Algorithm visualization in CS education

Luděk, Kučera
Charles University
Prague, Czech Republic
ludek@kam.mff.cuni.cz

CSIT Conference
24-28 September 2009
Yerevan, Armenia

Abstract

Algorithm visualization is a useful tool in teaching computer science. Existing algorithm visualization systems are well designed and sophisticated, when viewed as software products. However, they are often unsatisfactory from the point of view of cognition theory. The aim of the present paper is to argue that the visual information should be structured in a proper way to make the "cognitive complexity" of visualization as low as possible.

Moreover, a succesful product must not be a simple animation, but a complex environment that makes it possible to show not only the algorithm, but its variations as well. It should also be possible to explain motivation, applications, and make a student an active participant of the teaching process.

The abstract approach is illustrated on the algorithm visualization system Algovision, developped at the Charles University, Prague.

Keyvords: Algorithm visualization, algorithm animation, CS education

1 Introduction

With an advent of visual programming languages and systems in 90's (and to some extent even in 80's) early attempts to use algorithm visualization in teaching appeared accompanied with great hopes that this concept would revolutionize CS courses in universities. It seemed that a dynamic character of visualization using, e.g., Java or other visual languages, matched perfectly with temporal aspects of algorithms. Unfortunately, these hopes did not materialize and after almost two decades the algorithm visualization is, in the best case, an auxiliary tool that is used as an illustration of what was taught using classical methods.

The aim of the paper is to analyze why the algorithm visualization is still less successful than originally expected and to suggest an approach that could improve the present situation at least in certain aspects. The approach is illustrated on an algorithm visualization system Algovision, developped at the Charles University.

2 Algorithm visualization

There are numerous systems for algorithm visualization that are used in Computer Science education: let us mention, e.g., Algoviz [18], Alvie [2, 7], Animal [3, 17], JAWAA [11, 16], JHAVÉ [12, 14], Leonardo [4, 5], Mavis [13], Trakla [19, 15] (in alphabetic order). All of them are based on the following paradigm of creating a visualization:

- A method of (static) visual representation of data used by an algorithm is chosen. It is most common that a method that is used in textbooks and journal articles is selected. E.g., if an algorithm operates on graphs or networks, nodes of a graph are represented by circles, edges are shown as arcs connecting their end nodes, and the state of a node or an edge can be indicated using colors or special shapes.
- In order to visualize the computation of the algorithm, at each step of the computation a representation of data is updated to reflect data changes. E.g., in the case of a graph algorithm, state-representing colors of nodes and edges change in time and sometimes new nodes appear or existing nodes are deleted, but otherwise the graph remains unchanged.
- Some systems also present a simplified code or a pseudocode of the algorithm and show the control flow that corresponds to the changing visualized data.

Experience in teaching algorithm courses leads the author of the present paper to a conclusion that such a visualization brings at the same time too much and too little information to a student. Let us explain the statement at an example of an algorithm that is taught in almost any Algorithm and Data Structure course and hence it is visualized in practically any algorithm visualization system - the Dijkstra's shortest path algorithm [8, 6]:

• Why a typical visualization offers too much information: it would be more precise to say that too much information is put to the foreground, i.e., there is too much information that a learner has to follow in order to understand the visualization. A graph might look as shown in Fig. 1, where all nodes look similar and are placed randomly in the plane. The key step of the algorithm is to select a node for processing among nodes that have been reached, but not yet processed. In a typical visualization (e.g., in the systems cited above), the only way for a student to select the proper node is to *read numerical information* associated with nodes. In this way we are loosing all

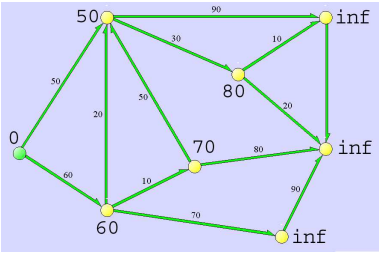

Figure 1: Static Dijkstra's algorithm

advantages of visualization, because, from the point of view of cognition, reading numbers is a completely different and much slower process than observing (moving) pictures.

Thus, even though a visualization offers full information about the computation, the "full information" is unstructured, and psychological experiments revealed that students are not able to distinguish a selection of nodes for processing made by Dijkstra's algorithm from a random order, unless they sequentionally read the numeric information. This is amplified by the fact that there is no correlation between a position of a node in the display window and the distance estimation, which is the variable used by the algorithm to select the next node for processing.

When looking as random, the animation doesn't help a student to understand the computation, and this is what strictly limits usefulness of this kind of algorithm visualization. It is even difficult for someone who already *understands the algorithm* to *understand the visualization*. The next section describes one possible way how to cope with this problem.

• Why a typical visualization offers too little information: there is a lot of information that is closely connected with a problem and the corresponding algorithm that should be conveyed to students, but is outside of the frame of standard visualization described above. Let us mention motivation for the problem, which goes hand-by-hand with possible applications; programming details and paradigms; complexity issues and, more generally, termination analysis of the algorithm; an algorithm correctness proof; a relation to similar problems, and many others. This is what we mean by saying that too little information is brought to students by a standard visualization. The section on visualization environments deals with this problem in detail.

3 Structured visualization

As it has already been mentioned in the introduction, visual information presented by a standard algorithm visualization is often unstructured and it is difficult to distinguish a sequence of color, shape and position changes generated by a computation from a random sequence of visual parameter changes.

Nevertheless, the sequence of changes generated by an algorithm is actu-

ally *far from being random*, as there is a simple mechanism that directs the changes.

This reminds the notion of Kolmogorov complexity of a sequence, which, roughly speaking, is given by the size of the shortest program that generates the sequence. In our case the algorithm itself is a witness of low complexity of the sequence of visual events.

However, the main problem in designing educational visualizations is in the field of cognition, not in mathematics or computer science, namely how much mental effort is needed to recognize that the complexity of the sequence of visual events is low. In other words, how much mental effort is necessary to be able to predict the next event after having seen several training sequences (which is essentially the same as understanding what is going on in the visualization).

Unsatisfiable results with many standard algorithm visualizations are due to our failure to present the visual information in such a way that the cognitive complexity of recognizing logical simplicity of the visual event sequence is minimized.

It is not easy to find an appropriate visualization method. As mentioned above when speaking about the example of Dijkstra's algorithm, using a static drawing of a graph, where only colors representing states could change and the values of the distance estimation are given numerically, gives an animation that looks random, i.e., it is very difficult to realize that a visual event sequence is logically simple.

A (pseudo)code of the algorithm with indication of execution control, if displayed and possibly linked in a certain way with the visualization, proves low complexity of a visual sequence, but the cognitive complexity of the proof is again too large. In fact, a pseudocode is also a written information, which requires *reading*, and, once again, we are loosing advantages of fast processing of visual information.

Our approach is based on another way of demonstrating a low complexity of a visual sequence, which is an *algorithm invariant*.

An algorithm invariant is usually used to prove algorithm correctness and termination. An invariant is a logical formula that remains fulfilled (i.e., invariant) during the whole computation and, together with a terminatin condition, implies the required property of the output (e.g., in the case of Dijkstra's algorithm, the property of being a description of the shortest path from the origin to the destination node). There is a well developped *formal* theory that studies a use of invariants in correctness proving. However, almost no research has been done to study the cognitive role of invariants.

We strongly believe that *knowledge of an invariant* is very close to *understanding of the algorithm*. The main problem in formal algorithm correctness proving is to find an invariant of a given algorithm. It is known that in general this problem is algorithmically unsolvable, and a metarule of the field says that the only method known at the moment is to understand the algorithm and to write down all logical restrictions of the data that follow.

If we go back to the example of Dijkstra's algorithm, the algorithm divides

graph nodes to three categories - processed, reached and unreached. The first set of invariants says that the shortest path length estimation (that is maintained by the algorithm for every node) is smallest for processed nodes, larger for reached (but unprocessed) nodes, and equal to infinity for unreached nodes.

The first invariant set can easily be visualized with very low cognitive complexity. Nodes are shown sliding along horizontal lines and their x-coordinate is proportional to the shortest path estimation maintained by the algorithm. In order to visualize the invariant, see Fig. 2, the window is divided into three stripes, and the left one (dark) contains processed nodes, the middle one (lighter) contains reached nodes, and the right one (light) is a container for yet unreached nodes with an infinite estimation. The invariant claim is that nodes could be partitioned in this way.

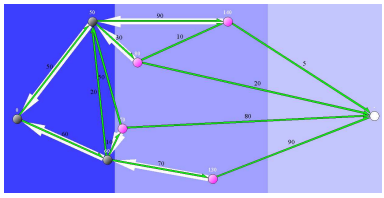

Figure 2: Dynamic Dijkstra's algorithm

This visual representation also clearly identifies the next node for processing by the algorithm - the leftmost node in the middle stripe - which makes the visual event sequence easily predictable. Relaxing an edge now appears as attracting a node by the node that is being processed.

The second set of invariants, the one that immediately implies correctness of the algorithm, says that the shortest path length estimation for a given node is the length of the shortest path to the node that passes through the *processed nodes* only. Again this invariant can be visualized by displaying such paths - see Fig. 2. where such paths are composed of wide arrows oriented to the left. The visualization immediately suggests in a purely graphical way not only how the computation continues (i.e., what is the sequence of visual event), but also why the algorithm finds the shortest paths.

The reason why the present paper illustrates the algorithm invariant approach using an example, and doesn't define it in a general way, is that the visualization that makes cognitive complexity of inferring future moves of the algorithm low is quite different for different algorithms, even when the algorithms are formally very similar. This is one of the main lessons we have learnt from Algovision: there is no general and uniform method of generating visualizations of low cognitive complexity, and each problem requires a visualization tailored to its nature. This also implies that attempts to create *educationally good* visualizations automatically are bound to fail.

4 Algorithm visualization environments

Our main goal is to create algorithm visualizations that can be used in a university course of algorithmics. A simple animation that shows how the data change during the computation, even if built according principles explained in the previous section, is not sufficient to reach our goals. Let us list several other features a good educational visualization should possess:

- It often happens that a certain measure or arrangement highly improves properties and a behavior of an algorithm. E.g., successful network flow algorithms (Dinitz [9], Edmonds-Karp [10]) use an augmenting path with the smallest number of edges. A good teacher should also show what happens when the successful strategy is *not* followed, e.g., show the consequences of using a long augmenting path to the network flow algorithm time complexity. Therefore, a usefull educational environment should implement and visualize much larger class of algorithms - not only the one that is taught, but also all its modifications and variations that (at least partially) fail, because they do not implement certain important features.
- Certain problems need no motivation. E.g., any traveler understands a use of knowledge of the shortest path, and any computer user has no doubts about usefullness of searching a string in a long text. On the other hand, some other problems need a very detailed exposition of the motivation and applications. E.g., an Algovision applet teaching Fast Fourier Transform algorithm must be preceeded by a quite complex applet that shows why Discrete Fourier Transform is important, and why we need a fast way of computation like FFT. In this particular case, our DFT applet uses a virtual device for the spectral analysis of a periodic function and the spectral search and compression.
- It is generally recognized that a learner should not be a passive observer of an educational visualization, but an active participant, especially if the program is used in the frame of individual and/or distant learning. This is why a program has to incorporate means for a learner to interact, e.g., to suggest how certain steps of a computation would be performed. Such features highly enhance usefullness of a visualization, but they also highly complicate its implementation.
- Certain correctness and/or complexity proofs are formulated as so called "adversary argument". A proof is designed as a game, and Bob wins when he is able to submit an input that is too difficult for the algorithm, while Alice (= the algorithm) wins if she solves successfully all Bob's inputs. It is conceivable to implement a visualization as a true game between two human players.
- Certain algorithms are implemented as computational circuits. In such a case we should not only visualize how the data flows through a circuit, but also show a step-by-step construction of a circuit that starts as a

"black box", and another animation shows how the black box evolves to a fully open circuit.

Even though the list is highly incomplete, it is clear that an educational visualization of a particular algorithm must be a complex visual environment that animates the algorithm in question and often a large class of related algorithms, can be used to show motivation and applications of the algorithm, the history of the algorithm, including animated implementation of its predecessors, etc.

5 Algovision

Algovision is a system for algorithm visualization that has been developped at Charles University, Prague. Its original aim was supporting a university course on Algorithms and Data Structures, but it has evolved to a general tool that covers a wide spectrum of algorithms in the range from standard data structures including unbalanced and balanced tree data structures and heaps, basic sorting algorithms, graph algorithms (extremal paths and spanning trees, flows in networks), carry look-ahead adder, FFT, convex hull and Voronoi diagram in the plane, string matching algorithms and the simplex algorithm of linear programming and other algorithms.

Algovision can be viewed as an illustration and application of the principles explained in the previous two section, because both of them represent an important part of the Algovision design philosophy, but in fact the causality is quite opposite - the principles slowly appeared as the system evolved under continuous evaluation and testing the applets in practical algorithm teaching.

In some sence it can be said that both direction were dictated by student acceptance or disapproval of particular applets and design strategies. Many steps of evolution of Algovision are in fact due to students, not only at Charles University, but also at many other universities, where the system is used or was presented.

The lecture delivered at the CSIT'09 conference showed several typical examples of algorithm visualization brought by Algovision that illustrate well the above-mentioned principles.

To a non-negligible extent, Algovision is also an armenian product: two students of Yerevan State University were visiting in Prague for certain time period and contributed to Algovision development and further co-operation is planned.

6 Acknowledgement

The Algovision project was partially supported by a development grant of the Czech Ministry of Education. Youth, and Sports.

References

[1] Algovision, http://kam.mff.cuni.cz/ ludek/Algovision

[2] Alvie, http://www.algoritmica.org/alvie

[3] Animal, http://www.animal.ahrgr.de

[4] V. Bonifaci, C. Demetrescu, I. Finocchi, L. Laura, "Visual editing of animated algorithms: the Leonardo Web builder", *AVI Conference*, Venezia, IT, pp. 476-479, 2006.

[5] V. Bonifaci, C. Demetrescu, I. Finocchi, G. Italiano, L. Laura, "Portraying Algorithms with Leonardo Web", *WISE Workshops*, pp. 73-83, 2005.

[6] T. Cormen, C. Leiserson, R. Rivest, C. Stein, "Introduction to Algorithms", 2nd Ed. MIT Press and McGraw-Hill, 2001.

[7] P. Crescenzi, C. Nocentini, "Fully integrating algorithm visualization into a cs2 course: a two-year experience", *ITiCSE*, pp. 296-300, 2007.

[8] E. W. Dijkstra, "A note on two problems in connexion with graphs", *Numerische Mathematik*, pp. 269271, 1959.

[9] Y. Dinitz, "Algorithm for solution of a problem of maximum flow in a network with power estimation", *Soviet Math. Doklady*, vol. 11, pp. 1277-1280, 1970.

[10] J. Edmonds, R. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems", *J. ACM*, 19(2), pp. 248-264, 1972.

[11] JAWAA, http://www.cs.duke.edu/csed/jawaa2/

[12] JHAVE, http://jhave.org

[13] I. Koifman, I. Shimshoni, A. Tal, "MAVIS: a multi-level algorithm visualization system within a collaborative distance learning environment", *Human Centric Computing Languages and Environments*, pp. 216 - 225, 2002.

[14] T. Naps, J. Eagan, L. Norton, "JHAVE - Supporting Algorithm Visualization Engagement", *J. IEEE Computer Graphics and Applications*, vol. 25, no. 5, 2005.

[15] J. Nikander, J. Helminen, A. Korhonen, "Experiences on Using TRAKLA2 to Teach Spatial Data Algorithms", In: Guido Rssling ed., Proceedings of the Fifth Program Visualization Workshop, 2008

[16] W. Pierson, S. Rodger, Web-based Animation of Data Structures Using JAWAA, *Twenty-ninth SIGCSE Technical Symposium on Computer Science Education*, pp. 267-271, 1998.

[17] G. Rling, "ANIMAL-FARM: An Extensible Framework for Algorithm Visualization", *VDM Verlag*, Saarbrcken, 2008.

[18] C. Shaffer, M. Cooper, S. Edwards, "Algorithm Visualization: A Report on the State of the Field", *SIGCSE'07*, Covington, USA, 2007.

[19] Trakla, http://www.cs.hut.fi/Research/TRAKLA2/