yaahp 算法库使用文档

1 目录、文件及基础环境设定

1.1 目录文件结构及说明

yaahplibs	算法库根目录
├── **dotnetfx**	Microsoft .NET Framework 2.0安装程序目录
│ ├── dotnetfx_x64.exe	64位Microsoft .NET Framework 2.0 安装程序
│ └── dotnetfx_x86.exe	32位Microsoft .NET Framework 2.0 安装程序
├── **example**	算法库示例程序目录
│ ├── bridge.single.xml	AHP模型及判断矩阵数据XML文件
│ ├── **libs**	示例程序所用到的库
│ ├── **yaahplib-demos**	算法库示例程序工程目录
│ ├── yaahplib-demos.sln	算法库示例程序Visual Studio 2015解决方案
│ └── yaahplib.license	算法库15天试用License
├── **vc-runtime**	Microsoft VC 2015运行时安装程序目录
│ ├── vc_redist_x64.exe	64位Microsoft VC 2015运行时安装程序
│ └── vc_redist_x86.exe	32位Microsoft VC 2015运行时安装程序
├── **x64**	yaahp 64位算法库目录
│ ├── API.chm	yaahp算法库API文档
│ ├── License.dll	使用授权库
│ ├── bridge.single.xml	AHP模型及判断矩阵数据XML文件
│ ├── psoahp.dll	粒子群算法实现的判断矩阵修正, 补全等算法库, 64位版本
│ ├── yaahplib-demos.exe	算法库示例程序可执行文件
│ ├── yaahplibs.dll	算法库主文件
│ └── yaahplib.license	算法库15天试用License
└── **x86**	yaahp 32位算法库目录
├── API.chm	yaahp算法库API文档
├── License.dll	使用授权库
├── bridge.single.xml	AHP模型及判断矩阵数据XML文件
├── psoahp.dll	粒子群算法实现的判断矩阵修正, 补全等算法库, 32位版本
├── yaahplib-demos.exe	算法库示例程序可执行文件
├── yaahplibs.dll	算法库主文件

└── yaahplib.license　　算法库15天试用License

1.2 算法库文件

算法库可执行文件3个(yaahplibs.dll, psoahp.dll, License.dll), 存放在x64(64位)及x86(32位)目录下; 另外, 为了正常使用算法库, 还需将使用授权文件(yaahplib.license)保存在算法库可执行文件相同目录下.

1.3 基础环境设定

1) .NET Framework 2.0

如果系统中未安装.NET Framework2.0, 需要自行安装. 根据Windows操作系统类型, 选择dotnetfx目录下的相应安装程序进行安装. 如果是Windows 7或更高版本, 会在第一次执行需要.NET Framework 2.0的程序时自动提示安装.

2) VC 2015 运行时

根据Windows操作系统类型, 选择vc-runtime目录下的相应安装程序进行安装.

3) 检验

执行x64(64位)/x86(32位)目录下的"yaahplib-demos.exe", 启动yaahp算法库示例程序.

2 算法库示例程序

本节对算法库示例程序功能进行介绍, 在此基础上参考此示例程序代码和API文档即可利用算法库进行开发.

图1 算法库示例程序主窗口

此示例程序使用了算法库的大部分接口, 演示了如何使用算法库完成AHP相关计算.

2.1 License信息

点击”License infomation”, 将弹出一个窗口显示目前使用授权信息, 如图2所示. 其中Trial days remain的值为剩余的使用天数, 小于0表示试用使用授权已过试用期, 99999表示正式使用授权. 试用使用授权则用户名显示为”Trial”, 正式使用授权显示授权用户名. 例如在图2中, 左侧为试用使用授权信息, 右侧是正式使用授权信息.

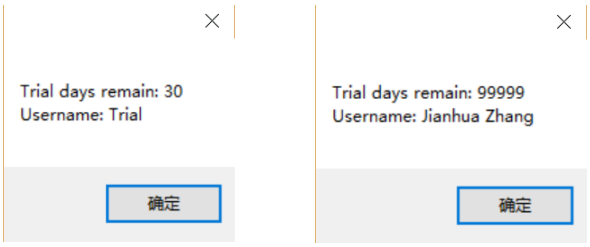

图2 使用授权信息窗口

2.2 判断矩阵计算

yaahp算法库中有两类计算: 1) 对象为单个判断矩阵的计算; 2) 对象为整个AHP模型的计算, 前者计算是后者的一部分. 利用单独暴露出判断矩阵计算接口, 可以方便地创建并计算单个的判断矩阵而不需要给出层次结构模型, 而整个AHP模型的计算必须以特定的数据格式给出AHP模型和判断矩阵数据.

● 判断矩阵初始化

”Matrix”组中, 上方的几个按钮点击后可以初始化判断矩阵, 点击”Init Consistent Matrix”按钮初始化一个满足基本一致性的判断矩阵; 点击”Init Inconsistent Matrix”按钮初始化一个不满足基本一致性的判断矩阵; 点击”Init Acceptable Matrix”按钮初始化一个残缺但可接受的的判断矩阵; 点击”Init DirectValue Matrix”按钮初始化一个包含直接数值(而不是标度上的值)的判断矩阵. 点击这四个初始化按钮中的任何一个后, 会弹出两次信息窗口, 第一次显示矩阵完成初始化, 但是并没有输入两两比较数据时的信息, 如图3左图所示; 第二次显示输入两两比较数据后的信息, 如图3右图所示.

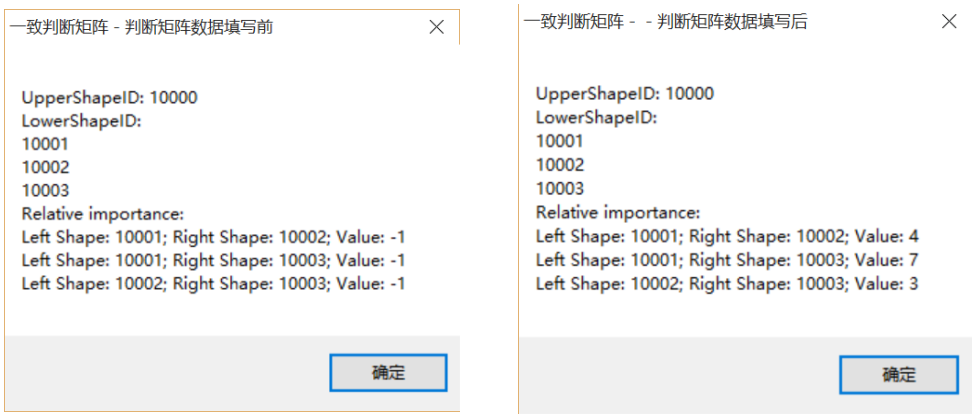

图3 判断矩阵初始化信息窗口

判断矩阵初始化后, “Matrix”组下方的四个判断矩阵相关计算按钮就可用了.

- 一致性

点击“Matrix”组下方的”Consistency”按钮, 将弹出一个窗口显示当前判断矩阵的一致性信息, 包括次序一致性和基本一致性. 图4中显示该判断矩阵满足次序一致性并且基本一致性为0.0311.

点击不同的判断矩阵初始化按钮, 然后点击”Consistency”按钮, 将会显示不同判断矩阵的一致性信息. 同样, 点击“Matrix”组下方的其他判断矩阵计算按钮, 将对当前的判断矩阵进行计算.

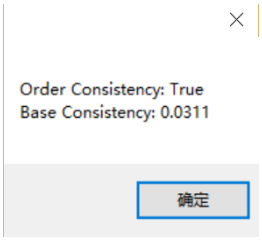

图4 判断矩阵一致性信息

- 对一致性影响最大的要素

点击“Matrix”组下方的”Max Influency”按钮, 将弹出一个窗口显示当前判断矩阵中对基本一致性比例影响最大的要素位置, 位置索引以0开始, 如图5所示. 图5中, 左侧为不满足基本一致性的判断矩阵情况; 右侧为满足基本一致性比例的情况.

图5 对判断矩阵一致性影响最大的要素

- 排序权重

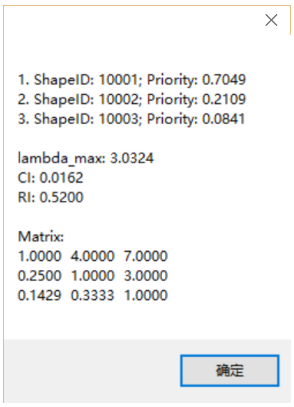

图6 判断矩阵排序计算结果

点击“Matrix”组下方的”Priorities”按钮, 将弹出一个窗口显示当前判断矩阵的排序权重计算结果, 如图6所示. 弹出的窗口中显示判断矩阵各个要素的ID和权重、lambda_max、CI、RI以及判断矩阵的完整数据.

● 一致性修正后的排序权重

点击“Matrix”组下方的”Adjust then Priorities”按钮, 将首先尝试对判断矩阵进行一致性修正, 如果判断矩阵满足基本一致性, 将会弹出窗口如图7左侧的提示窗口, 提示不需要修正; 否则, 将显示判断矩阵首先对一致性进行修正然后计算排序权重的结果, 如图7右图所示. 从图7右图中可以看出, 比不修正直接计算的结果多了顶部的修正前后基本一致性比例数据, 并且从下方的判断矩阵数据中可以看出对判断矩阵的修正结果.

图7 判断矩阵修正后的判断矩阵排序计算结果

2.3 完整AHP计算

● AHP模型与判断矩阵数据XML格式

完整的AHP计算需要XML格式的层次模型和判断矩阵数据, 格式说明如下. 以此为数据接口准备数据然后使用yaahp算法库进行计算即可. 文件“bridge.single.xml”是一个完整的数据示例.

```
<?xml version="1.0" encoding="utf-8"?>
<AhpFileXml>
  <!— XML头标识, 必须为”ahp6” —>
  <header>ahp6</header>

  <!— 唯一标示此模型及数据的GUID, 例如C#中使用”System.Guid.NewGuid().ToString()”生成 —>
  <GUID>cedc1e10-8460-4b14-aeb1-1525e1757a02</GUID>

  <!— 层次模型中要素部分, Shapes的每个子节点描述一个要素 —>
  <!— ShapeCount为包括决策目标, 各中间层要素和备选方案的总个数 —>
```

```
<Shapes ShapeCount=“18">
  <!— 要素的文本即显示的文字 —>
  <!— 要素的ID为一个long型的唯一数值—>
  <!— 要素的Tag值只能是”top”, “mid”和”bottom”, 标识要素类型, 分别对应决策目标, 中间层要素和备选方案 —>
  <Shape Text="过河收益" Tag="top" ID="632881677470937500" />
  <Shape Text="经济收益" Tag="mid" ID="632881677480156250" />
                                    ……
</Shapes>

<!— 层次模型中连接部分, Connections的每个子节点描述一个连接 —>
<!— ConnectionsCount为连接总个数 —>
<Connections ConnectionsCount=“47">
  <!— FromIndexOfShapeInArray值表示连接起点要素在Shapes列表中的索引位置 —>
  <!— 索引从1开始, 例如1表示Shapes的第一个子节点描述的要素 —>
  <!— FromName取值可能两种: “Top”和”Bottom”, 表示连接起点在起点要素上的位置 —>
  <!— ToIndexOfShapeInArray值表示连接终点要素在Shapes列表中的索引位置 —>
  <!— ToName取值可能两种: “Top”和”Bottom”, 表示连接终点在终点要素上的位置 —>
  <Connection FromIndexOfShapeInArray="1" FromName="Top" ToIndexOfShapeInArray="0" ToName="Bottom" />
  <Connection FromIndexOfShapeInArray="2" FromName="Top" ToIndexOfShapeInArray="0" ToName="Bottom" />
                                    ……
</Connections>

<!— matrixes错别字历史遗留 —>
<!— IsMatrixesSaved值指出判断矩阵数据是否保存 —>
<!— 如果使用基本算法库, 该值必须为true并且必须有下面的Matrixes部分数据—>
<IsMatrixesSaved>true</IsMatrixesSaved>

<!— 所有判断矩阵数据, Matrixes每个子节点描述一个判断矩阵, Count值为判断矩阵个数—>
<Matrixes Count=“15”>
  <!— 一个判断矩阵 —>
  <!— UpperShapeid判断矩阵Upper要素的ID —>
  <!— LowerShapeCount判断矩阵中要素个数 —>
  <!— RelativesCount判断矩阵中两两比较元素个数 —>
  <Matrix UpperShapeid="632881677470937500" LowerShapeCount="3" RelativesCount=“3">
    <!— 每个LowerShapeID给出一个判断矩阵中要素的ID —>
    <LowerShapeID>632881677480156250</LowerShapeID>
    <LowerShapeID>632881677495625000</LowerShapeID>
    <LowerShapeID>632881677499687500</LowerShapeID>
```

```
    <!— 每个Relative给定一个两两比较数据 —>
    <!— ShapeID1两两比较的左要素 —>
    <!— ShapeID2两两比较的右要素 —>
    <!— RelativeValue两两比较值—>
    <!— 取值: -9,-8,-7,-6,-5,-4,-3,-2,1,2,3,4,5,6,7,8,9, 小于0的值对应标度中小于1的标度, 例如-2对应1/2; -5对应1/5—>
    <!— IsDirectValue和RelativeValueDirect=“1”用于直接给定比较值, 而不再考虑标度转换 —>
    <!— 例如某个两两比较是1.65, 那么IsDirectValue设为”true”并将RelativeValueDirect设为”1.65”即可—>
    <!— 再如某个两两比较是0.39, 那么IsDirectValue设为”true”并将RelativeValueDirect设为”0.39”—>
    <!— Relative的IsDirectValue和RelativeValueDirect属性也可以没有, 没有的话默认值为False和1 —>
   <Relative ShapeID1="632881677480156250" ShapeID2="632881677495625000" RelativeValue="3"
IsDirectValue="False" RelativeValueDirect=“1"/>
    <!— 一个两两比较可以由UpperShapeId, ShapeID1和ShapeID2确定 —>
    <!— 表达的意思为: “对于UpperShapeId, ShapeID1和ShapeID2相比的重要性/优势如何” —>
    <Relative ShapeID1="632881677480156250" ShapeID2="632881677499687500" RelativeValue="6"
IsDirectValue="False" RelativeValueDirect="1" />
    <Relative ShapeID1="632881677495625000" ShapeID2="632881677499687500" RelativeValue="2"
IsDirectValue="False" RelativeValueDirect="1" />
   </Matrix>
                                ……
 </Matrixes>
</AhpFileXml>
```

- 从XML文件载入层次模型和判断矩阵数据

点击“AHP”组中的”Load from file”按钮, 将从上方文本框中指定的文件(点击文本框右侧的按钮可以选择文件)中载入层次模型和判断矩阵数据. 数据成功载入后会弹出消息框提示, 并且”AHP”组下方的”Metrices count”, “Set Adjust Flag”等按钮将变为可用状态.

- 从XML字符串载入层次模型和判断矩阵数据

点击“AHP”组中的”Load from string”按钮, 将从上方文本框中指定的文件(点击文本框右侧的按钮可以选择文件)中载入XML文件内容, 然后以从此字符串中载入层次模型和判断矩阵数据. 数据成功载入后会弹出消息框提示, 并且”AHP”组下方的”Metrices Count”, “Set Adjust Flag”等按钮将变为可用状态.

这种形式不需要以文件为数据交换介质, 能够直接在内存中完成数据交换.

- 判断矩阵数量

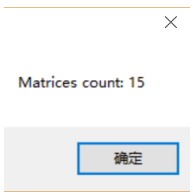

图8 判断矩阵数量

点击“AHP”组中的”Matrices Count”按钮, 将弹出一个窗口显示当前层次模型及判断矩阵数据中的判断矩阵数量, 如图8所示.

- 某个判断矩阵一致性比例

点击“AHP”组中的”Matrices Consistency”按钮, 将弹出一个窗口显示某个判断矩阵(通过指定判断矩阵ID来确定)的一致性比例, 如图9所示. 在此功能的示例代码中, 可以看到查询判断矩阵ID, 然后以判断矩阵ID对判断矩阵进行操作.

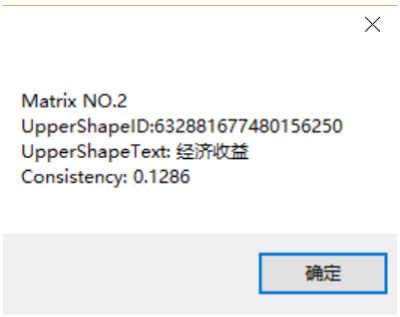

图9 某个判断矩阵的一致性比例

- 某个判断矩阵中对基本一致性比例影响最大的元素位置

点击“AHP”组中的”Max Influency”按钮, 将弹出一个窗口显示某个判断矩阵(通过指定判断矩阵ID来确定)中对一致性比例影响最大的元素, 如图10所示.

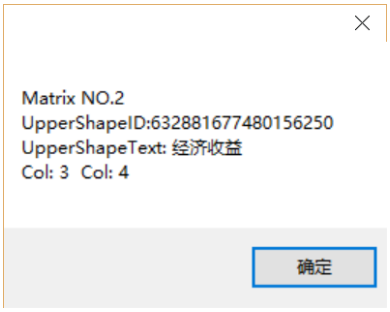

图10 某个判断矩阵中对一致性比例影响最大的元素

- 某个判断矩阵的排序权重

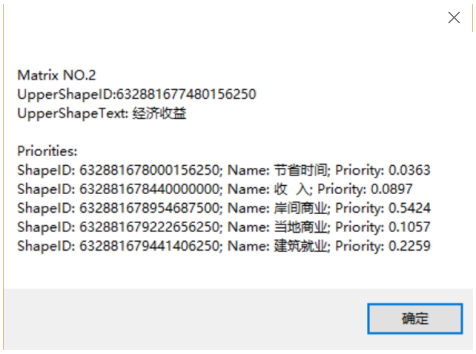

图8 判断矩阵数量

点击“AHP”组中的”Matrix Priorities”按钮, 将弹出一个窗口显示某个判断矩阵(通过指定判断矩阵ID来确定)的排序权重计算结果, 如图11所示.

- 标记某个判断矩阵为自动修正

点击“AHP”组中的”Set Adjust Flag”按钮, 将弹出一个窗口提示已将某个判断矩阵(通过指定判断矩阵ID来确定)标记为自动修正一致性比例, 如图12所示. 被标记为自动修正的判断矩阵在总排序权重计算时, 将会首先对基本一致性进行修正, 然后再计算排序权重.

图12 标记某个判断矩阵自动修正一致性

- 计算总排序权重

点击“AHP”组中的”Global Priorities”按钮, 将弹出总排序权重计算结果窗口, 如图13左图所示. 计算结果分为三个部分, 备选方案排序权重(也即总排序权重)、各中间层要素对决策目标的权重以及各判断矩阵详细数据, 这三部分在窗口中分为3行显示. 点击每行最右侧的小按钮, 点击的弹出窗口中可以看到更各部分的更详细的数据, 如图13右图所示.

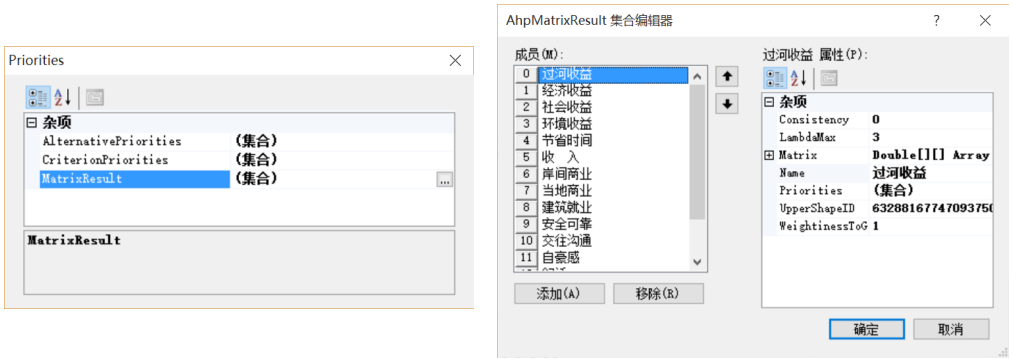

图13 总排序权重

- 计算子目标的总排序权重

点击“AHP”组中的”Sub-goal Priorities”按钮, 将弹出子目标的总排序权重计算结果窗口, 如图14所示. 与总排序权重非常类似, 区别在于子目标总排序权重涉及的判断矩阵数量少, 如图14右图所示.

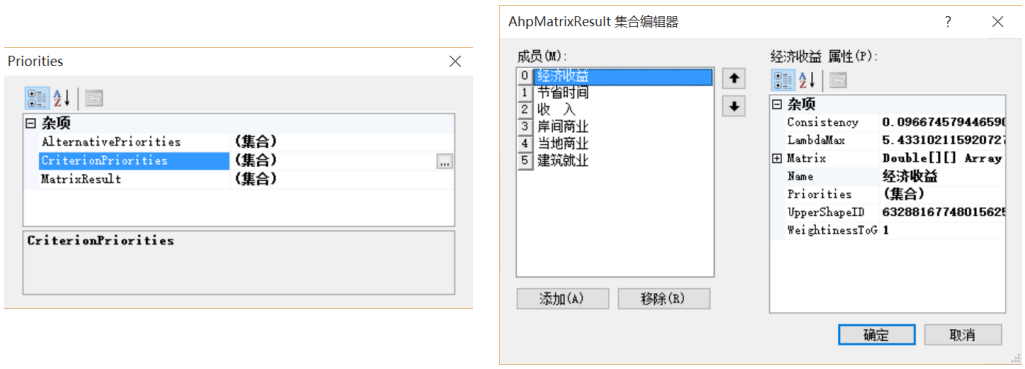

图14 子目标的总排序权重