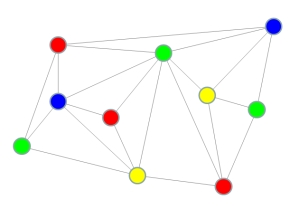
Anton Ridgway

December 17, 2014

**Project Objective and Scope**

The objective of this project was to develop a method to visualize constraint-satisfaction problem spaces in general. Intuitive methods exist to visualize specific CSP application domains, and graph representations are often useful to show problem elements and how they’re connected (for example, the graph coloring image included below). For this project, however, I was interested more in investigating new methods to show the entire problem *space* for CSP’s in general.



Graph-coloring example image from <http://web.engr.illinois.edu/~slazebni/fall13/assignment2.html>.  
The goal of this CSP is to color each node with one of four colors, where each node must be colored differently from its neighbors. A single configuration, with the topology of variables and binary constraints, is represented here.

Typically, problem spaces are represented as decision trees, where each branch represents some choice that helps to determine a given end-state. For constraint satisfaction problems, each level in the tree could be assigned to one of the variables in the problem, while each branch represents a domain-value assignment for that variable. Since any combination of variables and values is possible, however, the number of end-state leaf nodes in the tree expands rapidly, often making the resulting tree infeasible to show using unspecialized tree representations. For a simple problem with only eight variables and three domain values, for example, the resulting tree contains 6,561 leaf nodes and 9,841 nodes total.

Techniques to alleviate the space requirements of tree representation and produce better images might include using radial rather than hierarchically-organized trees, to make better use of 2D space as the tree expands on each level, or developing an interactive 3D representation, to provide more space through depth. I was also definitely interested in developing a mesh-based representation that could translate the general structure of a radial tree to a surface, using the number of satisfied constraints associated with each node to determine its height.

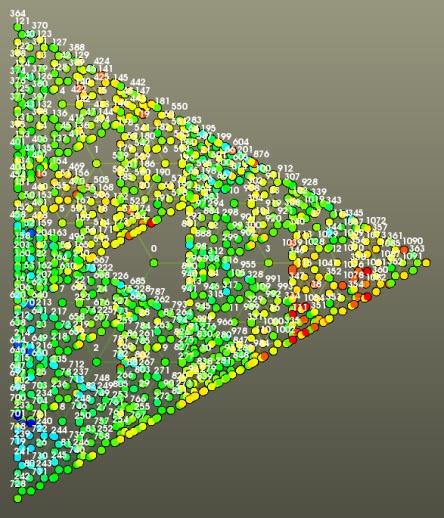
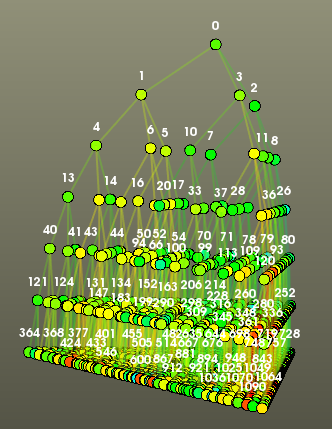
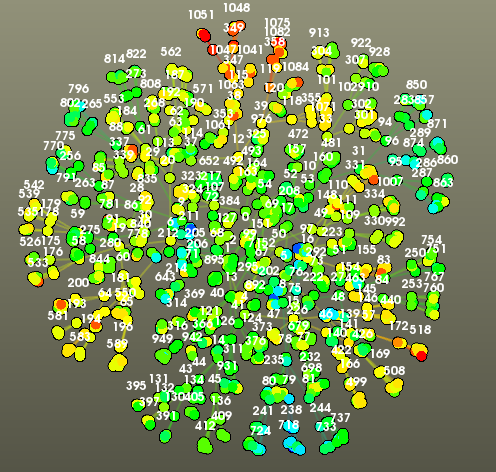
The scope of the work done in this project was to investigate CSP problem space representations that can be easily produced in VTK (for instance, graph visualizations), and then to investigate the process needed to make use of the mesh-based representation described above, and assess the result. This included the development of the basic “pipeline” program needed to generate the desired data formats from the initial CSP datasets available to me.

**Final Results**

Ultimately, I was able to produce some reasonably informative visualizations for the problem space using functionality provided in VTK and Microsoft Excel.

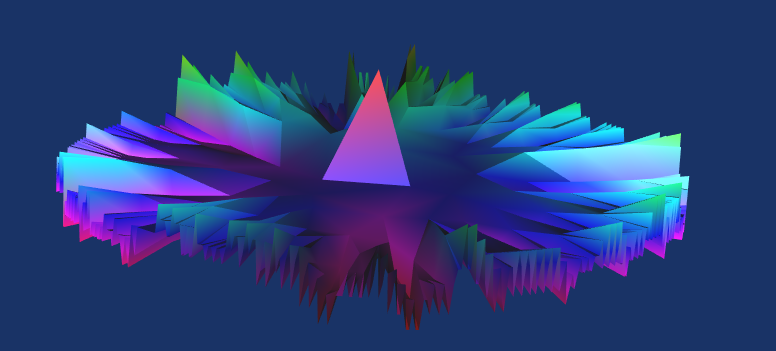
Using only the data from the leaf nodes of the tree, I was able to illustrate the general distribution of solved constraints for each branch of the problem space, which helps us to characterize problem difficulty. A variety of methods would be useful to extract trends from this data, but I focused on a simple histogram. Unfortunately, I wasn’t able to find a lot of support for one-dimensional data manipulation in VTK, so I made use of Excel’s histogram library to generate some simple graphs. These show how the uniform distribution of problem characteristics during problem generation led to a normal-like distribution of solution qualities. In general, these graphs could help to show qualitatively whether a problem instance is easy (when the distribution is weighted to the right) or hard (weighted to the left).

Working with graph-based methods, I was able to more fully represent the different branches of the problem space, and the leaf nodes’ relationships, while using color to represent relative value. I looked at a variety of different layouts, and generally found them to be more visually informative than the visualizations of leaf node data only. It also seems to suggest that high- and low-quality solutions are grouped together in different areas of the problem space (this is best illustrated by the spanning-tree visualization, to the right below). Unfortunately, the lack of tree-based layout support in VTK’s binary Windows distribution prevented me from experimenting with tree-based layouts, which would be the most natural choice.

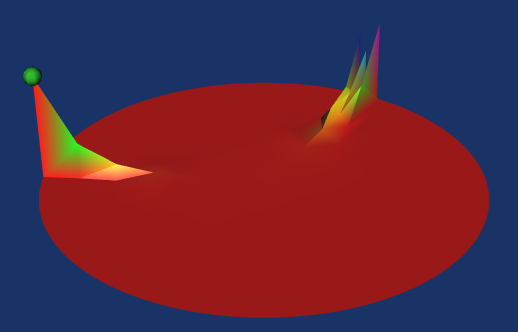
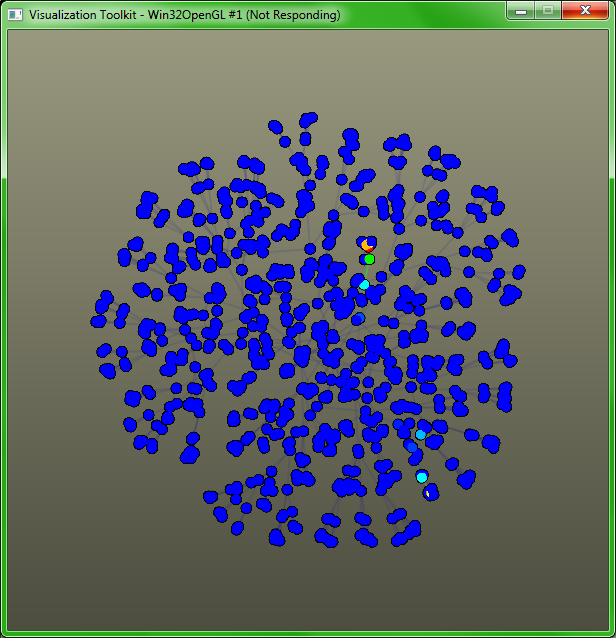
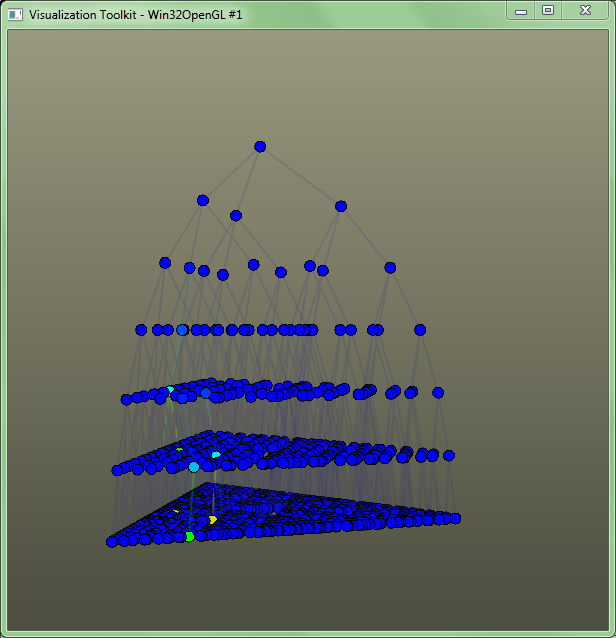


Finally, when working with the mesh-based representation, I found that I was able to show a large number of nodes in a fairly compact space. I generated the meshes using my pipeline tool, and added normals using VTK’s vtkPolyDataNormal class. I drew from a full hue range (so that red occurred at maximum and minimum values) to color each vertex according to its height.

An interactive view of the full problem space landscape allowed the user to see local and global maxima and minima as peaks and troughs, though the random distribution of values also lent itself to a very jagged outline. The general grouping of problem space areas is also visible here; in the image below, a lower-quality violet-red area faces the camera, while a healthier green area can be seen in the distance.



I also used VTK’s animation functionality to create animations that show algorithm progress through the space over time, by altering the mesh structure as new configurations are tried. In the mesh-based animation, a small sphere was also added to indicate the highest-point discovered at any given time, and it was turned green at the end of the animation to indicate that the algorithm had finished. In general these animations succeeded in showing how CSP algorithms move through the problem space over time.



However, because of the limited problem sizes I was able to work with in the time I had allotted for testing, most of the algorithms are able to solve their problems fairly quickly, and the resulting graphs are not as interesting or informative as they might be in full-scale. In principle, however, these animations were successful in meeting their goals—algorithmic behaviors can see be seen (algorithms exploring options in the same vicinity, or suddenly jumping elsewhere), and the algorithms compare with each other as we expect. For example, though the scale is small, we can still see that the AFCCBJ algorithm finds an answer to its problem faster than the algorithm it’s intended to improve on, APO (taking 5 steps instead of 6 to complete).

**Process and Tools**

I generated my initial datasets using the FarmX multi-agent systems simulator, which supports several different distributed CSP algorithms, and can generate problems for them randomly. The resulting data files had the following format:

* Total number of domain values on the first line
* Total number of constraints on the second line
* And for each successive line, a single time step in the algorithm’s process was represented, by:
  + The number of satisfied constraints on this step
  + The value assigned to each variable on this step

The pipeline tool I developed was written in Java, and designed to read in this format, and output CSV and OBJ files that VTK could read. Its general process was to build up problem-space trees based on each line from the input file, and use those trees to output files to represent each step in the problem-solution process. A number of combinatorics tricks were needed to solve the non-trivial problem of indexing vertices and assigning vertices to faces when creating the mesh output, but the other output formats were mostly straightforward. (This triangulation process could potentially be circumvented by only outputting a point cloud and using VTK’s vtkDelaunay3D class to perform triangulation for us, but I didn’t realize that that utility existed, and as such I can’t testify to its appropriateness for this situation.) Generating FarmX output using a trivial algorithm that walks through all possible CSP configurations also allowed me to generate output files that represent the whole problem landscape in the same way.

Last, I used VTK to create the actual visualizations. I created interactive, still visualizations of full problem spaces using VTK’s vtkGraphLayoutView and its associated interactor, as well as vtkPolyDataMapper and render window classes. Then, I created non-interactive animations that display multiple frames of algorithm operation (rotating the camera as necessary) by building on top of the structure of the interactive programs with VTK’s vtkAnimationScene and vtkAnimationCue classes. These work by allowing the user to specify time parameters for the whole scene, then for each set of elements to be animated together (via cues), and to add callback methods for the timing events these generate.

**Conclusions and Possible Future Work**

The datasets that I actually had time to work with were for fairly small trivial cases, but I was nevertheless impressed with the way that the different visualization approaches were able to convey a decent amount of information from a large number of initial data points. In moving toward working with larger problems, my guess is that a mesh-based solution would probably hold up the best of the three animation types tried, since the more intense point layout calculations are performed in advance, and the rendering is performed on a single entity composed of triangles instead of a much larger number of separated elements. Some form of data processing that sums nodes together into representative “average” nodes might also be a possible way to deal with bigger data moving forward.

It’s also true that, while the visualization met its proposed goals, the uniformly random nature of the problems I worked with made overall trends difficult to identify (for example, just by looking at the jagged edge on the mesh). Future work on the project might also entail finding ways of smoothing out the edges (or reorganizing the problem space tree) to better show general problem properties.

I would also be interested to see more types of animations—animations of how the problem landscape changes as the number of variables changes, or as the density or tightness of the problem is varied, could be very informative. I did generate datasets for these, but did not have the time to pursue creating fully-featured animations.

Finally, better ways of processing the initial data to generate the mesh might be needed for animation. Since in most cases CSP algorithms don’t place the variables in a hierarchy, it’s difficult to distinguish, when an algorithm moves to a different portion of the problem space, if all variables have drastically changed, or if only it’s simply that one variable considered high-up in the mesh’s tree representation has been altered. This is not an issue for full landscape datasets, since each branch does provide a meaningful grouping of CSP configurations by value, but the animated visualizations might benefit from a different topology.