Homework Module:
Bio-Inspired Distributed Problem Solving

Purpose: Learn to solve a few standard graph-theoretic problems using purely local signalling.

1 Background: Development of the Drosophila Brain

An interesting (and very bio-inspiring) example of search and emergence during development involves the sensory organ precursor (SOP) cells [2] of the Drosophila's (i.e., Fruit Fly's) nervous system. As depicted in Figure 1, SOP cells, which become sensory bristles, originally differentiate from a homogeneous population of proneural cells. Interestingly enough, this differentiation creates a population in which a) every non-SOP cell is adjacent to at least one SOP, and b) no SOP cells are adjacent. This pattern emerges from chemical signaling in which SOPs inhibit neighbor cells (from becoming SOPs) by emitting the protein Delta, which interacts with another protein, Notch, to prevent SOP formation. Although the details of these chemical processes are beyond the scope of this homework, their essence has been abstracted into a very elegant, fully distributed solution to a vexing problem in computer networks [1].

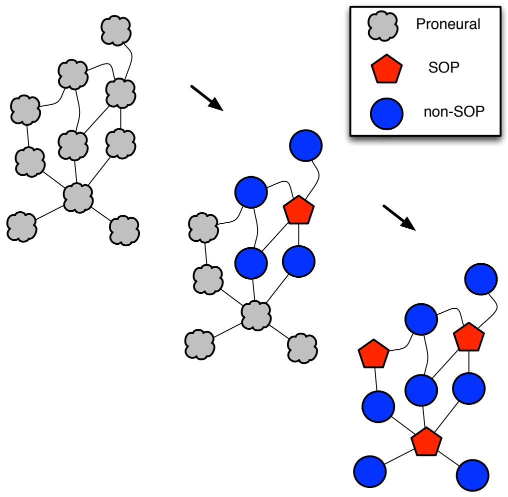

Figure 1: Emergent segregation of Drosophila proneural cells into SOP and non-SOP varieties. Note that all non-SOP's connect to at least one SOP, and no two SOPs are connected to one another.

1.1 Minimal Independent Set (MIS) Problem

The MIS problem involves finding a set of *local leader* nodes in a network such that a) all non-leaders are adjacent to a leader, and b) no leaders are adjacent to one another. Computer scientists and mathematicians struggled with distributed algorithms for solving this problem for decades, before Afek et. al. [1] looked to the Drosophila for a solution. The resulting algorithm highlights the role of localized search (within each node) in leading to the global solution: the MIS.

The procedure/simulation begins with a network of N nodes, the connections (a.k.a. edges) between them, and two empty pools, one for leaders (L) and one for non-leaders (NL). D is the maximum degree (i.e. number of edges) over all nodes in the network. For example, if the 5 nodes in a simple network have degrees 2,1,3,1,2, then D = 3.

Each step of the simulation involves two signaling rounds for each node (n) that has **not yet been assigned to either of the pools**. These nodes presumably execute this procedure synchronously, with the same step being concurrently executed by all nodes.

```
1. $p = \frac{1}{D}$
2. if $p \geq 1$ exit
3. Repeat $M log_2 N$ times (for all nodes in N - L - NL):
   (a) Round 1
       • With probability p do:
           – Broadcast message B to all neighbors(n)
           – state(n) $\leftarrow$1
       • If n receives B **in round 1**, then state(n) $\leftarrow$ 0
   (b) Round 2
       • If state(n) = 1 (i.e., n broadcast B but did not receive B in round 1):
           – Add n to L
           – Broadcast B to all neighbors(n).
       • If n receives B **in round 2**, then add n to NL.
4. p $\leftarrow$ 2p
5. GOTO Step 2
```

The authors show that when $M \geq 34$, success of the algorithm is, for all intents and purposes, guaranteed. Figure 2 shows three sample networks with MIS's found by this algorithm.

Notice that each unassigned node makes a simple stochastic decision on each pass through round 1: whether or not to broadcast B, which constitutes declaring its intention to become a leader node. These declarations, when met by similar messages from a neighbor, simply cancel one another: neither becomes a leader (on this pass). Initially, these declarations occur infrequently, but each round through the outer loop doubles p, leading to more declarations among the remaining uncommitted nodes. Thus, each node exhibits a probabilistic *try-and-try-again* behavior until either its eagerness is rewarded (thus gaining entrance to L) or its passivity is punished by an eager neighbor, thus relegating it to NL. The individual search behaviors are quite simple, but the end result is the emergent solution to a complex problem.

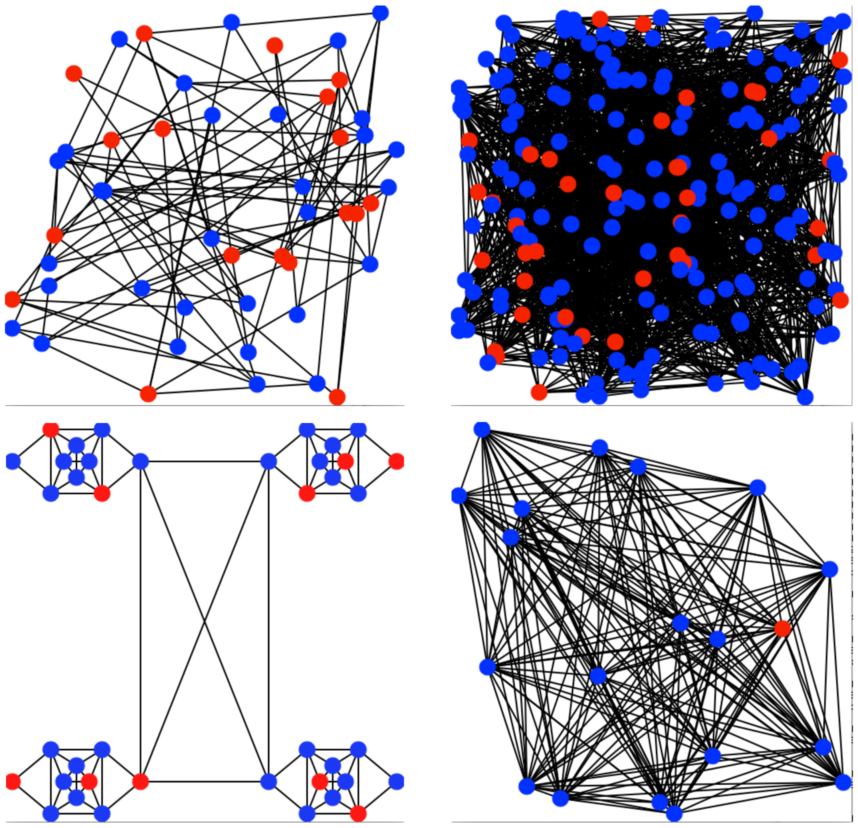

Figure 2: Networks of varying sizes and topologies for which a minimally independent set (MIS - red circles) is easily found using the distributed algorithm of [1], inspired by fruitfly neural development. The bottom right graph, a clique, requires only one leader node.

1.2 The Vertex Coloring (VC) Problem

VC is a classic NP-Complete problem in which a color (or other parameter assignment) is made to each vertex in a graph such that no two vertices that share an edge have the same color (or parameter value). VC relates to the very practical map-coloring problem, in which any two adjacent countries should have different colors, thus visually enhancing their border, as shown in Figure 3.

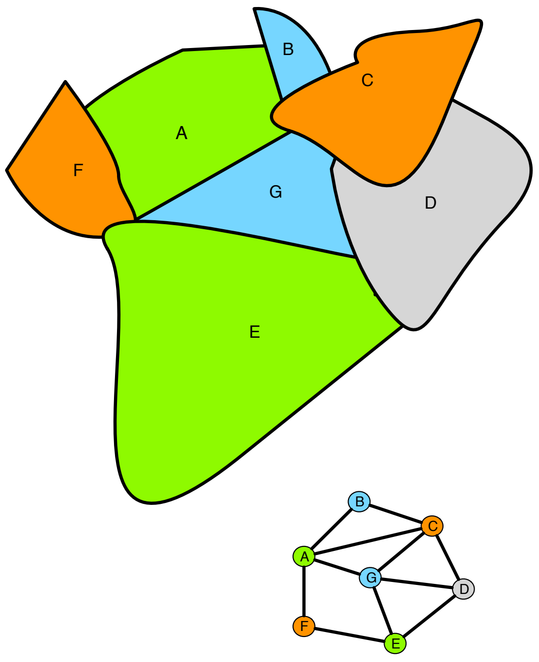

Figure 3: A simple illustration of a vertex-coloring problem (bottom) as an abstraction of map-coloring problem (top), in which any two countries that share a line border a) are represented by vertices with a connecting edge, and b) must have different colors.

The algorithm above for MIS requires only a few small modifications to solve VC problems, though it is difficult to guarantee an optimal coloring, i.e., one that maintains the different-colored-neighbors constraint while using the fewest possible number of colors.

The VC version of the algorithm also consists of two rounds. In the first, any node (n1) has a given probability of choosing its own color, c, which is then broadcast to all neighbors to inform them of n1's intended choice. If, during that same round, n1 receives no messages from any neighbor indicating that it too has chosen color c, then n1's choice of c becomes semi-permanent, and in round 2 it broadcasts the fact that it has now officially chosen c to all of its neighbors, which must then remove c from their list of color options.

An important additional element of the distributed VC algorithm is an ability to undo a color choice, thus implementing a primitive form of backtracking. A node (n1) can randomly choose to uncolor itself, but should only do so when n1 contributes to a conflict situation wherein one of n1's neighbors (n2) is uncolored but has no remaining possible colors (due to the color choices made by its neighbors). By uncoloring itself, n1 will, in some cases, give n2 a valid color option, thus removing the conflict. Your code will need to carefully keep track of the colors available to each node.

For this assignment, assume that one of the inputs to the VC algorithm is a parameter K, the maximum number of allowable colors. Your code should handle all values of K between (and including) 3 and 10.

Figure 4 provides several vertex colorings found by this algorithm.

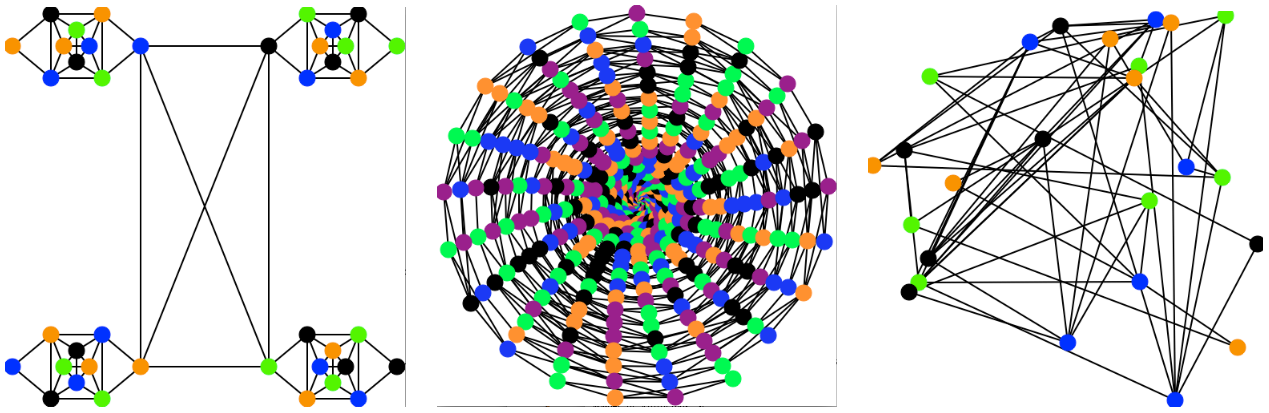

Figure 4: Solutions to vertex-coloring problems on an assortment of graphs.

2 Programming Requirements

1. You must implement (from scratch) the above distributed MIS algorithm, along with a modified version for solving VC problems. Both algorithms are very similar, so savvy use of object-oriented concepts can save some programming effort.

2. Your code should handle relatively large graphs containing hundreds or thousands of nodes and edges.

3. You must implement (or download) a visualization routine such that the complete graph (both vertices and edges) can be drawn in a window on your computer screen. These visualizations should use color coding of vertices to show the parameter assignments (i.e., L -vs- NL for MIS, or one of the K colors for VC). For small graphs, this should make it obvious whether or not your algorithm has found a legal solution. For the VC problem, you should be able to handle at least 10 different colors for the nodes. Though map-coloring can be done with 4 colors, most complicated graphs can require much larger numbers of colors.

4. Your code must check for and display the number of constraints that are violated by a solution.

   For example, in VC, if two adjacent nodes have the same color, then your code must detect that violation and include it in your final tally. In VC, uncolored nodes are also violations.

   In MIS, the three types of violations are a) two adjacent nodes that are both in L, b) a node in NL that does not have any neighbors in L, and c) a node that is neither in L nor NL.

5. When your algorithm finishes running on an MIS or VC problem, it must a) display the number of violations in the solution and b) present a window that visualizes the solution: the color-coded graph.

ALL of these requirements must be properly implemented for you to receive ANY credit for this assignment. They are the essential basis for the runs that you will perform during your demonstration, and it is those runs that will be the sole basis of your grade. If we cannot verify (via the graph visualization and constraint-violation count) that your algorithms have solved problems, then we cannot give you credit.

3 Deliverables

There is no report for this assignment. Everything is based on the performance of your code during the demo session.

Along with this project description, you will be given several files containing graph specifications. At the demo session, the instructor will select THREE of those graphs: you will need to run both the MIS and VC algorithms on each of the three graphs and show (via your visualization and constraint-violation count) that they produce legal solutions.

In addition, you will receive two new files during the demo session (in the same format as the other files). You will need to run the MIS and VC algorithms on those two graphs and produce legal solutions, which, again, can only be verified by visualization and the constraint-violation check.

Thus, there are 5 graphs in total, with both algorithms running on all 5. Each combination of algorithm and graph will be worth TWO points, for a grand total of 20 points.

These are stochastic algorithms, so they can produce different results on different runs on the same graph. However, for the problems given to you, it should not be too difficult to tune the algorithms so that they solve the problems most of the time.

During the demonstration session, you must produce a legal solution to a problem in order to get full credit for that portion of the assignment. However, you will be allowed to run your algorithm R=3 times during that solution search, and before each of the R runs, you will be free to alter any of the key parameters. However, these alternations can ONLY be done to either a) a parameter file, or b) a command-line input. You will not be permitted to modify your code in any way: only the inputs. So any and all parameters for which your algorithm's performance seems sensitive should be incorporated into either the command-line input or a separate parameter file. If you need to make changes to a source file during the demo, you will not get credit for the results produced by those modifications.

4 Important Notes

1. This is NOT a group project. You must work alone, and you will demonstrate your code alone.
2. It is OK to download and/or share code for graph visualization, but each person must write their own code for distributed MIS and VC solving, as well as their own constraint-violation checks. Feel free to use It's Learning to share or give tips on visualization code such as Python's NetworkX package.

References

[1] Y. Afek, N. Alon, O. Barad, E. Hornstein, N. Barkai, and Z. Bar-Joseph, *A biological solution to a fundamental distributed computing problem*, Science, 331 (2011), pp. 183–185.

[2] D. Sanes, T. Reh, and W. Harris, *Development of the Nervous System*, Elsevier Academic Press, Burlington, MA, 2006.