Unplugging Dijkstra's algorithm as a mechanical device

Riko Jacob[1][0000−0001−9470−1809] and Francesco Silvestri[2][0000−0002−9077−9921]

[1] IT University of Copenhagen, Copenhagen, Denmark
rikj@itu.dk
[2] University of Padua, Padua, Italy
silvestri@dei.unipd.it

Abstract. Graphs are a fundamental concept in computer science, effectively modeling diverse scenarios such as social networks, protein interactions, and mobility. Dijkstra's algorithm is crucial for computing single-source shortest path in graphs and is a key component of graph processing. This paper presents an educational activity designed to "unplug" graphs and Dijkstra's algorithm, making these topics accessible to a broad audience. The activity utilizes a physical graph with chains as edges and key rings with retractable badge holders as nodes. By pulling two nodes of this graph apart, it is possible to find a shortest path between these nodes. This can be used to visualize how Dijkstra's algorithm works, including how the graph models the world. It invites for discussing how much more efficient this is compared to enumerating all paths, and what additional insights computer scientists had to achieve impressive speedups over plain Dijkstra, allowing for route planning to be perceived as a solved problem, where we use the packaged solution without further thought. We discuss the implementation of this activity in public outreach events, such as Culture Nights and primary school classrooms.

Keywords: Dijkstra's algorithm · mechanical device · CS Unplugged

1 Introduction

Computer Science Unplugged (CS Unplugged) is a collection of teaching activities designed to introduce key concepts from Computer Science without using digital devices or programming, but instead exploiting simple materials (pencils, paper, cards, ...), games, and kinaesthetic engagement. CS Unplugged was introduced in the seminal work [4] by T. Bell, I. H. Witten, and M. Fellows which presents a variety of activities aimed at introducing computer science ideas to primary school students. Since then, hundreds of works have proposed activities covering a wide range of topics, including binary encoding, sorting, image representation, cryptography, and more. Many of these activities are accessible online through websites, research papers, or simple documents (see, e.g., www.csunplugged.org, [1, 2] and references therein). Although initially intended for primary school students, the activities have found broad applications in public outreach efforts.

The concept of graphs is highly significant in computer science, as it can model various scenarios, including applications in social networks, bioinformatics, and mobility. However, a non-technical audience, in particular primary school kids, might incur difficulties in understanding graphs and how they can model reality. To address this challenge, several CS Unplugged activities in the literature aim at introducing the concept of graph, some graph primitives like Minimum Spanning Tree, and intractable problems like graph coloring or dominating set. These activities are mostly based on paper and pencil activities, which require solving challenges like coloring a map or positioning ice cream vans on a map. Dijkstra's algorithm is among ones of the most studied graph processing algorithms and it solves the single-source shortest path problem: given a graph $G = (V, E)$ and a source vertex $s \in V$, find all shortest paths from s to every vertex $t \in V$. There are numerous educational videos and web pages aimed at explaining the intuition behind Dijkstra's algorithm and how to implement it: however, to the best of our knowledge, there is no previous work aiming at unplugging Dijkstra's algorithm and introducing it without any digital devices or coding.

In this paper, we propose an unplugged activity for introducing the concept of graph and Dijkstra's algorithm using a mechanical device; the activity has already been used in several events, but never formally described. The novelty of the approach is the use of a physical graph constructed on a board, with chains as edges and rings with retractable badge holders as vertexes (see Figure 1). A map is plotted on the board's surface, representing a geographical region of interest to the audience: the vertexes represent points of interest (e.g., cities), while the edges capture connections on the map (e.g. roads or train lines). The device allows one to find the shortest paths by simply vertically pulling up vertexes. To find the shortest path between two vertexes s and t of the graph, it suffices to take the two vertexes and pull them apart: the shortest path between s and t is going to be the taut chains that are suspended over the other chains, i.e., the path of chains that stops the two vertices from being able to be pulled further apart. Moreover, by just pulling up one vertex s, all other vertexes are raised (basically) in order of increasing distance from s, and the shortest path from s to a vertex t can be easily detected as it is the only taut chain path between the two nodes. When pulling a vertex, we are physically implementing Dijkstra's algorithm using the mechanical device, providing a visualization of the algorithm.

This activity can build on common knowledge like navigator apps and maps, and it can be used to introduce several concepts. Indeed, the device provides an example of how the concept of graphs captures real-world problems, like finding the best train combination to move from one city to another. Moreover, the device facilitates discussions on the complexity of solutions based on an exhaustive search for route planning, and on the significant improvements provided by fast solutions such as Dijkstra's algorithm. The goal of this paper is to describe the device and the associated activity, and how the device was constructed and used in different scenarios. The paper is structured as follows: in Section 2, we introduce some graph terminology used in the paper and relevant previous work;

then, in Section 3, we describe the activity, the relation to Dijkstra's algorithm, and how to construct the device; in Section 4, we describe our experience of the table in public outreach public in Culture Nights, and in fifth-grade classes (about 10-11 years) in an Italian Montessori primary school; we conclude with some final remarks in Section 5.

2 Preliminaries

2.1 Graphs and Dijkstra's algorithm

In this section, we introduce some graph notations and Dijkstra's algorithm; note that these concepts are not required for the main activity, which can be presented without any technical requirement. A *graph* is a pair $G = (V, E)$, where $V = \{v_1, \dots v_n\}$ is a set of elements called vertexes and $E \subseteq V \times V$ is a set of m unordered pairs from V, whose elements are called edges. A *weighted* graph is a graph where each edge $e \in E$ is given a weight $w_e \in \mathbb{R}$; we assume weights to be non-negative. A path $\mathcal{P}$ is a sequence $(v_{i_i}, v_{i_2}, \dots v_{i_\ell})$ of vertices in V such that there exists in E an edge $(v_{i_j}, v_{i_{j+1}})$ for each $j = 1, \dots, \ell - 1$. The *weight* (or cost) of a path $\mathcal{P} = (v_{i_i}, v_{i_2}, \dots v_{i_\ell})$ is the sum of the weights of the edge among the path, that is $\sum_{j=1}^{\ell-1} w_{(v_{i_j}, v_{i_{j+1}})}$. Given two vertices $s, t \in V$, the *shortest path* from s to t is a path with minimum weight among all paths from s to t; we denote with $d(s, t)$ the weight of this shortest path.

Given a graph $G = (V, E)$ and a vertex $s \in V$, the *single-source shortest path* (SSSP) problem requires computing the shortest path from $s \in V$ to every vertex $t \in V$. The *Dijkstra's algorithm* is one of the most common solutions to SSSP, and it was conceived by E. W. Dijkstra in 1956 [6]. We provide a simple explanation of the algorithm and we refer to [5] for a more detailed analysis. The algorithm iteratively constructs with a greedy approach a set $S \subseteq V$ of vertices for which we know a shortest path from the source s; for all vertexes $t \in V \setminus S$, we instead have an upper bound $\tilde{d}(s, t)$ on the cost to reach t from s by using only vertices in S; the bounds are improved in each iteration until we reach the exact values.

More specifically, we initially have $S = \{s\}$ since the source is connected to itself, and the upper bound $\tilde{d}(s, t)$ is set as:

$$\tilde{d}(s,t) = \begin{cases} w_{(s,t)} \text{ if } (s,t) \in E \\ +\infty \quad \text{otherwise} \end{cases}$$

Then, the algorithm repeats $n - 1$ times the following steps:

1. Add to S the vertex u in $V \setminus S$ with minimum upper bound on the cost; for this node, we indeed have $d(s, u) = \tilde{d}(s, u)$.
2. For each edge (u, t) with $t \in V \setminus S$, update the upper bound as follows:

$$\tilde{d}(s,t) = \min\{\tilde{d}(s,t), d(s,u) + w_{(u,t)}\}$$

We observe that the above algorithm exploits the fact that when u is added to S, then the current upper bound $\tilde{d}(s,u)$ is the cost of the shortest path $d(s,u)$. Although the above algorithm only computes the cost of the shortest path, it can easily be restated to compute the actual shortest path.

2.2 Previous work

There are numerous CS Unplugged activities documented in various websites and academic papers, as well as many activities that have yet to be formally described. For a more comprehensive examination, we recommend referring to the CS Unplugged website and book [4], as well as several surveys (e.g. [1, 3, 2]). For an overview of the different applications of CS Unplugged and discussions on the effectiveness and limits of the unplugged approach, we refer to the aforementioned [3] and reference therein. Given the importance of graphs and graph primitives in computer science, several unplugged activities on these topics have been presented. Already Bell, Witten, and Fellows in the book *Computer Science Unplugged* [4] provide several activities on graph primitives, such as Minimum Spanning Tree (MST), sorting network, graph coloring, dominating sets, Steiner trees.

In the MST activity, students are given a map of houses connected by muddy roads and are asked students to pave enough streets so that it is possible for everyone to travel from their house to anyone else's house only along paved roads; clearly, the paving cost should be as little as possible. The map with houses and roads nicely introduces the notion of graphs to kids. Moreover, the paving problem is an intuitive example of MST, that can be easily solved on small graphs, but also on slightly larger with Kruskal's algorithm. Sorting networks are an example of direct acyclic graphs used for describing computations: the activity asks students to sort elements by following the paths given by a sorting network drawn on the floor. The main goal of this activity is not to introduce graphs, but rather to reason about sorting algorithms; in fact, the graph structure of paths is usually not emphasized.

While the above activities focus on algorithm design, the activities on graph coloring, dominating sets, and Steiner trees focus more on intractability by showing that simple problems might be complex to solve even with small inputs. As in the previous activity, the activities build on a particular problem (e.g., setting ice cream vans, or coloring a map) to introduce the concept of a graph. Then, different input examples of increasing size can be given to the students: a warm-up with easy input instances allows students to better understand the problems; then, larger input instances challenge the students and show that finding a solution quickly becomes harder as soon as the input size increases.

Most activities on graphs are based on a "paper and pen" approach. An exception is provided on activities on sorting networks that can be seen as a kinaesthetic learning activity. To the best of our knowledge, there is no previous work on describing graph primitives, like Dijkstra's algorithm, using a mechanical device as proposed in this paper.

Fig. 1: The board used for explaining Dijkstra's algorithm: the left picture shows the complete graph on the board and the right picture how it was used during an activity.

3 Activity description

In this section, we describe the activity to introduce graphs and Dijkstra's algorithm with the mechanical device. We first describe how the activity was presented to two fifth-grade classes of an Italian primary school (about 10-11 year-old students). Then, we explain why the device implements Dijkstra's algorithm. Finally, we describe how to construct the mechanical device.

3.1 Narrative

Graphs and paths. The activity leverages on the board in Figure 1, which represents (part of) Europe. At the top, a set of rings highlights some cities: each ring has the name of the city and it is connected to the table via a retractable badge holder. Each holder allows to pull up the ring with the city name from the board, and then easily reposition it on the board; this feature will be useful later. The chains represent the *trains* or *ferries* that connect two cities (in both directions). The length of the chain (i.e., the number of links) represents the travel time from one city to another using that train/ferry: for instance, the chain representing the train from Milan to Lyon is much shorter than the ferry from Bilbao to Dublin, meaning that the ferry takes much more time than the train; we can say that each link is about 30 minutes. The rings and chains create a *graph*: the rings (i.e., cities) are called *vertexes*, the chains are named *edges*, and the traveling time is the *weight* of the edge. A sequence of trains/ferries for moving from one city to another one is called *path*: for instance, we can travel from Madrid to Brussels by taking the trains Madrid-Paris and Paris-Brussels. The total traveling time of the path is proportional to the number of links in the chains we use: the larger the number of links in the path, the longer the

Fig. 2: Two examples of activities on the mechanical device. On the left, two cities are pulled up at the same time and the top path denotes the shortest path between the two cities. On the right, we pull up one vertex: here, we are mechanically implementing Dijkstra's algorithm.

traveling time between the two vertexes. We assume that we do not spend time on a vertex for a train transfer.[3]

Shortest paths. Let's say we start from Padua and we want to reach Munich: which trains should we take if we want to minimize the traveling time? There are many train combinations: we can go via the paths Padua-Rome-Milan-Munich or Padua-Milan-Lyon-Paris-Frankfurt-Munich, which however do not minimize the traveling time. A quick look at the board shows that the best paths seem to be Padua-Milan-Munich or Padua-Wien-Munich. As the traveling time is given by the length of the chains, by counting chain links, we easily see that the fastest solution from Padua to Munich is via Milan (21 vs 41 chain links). A path that minimizes the total traveling time between two cities is named *shortest path*. We observe that for providing the exact solution, we have considered several train combinations: some combinations clearly take longer times, but others require a more fine analysis by counting chain links. If we now take two far away cities, like Padua and Copenhagen, we see that the set of possible train/ferry combinations is very large and we need a long time to find the shortest path by enumerating all possible train/ferry combinations.

Shortest path between two vertexes. To easily determine the shortest path, we can use a simple mechanical approach. Suppose we want to travel from Paris to Warsaw: we vertically lift the rings of the two cities, and the path that rises above the other chains is the shortest one. More specifically, take the two rings associated with the two cities and slowly lift them. As we do, the chains start to rise and, at some point, a path connecting the two cities appears with its chains

[3] Although the map is inspired by some existing train/ferry lines, the graph does not represent an existing network nor the traveling times between cities are real.

taut and above the others: this path is the shortest one. Refer to the left image of Figure 2 for an example.

At this point, we can challenge the participants by asking them to determine the shortest path between different cities or after changing the chain lengths. For instance, which is the shortest path between Milan and Frankfurt if there is a long delay on the Munich-Frankfurt train (i.e., we replace the chain with a longer one)? If the tunnel between Paris and London closes for maintenance, what is the shortest path from Madrid to London? (In this case, the only available path is represented by the very long chain of the ferry from Bilbao to Dublin.)

Single-source shortest path and Dijkstra's algorithm. To better understand why this approach yields the shortest path, let's assume we are in Frankfurt and want to compute the shortest path from Frankfurt to *every* city in the graph. This problem is called the *single-source shortest path*. If we slowly lift Frankfurt's ring, we see some chains rising, and eventually, another city's ring lifts. This city represents the closest one to Frankfurt in terms of travel time (Munich, in our case). By pulling up Frankfurt's ring, we lift its edges, and the shortest edge is the first to rise completely. Additionally, this edge is the only one fully stretched, while all other edges remain slightly loose. Refer to the right image of Figure 2 for an illustration.

If we continue lifting Frankfurt's ring, another city rises, specifically the second closest city to Frankfurt (Brussels, in our case). The shortest path from Frankfurt to this second city is given by the only taut path between the two cities. We observe that there cannot be any shorter path from Frankfurt to the second city; otherwise, the pulling would have already stretched this path.

If we continue, we find the shortest paths from Frankfurt to all the other cities. Additionally, by observing the order in which the cities rise, we see them sorted by travel time: the shortest path from Frankfurt to another city is given by the only taut path between the two cities. What is happening here? By pulling one vertex, we are mechanically implementing Dijkstra's algorithm, one of the most important algorithms in computer science. This algorithm computes the shortest paths from one vertex, called the source, to all other vertices in the graph.

Consider now the previous case where we want to compute the shortest path from Paris to Warsaw. We can simply lift Paris's ring: when Warsaw's ring rises too, we have found the shortest path between the two cities. We could achieve the same result by lifting Warsaw's ring instead of Paris's. When we pull both rings, we are applying Dijkstra's algorithm from both cities simultaneously. At some point, a rising path starting from Paris and a rising path starting from Warsaw meet (approximately in the central part of the shortest path), creating a single suspended path between the two cities. This approach allows us to better detect the shortest path, as it is completely hanging. Additionally, it requires less lifting compared to using a single ring and enables us to compute the shortest path even when the string in the badge handler is not sufficiently long.

3.2 What's behind?

We now explain why the above mechanical device is implementing Dijkstra's algorithm. As recalled in Section 2.1, Dijkstra's algorithm iteratively computes a set of vertexes S which contains all vertexes of the input graph $G = (V, E)$ for which we already know the shortest path from the source $s \in V$. When we slowly pull up the source vertex (e.g., Frankfurt), we are iteratively constructing S: specifically, all hanging vertexes are the ones in set S. In the beginning, only the source s is up, like in the initial step of Dijkstra's algorithm: the source is connected to itself by a path of length 0. By continuing lifting s, we apply a pulse to the chains that propagate with uniform speed among all edges adjacent to the source s: therefore the second node to rise is the one connected to s by the shortest edge; this is the second vertex added to S.

Assume that we have already raised $|S| \geq 2$ vertexes and we continue to lift s. The pulse propagates among all edges that connect a node in S with a node in $V \setminus S$. Let v be the next vertex to rise; we claim that v is the vertex with the smallest shortest path from s among the vertexes in $V \setminus S$. If this was not true, there would exist another vertex $w \in V \setminus S$ that lies on the board when v raises and with $d(s, w) < d(s, v)$. However since the pulse propagates with uniform speed, we must have that w would have raised before v, which is a contradiction. As Dijkstra's algorithm adds the vertex in $V \setminus S$ with the smallest shortest path, then v is also the next vertex added to S.

3.3 Construction of the device

Now, let's explain how the mechanical device is constructed. It consists of a particle board: as a reference, the board in Figure 1 measures 120 cm by 90 cm in size. Due to its weight, it may be cumbersome to move: to address this, we suggest adding a handle on the back for easier carrying, or dividing the board into two folding parts connected by hinges. The board is supported by two wooden folding sawhorses.

At the top of the board, we affix a map printed on an adhesive vinyl sheet. The specific geographic area depicted on the map can be chosen based on the audience and the educational context, as well as the dimension of the board. For example, at a Culture Night event in Copenhagen, we used a map of the city: this choice was ideal because the audience was familiar with local street names and with the opening/closing of bridges that affect traffic flows. Conversely, when working with fifth-grade students at an Italian Montessori school, we used a map of Europe: this decision aimed to complement their recent introduction to European geography and to spark curiosity about cities in different countries. Maps for the board can be obtained from OpenStreetMap using tools such as Big Map 2 (https://github.com/zverik/bigmap2), which allows downloading high-resolution maps suitable for large displays.

Each vertex on the top of the board is represented by a key ring connected to a retractable badge holder, which is positioned on the bottom part of the board. We repeat these steps for each vertex:

1. A small hole is drilled on the board where to position the vertex.
2. The retractable badge holder is placed on the bottom part of the board.
3. The thread from the badge holder is passed through the hole drilled on the board.
4. The thread is then knotted onto a ring on the top part of the board.
5. A label with the city name is added to identify the vertex.

The tension of the retractable holder keeps the ring attached to the board, but it still allows to raise it. It's important to note that during this operation, the end of the badge holder where badges are normally attached needs to be cut. Care must be taken during this step to prevent the thread from irreversibly retracting inside the holder once the end is cut.

Edges in our setup consist of metal chains with small links (e.g., 18mm). An alternative option is using non-elastic ropes. Chains are chosen for their ease of attachment to rings and the convenience of measuring length by counting links. To ensure simplicity, the graph should be planar to avoid complications where crossing edges might interfere with the pulling operation. Finally, we remark that the diameter of the rings might affect the pulling operation (i.e., by pulling a path slightly longer than the shortest path): however, as soon as the rings are significantly smaller than chains, the effect is negligible.

4 Case Studies

In this section, we describe our experience in using the mechanical device for public outreach. We remark that this is anecdotal evidence, and should not be seen as an experimental evaluation of the proposed method.

4.1 Broad audience

The first version of the described device was designed and built for an exhibition for the 25th anniversary of the Computer Science Department at ETH Zurich, and subsequent versions for Culture Nights at the Technical University of Munich and the IT University of Copenhagen. For the exhibition, we additionally had rice visualizing the number of top-to-bottom paths in a Manhattan-like graph of side-length n. We have many good experiences showing that the device is a nice activity for the general public. Given the size of the device, it attracts some attention. The explanation can easily be adapted to whatever pre-knowledge visitors come with. The robust mechanical design means that even young children will not damage it.

4.2 Primary School

The board was used within a 1.5-hour activity for two fifth-grade classes of the *Scuola Primaria M. Montessori* in Padua, an Italian primary school that follows the Montessori approach. The students, aged 10 and 11, were divided into two

classes with 16 and 19 students, respectively. The activity was conducted separately for each class in February 2024. The students were already familiar with concepts like bits and algorithms, thanks to various "plugged" and "unplugged" projects carried out by the school. The goal of the proposed activity was to show that there exist different types of computational problems: some problems can be solved in a short time (easy problems), while some problems can be solved in a very long time (hard problems). In more academic terms, we aimed to introduce the concept of complexity classes. Another objective was to introduce graphs as one of the most important topics in computer science.

The first part of the activity introduced the concept of graph and the SSSP problem as described in Section 3. The SSSP problem can be "easily" solved using Dijkstra's algorithm. This part was introduced with references to navigation apps like Google Maps and Apple Maps, which are familiar to students. The second part of the activity focused on "hard" problems and, in particular, on the graph coloring problem, following the approach in [4]. Students realized that even an intuitive problem, such as coloring the vertices of a graph, cannot be "easily" solved, even with small graphs.

The students showed curiosity and actively participated in the challenges presented in both parts; several questions were asked by students, even in the following days. The teachers were positive about the activity and student engagement, and they are interested in repeating the activity next year. We observed that, since the school follows a Montessori approach, the students were accustomed to using physical devices to better understand abstract concepts (e.g., mathematical operations, language, geography) and were curious about the mechanical device used to explain Dijkstra's algorithm.

5 Conclusions

In this paper, we have introduced a mechanical device to explain Dijkstra's algorithm. The device is a board with a graph made of chains and rings connected to the table via a retractable badge holder; by suitably pulling nodes, we have a visual representation of Dijksta's algorithm. As future work, it would be interesting to perform a formal evaluation of the effectiveness of the device in teaching, and to analyze which other graph algorithms can be explained using the proposed device. Moreover, it would be interesting to develop other mechanical devices to visualize other algorithms, for instance for sorting and searching but also for some simple learning methods.

Acknowledgments

Riko Jacob would like to express his gratitude towards all the people at ETH Zurich, TUM in Munich and IT University of Copenhagen, that were involved in designing and creating the different mechanical devices and exhibitions, and for showing them in class and to the public. Francesco Silvestri is grateful to teachers Ylenia Costa and Cinzia Paccagnella of the *Scuola Primaria M. Montessori*

(Padua, Italy) for the opportunity to present the activity to their students. Silvestri was in part supported by project "Big Data Analytics for Mobility" under the UniImpresa call of the University of Padua, and by the PRIN project n. 2022TS4Y3N "EXPAND: scalable algorithms for EXPloratory Analyses of heterogeneous and dynamic Networked Data", funded by the Italian Ministry of University and Research (MUR).

References

1. Battal, A., Adanır, G.A., Gülbahar, Y.: Computer science unplugged: A systematic literature review. Journal of Educational Technology Systems **50**(1), 24–47 (2021)
2. Bell, T., Rosamond, F., Casey, N.: Computer Science Unplugged and Related Projects in Math and Computer Science Popularization, pp. 398–456. Springer Berlin Heidelberg (2012)
3. Bell, T., Vahrenhold, J.: CS Unplugged—How Is It Used, and Does It Work?, pp. 497–521. Springer International Publishing (2018)
4. Bell, T., Witten, I., Fellows, M.: Computer Science Unplugged - an enrichment and extension programme for primary-aged children. (2002)
5. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, Third Edition. The MIT Press, 3rd edn. (2009)
6. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numer. Math. **1**(1), 269–271 (dec 1959)