PATH FINDING VISUALIZER

Aakansha N. Tabhane[1], Nikhil Likhar[2], Kalyani Mohod[3],Manali Kadbe[4]
KDK College of Engineering

Abstract

Algorithm visualization has been high topic in Computer science education for years, but it did not make its way to schools/collages lecture halls as the main educational tool. The present paper identifies two key circumstances that an algorithm visualization must fulfil to be successful: general availability of used software, and visualization of why an algorithm solves the problem rather than what it is doing. One possible method of "why" algorithm visualization is using algorithm unvarying rather than showing the data conversion only. Invariants are known in Program faultleness.

Theory and Software authentication and many researchers believe that knowledge of invariants is essentially correspondent to understanding the algorithm. Algorithm stable visualizing leads to codes that are computationally very commanding, and powerful software tools require downloading/installing compilers and/or runtime machines, which restrict the opportunity of users. One our important finding is that, due to computing power of the recent hardware, even very entangle visualization involving 3D animation (e.g., For-tune's algorithm, see Section 4) could be successfully implemented using interpreted graphic script languages like JavaScript that are available to every web user without any installation. The use of images to deliver some useful information about algorithms. Algorithm Visualization. In addition to the mathematical and verifiable analyses of algorithms, there is yet a third way to study algorithms.

Keywords:*visualization, animation, algorithm, Invariant.*

INTRODUCTION

Algorithm visualization (often called algorithm animation) uses dynamic graphics to visualize computation of a given algorithm. First attempts to animate algorithms date to mid 80's(Brown, 1988; Brown and Sedgewick, 1985), and the golden age of algorithm visualization was around the year 2000, when magnificent software tools for anenergetic algorithm visualization (e.g., the language Java and its graphic libraries) and plenty of powerful hardware were already available. It was expected that algorithm visualization would completely change the way algorithms are taught. Many algorithm animations had appeared, mostly for simple problems like primary tree data structures and sorting. There were even attempts to robotize development of animated algorithms and algorithm visualization. Another guidance was to develop tools that would allow learners to prepare their own animations comfortably. Instead of giving appropriate references to algorithm animation papers, the reader is directed to a super-reference (Algoviz,) that brings a list of more than 650 authors/creator, some of them even with 29 references in algorithm animation and visualization.

However, algorithm visualization and animation has not fulfilled the desire, and it is still not used too much in computer Science courses. (bassat Levy and Ben-Ari, 2007), complaining about low approval of algorithm animation tools by teachers. The number of articles, reports, and visualization tools sensibly declined in the second decade of the new golden age. The present paper is an attempt to find why algorithm animation and visualization is used much fewer in instruction then we desire 10 or 20 years ago.

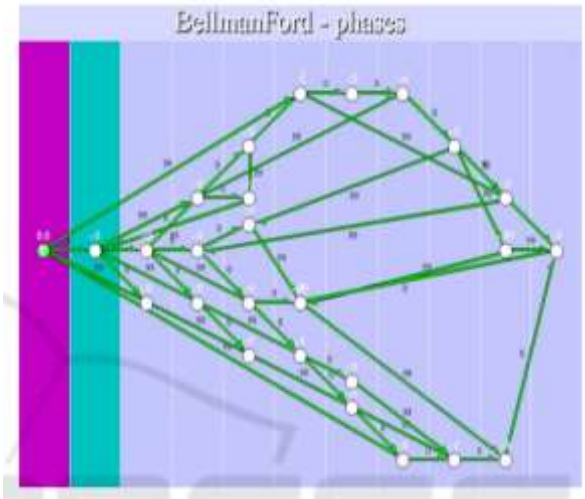
We strongly understand that the reason is relative basic: An algorithm operates on some data (the input data, working variables, and the output data). Usually, in any particular scope of Computer Science, there is a fundamental way of visualization of data - graphs and trees are drawn as circles linked by line segments, number chain could be visualized as collections of vertical bars, there are fundamental ways of drawing matrices, vectors, real functions, etc. An algorithm animation is usually enforce by running the algorithm slowly or in steps, and simply reorganize the visual portrayalof the data in the screen. A person who knows and understands the algorithm in question can see how the algorithm progresses, but a learner user just sees visual objects moving and changing their shapes and colours, but finding out why the movie runs in that way is usually too difficult for him or her.

ALGORITHMIC IDEA VISUALIZATION

Even though the example given in this segment can also be understood as an algorithm stable visualization, it is perhaps more convenient to speak about algorithmic idea visualization (AIV). Once more, there is no general method of AIV, because the underlying ideas of different algorithms in different fields have nothing in common, and each idea is unique and requires uncommon method of representation by dynamic graphic means. Well, in case, there is one general method of AIV. Even though very little is known about productive mental process that leads to discovery of new algorithms, we understand (based on our introspection) that a researcher visualization is perhaps often based, as the word recommend, on mental images - and AIV is just a straight forward projection of such mental images to a demonstration of a computer.

Due to the space limitation, we give just one example - Fortune's algorithm (Fortune, 1987) for Voronoi diagram in the plane. There are several animations of the algorithm in the web, the reader is invited to look at them. It can be seen that the Voronoi diagram is eventually drawn, but the animations give totally no idea what the moving arcs mean and why and how they build the diagram. The algorithmic idea behind the method is following: imagine the plane containing sites are enclosed as a horizontal plane into the 3-dimensional space. For each site, create a circular cone that has a vertical axis and uses the site as its apex.

Observe the cone surfaces vertically from the limitless (to avoid effects of perspective). The junction of cones project to the site plane as the Voronoi diagram we are looking for. Moreover, if the "mountains" of the cones are swept by an inclined plane, the junction of the plane with the clear parts of the cones appear as the arcs that are visual in the planar animations. We tried to show this, but the reader is invited to look kindly to (Kucera,), where he or she can see a full visualization of the 3D situation.As shown in fig:1,2&3.



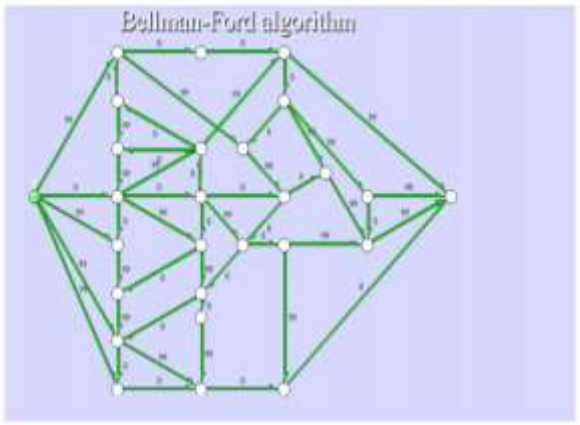


Figure:1 **Figure:2**

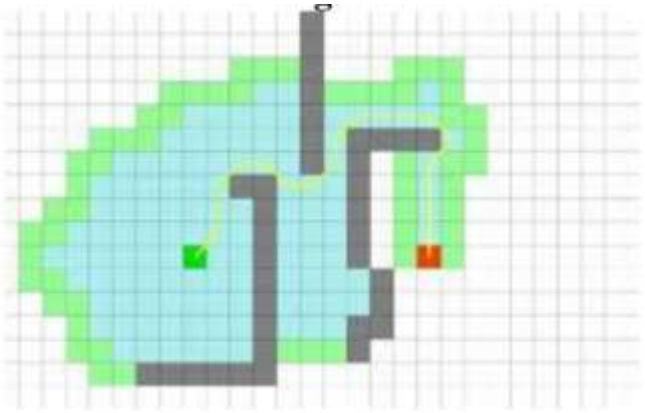


Figure:3

SOFTWARE OF ALGORITHM VISUALIZATION

A dynamic visualization system for algorithm animation should satisfy the following conditions:

• Animation speed - the system should be able to present a good dynamic visualization.

• Programming effort - it should be easy to writea visualization code.

• Widespread access - it must be easy to run a visualization code without (much of) downloading and installing software.Of course, the first condition is the principal one:in many cases, a visualization involves a continuous transformation of the displayed picture, and it might be computationally very demanding to deliver at least 20-25 frames per second to guarantee a smooth animation. A failure in this point would make the system useless. Programming languages can be divided into three classes:

• Compiled languages - a code written by a programmer is compiled into the machine languageand runs at the maximum possible speed. Examples are the languages C and C++ that also offerlibraries of graphical functions (e.g., graphics.h).

• Semi-compiled languages - The code written bya programmer is transformed into a simpler codethat is then interpreted by a special software . An example is Java - the intermediate code is interpreted by JVM program (Java Virtual Machine)

• Interpreted languages - the runtime system reads human written program instructions in runtime and interpret them. An example is JavaScript, see below.

There are really big differences in the speedamong the above classes. While one simple instruction is often executed in just several machineclock tacts, if the same instruction is interpreted, the software must first read and parse the corresponding code, use tables to find the equivalent machine instruction, and only after that the instruction is executed. Interpreted languages are often several order of magnitude slower than compiled languages.

Typical animations that can be found in the web are quite simple and computationally almost insignificant. Consequently, practically any system that allows dynamic animation can be used, preference is given to simple scripting languages.

CONCLUSION**:**

This paper concludes that visualizer can be achievable in near future in each and every algorithm learning. It is part of our project; the new pathfinding has been designed and implemented. Pathfinding is plotting by computer application of the shortest route between two points it is more particular variant on solving maze. A part of project describes about develop pathfinding algorithm and implementation of still behaviour.

REFERENCES

[1]. bassat Levy, R. B. and Ben-Ari, M. (2007). We work so hard and they don't use it: acceptance of software tools by teachers. In ITiCSE '07: Proceedings of the 12th annual SIGCSE conference on Innovation and technology in computer science education, Dundee, Scotland. ACM Press.
[2]. Brown, M. and Sedgewick, R. (1985). Techniques for algorithm animation. IEEE Software, 2:28–39.
[3]. Brown, M. H. (1988). Algorithm Animation. MIT Press.
[4]. Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2001). Introduction to Algorithms, 2nd edition. The MIT Press, Cambridge, Massachusetts, and McGraw Hill, Boston.