Shortest-path algorithms and their application
to threshold fields in varistors

Robert J. Clark
Departments of Physics and Chemistry, Ohio Northern University

P.M. Duxbury
Department of Physics and Astronomy, Michigan State University

Abstract

This project involved the study of various types of shortest-path algorithms. A series of algorithms were written in which only a local knowledge of the lattice being traversed existed. Their data were found to conform, within a small margin of error, to known traits of diffusion-type paths. This data was compared to data from Dijkstra’s algorithm, a global shortest-path algorithm. While the results for this algorithm differed to a small extent from theoretical values, it is thought that further tests will prove the version used here does indeed conform to them. Dijkstra’s algorithm was then used to calculate V-I curves including threshold fields for varistors, which are modeled in these simulations as two-dimensional lattices.
I. Introduction

Before a discussion of the specific algorithms used in this study can take place, some preliminary definitions should be put forth. In this paper, “graph” refers to any set of points, or “nodes,” connected by a set of line segments, which are called interchangeably “arcs” or “bonds.” A directed graph is one that only permits movement in one direction across a bond. The graphs used in this study are directed to an extent; bonds in the direction the lattice is being traversed are directed, but bonds normal to those are not. With each arc there is also an associated cost. More information will follow on the relationship of these costs to actual physical situations.

A shortest-path algorithm, then, is one that seeks to find the minimum cost path across a graph of fixed nodes, bonds, and costs. Applications of this sort of algorithm are numerous: superconductors, semiconductors, computer networks, and even traveling salesmen can make use of such algorithms. There are two types of shortest-path algorithm. A local algorithm is one in which the particle crossing the graph has a knowledge of the structure of the graph only in its immediate vicinity. It must select from the bonds it is adjacent to based on which of those has the lowest cost. While a local algorithm can find the lowest-cost path, especially in the case of a small graph, it will not always. An algorithm that always finds the lowest cost path is called a global algorithm. The traversing entity then has knowledge of every other bond in the lattice. This type of algorithm chooses the lowest-cost path from all the possible paths the particle can take.
II. Prim’s and Clark’s Local Algorithms

Probably the most famous of the algorithms which work by selecting only the cheapest bond is Prim’s algorithm. It begins at the lowest-cost bond in the graph, and produces a spanning tree by choosing the minimum-cost bond at each step. A spanning tree is just a connection of the nodes such that every node on the graph is visited. A minimal spanning tree is the tree with the smallest number of bonds that touches every site\(^1,3\). Prim’s algorithm has many uses, for example, forest fire closely resembles Prim minimal spanning trees.

Clark’s algorithm is a local algorithm that creates, instead of a minimum spanning tree, merely a path that a particle with local knowledge would take through a lattice of random costs. This algorithm is similar to those used to model the diffusion of single gas molecules, and it will be shown how some of the statistics compare to those of the diffusion models. In addition, a comparison can be made to a drunk person on a mountain versus a sober person who has remembered to bring a map. The diffusion path might be likened to that of the drunk man traversing the mountain based only on his limited vision. A shorter path will be taken by the person with a map, who can see the entire terrain at once. Well, here’s what it’s like to be drunk on a mountain, a description of Clark’s algorithm:

**INPUT** lattice size, starting point

**GENERATE** random numbers for the proper number of bonds

**SELECT** the shortest neighboring bond

**WRITE** to file total cost so far, x and y coordinate

**REPEAT** until other side is reached.
PRINT total cost, number of iterations

The source code can be found online at http://www.pa.msu.edu/~clarkr/research.html, or requested from me at r-clark@onu.edu. The above is merely a simplified description of how it works. There are really several algorithms that grew out of that one and are worth mentioning:

**S2:** Goes from one corner to the opposite corner. Can only go up and to the right. Thus, it has two degrees of freedom. (Fig. 1)

**S2P:** Goes from midpoint of one side to the opposite side. Moves ahead one bond each time and either up or down based on which is the shorter bond. Plots a probability distribution for the final x-coordinate and total cost. (Fig. 2)

**C22:** Corrected version of the regular algorithm outlined above, prohibiting visiting sites that have already been visited. Freedom to move forward and to either side gives it three degrees of freedom. (Fig. 3)

**C2P:** Similar to S2P, graphs probability distributions for final x-coordinate and total cost for C22. (Fig. 4)

**C26:** Finds average final height over a large number of runs for several lattices, in hopes of finding the dependence of path length on lattice size. (Fig. 5)
III. Dijkstra’s Algorithm

The most common algorithm for finding the absolute lowest-cost path across a graph is Dijkstra’s algorithm. It works in a manner similar to Prim’s, except that instead of selecting only the lowest-cost bond, it finds the shortest of all possible paths from the next bond to the first node\(^1\). In this manner it generates the true shortest path. Dijkstra’s algorithm is used to model, as one example, rivers, which take the lowest-energy path across terrain. Here the type of potential is gravitational, but Dijkstra’s algorithm can be applied to electrical systems as well. As will be shown later, it is useful in finding the minimum voltage at which a semiconductor such as a varistor will conduct electricity, as well as determining the entire V-I characteristic for the device. Fig. 6 shows a graph which will be used to explain how Dijkstra’s algorithm works.

Dijkstra’s algorithm begins like a local algorithm in that the first node grown toward is the cheapest reachable from the starting node. However, in growing out to each node you must choose the cheapest route from the starting node. This requires keeping track of all the possible paths to the site. In the end, Dijkstra’s algorithm finds the shortest path from the starting node to all nodes in the graph. (This makes sense since all paths to each node must be remembered). A picture of a Dijkstra shortest path tree (a tree grown using Dijkstra instead of Prim) is given in Fig 7.

Some analysis was also done comparing the scaling exponent of final heights in Dijkstra’s algorithm with that of Clark’s algorithm. This scaling exponent is the power \(\theta\) in the expression \(\langle h \rangle \sim L^\theta\), where \(h\) is the absolute value of the difference in starting and final height on the lattice, and \(L\) is the lattice size (for a square lattice). The theoretical value of \(\theta\) for a local algorithm is 1/2 and for a global algorithm is 2/3. I obtained some interesting (some would say
wrong) results for these two values. For Clark’s algorithm \( \theta \) worked out to be about .543, as is shown in Fig. 5. For Dijkstra’s algorithm the exponent had a value of .787, a percent error of 18%, but some explanation for this is that not enough runs were taken to find correct average final height differences. A rather primitive approach was taken which involved rather tedious counting of graph divisions and averaging by hand, not enough time existing to alter the source code. Despite this, a graph of this fit is given in Fig. 8. In the future, more work should be done (by someone other than an REU student who only has ten weeks) to confirm that this exponent is indeed 2/3.

IV. The Varistor Problem

The application of Dijkstra’s algorithm I explored involves variable resistors, or varistors. These are semiconductors which exhibit extremely non-ohmic behavior, i.e. their resistance varies (hence the name) greatly across a range of voltages, producing a highly nonlinear V-I curve. Typically made of grains of ZnO or some similar material, the resistance to current flow occurs chiefly across grain boundaries, not within the grains. Of particular interest to us is the minimum voltage at which a varistor will begin conducting appreciable amounts of electricity. The electric field associated with such a voltage is called the \textit{threshold field}. It turns out that the threshold field is mathematically related to the lowest-cost path across the varistor. Since the grains and boundaries of the varistor make a good analogy to the nodes and bonds of a graph, we can use a shortest-path algorithm such as Dijkstra’s to find this threshold field.

One may wonder why it is that a global algorithm rather than a local one is used. After all, when an electron is going through a varistor, how does it know which path will work out to
be the shortest? Shouldn’t it be akin to the drunk guy on the mountain, taking the easiest path at every turn? The answer is yes, at first, when the first electrons are entering the varistor. However, the threshold field is related to steady-state flow, for which a certain set of conditions must be met. First of all, the cost function must be minimized. This function is expressed by

$$Cost = \int_{0}^{I'} V(I')dI'$$

For an Ohmic resistor, this expression is proportional to the power, $I^2R$. It is clear that current will flow best when the energy dissipated by the varistor per unit time (power or cost) is minimized. Additionally, Kirchoff current conservation applies to the varistor, which is another constraint the flow must conform to. These two conditions provide a steady-state solution to the problem of where the current is to go, and it is in this manner that each node “knows” the cost function of every boundary. That could not happen immediately, in fact a local algorithm is appropriate at first, but it does happen after a short period of time.

Fig. 9 shows a V-I curve for a varistor generated using Dijkstra’s algorithm combined with a graphing program (similar to the one used to generate the tree in Fig. 7). Notice the sharp turn in the graph: this is the point at which the threshold field is reached.

V. Remarks

The subject of study this summer involves the nexus of several interesting, but different fields. There is, of course, the field of computer science, which provides the algorithms. Physics, moreover, gives these constructions meaning in the real world. This type of algorithm can be applied to many types of quantum statistical and otherwise disordered systems, including random magnets, quantum spin problems, percolation, and even problems such as the proper way to
remove oil from a well. It is this combination of intriguing theory and myriad applications that makes this field so attractive.

My chief accomplishment was inventing several incarnations of a new local algorithm that hopefully will be found useful by someone. In addition, data was generated using Dijkstra’s algorithm which models quite well known properties of varistors. The most rewarding things about it for me were learning a lot more about computer programming and experiencing a kind of scientific research that was new to me and, well, kind of fun.
Fig. 1:

The path across a 1000 by 1000 lattice generated by S2
Fig. 2

Distribution of Final Heights for an S2-type Algorithm

100 by 100 lattice, 100 runs
Fig. 3

The path generated by C22

1000 by 1000 lattice
Fig. 4

Distribution of final heights for C2-type algorithm

100 by 100 lattice, 50 runs
Fig. 5

Curve fit of average final height difference vs. lattice size

Exponent is .543, as compared to the 1/2 normally reported for diffusion-type paths.
Fig. 6

Explanation of Dijkstra’s Algorithm
Fig. 7

Tree generated using Dijkstra’s algorithm
Fig. 8

Plot of Final Height vs. Lattice Size using Dijkstra’s Algorithm

Exponent is .786
Fig. 9

V-I Curve for a Varistor
Bibliography

1. *Exact Combinatorial Algorithms: Ground States of Disordered Systems*, Duxbury et.al.,


Acknowledgements

Special thanks to Aleksander Donev for providing Fig. 7 and the plotting algorithm, and to Corey Musolff for help with the V-I plot. Thanks also to Radu Dobrin, Jan Meinke, and especially Dr. Duxbury for all your help this summer. You have taught me a lot- best of luck to all of you in the future.