Design: Fiercely Independent Educational Web App

Goal

The goal of this web app game is to provide a gentle and discovery-based introduction to NP-completeness. The user is asked to manually find a largest independent set in a randomly-generated unweighted, undirected graph. The puzzles will become increasingly large and edge-dense (both increasingly logarithmically), incresing the difficulty and familiarizing the user with a detailed understanding of the complexity of NP-complete problems.

We will track the total score, i.e. the sum of the sizes of all independent sets of all puzzles the user has completed, and store these statistics in a database, to lend progress-based motivation to the user.

1 Implementation

Hosted by pythonanywhere.com; login and register pages as specified by a database-backed website tutorial. Main page will feature an interactive canvas element where the graph puzzle appears. Initial GUI version will display nodes in an evenly-spaced circle. Later improvements to the GUI will allow user rearrangement of the graph (click to drag?).

1.1 Interfaces

The internal design will be object-oriented.

1.1.1 Loc Class: Methods

The `Loc` class will maintain the pixel-location fields `Loc.i` and `Loc.j`, where the center of the canvas element is $(0, 0)$. `Loc` objects will be used to specify the location of graph nodes, click events, and of endpoints of line-segments (for edge drawings). Supports the `locEquals()` method for deep equality testing.

1.1.2 NodeDrawing Class

Attributes: location (`Loc` object), color (whether selected). Methods: `drawNode()`.

1.1.3 EdgeDrawing Class

Attributes: locations first and second. Methods: `drawEdge()`.

1.1.4 Graph Class

Attributes: screen size, adjacency matrix, number of vertices, number selected, independence number, array of node drawing `NodeDrawing` objects.

This class will also track the canvas state to enable drag-and-drop animation. Attributes: mouse state and position, and currently selected node.

1.2 Design

On initialization, a graph will be randomly generated within the graph size and density specifications. The graph independence number will be computed and stored in the `Graph.indepNum` attribute. The nodes will be drawn evenly spaced on a circle whose radius is about $\frac{1}{3}$ of the square canvas element side length. The edges between nodes will be drawn in dark green as specified by the adjacency matrix.

When a user clicks on a node, it will be selected and colored dark green, and all of its neighbors will be deselected and returned to the original blue. When the user manages to reach a state where a maximum independent set is selected (α nodes selected, where α is the independent number of the graph), the puzzle is considered solved. Score is incremented and stored in the database; a congratulatory message is printed, and upon user taking option to proceed, a new graph puzzle is generated.

Drag and drop as previously described will be available in later versions.[1]

[1] This resource will be helpful for building interactive canvas shapes from scratch: https://simonsarris.com/making-html5-canvas-useful/