MULTI-ATTRIBUTE VALUE FUNCTION FOR ORDERING WEB PAGES

1. INTRODUCTION

1.1 The Problem Considered

The Internet hosts a massive collection of web pages, and with the sheer volume of information available it is often hard to find suitable pages. Most search engines ask the user to specify a set of keywords, and the pages that contain most keywords or relate in some way to the keywords are listed in the result of the search. One problem with these engines is the amount of results that are usually returned. Most queries result in thousands of pages, and even with the "best" matches listed first the decision as to which pages to visit can be hard. The only way you can try to limit the number of pages returned is to provide more keywords, which can cause problems especially if a vague or incorrect word is entered. What is needed is some method of aiding the user in his decision of which pages to visit first.

This project implements a multi-attribute value function and uses it to order a set of web pages so that the ones most suited to the users preferences appear first. A problem with search engines is their limited scope. Most of them only search for keywords and do not look at any other page content such as the number of links. The multi-attribute value function implemented takes a set of web pages and orders them according to user defined preferences for the various web page attributes. The attributes are primarily based on the stylistic content of a web page, such as the number of images, or whether the page has frames or not. The multi-attribute value function will ideally be used alongside a search engine to re-order a number of its results according to the pages content. For instance, if the user is looking for a comparison between hardware performance they may search for "hardware" and then order the pages using the multi-attribute value function according to the number of tables they have.

1.2 Working Environment

This project was developed using Sun's JDK v1.3 (Java Development Kit) primarily using Windows NT. Compatibility with previous versions of is probable for most of the code implementing the web page preferencer (as I have called it), although the graphical interface uses the javax.swing package which is less likely to be backwards compatible.

1.3 Outline Of Remaining Report

The remainder of the report consists of 3 more sections, as follows. Section 2 details the theory behind the multi-attribute value function and how I have applied it to web pages. Section 3 describes the project implementation in Java. Finally, section 4 draws conclusions and mentions possible improvements that could be made in the future.

2. THEORY

This section describes the theory behind my program. All numerical values specified are as implemented.

2.1 Multi-attribute Value Function

A multi-attribute value function is the weighted sum of many value functions, each operating on a different attribute. So, given n attributes, the value V is given by,

$$V = \sum_{i=1}^{n} \lambda_i f(a_i) \qquad (2.1)$$

where λ_i is the weight of attribute a_i and f is a value function.

The weights give the relative importance of each attribute. They can take any value, but when running my program using the graphical interface they range from 0-100 because it is easier for the user to order their preferences on a scale of 0-100 rather than, say, 0.0-1.0.

The value functions return a value between 0.0 and 1.0. Theoretically, a value function can be any function. In my implementation there are 3 main types of value function to correspond to the 3 possible types of attribute. These are, attributes that take a boolean value (or a value to be matched with another and so will evaluate at some point as boolean), attributes that take any integer value, and attributes that take both a boolean and an integer value.

For boolean attributes the value function returns either 1.0 if the attribute is true, or 0.0 otherwise. See rule 2.2 below.

$$f(a) = \begin{cases} 1.0 & \text{if } a == true \\ 0.0 & \text{otherwise} \end{cases} \qquad (2.2)$$

For instance, whether the page has frames or not. The number of frames is irrelevant.

A related function compares a value against a pre-stored value and returns 1 if the values are identical, and 0 otherwise. See rule 2.3 below.

$$f(a) = \begin{cases} 1.0 & \text{if } a == b \\ 0.0 & \text{otherwise} \end{cases} \qquad (2.3)$$

where b is a pre-stored value.

For integer valued attributes the two following related functions have been implemented. Theoretically any function is possible, but all that is necessary when applied to web pages is to specify whether more or less of a particular attribute is desired. So for simplicities sake, the functions are strictly linear. On the left of figure 2.1 is the function used when the user is looking for more of a particular attribute, and on the right is the function used when he is looking for less.

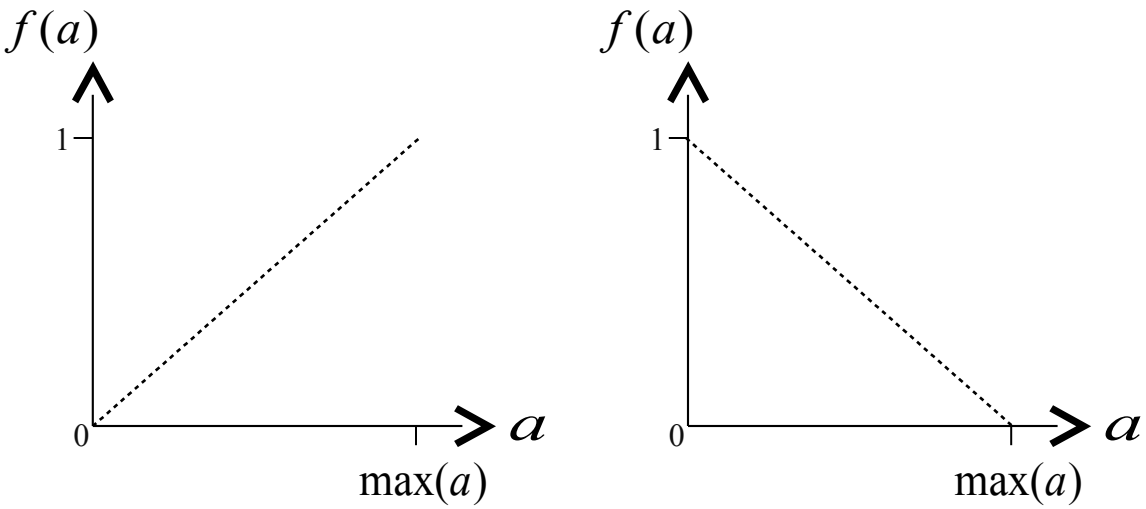

Figure 2.1

There is the constraint that $0 < a < \max(a)$, where $\max(a)$ is the maximum attribute value encountered so far, and so the function will not return < 0.0 or > 1.0

The third type of value function is a combination of the two previous value functions. It takes an attribute value, matches it to a pre-stored value, and returns 0.0 if there is no match or evaluates the attribute with one of the functions shown in 2.1 if there is a match. This is shown below in rule 2.4.

$$g(a) = \begin{cases} f(a) & \text{if } a == b \\ 0.0 & \text{otherwise} \end{cases} \qquad (2.4)$$

Looking at the possible weight and function values, it is obvious that the multi-attribute value function returns a value between 0.0 and 100.0 n . To prevent this figure becoming too large and to cater for the user's familiarity with percentage values (although it isn't fundamentally a percentage value) I have normalised it so that value ranges from 0 to 100.0.

2.2 Web Pages Attributes

The attributes of a web page that I will be using are shown in table 2.1 below.

Attribute	Type
Amount of text	Integer (count)
Number of images	Integer (count)
Number of tables	Integer (count)
Number of links	Integer (count)
Number of downloads - all - images - videos - other	Boolean (selected?) and Integer (count) (note: this attribute is represented as a string e.g. "images" in the value function shown in 2.4)
Presence of frames	Boolean (exists?)
Extension of site name	String (matches?) and Integer (count)

Table 2.1

These attributes are all represented in a web page as simple tags. For instance, images are represented by the <IMG SRC=...> tag.

It's worth noting that the function used to evaluate the "number of downloads" attribute is treated slightly differently to the others. This function is shown in 2.4. The user selects between the type of download he wishes. This type is then stored as a string in the function (the pre-set value), for example as "images". When the web pages are ordered this string is compared with the desired download type, and the appropriate attribute count (in this case the number of images) is passed as a parameter to the value function for evaluation.

2.3 Application

The user specifies how important each attribute is by assigning weights to them. The web page is parsed to generate measures of each attribute, which are then translated into values using a value function. The weighted sum of these values gives the total value of the page. By comparing the values of each of the candidate web pages an order of preference is determined.

3. IMPLEMENTATION

This section will describe the basic structure of my program. Basic knowledge of object-oriented software, and Java in particular, is assumed. The class diagrams in this section are simplified UML diagrams.

3.1 Choice Of Language

This project is implemented in Java. There are several reasons for this. Firstly, Java is an object oriented language. This is of great use when coding the multi-attribute value function, which consists of many other functions. These functions can vary and the use of an object-oriented language allows the easy interchange between different functions - they simply have to extend a common superclass. Secondly, since Java is a language designed a great deal around the Internet it already has classes in the javax.swing.text.HTML package that implement HTML file parsing. This prevented me having to write complex code to parse the web pages. Thirdly, Java is highly portable and can be run on any platform providing it has the Java runtime environment installed. Finally, I have experience in coding in Java and the clear structure, automation of memory management (no pointers!), and flexibility make it a pleasure to code in.

3.2 Design

This project consists of three main parts. Firstly, there is the code that implements the multi-attribute value function in the class MultiAttributeValueFunction. This was designed to be as flexible as possible so that it can be used in the future in different applications. This code is used by the class PagePref that is the core application and consists of a number of public methods to parse and sort web pages. Finally, there is the graphical interface implemented in the class PagePrefGUI. It is the "main" method of this class that is called from the command line in order to run the program. While it was possible to make the class PagePref the graphical interface itself, I chose to separate the interface functionality for clarity and so that the web page ordering code can be called from other programs if necessary without the extra bulk of the interface code.

3.2.1 Multi-attribute Value Function

The multi-attribute value function is implemented by the class MultiAttributeValue-Function. This class stores a number of ValueFunction objects, which implement the value functions f . These functions are applied to the attributes a by calling a method "evaluate", passing it a set of Object objects that represent the attributes. Since the attributes are simple types (integer, boolean, or string variables) there was no need to create a class to represent them – they can be instantiated as objects and referred to collectively by the Object class. Also, it is intended that this code is as general as possible and can be used in the future in other application areas. For this reason the attributes were not defined in such a way as to be unique to web pages (for instance if I had decided to send the MultiAttributeValue-Function a WebPage object rather than a set of Objects). See figure 3.1 below.

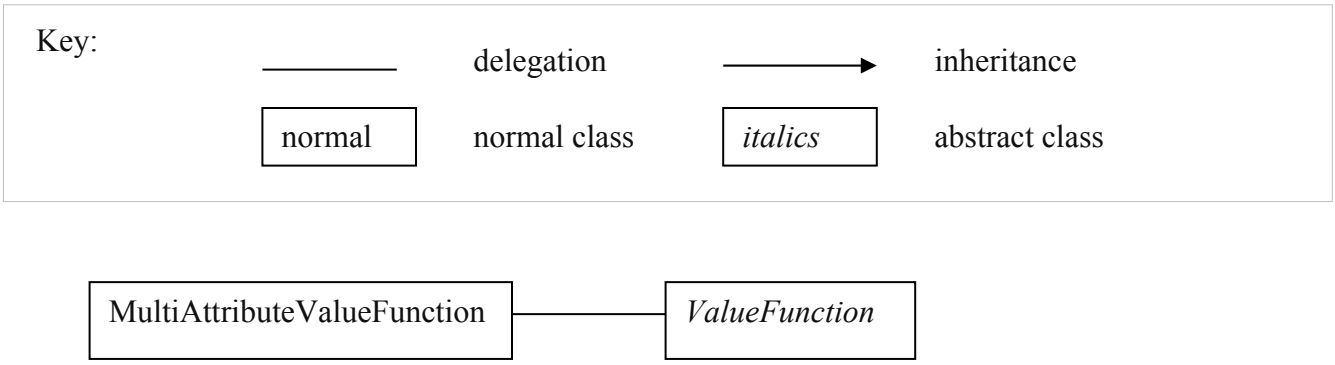

Figure 3.1

The ValueFunction class is an abstract class that insists a method called "evaluate" is implemented by all subclasses. This method must take an Object (the attribute) and return a floating point number. The MultiAttributeValueFunction class calls this method on each of its ValueFunction objects in turn, passing the corresponding attribute Object from the set passed to its own "evaluate" method (note that if the function requires a collection of attributes they can be passed as an array, which is itself an Object).

The ValueFunction class is extended by three other classes. These are CountValue-Function, ExistValueFunction, and ChoiceValueFunction.

CountValueFunction is an abstract class for any value function that takes an integer valued attribute as input. It requires a maximum value to be set. This class is extended by PositiveValueFunction and NegativeValueFunction, which implement the functions shown in the left and right side of figure 2.1 respectively. These are the classes used for evaluating attributes that are counted, such as the number of links in a page.

ExistValueFunction is an abstract class that is intended to evaluate a boolean valued attribute. It is extended by SomeValueFunction that implements the function given by rule 2.2, and by NoneValueFunction that implements the reverse of SomeValueFunction (i.e. if the attribute is true it returns 0.0, and 1.0 otherwise). These are used for attributes that will either exist or not, such as whether a page has frames.

Finally, ChoiceValueFunction is a variation of ExistValueFunction (but not similar enough to extend it) that takes a string attribute and compares it against a pre-stored string to determine its output. This implements the function given by rule 2.3. It is used when there are a number of alternatives to choose from, such as the extension of the server name that a web page is located on (i.e. .com, .org, etc…). This class is extended by CountChoiceValueFunction, that takes an attribute and returns 0.0 if the attribute does not match the pre-stored attribute, or uses a CountValueFunction to evaluate the attribute if it does match. This is the function given by rule 2.4. It is used to switch between attributes that can be counted, such as image or video downloads.

The class diagram for the value functions are shown below in figure 3.2.

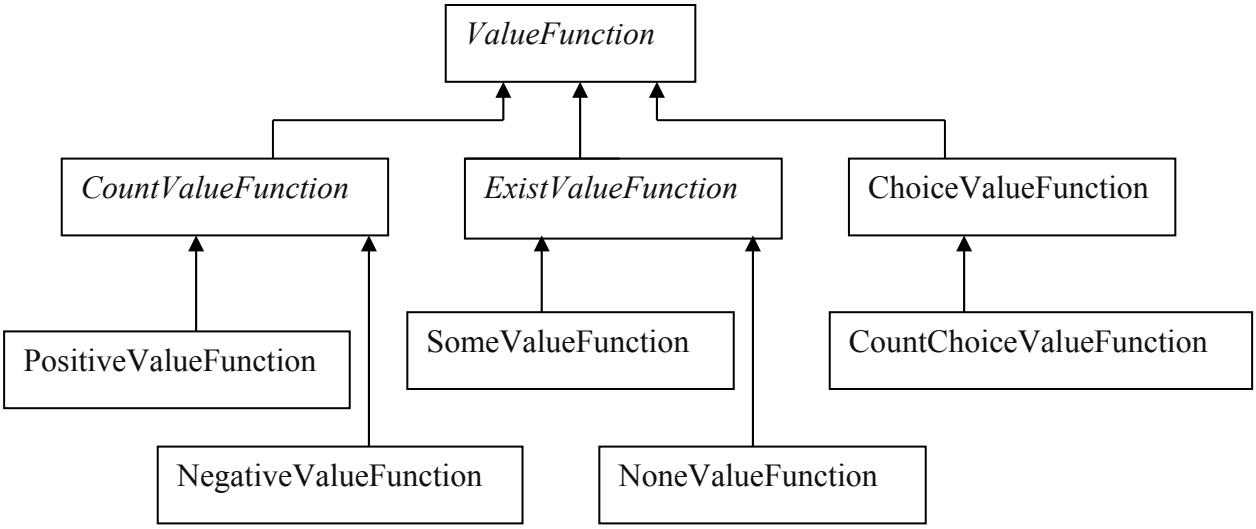

Figure 3.2

3.2.2 Web Page Preferencer

At the top-most level of the program is the class PagePref. This class stores a number of web pages, represented by WebPage objects, and a MultiAttributeValueFunction object. It also has to keep a record of the largest attribute values encountered, for use by any CountValueFunction objects (stored in the MultiAttributeValueFunction object) which require a maximum attribute value to be set. PagePref has methods to get and set the MultiAttributeValueFunction, get and set the WebPage objects, and parse and order the WebPage objects.

The WebPage class stores simple information about a web page, such as its name, location and value, and information about the various page attributes.

The parsing of the web pages is implemented by code already provided in the package javax.swing.text.HTML by a class called ParserDelegator. This class defines a class called CallBack to perform any operations required when the various web page tags or text are encountered. I have extended this class with MyCallBack, which is instantiated with a reference to a WebPage object, and simply updates the relevant attribute counts in that WebPage object. The parsing of the web pages is performed by a method in PagePref called “parsePages”. One MyCallBack object is instantiated for each WebPage object, and the “parse” method from the ParserDelegator class is called.
The ordering of the pages is performed by a method in PagePref called “orderPages”. It looks at each WebPage object in turn, extracting the relevant attribute information, instantiating them as objects and passing them to the “evaluate” method of MultiAttributeValue-Function, which in turn returns a value that is set as the value of the web page. Figure 3.3 shows the class structure of this part of the code.

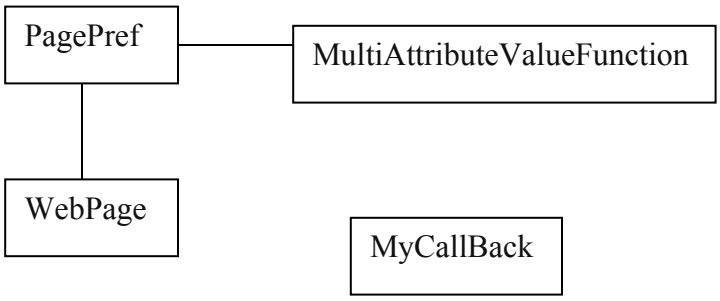

Figure 3.3

3.2.2 Graphical User Interface

The graphical interface is implemented by the class PagePrefGUI. This class contains a public "main" method, which when called instantiates a new PagePref object and creates an interface window. The window created contains a toolbar of main commands (a subclass of the JToolBar class in package javax.swing) and a number of panels (each a subclass of the JPanel class in package javax.swing) that are responsible for getting user input. Each of these panels have methods to retrieve the values set by the user and instantiate and return any classes that are needed by PagePref.

Class CommandToolbar contains three buttons. Button "parse" calls the "parsePages" method of PagePref, button "order" calls the "orderPages" method of PagePref, and button "quit" quits the program. The attributes needed by the PagePref methods are retrieved from the various panels.

Class AttributePanel is responsible for switching the value function for each attribute on or off. It does this by simply changing the function weight to 0.0. Class FunctionPanel is responsible for getting the user selection of the value function that will be used for a particular attribute. It contains a method "getFunctions" that returns an array of ValueFunction objects. Class WeightPanel has a method "getWeights" to return an array the weights set by the user. Finally, class WebPagePanel is responsible getting the web page locations (either individually or from a list in a text file) and for displaying the name and attributes of the web pages after they have been parsed. The class diagram for the interface is shown below in figure 3.4. Not suprisingly, there is a similarity between the class heirarchy of the classes used by PagePref and the classes used by PagePrefGUI.

Figure 3.5 shows an example of the interface. The table in the web page panel can scroll right to reveal the attribute values.

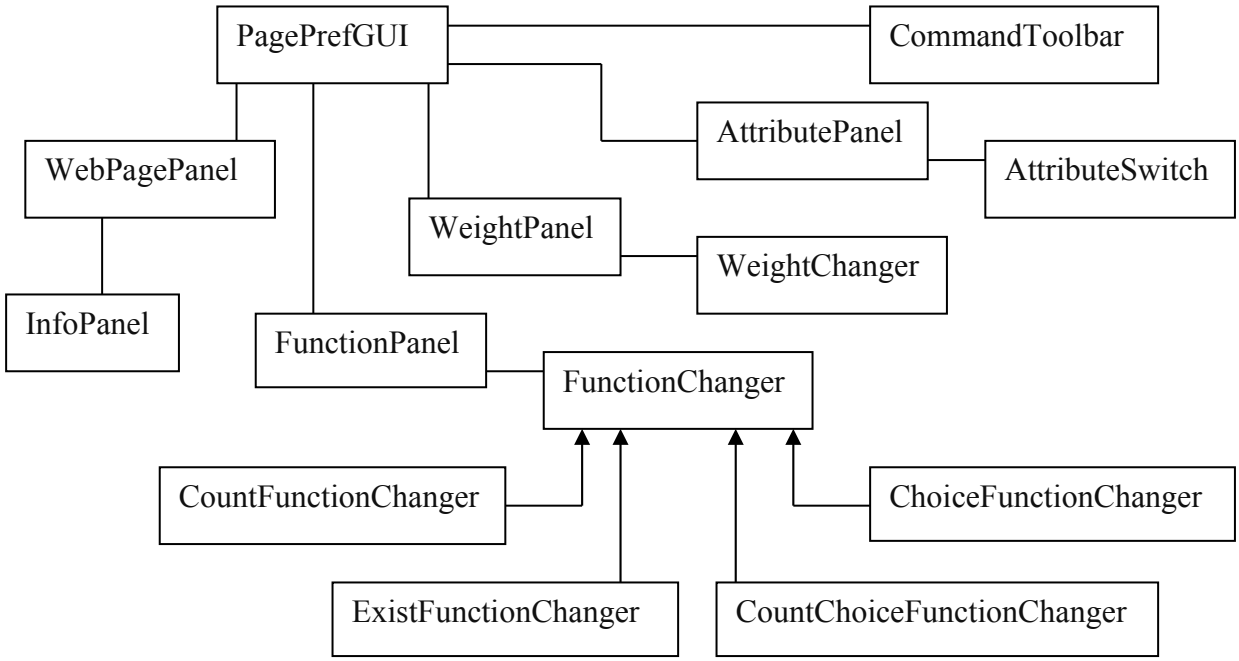

Figure 3.4

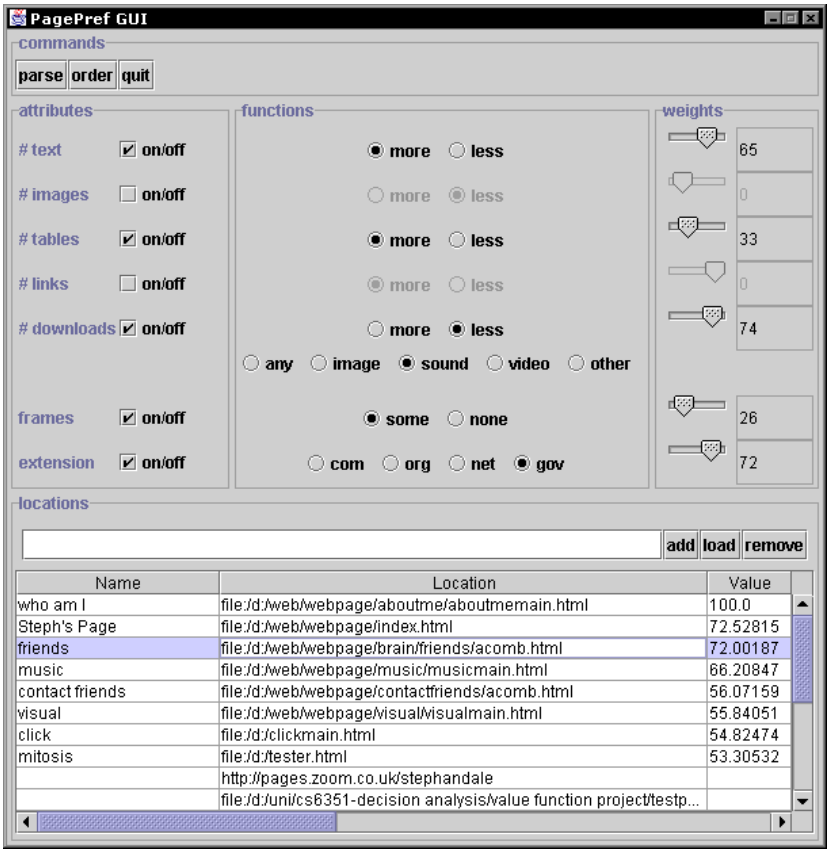

Figure 3.5

4. CONCLUSIONS

The program implemented successfully takes a set of candidate pages and re-orders them according to user defined preferences. The parsing of the web pages is achieved in a reasonable amount of time, and is fast enough to be used in addition to the use of standard search engines.

At present though, the program has a number of limitations that need to be improved upon before it can be considered as a useful aid.

4.1 Extensions

The method of ordering web pages relies on the fact that the user already knows the location of the candidate pages. This will not be the case most of the time. Ideally this program should be linked to a search engine to allow the user to perform a standard keyword search to generate the candidate pages. A given number of these pages (say the first 20) can then be re-ordered using the multi-attribute value function.

The ability to double click a row of the table of web page entries and have them open up in a browser is highly desirable. Currently, the user will have to enter the entire location of a page into a browser manually before being able to view the page.

These two extensions are necessary before the program is of any great use. The time taken to manually enter the page locations in the program, and to re-enter the location in a browser after parsing negates the benefit of using the program, which is intended to speed up the location of suitable web pages.

4.2 Improvements

Currently there is no set relationship between the classes that implement the value functions and the attributes. The value functions can only operate on certain attributes, and as yet there is no enforcement of which functions take which attributes. If a developer wishes to use my code in the future it is left entirely up to him to ensure that the correct attribute classes are sent to the correct value function classes. It would be better if the possibility of error was completely removed in the implementation by somehow linking the two.

The attributes are not stored in any kind of structure in the WebPage class. This is fine for the small amount of attributes that are being used currently, but in the future it may become necessary to implement some kind of attribute set. This set should allow new kinds of attributes to be added or old attributes to be removed without needing to alter any other code. This is especially relevant when dealing with Internet, which is a relatively new technology and as such the functionality of HTML is rapidly changing. Currently, code in the PagePref class will need to be altered if attributes are added or removed from the WebPage class.

Currently the only way to send WebPage objects to and from the WebPagePanel is by sending them all together. When the pages are to be parsed all of the WebPage objects are retrieved from the WebPagePanel object and passed to PagePref. When the pages have been ordered by PagePref they are

all sent back to the WebPagePanel object, which displays them in the order they are already in. This is inefficient. It would be better to simply send only the new pages to PagePref, and have a method in the WebPagePanel class that re-orders the table according to the page values.

Because of the passing of all web pages together (mentioned above), the results of the web page parsing does not show up in the table until they have all been parsed. For a small number of pages this is not a problem, but if dealing with more than ~15 it would be better to order the pages (and show the ordering) as they are parsed.

4.3 Summary

In summary, the program is correct and performs adequately within its limitations. A general purpose multi-attribute value function has been produced, which can make use of a variety of value functions to deal with different attributes. However, the limitations of the interface must be improved upon before it can be of real use as a decision aid. The improvements are simple, but given the extra time they would require they were deemed unnecessary for the purpose of the project, which has its focus on the multi-attribute value function.