorithms.

References

Abdelaal M, Hlawatsch M, Burch M, Weiskopf D (2018) Clustering for stacked edge splatting. In: Proceedings of symposium on vision, modeling & visualization, VMV, pp 127–134. https://doi.org/10.2312/vmv.20181262

Archambault DW, Purchase HC (2016) Can animation support the visualisation of dynamic graphs? Inf Sci 330:495–509. https://doi.org/10.1016/j.ins.2015.04.017

Archambault DW, Purchase HC, Pinaud B (2011) Animation, small multiples, and the effect of mental map preservation in dynamic graphs. IEEE Trans Vis Comput Graph 17(4):539–552. https://doi.org/10.1109/TVCG.2010.78

Baecker, R.: Sorting out sorting: A case study of software visualization for teaching computer science. In: Software Visualization: Programming as a Multimedia Experience, pp. 369–381 (1998)

Beck F, Burch M, Diehl S, Weiskopf D (2017) A taxonomy and survey of dynamic graph visualization. Comput Graph Forum 36(1):133–159. https://doi.org/10.1111/cgf.12791

Bedi P, Sharma C (2016) Community detection in social networks. WIREs Data Min Knowl Discov 6(3):115–135. https://doi.org/10.1002/widm.1178

Brown MH (1991) Zeus: a system for algorithm animation and multi-view editing. In: Proceedings of the 1991 IEEE workshop on visual languages, pp 4–9

Brown MH, Sedgewick R (1984) A system for algorithm animation. In: Christiansen H (ed) Proceedings of the 11th annual conference on computer graphics and interactive techniques, SIGGRAPH. ACM, New York, pp 177–186. https://doi.org/10.1145/800031.808596

Brown MH (1988) Exploring algorithms using Balsa-II. IEEE Comput 21(5):14–36. https://doi.org/10.1109/2.56

Brown MH, Sedgewick R (1985) Techniques for algorithm animation. IEEE Softw 2(1):28–39. https://doi.org/10.1109/MS.1985.229778

Burch M, Vehlow C, Beck F, Diehl S, Weiskopf D (2011) Parallel edge splatting for scalable dynamic graph visualization. IEEE Trans Vis Comput Graph 17(12):2344–2353. https://doi.org/10.1109/TVCG.2011.226

Burch M, Hlawatsch M, Weiskopf D (2017) Visualizing a sequence of a thousand graphs (or even more). Comput Graph Forum 36(3):261–271. https://doi.org/10.1111/cgf.13185

Burch M, Melby E (2019) Teaching and evaluating collaborative group work in large visualization courses. In: Proceedings of the 12th international symposium on visual information communication and interaction, VINCI, pp 17–1178. https://doi.org/10.1145/3356422.3356447

Burch M, Wallner G, van de Wetering H, Rooks F, Morra O (2021) Visual analysis of graph algorithm dynamics. In: Klein K, Burch M, Limberger D, Trapp M (eds) Proceedings of the 14th international symposium on visual information communication and interaction, VINCI. ACM, New York, pp 16–1165

Bureau of transportation statistics: reporting carrier on-time performance (1987-present) (2019). https://www.transtats.bts.gov/DL_SelectFields.asp?Table_ID=236. Accessed April 2020

Cockburn A, Karlson AK, Bederson BB (2008) A review of overview+detail, zooming, and focus+context interfaces. ACM Comput Surv 41(1):2–1231. https://doi.org/10.1145/1456650.1456652

Diehl S, Görg C (2002) Graphs, they are changing. In: Proceedings of the international symposium on graph drawing, pp 23–31. https://doi.org/10.1007/3-540-36151-0_3
Dijkstra EW (1959) A note on two problems in connexion with graphs. Numer Math 1(1):269–271
Eades P, Klein K (2018) Graph visualization. Graph data management. Fundamental issues and recent developments. Springer, Cham, pp 33–70
Elmqvist N, Do T, Goodell H, Henry N, Fekete J (2008) ZAME: interactive large-scale graph visualization. In: Proceedings of IEEE VGTC pacific visualization symposium, pp 215–222. https://doi.org/10.1109/PACIFICVIS.2008.4475479
Euler L (1741) Solutio problematis ad geometriam situs pertinentis. Commentarii Academiae Scientiarum Petropolitanae 8:128–140
Frishman Y, Tal A (2004) Dynamic drawing of clustered graphs. In: Proceedings of the IEEE symposium on information visualization (InfoVis), pp 191–198. https://doi.org/10.1109/INFVIS.2004.18
Görg C, Birke P, Pohl M, Diehl S (2004) Dynamic graph drawing of sequences of orthogonal and hierarchical graphs. In: Proceedings of 12th international symposium on graph drawing, GD, pp 228–238. https://doi.org/10.1007/978-3-540-31843-9_24
Greilich M, Burch M, Diehl S (2009) Visualizing the evolution of compound digraphs with TimeArcTrees. Comput Graph Forum 28(3):975–982. https://doi.org/10.1111/j.1467-8659.2009.01451.x
Gross JL, Yellen J (2005) Graph theory and its applications. Textbooks in mathematics. CRC Press, Boca Raton
Henry N, Fekete J (2006) MatrixExplorer: a dual-representation system to explore social networks. IEEE Trans Vis Comput Graph 12(5):677–684. https://doi.org/10.1109/TVCG.2006.160
Lee B, Plaisant C, Parr CS, Fekete J, Henry N (2006) Task taxonomy for graph visualization. In: Bertini E, Plaisant C, Santucci G (eds) Proceedings of the 2006 AVI workshop on BEyond time and errors: novel evaluation methods for information visualization, BELIV 2006, Venice, Italy. ACM Press, New York, pp 1–5 (2006)
Leskovec J, Kleinberg J, Faloutsos C (2005) Graphs over time: densification laws, shrinking diameters and possible explanations. In: Proceedings of the eleventh ACM SIGKDD international conference on knowledge discovery in data mining. KDD'05. Association for Computing Machinery, New York, pp 177–187. https://doi.org/10.1145/1081870.1081893
Michail O (2015) In: Zaroliagis C, Pantziou G, Kontogiannis S (eds) An introduction to temporal graphs: an algorithmic perspective. Springer, Cham, pp 308–343. https://doi.org/10.1007/978-3-319-24024-4_18
Misue K, Eades P, Lai W, Sugiyama K (1995) Layout adjustment and the mental map. J Vis Lang Comput 6(2):183–210. https://doi.org/10.1006/jvlc.1995.1010
Prim RC (1957) Shortest connection networks and some generalizations. Bell Syst Tech J 36(6):1389–1401
Purchase HC, Hoggan E, Görg C (2007) How important is the "mental map"?—An empirical investigation of a dynamic graph layout algorithm. In: Proceedings of the international symposium on graph drawing, pp 184–195 . https://doi.org/10.1007/978-3-540-70904-6_19
Shneiderman B (1996) The eyes have it: a task by data type taxonomy for information visualizations. In: Proceedings of visual languages, pp 336–343. https://doi.org/10.1109/VL.1996.545307
Stasko JT (1990) Simplifying algorithm animation with TANGO. In: Proceedings of the 1990 IEEE workshop on visual languages, pp 1–6. https://doi.org/10.1109/WVL.1990.128374
Stasko JT (1990) Tango: a framework and system for algorithm animation. IEEE Comput 23(9):27–39. https://doi.org/10.1109/2.58216
Tversky B, Morrison JB, Bétrancourt M (2002) Animation: can it facilitate? Int J Hum Comput Stud 57(4):247–262. https://doi.org/10.1006/ijhc.2002.1017
van Liere R, de Leeuw WC (2003) Graphsplatting: visualizing graphs as continuous fields. IEEE Trans Visu Comput Graph 9(2):206–212. https://doi.org/10.1109/TVCG.2003.1196007
Végh L, Stoffová V (2017) Algorithm animations for teaching and learning the main ideas of basic sortings. Inform Educ 16(1):121–140
von Landesberger T, Kuijper A, Schreck T, Kohlhammer J, van Wijk JJ, Fekete J, Fellner DW (2011) Visual analysis of large graphs: state-of-the-art and future research challenges. Comput Graph Forum 30(6):1719–1749. https://doi.org/10.1111/j.1467-8659.2011.01898.x
Ware C (2008) Visual thinking: for design. Morgan Kaufmann Series in Interactive Technologies, Paperback
Yi JS, Ah Kang Y, Stasko JT, Jacko JA (2007) Toward a deeper understanding of the role of interaction in information visualization. IEEE Trans Vis Comput Graph 13(6):1224–1231. https://doi.org/10.1109/TVCG.2007.70515