Simulation Exercise 1

Be concise: Try to stay under a total of about 1200 words in answering all the questions. Note that some questions have multiple parts; be sure to address all of them.

Jets and Sharks

McLeod, Plunkett & Rolls (1998, Chapter 2) describe a constraint satisfaction network that encodes knowledge about two fictitious gangs, the "Jets" and the "Sharks". You can run this network by opening a terminal, going into the Examples folder of Lens and typing

```
lens jets-n-sharks.in
```

Alternatively, start lens by double-clicking its icon, then click "Run Script" from the control panel, navigate to the Examples folder using the top of the left panel, select jets-n-sharks.in in the right panel, and click OK.

Clicking on "Unit Viewer" from the control panel will bring up a display of the network and a list of examples. The display is organized similarly to the one in the text, with the "Instance" units in the middle (whose names all end in "-in") and groups of Property units around it (including "Name" units in the upper left—it's important not to confuse these with the Instance units).

Currently none of the weights in the network are displayed, but if you right-click on a unit, its weight values will be displayed (where grey indicates no connection). *[On a Mac without a two-button mouse, the standard method of using Control-click to generate a right-click may not work—you may have to click with two fingers together on the trackpad (and make sure this method is enabled in System Preferences->Trackpad).]*

In the default Palette (which can be changed at the top), positive weights are red and negative weights are blue. Note that all of the positive weights in the network are equal to 1.0 and all of the negative weights are equal to -1.0. If you right-click on a unit for some value of a property (e.g., the 40's unit for Age) you'll see that it has positive weights to the Instance units for individuals who have that value (in this case, Art-in, Clyde-in, Karl-in, and Earl-in) and negative weights to units for other values of the property (30's and 20's). Of course, for the Name property, only one Instance unit has a given value. Right-clicking on the unit again will re-display the unit activations. You can also see the weights by opening up the Link Viewer from the main control panel and then selecting specific Sending Groups and Receiving Groups using the menus, but the display may be less intuitive.

Now click on the "Ken" example in the Unit Viewer (while displaying unit activations). This provides input to the Ken name unit and then updates all the unit activations repeatedly. You won't see these changes while they're being computed, but you can use the arrow controls at the upper right of the display to step through the sequence of unit activations. You can also use the arrows in the second row next to the slider to watch a "movie" of the activations changing over time. In answering the following questions, it may be useful to examining the dynamics of processing over time in this way, although be sure to base your answers on the **final activations** (where the rightward-facing arrows are grayed-out).

For each question, be sure to explain why a certain property holds in terms of the detailed operation of the network. Here's an example (which you should be sure to understand before going on to answer the actual questions):

Question: Why are some of Ken's properties more strongly activated than others?
Example answer: The example provides direct (external) input to the Ken name unit. As this unit becomes active, it activates the Ken_in instance unit, which then activates the rest of Ken's properties (Sharks, 20s, HS, Single, Burglar). These property units, in turn, partially activate the instance units of other individuals who are similar to Ken in that they share many of his properties (e.g., Nick, Neal). These partially activated instance units then provide additional support to their own property units, some of which agree with (support) those of Ken and some of which compete with his properties, causing differing levels of final activation.

Before answering the first question, be sure again that you've clicked on the "Ken" example in the Unit Viewer and are looking at the final activations.

> **Q1:** Why are some of the instance units (other than Ken) more active than others? Explain the differences. (Note that, although this question is related to the example question above, your answer should go beyond what is stated in the example answer.)

Now click on the "Sharks 20's" example. This provides input to both the Sharks unit and the 20's unit, which together uniquely specify Ken. Note that the network settles to states that are very similar but not identical to those generated by the "Ken" example. Again, be sure that you're looking at the final activations of the network. Once the rightward arrows are grayed out, you can click back and forth between the "Ken" and "Sharks 20's" examples to compare the final states of each.

> **Q2:** Explain why the occupation units show partial activations of units other than Ken's occupation, which is Burglar. Be sure to contrast the current case with the one with Ken as input.

Sometimes we do not know something about an individual; for example, we may never learn that Lance is a Burglar. Yet, we are able to give plausible guesses about such missing information. To explore the network's ability to do this, first disconnect the Lance instance unit and the Burglar unit by running the following two commands:

```
lens> disconnectUnits Lance-in Burglar
lens> disconnectUnits Burglar Lance-in
```

(Two commands are needed because there is a connection in each direction.) Now click on the "Lance" example.

> **Q3:** Explain how the model was able to fill in the correct occupation for Lance. Also, explain why the model tends to activate the Div. (divorced) unit as well as the Mar. (married) unit.

You can reinstate the connections by running the following command:

```
lens> connectUnits Lance-in Burglar -mean 1.0 -range 0.0 -bidirectional
```

Schemas

In order to carry out this part of the homework, you will need to download and unzip the file schema.zip, which contains three text files: schema.in, schema.ex, and schema.wt. Just put these in the Examples folder. Then, from there, start the simulation by running "lens schema.in" and open the Unit Viewer (as in the Jets and Sharks simulation). You can find the schema files in the Simulation Exercises folder.

Rumelhart and colleagues (PDP2, Chapter 14) provide an alternative formulation of the traditional notion of a schema, based on parallel constraint satisfaction. An important property of schemas is that they can embed. Rumelhart et al. (pp. 35-36) define what it means for something to be a "subschema" under their interpretation and provide an example of a "window" subschema composed of the "window" and "drapes" descriptors (see their Figure 13).

> **Q4:** Critically evaluate whether or not the "windows" and "drapes" features do, in fact, form a subschema. Be sure to provide evidence from running the simulation (e.g., goodness values) to support your argument. (Note that Figure 13, in and of itself, is not sufficient to establish their claim.)

In order to answer Q4 (and Q5), you will need to be able to run the network after providing input to particular units by hand, rather than by simply clicking on pre-defined examples . This is a bit awkward in Lens but it can be done. To turn units on (e.g., "windows" and "drapes"), run the following:

```
lens> setObj preEventProc "turnOn {window drapes}"
```

To turn the units off, use turnOff instead of turnOn above. To turn one on and one off, run

```
lens> setObj preEventProc "turnOn {window} ; turnOff {drapes}"
```

Then click on the "empty" example (if you want input only to the specified units) or click on another example (if you want to combine them with a standard cue or with one of the full prototypes). To remove the hand-specified inputs, just run

```
lens> setObj preEventProc ""
```

The value of `preEventProc` can also be set from within the Object Viewer (created by clicking on the "New Object Viewer" button of the main panel).

Now, pick two features which are strongly active in different rooms but are not active together in any of them (e.g., "bed" and "oven", as discussed in class---you can use one of these but not both). For each of your two features, clamp it on (as in 2C), run the network on the "empty" example, and examine the goodness value and final pattern it produces (you should save and print each of these final activation displays, using the "Hinton Diagram" display palatte). Then clamp both of the features on together, run the network, and examine the resulting "combined" goodness and final pattern (and save and print it).

> **Q5:** Identify the ways in which the combined final pattern differs from each of the single feature patterns, and try to explain those differences. Does the pattern produced by one of the features predominate, or is the mix fairly even, and why? How does the goodness value of the combined pattern compare with those of the single-feature patterns, and why? You will find it useful to examine the pattern of weights among units (see the actual network or Figure 5 in the Rumelhart et al. chapter) in explaining why the network behaves as it does.

For this question, you will need to include images of three activation displays in your response. There are two possible ways to do this. The first and probably simplest is to take a screenshot and then clip out the relevant portion of the resulting image file (exactly how to do this depends on the operating system you're running — ask for help if you need it). The second is, within Lens, to select "Print" under the "Viewer" menu of the Unit display. Then select "File" at the top of the new pop-up and give a filename ending in ".ps" (postscript). This will create a postscript file *in the folder in which you started Lens*. This file can then be imported into a document (again, ask for help if needed).