Harsh Patel

harshp

ECE 4420/6420 Section 001

Takehome #1

Graph Vertex Coloring using Embedded CLIPS

I want to extend the labeling problem that was shown in lecture to implement a more general graph vertex coloring problem. The idea is to use C to read in graph data (from a file), find a valid coloring using CLIPS and then output back to C to create a visual of the results. By coloring I mean filling in a graph of X vertices using Y different colors such that no adjacent vertices have the same color. Embedding CLIPS here is useful because it allows C to handle all of the I/O and graph generation, which is great for testing and usability.

In terms of references I would probably only need to use the CLIPS basic and advanced programming guides. I might need to refer back to the book and/or lecture notes as well.

Goals:

- Create a C driver that reads in the graph data from a text file along with the number of colors that user wants to allow for coloring with. (2 through X, where X is an undecided integer).
- Create the templates, rules and initial facts for CLIPS using the obtained data and color constraints.
    - This will be done using the functions outlined in the advanced programming guide.
    - This is where using C is particularly useful. We can dynamically create rules and facts based on different input data.
- Run CLIPS and obtain the valid coloring if one did exist.
- Output the result in some sort of visual representation (maybe just a grid of squares that are colored accordingly).

Deliverables:

- A main.c file along with all dependencies and a makefile required to use embedded CLIPS
- Some (3-5) text files containing sample data along with the output (.jpg or similar file) of the resulting coloring with a varying number of total colors (so more than 1 output file per input).
- A readme.txt file to explain the syntax that sample graphs should have in order to be used with the main file.
    - Perhaps another executable that will create a random graph of X vertices and output it in the correct format for the main program to use.

Project Overview

Requirements:

1) The files and Clips_Core directory which is provided with this report.
2) Graphviz – This library is used to render the .dot files for the colored graphs.
    a. ECE Linux machines have it installed already.
    b. Install on your machine using: sudo apt-get install graphviz.
    c. Alternatively go to www.webgraphviz.com and paste the contents of the .dot file into the text box and hit create graph.

Components:

Makefile:

- Has 4 basic scripts, make all, make clean, make jpg and make png (more on the last 2 later)
- Calling "make all" gives you the two executables graphGen and graphColor

Graph Generation – uses graphGen.c and executable graphGen

- **Usage: ./graphGen <num_nodes> <output_file>**
- Basically this is the random graph generator which outputs graph layout data in a specific format which is used by the second executable.
- There is an example of the expected format for this file shown in **Appendix A**.
    - The first line of the file should have the format "X Y" where X is the number of nodes in the graph and Y is the number of edges.
    - Following this, each edge is listed in the format "A--B" where A and B are nodes and 0 > A != B > number of Nodes.
    - Also each node must have at least one edge (this is a constraint I placed on the format to make the CLIPS generation easier)
    - It is recommended you use this generator to create your own graphs
    - **Another Caveat:** It's highly recommended you keep the max number of nodes to 20, the program will even warn you so. The explanation for this provided within the Embedded CLIPS section below.

CLIPS Generation – uses clipsGen.c (and h) and is part of the graphColor executable

- This is not the CLIPS engine, this is just a group of functions that use the graph data from the provided input file to create the .clips file for the CLIPS engine to use.
- The reason I went with this format instead of fully relying on the functions provided in clips.h is because this method is easier to debug and change as I go. Along with this, I like to able to keep copies of the CLIPS files for reference later.
- An example of what the output file created by these functions loops like is shown in **Appendix C.**
- The inspiration for the design of the .clips file was taken from the CSP and labelling lecture **[1]**.
- There is a template to define an adjacency and one to define the possible coloring of each node.

- There is a rule for symmetry between adjacencies and also a rule to find a valid coloring
- Lastly, the initials facts are based on how many colors the user chose to color the graph with (number between 2-4).

Embedded CLIPS – uses main.c ,clipsGen.c and the graphColor executable

- **Usage: ./graphColor <graph_input_file> <num_colors>**
- This driver starts off by using the clipsGen functions to create the .clips file and then it creates the CLIPS environment and reads in that file.
- See **Figure I** for a detailed look at the code.
- First this program obtains a pointer to the template "coloring", which is used to store the final coloring. This pointer is required to use other clips library functions. **[3]**
- Following this, The CLIPS engine is allowed to fire one rule at a time until either the agenda is empty or the first fact with the template "coloring" is found.
- If a valid coloring was found, the file coloring.dot is created, otherwise the program notifies the user and ends. See an example if this file's format in **Appendix B.**
- Once again to use the .dot file, you can have graphviz installed and then just use the make file to generate an image.
    - Depending on the version of graphviz one of the make scripts might not work.
    - make jpg works on the schools Linux machines
    - make png works on Ubuntu 16.04.
        - This script creates a .ps file first and then converts it. The reason for this is that this version of graphviz doesn't support direct conversion to .jpg and converting a .ps file to .jpg makes it blurry so I went with .png instead.
- Discussion on Limitations:
    - Since I used the labelling problem as a stepping stone, the size of the templates and rules are based on the number of nodes and edges.
    - Due to this fact the safe operating limit of this program is set between 2 to 20 nodes, with 25 working most of the time.
    - The reason this is so is because while using CLIPS to determine a coloring for a graph gives much Improved computational times, the memory requirement is immense.
    - Especially considering the Rete algorithm keeps track of tokens with Beta nodes in particular, this program is very resource intensive for trying to color larger graphs. **[2]**
    - There may be a way to split up the rules and templates into smaller chunks, however I was not able to determine a reliable method to do so.
    - So for now, this program can color graphs up to 20 nodes with 2 to 4 colors.
        - Tested a graph of 30 nodes and just firing a couple of the adjacency symmetry rules I was already at 6GB of RAM usage with no coloring found.

Figures and Results

```
// Obtain  pointer to the template named color
void * deftemplatePtr = EnvFindDeftemplate(theEnv, "coloring");

// Fire one rule
long long numRun = EnvRun (theEnv, 1);

void * factPtr = EnvGetNextFactInTemplate(theEnv, deftemplatePtr, NULL);
int i = 0;

const int MAX_RULES = 25;

/* Fire the rules in the agenda until:
*  agenda is empty, i == MAX_RULES : Meaning no coloring found
*  or the rule with template "coloring" is found
*/
while(numRun != 0 && i < MAX_RULES && factPtr == NULL){
  numRun = EnvRun (theEnv, 1);
  factPtr = EnvGetNextFactInTemplate(theEnv, deftemplatePtr, NULL);
  i++;
}
```

Figure 1: CLIPS driver code

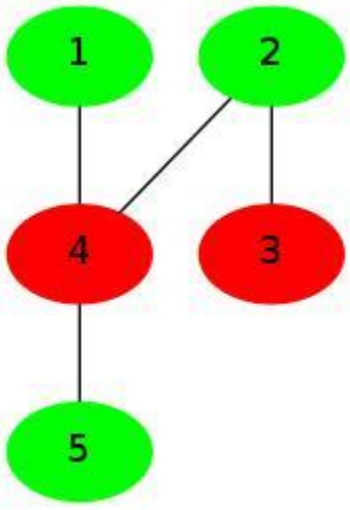

Figure 2: Five Node Graph

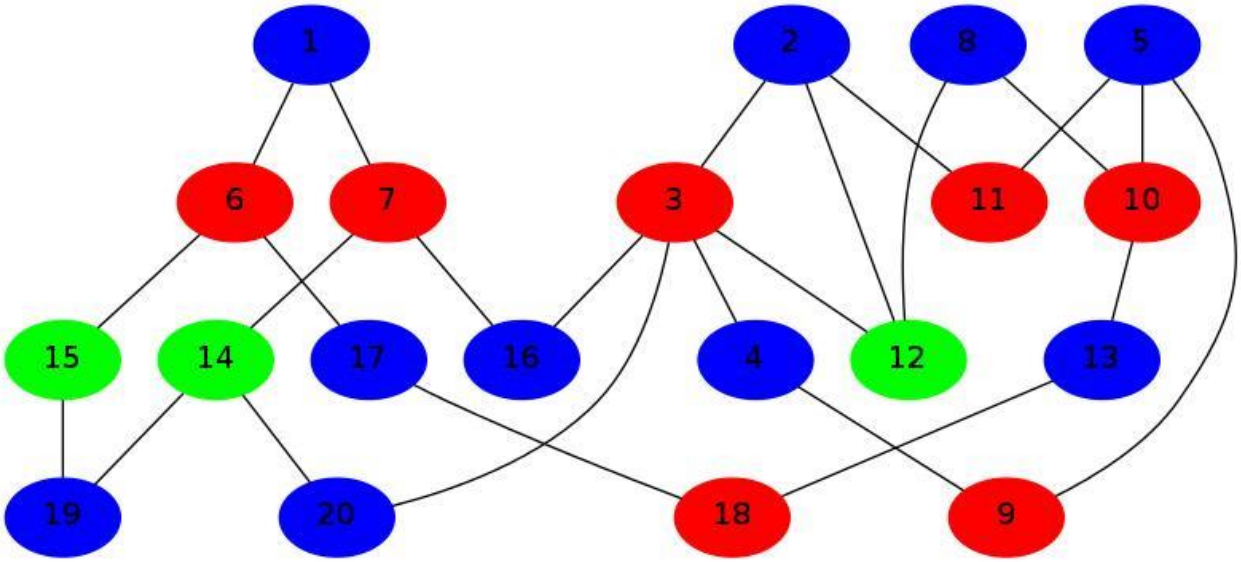

Figure 3: Twenty Node Graph

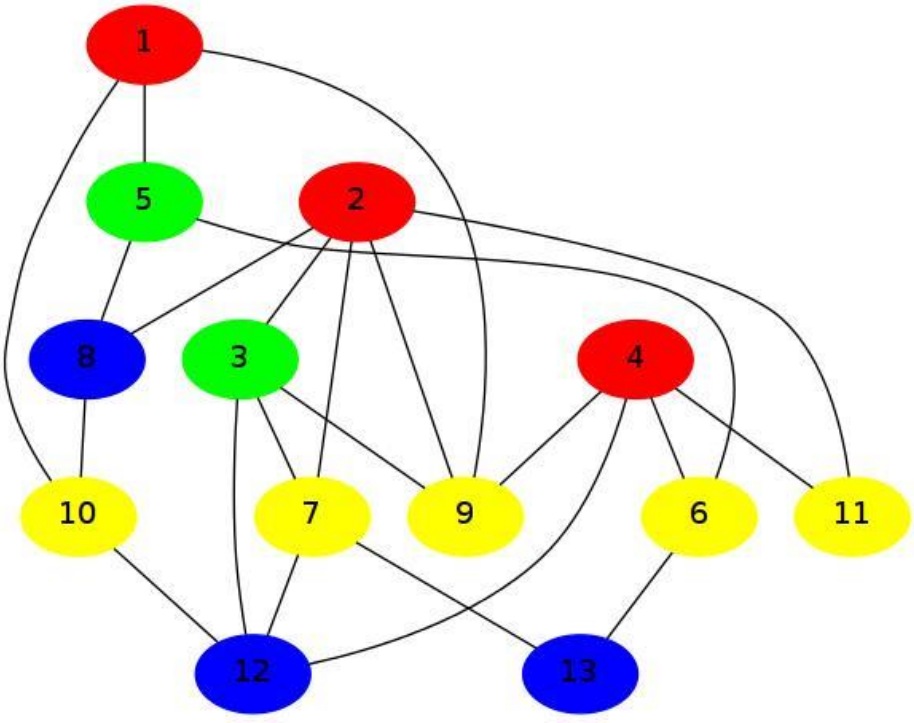

Figure 4: 13 Node Graph

** Find more samples in the Sample_Images directory **

Summary and Future Direction

The final product presents a good overview for how embedded CLIPS could be used in conjunction with other programs or tools to create more complex applications.

Benefits of Embedded CLIPS:

- Using this setup you can create a .clips file for any configuration of graph you'd like, and assuming you have enough memory, quickly compute the coloring sequence.
- Allows for visualization of the data obtained from CLIPS.
- Very easy to use and package.
- This simple example can be extended to include other functions like a shortest path visualization tool.

While we have seen some of the short comings involved with the Rete algorithm, the computational speed is a tremendous asset. Of course for future development on such projects, it is important to discover a better way to organize the CLIPS code such that the memory requirement isn't so high. This application was just a showcase for what embedded C can bring to the table. Once a reliable method is discovered that can be used with multiple graphs of various layouts, this simple showcase can become a more practical tool.

It would probably be useful to figure out a way to package the graphviz tool into the application as a c library. There is a way to use cgraph, which is one of the provided c libraries in the graphviz package, however this still involves having graphviz installed on the system. Creating a static library for graphviz is not as simple as CLIPS, but it surely can be done.

All Examples correspond to the file 8.txt found in Sample_Graphs

Appendix A:

```
8 12
1--3
1--4
1--6
1--7
1--8
2--3
2--6
2--7
3--5
3--7
5--8
6--7
```

Appendix B: Format of coloring.dot [4]

```
graph G{
      1--3;
      1--4;
      1--6;
      1--7;
      1--8;
      2--3;
      2--6;
      2--7;
      3--5;
      3--7;
      5--8;
      6--7;
      1[color= red,style=filled];
      2[color= red,style=filled];
      3[color= blue,style=filled];
      4[color= green,style=filled];
      5[color= green,style=filled];
      6[color= blue,style=filled];
      7[color= green,style=filled];
      8[color= blue,style=filled];
}
```

Appendix C: color.clips

```
(deftemplate adj-to (slot node1) (slot node2))

(deftemplate coloring
   (slot col1)
   (slot col2)
   (slot col3)
   (slot col4)
   (slot col5)
   (slot col6)
   (slot col7)
   (slot col8)
)

(defrule rulesym
   (adj-to (node1 ?X) (node2 ?Y))
 =>
   (assert (adj-to (node1 ?Y) (node2 ?X)))
)

(defrule rulecol
   (adj-to (node1 ?R1) (node2 ?R3))
   (adj-to (node1 ?R1) (node2 ?R4))
   (adj-to (node1 ?R1) (node2 ?R6))
   (adj-to (node1 ?R1) (node2 ?R7))
   (adj-to (node1 ?R1) (node2 ?R8))
   (adj-to (node1 ?R2) (node2 ?R3))
   (adj-to (node1 ?R2) (node2 ?R6))
   (adj-to (node1 ?R2) (node2 ?R7))
   (adj-to (node1 ?R3) (node2 ?R5))
   (adj-to (node1 ?R3) (node2 ?R7))
   (adj-to (node1 ?R5) (node2 ?R8))
   (adj-to (node1 ?R6) (node2 ?R7))
 =>
   (assert
      (coloring
         (col1 ?R1)
         (col2 ?R2)
         (col3 ?R3)
         (col4 ?R4)
         (col5 ?R5)
         (col6 ?R6)
         (col7 ?R7)
         (col8 ?R8)
         )
      )
)

(deffacts startup
   (adj-to (node1 red) (node2 green))
   (adj-to (node1 red) (node2 blue))
   (adj-to (node1 green) (node2 blue))
)
```

References:

[1] Robert Schalkoff, J. Slides to Accompany Intelligent Systems, Application to Constraint Satisfaction Problems [Online]. Available: https://clemson.instructure.com/courses/30291/pages/clips-advanced?module_item_id=516653. Cited: 10/23/17.

[2] Robert Schalkoff, J. (2011). *Intelligent Systems Principles, Paradigms, and Pragmatics* [Text]. Cited: 10/23/17.

[3] Gary Riley, (2015). CLIPS Reference Manual Vol. II Advanced Programming Guide [Online]. Available: http://clipsrules.sourceforge.net/documentation/v630/apg.pdf. Cited: 10/23/17.

[4] Emden Gansner, (2006). Drawing graphs with dot [Online]. Available: http://www.graphviz.org/Documentation/dotguide.pdf. Cited: 10/23/17.