The *Bleak House* Character Map Project

Abraham Asfaw*

May 2012

1 Introduction

Inspired by Franco Moretti's visual interpretation of Victorian novels, the students in ENGL270, Crime and Detection (Spring 2012) set out to create their own character map of Charles Dickens' popular novel entitled *Bleak House*. Literary critics mark this novel as one of Dickens's best works; however, it is also one of his richest creations with several characters. Critical reading of the novel requires recollection of relationships that have formed between characters in the novel. In order to simplify the process of recalling character relationships, this project was started by the author. Following the approval of Prof. James Arnett (Ph.D.), the project began as a collaboration between the students enrolled in the course.

2 Project Outline

A website was created[1] that allowed students to enter relationships between characters as they discovered these relationships throughout the course reading assignments. Within a month and a half, the website had collected enough information to allow for the creation of a character map with several customizations. In this picture, we show three types of characters based on popularity. The first kinds of characters are relatively unpopular throughout the story, and develop fewer relationships than all others. These characters

*asfaw@princeton.edu

[1]Copyright © 2012, Abraham Asfaw `http://www.abrahamasfaw.com/bh`

and their relationships are shown in varying shades of green based on their ranked popularity among each other. The second kinds of characters are more popular and have more complicated relationships throughout the story. They are shown in varying shades of teal, again ranked by their popularity among each other. Finally, we have the third kinds of characters which are central to the story and form relationships with almost all other characters in the story. They are shown in varying shades of purple based on their popularities among each other. The most popular characters among these go beyond purple into dark violet and eventually into black colors.

The meaning of a relationship in this context is loosely defined to be any form of interaction between characters, ranging from a brief meeting to an elaborate conversation to familial or marietal engagement.

3 Analysis and Conclusion

The project has achieved its goal of visualizing the relationships between characters in the novel. The poster presented here is available online at `http://www.abrahamasfaw.com/bh`.

The use of lines to indicate relationships between characters is very intuitive and can easily be understood by readers. To the technically inclined readers, parallels exist between this model of relationship analysis and graph theory.

The use of colors to rank characters by popularity achieves two goals. First, the division of the characters into two groups based on overall popularity allows us to select dim and undistracting colors for the unpopular characters. This allows greater emphasis on the popular ones. Second, the ranking of characters in each group by varying shades of color allows us to perceive an additional dimension to our analysis – that of ranking within the group. That way, we develop several tiers of character popularity levels.

To the students enrolled in the course, including the author, this project has had an immense impact on the learning process throughout the semester. Students were allowed to participate beyond the regular coursework requirements by building a character map, thereby working in teams to tackle specific characters and find relationships that exist between these characters.

To the author, this project has allowed for a fantastic learning opportunity in which concepts from mathematics and computer science (particularly those in graph theory) were tied to concepts in literary analysis. Through several revisions of the script that drew the character maps to the management of the project website (which rose to first rank on google in less than a month), a number of invaluable skills were learned throughout the semester.

4 Future Research

Future research could be have two possible directions. The first direction would be the duplication of this type of project in a different course where the reading material may be similar or different. The second direction would be that of developing the tools written in this project to full-blown literary analysis programs. For instance, the relationships are drawn from character interactions. An added dimension would be that of where characters met and the type of relationship between the characters. While the collection of such information from the reading is not difficult, visualizing it requires attention to intricate details.

`http://abrahamasfaw.com/bh`

5 Acknowledgments

The author would like to thank Prof. James Arnett for the fantastic opportunity of realizing this project. Prof. Arnett kept an open mind and provided encouragement throughout the semester as several revisions of the analysis tools were made.

The following is a list of students who participated in the project.

- Abraham Asfaw
- Alexandra Belmont
- Samantha DiMella
- Ritchie Jean-Louis
- Erin Kelly
- Brendan Kiely
- Rachael Losco
- Paolo Martusciello
- Megan McCarthy
- Dimitri Mezidor
- Andrew Michitsch
- Franchesca Santos-Cucalon
- Christopher Smith
- Joshua Steinberg
- Allison Sweeney
- Mayra Yepes
- Marco Zefi

```python
# Relationship analyzer
# Written for The Bleak House Character Map Project
# By Abraham Asfaw '12
# asfaw@princeton.edu
# Released under the MIT License --
# Do whatever you want with this code, but don't bother
# me if something goes wrong. You're welcome to contact
# me with suggestions.

from random import randint

# generate random hex RGB color code
def randcolorstr():
    startcolor = 0
    finishcolor = 255
    r = randint(startcolor,finishcolor)
    g = randint(startcolor,finishcolor)
    b = randint(startcolor,finishcolor)
    rstr = "%x" % r
    gstr = "%x" % g
    bstr = "%x" % b
    if len(rstr) == 1: rstr = "0" + rstr
    if len(gstr) == 1: gstr = "0" + gstr
    if len(bstr) == 1: bstr = "0" + bstr
    retstr = "#" + rstr + gstr + bstr
    return retstr.upper()

# convert hsl to rgb based on lightness
# low lightness l means popular character
# popularity vs color index
# unpopular -- shades of green
# somewhat popular -- shades of teal
# very popular -- shades of purple
# assignment of colors made by project participants
def hsltorgbstr(l):
    if l < 0.3: # intense
        h,s,l = 300, 1.000, l/0.3
    elif l < 0.67:
        h,s,l = 180, 1.000, (l-0.3)/0.37
    else: # weak
        h,s,l = 120, 1.000, (l-0.67)/0.33

    c = (1.0-abs(2.0*l-1.0))*s
    hp = h/60.0
    x = c*(1.0-abs((hp%2.0)-1.0))
    if hp >= 0 and hp < 1:
        r1,g1,b1 = c,x,0
    elif hp >= 1 and hp < 2:
        r1,g1,b1 = x,c,0
    elif hp >= 2 and hp < 3:
        r1,g1,b1 = 0,c,x
    elif hp >= 3 and hp < 4:
        r1,g1,b1 = 0,x,c
    elif hp >= 4 and hp < 5:
        r1,g1,b1 = x,0,c
    elif hp >= 5 and hp < 6:
        r1,g1,b1 = c,0,x
    else:
        r1,g1,b1 = 0,0,0
    m = l-0.5*c
    r,g,b = int((r1 + m)*255), int((g1 + m)*255), int((b1 + m)*255)
    rstr, gstr, bstr = str("%x" % r) , str("%x" % g) , str("%x" % b)
    if len(rstr) == 1: rstr = "0" + rstr
    if len(gstr) == 1: gstr = "0" + gstr
    if len(bstr) == 1: bstr = "0" + bstr
    retstr = "#" + rstr + gstr + bstr
    return retstr.upper()

# generate hex RGB color code
def colorstr(rank,total):
    return hsltorgbstr(1.0-(rank/total))

# read the dictionary file containing relationships
# format of input is a dict
# each key is the character name
# each item is an array with 2 arrays
# [[relationships], [descriptions]]
dictrelsdescs = eval(open("relsnew").read().strip())

# words to avoid when looking for a character in relationships
unmatch = ["MR.", "MRS.", "SIR", "LADY", "THE", "MAN", "FROM", "MR", "MRS"]
# chars to avoid in names when matching
badchars = ["(", ")","{","}"]

# setup procedures
# create charnames which will contain the names of all characters
# create dictrels which will contain key=charname and item=array[relatedchars]
# create dictn2i which will contain key=charname and item=intindex
# create dicti2n which will contain key=intindex and key=charname
dictrels = {}
```

```
charnames = dictrelsdescs.keys()
from sets import Set
for charname in charnames:
    dictrels[charname] = Set()
dictn2i = {}
dicti2n = {}
icounter = 1
for charname in charnames:
    dictn2i[charname] = str(icounter)
    dicti2n[icounter] = charname
    icounter += 1

# look for relationships and populate dictrels
# -> change to uppercase during matching
# -> remove bad characters in badchars
# -> avoid matching words in unmatch
for character in charnames:
    rels = "||".join(dictrelsdescs[character][0])
    for charname in charnames:
        if charname != character:
            for eachword in charname.split(" "):
                ewmod = eachword.upper()
                for char in ewmod:
                    if char in badchars:
                        ewmod = ewmod.replace(char,"").strip()
                if ewmod in rels.upper() and ewmod not in unmatch:
                    dictrels[character].add(charname)

# prepare dot file for writing
fhandle = open("relsnewdot",'w')
fhandle.write("graph g{\n")

# populate excepts with characters whose names will not be in the dotfile
# populate dictcounts[charname] = [count_as_source,count_as_referenced]
excepts = []
dictcounts = {}
for character in charnames:
    mentions = 0
    for _,ends in dictrels.items():
        for end in ends:
            if character == end:
                mentions += 1
    if mentions == 0 and len(dictrels[character]) == 0:
        excepts.append(character)
    else:
        dictcounts[character] = [len(dictrels[character]), mentions]
totalcounts = []
for character, counts in dictcounts.items():
    totalcounts.append([counts[0],character])
totalcounts.sort()
ranked = [i[1] for i in totalcounts]

# populate dictcolors and write characters into dot file with their colors as nodes
dictcolors = {}
for character in charnames:
    if character not in excepts:
        thischartotals = sum(dictcounts[character])
        # enable for random color selection
        #dictcolors[character] = randcolorstr()
        # enable for ranked color selection
        dictcolors[character] = colorstr(ranked.index(character)+1.0, len(ranked))
        fhandle.write(dictn2i[character] + '[label="' +
                character + " (" + str(dictcounts[character][0]) +
                "," + str(dictcounts[character][1]) + ')", color="' +
                dictcolors[character]  + '"];\n')

# write the edges into the dot file
for origin, ends in dictrels.items():
    if origin not in excepts:
        thiscolor = dictcolors[origin]
        for end in ends:
            fhandle.write(dictn2i[origin] + "--" +
                    dictn2i[end] + ' [color="' + thiscolor + '"];\n')

# finish writing dot file
fhandle.write("}")
fhandle.close()
```