huffman

drawing binary Huffman trees
with METAPOST and METAOBJ

Contributor
Maxime CHUPIN
notezik@gmail.com

Version 0.1, 2023, May, 10th
https://plmlab.math.cnrs.fr/mchupin/huffman
Abstract
This METAPOST package allows to draw binary Huffman trees from two arrays: an array of strings, and an array of weights (numeric). It is based on the METAOBJ package which provides many tools to build trees in general.

https://plmlab.math.cnrs.fr/mchupin/huffman
https://github.com/chupinmaxime/huffman

Contents
1 Installation 2
  1.1 With \TeX{}Live under Linux or macOS . . . . . . . . . . . . . . . . . . . 2
  1.2 With \TeX{} and Windows . . . . . . . . . . . . . . . . . . . . . . . . . . 3
  1.3 Dependencies . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

2 Main Command 3

3 Package Options 4

4 METAOBJ Tree Options 6

5 Access to Nodes and Leaves 7

6 Constructors 9

This package is in beta version—do not hesitate to report bugs, as well as requests for improvement.

1 Installation

huffman is on CTAN and can also be installed via the package manager of your distribution.

https://www.ctan.org/pkg/huffman

1.1 With \TeX{}Live under Linux or macOS

To install huffman with \TeX{}Live, you will have to create the directory texmf in your home.

user $> mkdir ~/texmf

Then, you will have to place the huffman.mp file in

~/texmf/metapost/huffman/

Once this is done, huffman will be loaded with the classic METAPOST input code
1.2 With MikTeX and Windows

These two systems are unknown to the author of \texttt{huffman}, so we refer you to the MikTeX documentation concerning the addition of local packages:

\texttt{http://docs.miktex.org/manual/localadditions.html}

1.3 Dependencies

\texttt{huffman} depends, of course on \texttt{METAPOST} \cite{2}, as well as the packages \texttt{metaobj} \cite{1} and—if \texttt{huffman} is not used with \texttt{Lua\LaTeX} and the \texttt{luamplib} package—the \texttt{latexmp} package.

2 Main Command

The package \texttt{huffman} provides one principal command (which is a \texttt{METAOBJ} like constructor):

\texttt{newBinHuffman.(name)((sizeofarrays))((symbarray),(valuearray))}

\begin{itemize}
  \item \texttt{name}: is the name of the object;
  \item \texttt{sizeofarrays}: is the size (integer) of the arrays;
  \item \texttt{symbarray}: is the array of \texttt{string} containing the symbols;
  \item \texttt{valuearray}: is the array of \texttt{numeric} containing the weights associated to the symbols.
\end{itemize}

The data arrays should begin at index 1.

\subsection*{Exemple 1}

\begin{verbatim}
input huffman

beginfig(0);
string charList[];
numeric frequency[];
charList[1]:="a"; frequency[1]:=0.04;
charList[2]:="b"; frequency[2]:=0.05;
charList[3]:="c"; frequency[3]:=0.06;
charList[4]:="d"; frequency[4]:=0.07;
charList[5]:="e"; frequency[5]:=0.1;
charList[6]:="f"; frequency[6]:=0.1;
charList[7]:="g"; frequency[7]:=0.18;
charList[8]:="h"; frequency[8]:=0.4;

newBinHuffman.myHuff(8)(charList,frequency);
\end{verbatim}
Package Options

You can modify the size of the internal nodes of the tree with the following command:

\texttt{set_node_size(\langle dim \rangle)}

\texttt{(dim)}: is the diameter with unit of the circle (default: 13pt).

You can change the color for the symbol boxes with the following command:

\texttt{set_leaf_color(\langle color \rangle)}

\texttt{(color)}: is a METAPOST \texttt{color}.

You can hide the bit values printed on the edges of the tree with the following boolean (\texttt{true} by default):

\texttt{show_bits}
Similarly, you can set the following boolean to \texttt{false} to hide the node values:

\begin{verbatim}
show_node_values
\end{verbatim}

Finally, you can set the following boolean to \texttt{false} to hide the leaf values:

\begin{verbatim}
show_leaf_values
\end{verbatim}

Here an example combining all these commands and variables.

\begin{verbatim}
Exemple 2

input huffman

beginfig(0);
string charList[];
numeric frequency[];
charList[1]:="s_1"; frequency[1]:=2;
charList[2]:="s_2"; frequency[2]:=4;
charList[3]:="s_3"; frequency[3]:=2;
charList[4]:="s_4"; frequency[4]:=12;
charList[5]:="s_5"; frequency[5]:=8;

set_leaf_color(0.2[white,green]);
set_node_size(8pt);
show_bits:=false;
show_node_values:=false;
show_leaf_values:=false;
newBinHuffman.myHuff(5)(charList,frequency);
myHuff.c=origin;
drawObj(myHuff);
endfig;
\end{verbatim}
4 METAOBJ Tree Options

Because the Huffman tree is built using the METAOBJ tree constructor, the METAOBJ tree options are available. All of them are not well suited for this application mostly because the Huffman tree is built using elementary trees. The options we give to the Huffman constructor are passed to all the subtrees.

We give in table 1 the METAOBJ options for the trees that could be used for the Huffman constructor.

<table>
<thead>
<tr>
<th>Option</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>treemode</td>
<td>string</td>
<td>&quot;D&quot;</td>
<td>direction in which the tree develops; there are four different possible values: &quot;D&quot; (default), &quot;U&quot;, &quot;L&quot; and &quot;R&quot;</td>
</tr>
<tr>
<td>treeflip</td>
<td>boolean</td>
<td>false</td>
<td>if true, reverses the order of the sub-trees</td>
</tr>
<tr>
<td>treenodehsize</td>
<td>numeric</td>
<td>-1pt</td>
<td>if non-negative, all the nodes are assumed to have this width</td>
</tr>
<tr>
<td>treenodevsize</td>
<td>numeric</td>
<td>-1pt</td>
<td>if non-negative, all the nodes are assumed to have this height</td>
</tr>
<tr>
<td>dx</td>
<td>numeric</td>
<td>0</td>
<td>horizontal clearance around the tree</td>
</tr>
<tr>
<td>dy</td>
<td>numeric</td>
<td>0</td>
<td>vertical clearance around the tree</td>
</tr>
<tr>
<td>hsep</td>
<td>numeric</td>
<td>1cm</td>
<td>for a horizontal tree, this is the separation between the root and the sub-trees</td>
</tr>
<tr>
<td>vsep</td>
<td>numeric</td>
<td>1cm</td>
<td>for a vertical tree, this is the separation between the root and the sub-trees</td>
</tr>
<tr>
<td>hbsen</td>
<td>numeric</td>
<td>1cm</td>
<td>for a vertical tree, this is the horizontal separation between sub-trees; the sub-trees are actually put in a HBox and the value of this option is passed to the HBox constructor</td>
</tr>
<tr>
<td>vbsep</td>
<td>numeric</td>
<td>1cm</td>
<td>for an horizontal tree, this is the vertical separation between sub-trees; the sub-trees are actually put in a HBox and the value of this option is passed to the HBox constructor</td>
</tr>
<tr>
<td>edge</td>
<td>string</td>
<td>&quot;ncline&quot;</td>
<td>name of a connection command (see METAOBJ documentation)</td>
</tr>
<tr>
<td>Dalign</td>
<td>string</td>
<td>&quot;top&quot;</td>
<td>vertical alignment of sub-trees for trees that go down (the root on the top); the other possible values are &quot;center&quot; and &quot;bot&quot;</td>
</tr>
</tbody>
</table>

Table 1: Table of METAOBJ tree options.

Here is an example using some of these options.

Exemple 3

```plaintext
input huffman
beginfig(0);
```
5 Access to Nodes and Leaves

To access the nodes and the leaves, you can use the \texttt{treeroot} command from \texttt{METAOBJ}, see the documentation for details.

\texttt{ntreepos(Obj((\textit{name}))(\langle\textit{int}\rangle,\langle\textit{int}\rangle,\textit{etc.}))}

The sequence of \texttt{\langle\textit{int}\rangle} gives the choice of branch where the children are numbered from 1 to \textit{n}.

The following example shows a use of this mechanism.

Exemple 4

\begin{verbatim}
input huffman;
beginfig(0);
string charList[];
numeric frequency[];
charList[1]:="s_1"; frequency[1]:=2;
charList[2]:="s_2"; frequency[2]:=4;
charList[3]:="s_3"; frequency[3]:=2;
charList[4]:="s_4"; frequency[4]:=12;
charList[5]:="s_5"; frequency[5]:=8;

newBinHuffman.myHuff(5)(charList,frequency)
"treemode(R)","treeflip(true)","hsep(1.5cm)", "edge(nccurve)"
,"angleA(0)", "angleB(0)"
myHuff.c=origin;
drawObj(myHuff);
endfig;
\end{verbatim}
charList[1]="s_1"; frequency[1]=2;

newBinHuffman.myHuff(5)(charList,frequency);
myHuff.c=origin;
ncarcbox(ntreepos(Obj(myHuff))(2,1,2))(ntreepos(Obj(myHuff))(2,2))
"linestyle(dashed evenly)", "nodesepA(5mm)", "nodesepB(5mm)";
drawObj(myHuff);
endfig;

Of course, this can only be used in two steps, first build the tree, then annotate it.
You can also access the leaves and the nodes using names. During the construction of the tree, names are given to leaves and nodes. Because you may want to build several Huffman trees, the trees are numbered. You can get the current tree number with the following command:

get_huffmanTreeNbr()

The leaves are numbered during the construction, and the corresponding METAOBJ object is named as follows:

leaf(leaf number)(tree number)

The nodes are numbered during the construction, and the corresponding METAOBJ object is named as follows:

node(node number)(tree number)
Thanks to METAOBJ you can annotate the tree using all the tools METAOBJ provides.

Exemple 5

```plaintext
input huffman;
beginfig(0);
string charList[];
numeric frequency[];
charList[1]:="s_1"; frequency[1]:=2;
charList[2]:="s_2"; frequency[2]:=4;
charList[3]:="s_3"; frequency[3]:=2;
charList[4]:="s_4"; frequency[4]:=12;
charList[5]:="s_5"; frequency[5]:=8;
newBinHuffman.myHuff(5)(charList,frequency);
myHuff.c=origin;
ncarcbox(node2_1)(leaf4_1)
"linestyle(dashed evenly)", "nodesepA(5mm)", "nodesepB(5mm)" ;
drawObj(myHuff);
endfig;
```

6 Constructors

The Huffman algorithm uses only three constructors that you can redefine to adapt the tree to your needs. Here are the three constructors (roughly commented in French) defined in this package. We will not discuss the code here but you are free to redefine and adapt it.
% style 'dune feuille caractère et proba

vardef newHuffmanLeaf@#(expr ch)(expr v) text options=
  % @# est l'identifiant de la feuille
  % c est le caractère considéré (ou la chaine)
  % v est la proba ou l'entier associé
  save _text_v,
  _text_token,_height_v,_height_token,_height_max,
  _width_token,_width_v;
  picture _text_v,_text_token;
  % on calcule le height max des deux écritures pour faire deux boites de même
  % hauteur
  _text_v := textext(v);
  _text_token := textext("$"&ch&"$");
  _height_v := abs((ulcorner _text_v) - (llcorner _text_v));
  _width_v := abs(urcorner _text_v - ulcorner _text_v);
  _height_token := abs(ulcorner _text_token - llcorner _text_token);
  _width_token := abs(urcorner _text_token - ulcorner _text_token);
  _height_max := max(_height_token,_height_v);
  % on fabrique deux boîtes vides aux bonne dimensions
  % et on ajoute un label au centre de celles-ci
  if(show_leaf_values):
    newEmptyBox.scantokens(str @# & "ch1")(_width_token+4,2*_height_max)
    "framed(true)" ,"fillcolor(_huffmanLeaf)" , "filled (true)" , options;
    ObjLabel.Obj(scantokens(str @# & "ch1"))(textext("$" & ch & "$"));
  newEmptyBox.scantokens(str @# & "ch2")(_width_v +4,2*_height_max)
    "framed(true)" , options;
    ObjLabel.Obj(scantokens(str @# & "ch2"))(textext(v));
  % on fixe relativement les coordonnées des deux boîtes pour qu'elles se touchent
  scantokens(str @# & "ch1").e=scantokens(str @# & "ch2").w;
  % on fabrique un container qui les regroupe et qui sera la feuille
  newContainer.@#(scantokens(str @# & "ch1"),
    scantokens(str @# & "ch2"));
else:
% si seulement le symbole
newBox.@#(textext("$" & ch & "$"))
"framed(true)="/fillcolor(_huffmanLeaf)", "filled (true)", options;
fi
enddef;

Node Code

% style 'dun nœud interne (non feuille) de 'larbre
vardef newHuffmanNode@#(expr v) text options=
    newCircle.@#(""") "circmargin(_node_size)",options;
if(show_node_values):
    ObjLabel.Obj(scantokens(str @#))(textext(v));
fi
enddef;

Tree Code

% style de 'larbre binaire de Huffman
vardef newHuffmanBinTree@#(suffix theroot)(text subtrees) text options=
    % un simple arbre
    newTree.@#(theroot)(subtrees) "Dalign(top)" , "hbsep (0.3cm)",options;
if(show_bits):
    % et on met 0 et 1 sur ses deux connections
    ObjLabel.Obj(@#)(btex 0 etex)
    "labpathid(1)" , "laberase(true)" , "labcolor(_huffmanBit)"
    ObjLabel.Obj(@#)(btex 1 etex)
    "labpathid(2)" , "laberase(true)" , "labcolor(_huffmanBit)"
fi
enddef;

References

[1] Denis B. Roegel. The metaobj package. MetaPost package providing high-
Command Index

get_huffmanTreeNbr, 8
newBinHuffman, 3
ntreepos, 7

set_leaf_color, 4
set_node_size, 4
show_bits, 4
show_leaf_values, 5
show_node_values, 5