Graph Theory in GeoGebra

Trevor Kenneth Williams & David E. Brown

Abstract

Graph theory is a visual field of mathematics. It is useful for graph theorists and students to visualize the graphs they are studying. Although software to visualize and analyze graphs such as SAGE exist, they can be difficult to learn. GeoGebra, although user-friendly, provides no automated way to make or analyze graphs. In the following article, the authors illustrate how JavaScript may be used to extend the capabilities of GeoGebra to build graph theory tools. This work is based on a chapter from the Master's Thesis (Williams, 2017).

Keywords: Graph Theory, JavaScript

1 INTRODUCTION

Graph theory is the study of objects and the relationships between them. These objects and their relationships are studied using graphs. In graph theory, a *graph* is a collection of vertices (which represent the objects) and edges (which represent the relationships between the objects). Graphs are usually defined as an ordered pair, $G = (V, E)$, where V is a set of objects, called the vertex set, and E, the edge set, contains subsets of size two of V. Two vertices, u, v, are called *adjacent* if $\{u, v\} \in E$. The 2-set $\{u, v\}$ is an edge. An edge is *incident* to a vertex if the vertex is contained in the edge.

It is often convenient to represent a graph as a diagram. For example, consider the graph, G, with vertex set

$$V = \{v_1, v_2, v_3, v_4\} \quad \text{and edge set} \quad E = \{\{v_1, v_2\}, \{v_2, v_3\}, \{v_3, v_4\}, \{v_1, v_4\}, \{v_2, v_4\}\}$$

A diagram of graph G is shown in Figure 1. It is common to refer to these diagrams as graphs themselves. Note that G has four vertices and five edges; vertices v_1 and v_4 are adjacent, but vertices v_1 and v_3 are not. Edges are generally named based on the vertices they connect. For example, the edge that connects v_1 and v_4 may be called edge $\{v_1, v_4\}$. Also, note that $\{v_1, v_4\}$ is incident to vertex v_1 and vertex v_4.

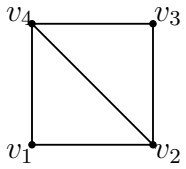
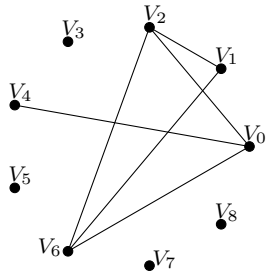


Figure 1. A graph with 4 vertices and 5 edges.

2 Graphs and Adjacency Matrices

Adjacency matrices provide us with an alternative way to represent a graph. Let G be a graph on $V(G) = \{V_1, \ldots, V_n\}$. The adjacency matrix of G is the $(0,1)$-matrix $A = [a_{ij}]$ with $a_{ij} = 1$ if and only if $\{V_i, V_j\} \in E(G)$. Figure 2 shows an example of a graph and its adjacency matrix.



$$\begin{array}{c} \\ V_0 \\ V_1 \\ V_2 \\ V_3 \\ V_4 \\ V_5 \\ V_6 \\ V_7 \\ V_8 \end{array} \begin{array}{c} \begin{array}{ccccccccc} V_0 & V_1 & V_2 & V_3 & V_4 & V_5 & V_6 & V_7 & V_8 \end{array} \\ \begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

Figure 2. A graph and its adjacency matrix.

The first applet that we created generates a random graph with 1 to 10 vertices along with its corresponding adjacency matrix. The applet contains two JavaScript buttons, `Create Graph` and `Generate Matrix`. Students press the `Create Graph` button, generate the corresponding adjacency matrix by hand, then check their work by pressing the `Generate Matrix` button.

2.1 Generating the Graph

In order for our applet to be useful, it needs to be used repeatedly. Thus, when students press the `Create Graph` button, the applet must delete any previous graphs and adjacency matrices. The JavaScript highlighted in Code Block 1 provides this functionality.

Code Block 1. Deleting previous graphs and matrices.

```
for (var i=0; i < 11; i++){
  ggbApplet.deleteObject("V_{"+i+"}");}
ggbApplet.deleteObject("text1");
ggbApplet.deleteObject("text2");
```

Next, the applet creates a random number, n, between 1 and 10 that represents the number of vertices in the graph. Variable n is also used in the `Generate Matrix` button code. Next, vertices are created and polar coordinates are used to space them evenly. Code that executes these steps is shown in Code Block 2.

Code Block 2. Creating a random number of vertices.

```
// Generate a random number of vertices
var n= Math.floor((Math.random() * 10) + 1);
// Store the number of vertices in GeoGebra for other button
ggbApplet.evalCommand("n="+n)
// Create the vertices so that they are evenly spaced (using polar coordinates)
for(var k =0;k<n;k++) {
  ggbApplet.evalCommand("V_{"+k+"}=(4;("+k+"*(360/"+n+"))\textdegree)");
  // Make all the points bigger
  ggbApplet.setPointSize("V_{"+k+"}", 5);}
```

Next, edges are created randomly using the code shown in Code Block 3.

Code Block 3. Creating random edges.

```
for (var q=0; q<n; q++){
   //Generate a random number associated with vertex q.
   var r= Math.random();
   for(var w=0; w<n; w++){
     //Generate a random number associated with vertex w.
     var t = Math.random();
     //If there is already an edge between the vertices, do nothing
     if(ggbApplet.exists("e_{"+w+","+q+"]")){
       break;}
     //If the edge doesn't exist do the following
     else{
       //Condition on the random numbers for vertices q and w to create random edges
       if( r < 0.7 && t < 0.7){
         //if w and q are the same vertex do nothing
         if(w==q){
           break;}
         //if not make the edge
         else{
           ggbApplet.evalCommand("e_{"+q+","+w+"}=Segment[V_{"+q+"},V_{"+w+"}]");}}
       else{
         break;}}}}
```

2.2 *Generating the Matrix*

After this, the applet generates the associated adjacency matrix using the code highlighted in Code Block 4. Note that this functionality requires JavaScript code that names edge segments in such a way that incidence is implied by the name of the edge. The code in Block 4 first checks segment names to determine if two vertices are adjacent and then builds a matrix.

Code Block 4. Analyzing the graph and building the matrix.

```
// Get the number of vertices from GeoGebra
var n= ggbApplet.getValue("n");
// Create an empty string to put our matrix values in later
var matrix = ""
for (var i=0; i<n; i++){
  //Create an empty array so that the edges can be given a value
  var e=[]
  for (var j=0; j<n; j++){
    // If there is an edge between vertex i and j then make variable e[i,j]=1
    if (ggbApplet.exists("e_{"+i+","+j+"}") || ggbApplet.exists("e_{"+j+","+i+"}")){
      e[i,j] ="1"}
    // Otherwise make the value of e[i,j]=0
    else{
      e[i,j]="0"}}
  // Make an empty variable so that the rows can be built
  var row = []
  // Make the first row
  row[i]=e[i,0]
  // Make the remaining rows and get them in the form that GeoGebra will use
  for (var k=1; k<n; k++){
    row[i]=row[i] + ","+ e[i,k]}
  row[i]= "{" + row[i] +"}"
  // Gather all our rows into one variable in the form GeoGebra will use
  if(i==0){
    matrix = matrix + row[i];}
  else{
    matrix = matrix +","+ row[i];}}
```

Code Block 5 on the next page lists code that tells GeoGebra to display the matrix in the graphics window.

Code Block 5. Placing the matrix in GeoGebra.

```
// Tell GeoGebra to make the matrix
ggbApplet.evalCommand("text1=FormulaText[{"+matrix+"}]")

// Make the matrix invisible
ggbApplet.setVisible("text1", false)

// Copy the matrix and tell GeoGebra where to place it
ggbApplet.evalCommand("text2=Text[ text1, (5.58, 1.7), false, true ]")
```

The `Create Graph` and `Generate Matrix` buttons provide students with simple and effective tools for understanding graphs and their adjacency matrices. A screenshot of the finished applet is shown in Figure 3.

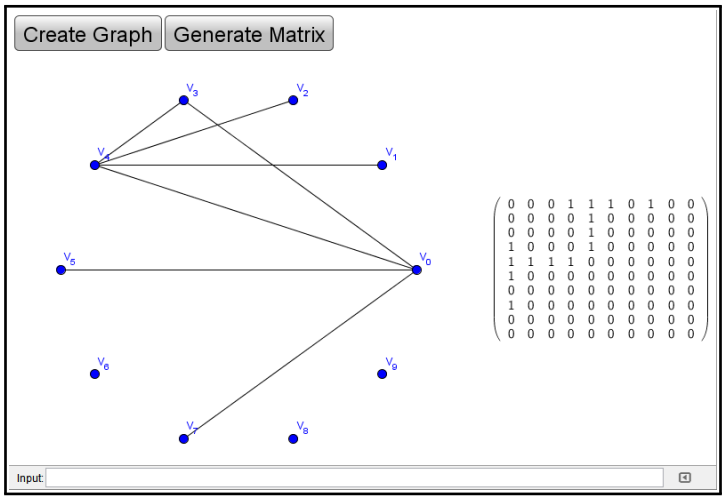


Figure 3. Adjacency matrix applet.

3 Tournaments, Score, and Kings

We constructed a second applet to help our students explore a specific class of graphs called tournaments. A *tournament* is a directed graph with an arc between every pair of vertices; that is, between any two vertices x and y either there is an arc from x to y, denoted $x \to y$, or an arc from y to x, denoted $y \to x$. If there is an arc from x to y we say x *beats* y. A round-robin tournament is a well-known example. In a round robin tournament, each team plays every other team with no ties. For example, consider the tournament depicted in Figure 4.

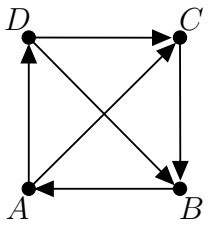


Figure 4. A tournament with 4 vertices.

The nature of a tournament leads to an ambiguous interpretation of the most dominant vertex. If the vertices represent sports teams and the arcs represent games played between the teams, then it is unclear which team is the "best." As the figure suggests, team A and team D both won two games. Team A beat team D, however, team B was beaten by team D and beat team A.

In an attempt to determine the dominant vertices in a tournament, mathematical sociologist H. G. Landau developed the notion of a king and proved that every tournament has at least one (Landau, 1951). A *king* in a tournament is any vertex, v, such that for every other vertex, x, either $v \to x$, or there exists a vertex, y, such that $v \to y$ and $y \to x$. It is also valuable to note how many vertices a vertex beats. This is called the score of a vertex.

Our second applet can determine the score of each vertex and which vertices are kings. This applet was built to use data from the website of Brendan McKay of Australian National University (see `http://users.cecs.anu.edu.au/~bdm/data/digraphs.html`). McKay stores tournaments as a string of 1's and 0's. These strings are the upper triangle of the tournament's adjacency matrix. Figure 5 provides an example of a tournament and its adjacency matrix.

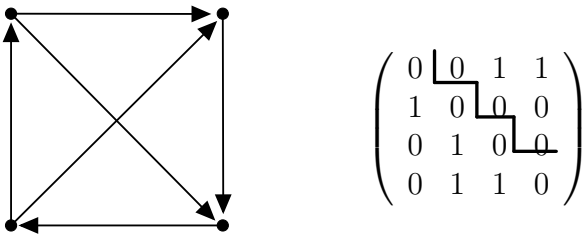


Figure 5. A tournament with 4 vertices and its adjacency matrix.

Notice that the diagonal entries of the adjacency matrix in Figure 5 are all 0. Furthermore, when entry (i, j) is 0, then entry (j, i) is 1. For example, for the adjacency matrix in Figure 5 there is a 0 in row 1 column 2, and there is a 1 in row 2 column 1. This is because the beats relationship in a tournament is anti-symmetric. If the upper triangle of the matrix is known then the entire matrix can be generated. Using McKay's method, the tournament in Figure 5 is stored as the string 011000 which is simply the upper triangle of the matrix read from left to right, top to bottom, excluding the diagonal entries. Our applet consists of an input bar, instructions, and four buttons—namely, `Show Kings`, `Hide Kings`, `Show Scores`, and `Show Vertex Names`. Each button and the input bar use JavaScript code to accomplish their tasks. We present this code below.

3.1 Input Bar

The input bar is used to copy and paste the tournament string and contains the code to build the tournament. The JavaScript code is shown in Code Block 6.

Code Block 6. `Input Bar` code.

```
// Delete Previously Existing Tournament
for (var i=0; i < 50; i++){
  ggbApplet.deleteObject("V_{"+i+"}");
}
// Determine number of vertices from uppertriangle of adjacency matrix
var str= ggbApplet.getValueString("t");
var n = Math.floor((1 + Math.sqrt(8 * str.length() + 1)) / 2);
// Create n vertices evenly spaced and correct size
var v = 0;
for(var k =0;k<n;k++) {
  ggbApplet.evalCommand("V_{"+k+"}=(31;("+k+"*(360/"+n+"))\textdegree)");}
for(var m=0;m<n;m++){
  ggbApplet.setPointSize("V_{"+m+"}", 5);
}
// Use the uppertriangle of adajacency matrix to build relationships
for (var i=0; i < n; i++){
  for (var j=i+1; j<n; j++){
    var p=String(str.charAt(v));
    if (p == 1){
      ggbApplet.evalCommand("v_{"+i+","+j+"}=Vector[V_{"+i+"},V_{"+j+"}]");
    }else{
      ggbApplet.evalCommand("v_{"+j+","+i+"}=Vector[V_{"+j+"},V_{"+i+"}]");
    } v++;
  }
}
```

3.2 Show Kings

The `Show Kings` button executes code that analyzes the tournament and determines which vertices are kings, then runs more code that highlights those vertices. Recall that a king is a vertex, k, such that for every other vertex in the tournament, x, either $k \to x$ or there exists a vertex, y, such that $k \to y$ and $y \to x$. JavaScript associated with this functionality is shown in Code Block 7.

Code Block 7. `Show Kings` button code.

```
// Find out how many vertices are in the tournament
var str= ggbApplet.getValueString("t");
var n = Math.floor((1 + Math.sqrt(8 * str.length() + 1)) / 2);
// For each vertex i does it beat each other vertex j?
// if not does it beat a vertex that does beat j?
function isKingOf(i, j){
  if(ggbApplet.exists("V_{"+i+"}") == false){
    return true;}
  if(ggbApplet.exists("V_{"+j+"}") == false){
    return true;}
  if(i == j){
    return true;}
  if (ggbApplet.exists("v_{"+i+","+j+"}")){
    return true;}
  for (var k=0; k < n; k++){
    if (ggbApplet.exists("v_{"+i+","+k+"}") && ggbApplet.exists("v_{"+k+","+j+"}"))
      return true; }
  return false;}

function isKing(i){
  for (var j=0; j < n; j++){
    if (isKingOf(i, j) == false)
      return false;}
  return true;}
// If vertex i is a king change its color to red
for (var i=0; i < n; i++){
  if (isKing(i)) {
    ggbApplet.setColor("V_{"+i+"}",255,0,0);}
  else{
    ggbApplet.setColor("V_{"+i+"}",0,0,255);}}
```

3.3 Hide Kings

The code executed by pressing the `Hide Kings` button changes all the vertices back to the same color. This code is shown in Code Block 8.

Code Block 8. `Hide Kings` button code.

```
// Find out how many vertices are in the tournament
var str= ggbApplet.getValueString("t");
var n = Math.floor((1 + Math.sqrt(8 * str.length() + 1)) / 2);

// Turn all vertices blue
for(var i=0;i < n; i++){
  ggbApplet.setColor("V_{"+i+"}",0,0,255);
}
```

3.4 Show Score

As Code Block 9 suggests, the code executed by pressing the `Show Score` button analyzes the tournament to determine the score of each vertex.

Code Block 9. `Show Score` button code.

```
// Find out how many vertices are in the tournament
var str= ggbApplet.getValueString("t");
var n = Math.floor((1 + Math.sqrt(8 * str.length() + 1)) / 2);

// Find out how many other vertices each vertex beats
for(var i=0; i < n; i++){
  var t = 0;
  for (var j=0; j < n; j++) {
    if (ggbApplet.exists("v_{"+i+","+j+"}")) {
      t = t + 1;
    }
  }
    // Change the label of the vertex to be the score
    ggbApplet.evalCommand("SetCaption[V_{"+i+"},\""+ t+"\"]");
}
```

3.5 Show Vertex Names

The code executed by the `Show Vertex Names` button changes the visible labels on the vertices to their given names. This is shown in Code Block 10.

Code Block 10. `Show Vertex Names` button code.

```
// Find out how many vertices are in the tournament
var str= ggbApplet.getValueString("t");
var n = Math.floor((1 + Math.sqrt(8 * str.length() + 1)) / 2);

// For each vertex change its label to its name
for(var i=0; i < n; i++){
  ggbApplet.setLabelStyle("V_{"+i+"}", 0)
}
```

Figure 6 shows a screenshot of the tournament applet.

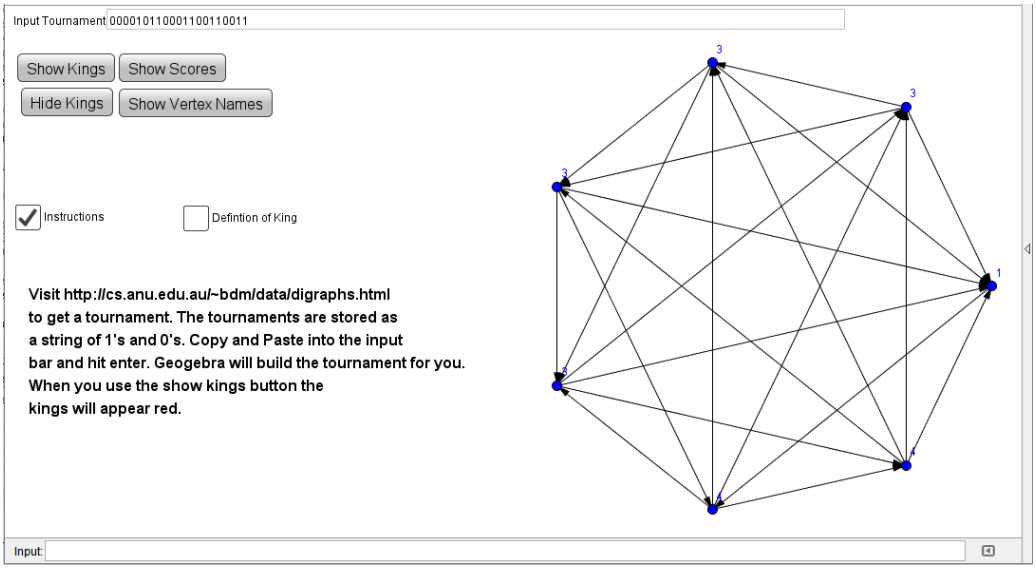


Figure 6. Tournament applet.

3.6 Pedagogical Value

The tournament applet provides our students with the ability to explore graphs in a manner that is not possible with pencil and paper, chalk, or a textbook alone. The applet enables students to explore tournaments interactively, "playing" with them as they pose and prove conjectures. For instance, we've used the sketches to motivate the following theorems in our course:

Theorem 1. *(Landau, 1951) Every tournament contains a king.*

Theorem 2. *(Landau, 1951) Let s be the highest score in a tournament, then any vertex with a score of s is a king.*

Theorem 3. *(Maurer, 1980) In a tournament, every vertex that is beaten is beaten by a king.*

Theorem 4. *(Maurer, 1980) No tournament has exactly 2 kings.*

Theorem 5. *(Maurer, 1980) A tournament has exactly one king, k, if and only if k beats every other vertex in the tournament.*

Theorem 6. *(Maurer, 1980) A tournament with 4 vertices cannot have 4 kings.*

As students engage with the applet, they develop conjectures and ultimately prove facts about tournaments in ways that cannot be achieved through reading alone.

4 Conclusion

With the use of JavaScript, GeoGebra is a useful tool for visualizing and analyzing graphs and tournaments. The applets we've highlighted in this paper have helped our students strengthen their understanding of these concepts. The applets represent a small portion of what is possible in GeoGebra. We encourage you to download them at `https://www.geogebra.org/u/johndoe314` and extend their functionality. Note that the applets must be downloaded and opened in the GeoGebra standalone application as they do not function properly on the GeoGebra website.

REFERENCES

Landau, H. G. (1951). On dominance relations and the structure of animal societies i:. effect of inherent characteristics. *The Bulletin of Mathematical Biophysics*, *13*(1), 1–19. doi: 10.1007/bf02478336

Maurer, S. B. (1980). The king chicken theorems. *Mathematics Magazine*, *53*(2), 67–80. doi: 10.2307/2689952

Williams, T. (2017). *Combinatorial games on graphs* (Unpublished master's thesis). Utah State University. (https://digitalcommons.usu.edu/etd/6502)

Trevor Williams, is a PhD student at Florida Atlantic University. He holds a Bachelor's degree in Mathematics Education and a Master's degree in Mathematics, both from Utah State University. He has many years of experience teaching mathematics subjects at all levels. His current research interests are in undergraduate mathematics education and combinatorial game theory.

David E. Brown is an Associate Professor of Mathematics at Utah State University and Associate Head of the Department of Mathematics and Statistics.