A MATLAB Toolbox for Two-Dimensional Rigidity Percolation:

The Pebble Game

Preface

Welcome to MATLAB toolbox for 2D rigidity percolation. The generical rigidity of graphs plays an essential role in the field of sensor network location. Jacobs et al. provided an exciting and effective method that uses pebble games to percolate the rigidity of a network [2]. To meet the need of research and teaching on MATLAB, we implement the algorithm and build a toolbox. The implementation disassembles the pebble game into elemental actions, which provides interactivity and reusability. Benefitting from the build-in graph operation from MATLAB, the toolbox has good compatibility. The practice shows the following convenience:

- Analyzing the rigidity of a network with less than ten lines of code.
- Observing the work of pebble games step by step.
- Never generating extra variables to contaminate workspace.
- Generating good figures for papers.

We sincerely hope our toolbox facilitates research and teaching. Any suggestions, corrections, and improvements are welcome.

Please Email: bluebirdhouse@me.com

Table of Contents

1. Concepts

Pebbling is a game that involves placing pebbles on the vertices of a directed acyclic graph according to specific rules [1]. When the pebble game is used to percolate the rigidity of 2D generic networks, the rules are as follows. Each vertex is given two pebbles. The edges are added to the graph one at a time, and pebbles are rearranged to cover the edge. First, we look at the vertices that incident to the newly added edge. If either vertex has a free pebble, then use it to cover the edge. In the meantime, the edge is directed away from the vertex that offers a pebble. If neither of them has free pebbles, their pebbles must have been used to cover existing edges. We should try to free up a pebble for the newly added edge. If a vertex at the other end of existing edges has a free pebble, then that pebble can cover the associated edge. The swap will free up a pebble. More formally, we are searching for a free pebble that follows the directions of edges. This search of a free pebble continues until either a pebble is found and a sequence of swaps allows the new edge to be covered, or else no more vertices can be reached and no free pebbles have been discovered. If we fail to find a free pebble, the new edge will be removed and play no roles in pebble games.

Two tasks should be accomplished when percolating the rigidity of a 2D generic network: identifying independent edges and exploring rigid clusters.

The first task starts from a graph that contains no edge. We randomly add one edge that should be tested to the graph and quadruple the edge; then, four edges are added to the graph one after another to find pebble covering. If four edges are all covered by pebbles, three will be removed from the graph. The left one is an independent edge. If not, we remove all of them and mark the edge redundant.

After every edge is tested, rigid clusters will be explored. Since an independent edge can only belong to one rigid cluster, the triple edge will exhaust three free pebbles of the current rigid cluster. Any vertex that belongs to the same rigid cluster will not collect pebbles from the associated two vertices. The edges between two vertices that fail to find a free pebble are marked as the same rigid cluster.

Our toolbox differs from the work [2] in two ways. The toolbox performs a breadth-first search; then, it moves through the search results to find a free pebble. This strategy finds free pebbles faster and is easier to program, but it will consume more memory. Furthermore, we believe that pinning free pebbles at a vertex is equivalent to using free pebbles to cover the edge (Theorem 1). Thus, we pin three pebbles and use the fourth one to cover the edge rather than cover quadruple edges. We will improve this claim.

Theorem 1. *Pinning free pebbles at a vertex is equivalent to using free pebbles to cover the edge associated with the vertex.*

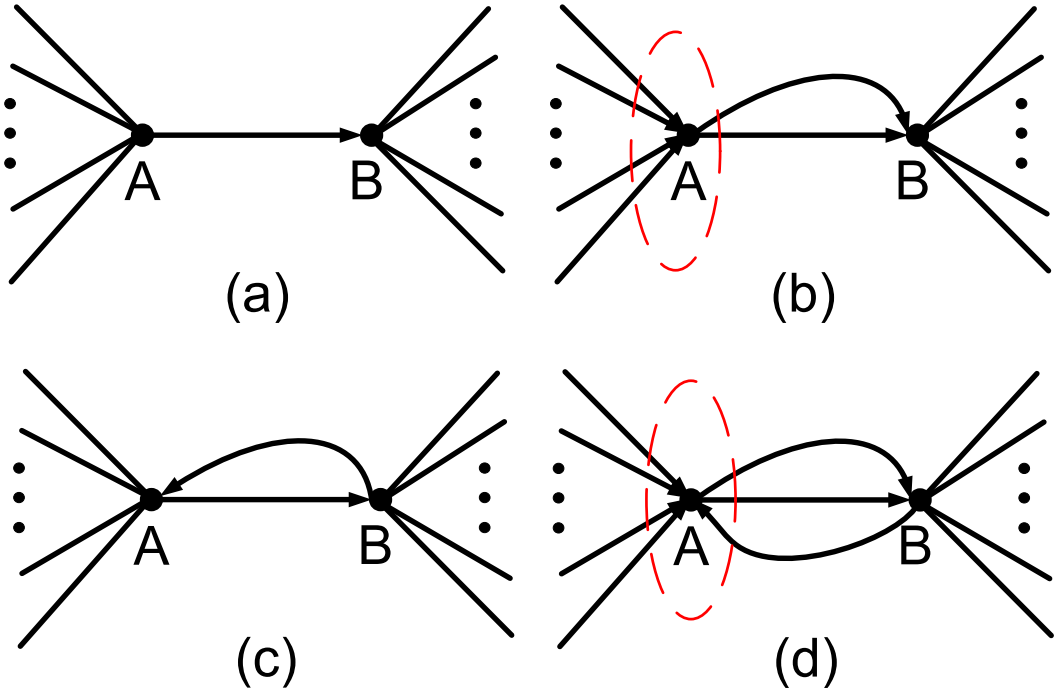

Figure 1: Testing an independent edge.

Proof. Without loss of generality, we would test the independence of edge AB (Figure 1(a)). According to Lemma 2.6 [2], any triple edge has a pebble covering. So, vertex A can collect the first free pebble. Covering the edge AB with the pebble will direct the edge away from vertex A (Figure 1(a)). We will show the existence of directed edge $\overrightarrow{AB}$ does not affect collecting the second pebble. If the second pebble is collected from vertex B, this is equal to the condition that B collects a free pebble. But, vertex B can collect a pebble with or without the directed edge $\overrightarrow{AB}$. If this pebble is collected through other edges that are directed away from an edge except $\overrightarrow{AB}$, then this action has nothing to do with the edge $\overrightarrow{AB}$. Thus, we conclude that A will collect the second free pebble with or without the directed edge $\overrightarrow{AB}$.

When the second pebble is collected by vertex A and is used to cover the double edge, the condition is like Figure 1(b). Note that there are only two edges directed away from A and point to B. Other edges which are associated with A will point to A. We will show two edges which directed away from A do not affect collecting the third pebble. In this condition, vertex A will never collect more pebbles from the vertices except for B. All the edges directed away from A are pointing to B. If vertex B collects the third pebble, it must not come from A because vertex A has no free pebbles. So, when B collects the third pebble, it has nothing to do with the existence of double $\overrightarrow{AB}$.

If a pebble from vertex B covers the double edge, the condition is similar to Figure 1(c). In this condition, freeing a pebble from A to B will lead to the condition of Figure 1(b). However, if the third pebble is collected through other paths, the collection has nothing to do with directed edges between A and B.

Two pebbles from vertex A and one from vertex B have been used to cover tripled edges in the last condition (Figure 1(d)). The fourth pebble can only be collected by vertex B and will not from vertex A. The existed three directed edges between A and B do not affect vertex B collecting pebbles.

Since the directed edges between vertex A and vertex B will not affect collecting more pebbles,

we can pin pebbles on the vertices rather than covering the first, double, and triple edges.

Q.E.D.

2. Examples and How-To

The toolbox is based on the build-in graph operation of MATLAB, so it should be compatible with version 7.6 and later. The user may add toolbox files to MATLAB's search path with m-file 'SetPath.' There are examples in the toolbox to play with; then, users get out to build personal stage of pebble game.

2.1. Building the Pebble Game Play Stage

The interface of **PebbleGamePlayStage** is the same as the function **graph**, which is a built-in MATLAB class. The first step is to create a table that contains a variable **EndNodes**, such as:

```
T1 = [1 2; 1 4; 1 6; 2 3; 2 4; 2 5; 2 6; 3 4; 3 6; 4 5; 4 6; 5 6];
T2 = [3 7; 3 9; 3 11; 7 8; 7 9; 7 10; 7 11; 8 9; 8 11; 9 10; 9 11;
10 11];
T3 = [5 12; 5 14; 5 16; 12 13; 12 14; 12 15; 12 16; 13 14; 13 16;
14 15; 14 16; 15 16];
T = [T1;T2;T3];
EdgeTable = table(T,'VariableNames',{'EndNodes'});
```

The table is used as the parameter to instantiate **PebbleGamePlayStage**.

```
PebbleGame = PebbleGamePlayStage(EdgeTable);
```

Then, store two associated vertices of an edge into different matrices.

```
s = [1 1 1 2 2 2 2 3 3 4 4 5];
t = [2 4 6 3 4 5 6 4 6 5 6 6];
PebbleGame = PebbleGamePlayStage(s,t);
```

When the instantiation **PebbleGame** is created, the class **UserInPut_Group** is instantiated automatically. Its handle is copied to **PebbleGame.UserInPut**. This class shows the user's input and provides data for instantiating **Operation_Group**. The latter, whose handle is **PebbleGame.Operation**, identifies independent edges and explores rigid clusters. The MATLAB will pop up two figures, such as Figure 2 and Figure 3. One shows the user's input, the other displays the process of the pebble game.

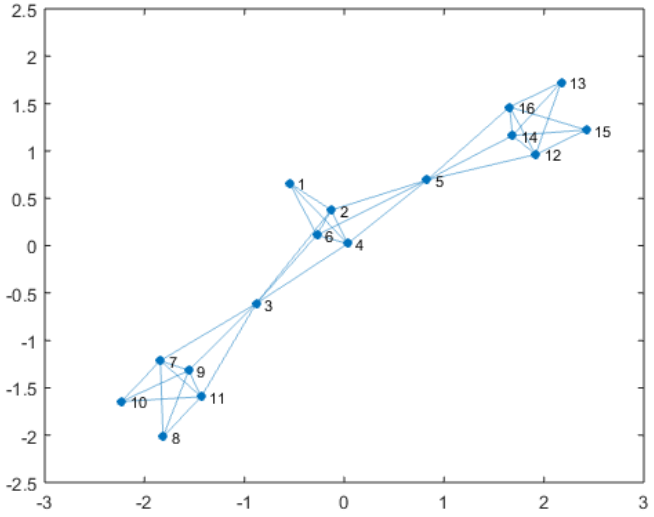

Figure 2: An example of user's input graph. It is used to confirm the input or be treated as a reference during the pebble game.

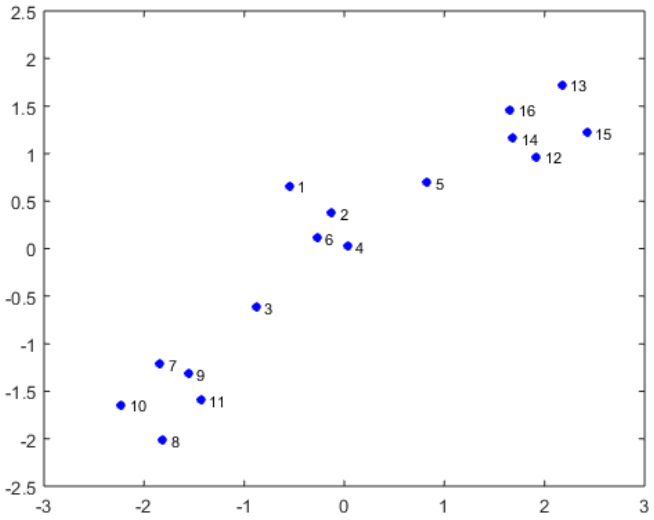

Figure 3: An Example of the initial status of a pebble game. No edge has been added to the graph yet. The vertices are painted blue, which means each vertex has two free pebbles.

2.2. Playing Pebble Games

The toolbox contains multiple functions to customize pebble games. However, if the user prefers identifying independent edges with the classic pebble game [2], only one function should be enough.

```
for i = 1:1:(12*3)
    PebbleGame.Operation.IndependentEdge(1)
    drawnow();
end
```

Each time the function **IndependentEdge** is called, it picks an edge from **PebbleGame.Operation.EdgeReadyForAdd** and try to find a free pebble to cover it. If the edge is independent, the function adds it to the graph; otherwise, the function adds it to **PebbleGame.Operation.EdgeUnableAdd**. Eventually, the graph will be like Figure 4.

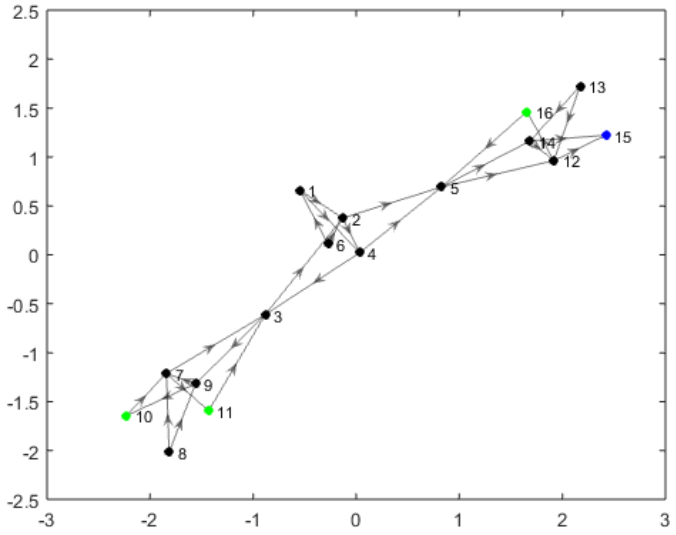

Figure 4: Pebbles covering all the independent edges. Their edges are directed away from the vertices that offer the pebbles. Most of the vertices use up their free pebbles and change color to black. Three greens remain one pebble. Redundant edges are missing because they play no roles in pebble covering.

2.3. Identifying Rigid Clusters and Coloring Them

Before exploring rigid clusters, the user executes function **StartIdentifyRigidCluster** to get the toolbox ready.

```
PebbleGame.Operation.StartIdentifyRigidCluster();
```

The code below explores rigid clusters as many as there are.

```
while(PebbleGame.Operation.IdentifyARigidCluster() == false)
  PebbleGame.Operation.ShowRigidCluster();
  drawnow();
end
```

As shown in Figure 5, the function **ShowRigidCluster** can color the rigid clusters. Use **drawnow** wisely to show a dynamic figure.

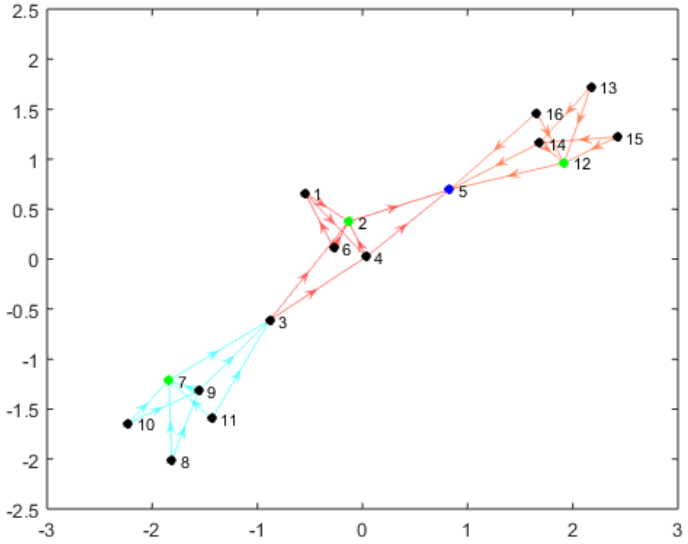

Figure 5: Rigid clusters are identified and colored.

3. Using Objects

3.1. PebbleGamePlayStage

obj = PebbleGamePlayStage(varargin)

3.1.1. Description

本工具箱不会在 MATLAB 的工作区内产生任何附加变量，用户通过操纵 PebbleGamePlayStage 实例的属性和方法完成所有操作。

初始化 PebbleGamePlayStage 与 MATLAB 内置方法 graph 完全相同，请参阅 MATLAB 相关文档。

3.1.2. Properties

a) UserInPut
b) Operation
c) Result

3.1.3. Object Functions

3.2. UserInPut_Group

obj = UserInPut_Group(varargin)

3.2.1. Description

用于承载和显示用户输入，并将相关的信息转换为内部其他对象可以接受的形式。

如仅利用本工具箱判断某图的 generally rigidity 属性，则没有必要专门创建此对象。它会被 PebbleGamePlayStage 创建，并将句柄放在 UserInPut 属性上。请见 **Examples and How To** 一章

3.2.2. Properties

a) Graph
b) GraphPlot

3.2.3. Object Functions

a) Show

3.3. Operation_Group

obj = Operation_Group(names,xdata,ydata,edgereadyforadd)

3.3.1. Description

完成 Pebble Game 并显示实时过程。萃取结果并显示。

如仅利用本工具箱判断某图的 generally rigidity 属性，则没有必要专门创建此对象。它会被 PebbleGamePlayStage 创建，并将句柄放在 Operation 属性上。请见 **Examples and How To** 一章。

3.3.2. Properties

a) Graph
b) GraphPlot
c) GraphFigure
d) XData
e) YData
f) EdgeReadyForAdd
g) EdgeUnableAdd
h) WorkingEdge
i) LamanNodesList
j) LamanEdgesList
k) EdgeTempletTable
l) RigidClusterEdgeColor
m) UnCoverCoverInfoBridge
n) UnCoverCoverInfo

3.3.3. Object Functions

a) [] = IDIntheRange(NodeID)
b) [] = EdgeNodeMatch(NodeID,[VertexA, VertexB])
c) [isCovered] = EdgeisCovered([VertexA, VertexB])
d) [FreePebbleNumber] = ReadFreePebbleNumber(NodeID)
e) [] = WriteFreePebbleNumber_(NodeID, FreePebbleNumber)
f) [Pin Number] = ReadPin(NodeID)
g) [] = WritePin_(NodeID, Pin Number)
h) [] = PinPebble(NodeID, NumbertoPin)
i) [] = UnPinPebble(NodeID, NumbertoUnPin)
j) [] = AutoUnPinAPebble()
k) [TotalFreePebbleNumber] = HowManyFreePebble()
l) [TotalPinNumber] = HowManyPin()
m) [] = CovertheEdge_([VertexA, VertexB])
n) [] = UnCovertheEdge_([VertexA, VertexB])
o) [] = TryToAddaEdge(EdgeReadyForAddNumber, ‘database’)
p) [] = CoverWorkingEdge()
q) [] = UnabletoCoverWorkingEdge()

r) [PathtoFreePebble, BreadthFirstSearch] = FindAPebble(nodeID)
s) [] = RearrangePebble(PathtoFreePebble)
t) [Independent] = IndependentEdge(EdgeReadyForAddNumber)
u) [] = StartIdentifyRigidCluster()
v) [ClusterNumber] = ReadCluster(Edge)
w) [ClusterNumber] = ReadColour(Edge)
x) [] = WriteAllCluster_(NodeIDs, ClusterNumber)
y) [] = WriteAllColour_(NodeIDs, ColourInformation)
z) [Neighbours] = AllNeighbour(NodeID)
aa) [] = WriteAllVisited_(NodeIDs, Logical)
bb) [Logical] = IsVisited(NodeID)
cc) [Explore, NodeIDs] = ExploreARigidClusterNode(Edge, ExploreNode)
dd) [IdentifyFinished] = IdentifyARigidCluster()
ee) ShowRigidCluster()
ff) [] = Show()

3.4. ZipNumList

obj = ZipNumList(initialCapacity)

3.4.1. Description

ZipNumList 是一个仅允许承接数组元素的列表。该列表带有压缩重复元素的功能。

3.4.2. Properties

3.4.3. Object Functions

a) appendElement(obj,Arrays,varargin)

b) ZipList(obj,varargin)

3.5. ContrastColor

obj = ContrastColor()

3.5.1. Description

用来产生顺序计数和互补的 RGB 颜色。

3.5.2. Properties

a) Counter
b) ColorRGB
c) Hue
d) Saturation
e) Value
f) HueFactor
g) SaturationFactor

3.5.3. Object Functions

a) [] = Show()
b) [] = Flash()

4. Properties

这一章列出用户在使用本工具箱时需要用到的属性。部分内置属性没有列出，修改它们可能造成未知的问题。

4.1. UserInPut

指向 UserInPut_Group 的 handle。该对象用于接受用户输入并将其转化为工具箱内其他对象可以接受的形式。

4.2. Operation

指向 Operation_Group 的 handle。该对象由工具箱自动生成，完成 Pebble Game 并显示实时过程。萃取结果并显示。

4.3. Result

空对象，备用。

4.4. Graph

UserInPut_Group 的 Graph 是 Matlab 内置无向图对象 graph。用于存储用户输入。

Operation_Group 的 Graph 是 Matlab 内置有向图对象 diraph，用于存储 Pebble Game 经行中的各种参数。

4.5. GraphPlot

UserInPut_Group 和 Operation_Group 的 GraphPlot 均为 Matlab 内置对象 Graph plot。为调用 Matlab 内置方法 plot 所生成的 handle，用于显示控制。

4.6. GraphFigure

Operation_Group 的 GraphPlot 为 Matlab 内置对象 figure。为控制绘图窗口的 handle。

4.7. XData

Operation_Group 的 XData 为 UserInPut_Group 中 GraphPlot 的横坐标信息。以保证 UserInPut_Group 和 Operation_Group 可以得到相同的绘图。

4.8. YData

Operation_Group 的 YData 为 UserInPut_Group 中 GraphPlot 的纵坐标信息。以保证 UserInPut_Group 和 Operation_Group 可以得到相同的绘图。

4.9. EdgeReadyForAdd

Operation_Group 的 EdgeReadyForAdd 为还没有被增加到图上的边。来自 UserInPut_Group 的 Graph，并与 Edges 具有相同的表结构。Pebble Game 在工作过程中，增加的边均来自这里。

4.10. EdgeUnableAdd

Operation_Group 的 EdgeUnableAdd 为 Pebble Game 运行中不能被增加到图上的边，及重复边（redundant bound）。与 UserInPut_Group 的 Graph.Edges 具有相同的表结构。

4.11. WorkingEdge

当前正在尝试增加的边，初始化为 0。是二元数组。在使用 **Examples and How To** 一节介绍的方法时，每一次 Pebble 的覆盖均只对 WorkingEdge 进行操作。WorkingEdge 显示为 MATLAB 内置颜色 yellow。

4.12. LamanNodesList

containers.ArrayList(k)。k = (n+1)n/2。拉曼图的节点列表。当不能找到第四个泡泡的时候，搜索的路径经过的那些节点。由于边是逐渐增加上去的，所以这个列表内的很多项都需要合并。handle 类。

4.13. EdgeTempletTable

用于在存储边的时候作为模板，任何操作请不要改写该模板。其结构与 EdgeReadyForAdd 一样，但是一个空表。

4.14. LamanEdgesList

LamanEdgesList = containers.ArrayList(k)。用来保存 LamanEdges 的列表。这个列表后期经过合并，最后得到 LamanSubgraphs 各图包含的边。LamanEdgesList 的元素根据 LamanNodesList 来生成。是 handle 类，并在用到的时候初始化。。

4.15. RigidClusterEdgeColor

指向 ContrastColor 的 handle 着色类。其中含有 RigidCluster 的编号，对应边的颜色。在 StartIdentifyRigidCluster()中首次初始化。

4.16. UnCoverCoverInfoBridge

布尔类型。在改变边的方向过程中，有部分数据需保存。当这个属性为真时，这些数据得以保存。反之这些数据则会丢失。

4.17. UnCoverCoverInfo

与 Operation_Group 的 Graph.Edges 具有相同的表结构，但是只有一行元素。暂存 UnCoverCoverInfoBridge 为真时需要保存的信息。

4.18. Counter

ContrastColor 的 Counter 表示当前正在使用的 RigidCluster 的编号。

4.19. ColorRGB

ContrastColor 的 ColorRGB 表示当前正在使用的 RigidCluster 的 RGB 颜色信息。

5. Tables

本工具箱使用的表数据均存储在 Operation_Group 和 UserInPut_Group 的 Graph 对象中，分别为 Nodes 表和 Edges 表。这两个表也是使用 Matlab 内置的 Graph 对象自动生成的。本工具箱根据需要对其做了定制。表的结构根据需要逐渐复杂起来的，以最大限度的节省资源。在 UserInPut_Group，Operation_Group 里表结构有不同.

5.1. Nodes

5.1.1. Name

UserInPut_Group 中 Graph.Nodes.Name 可以由用户定制。其方法与 Maltab 内置方法 graph 定制节点的名称相同。

Operation_Group 中 Graph.Nodes.Name 为字符 1，2，3 等，顺序取决于 UserInPut_Group 的 Graph 节点顺序。

5.1.2. FreePebble

UserInPut_Group 中 Graph.Nodes.FreePebble 记录了对应结点具有自由 Pebble 的个数。

5.1.3. Pin

UserInPut_Group 中 Graph.Nodes. Pin 记录了对应结点被订住的 Pebble 个数。

5.1.4. Visited

在判断 Rigid Cluster 时，需要记录节点是否被访问过。

False 表示没有访问过。

True 表示访问过，且为 rigid。

2 表示访问过，且为 floppy。

5.2. Edges

5.2.1. EndNodes

UserInPut_Group 中 Graph.Nodes.EndNodes 表示用户输入图的边，其终端节点名称可由用户定制。

Operation_Group 中 Graph.Nodes.EndNodes 表示已经被 Pebble 覆盖边的边的终端节点。

5.2.2. Weight

UserInPut_Group 和 Operation_Group 中的 Graph.Nodes.Weight 表示边的权重。在本工具箱中没有作用。

5.2.3. Cluster

Operation_Group 中的 Graph.Nodes.Cluster 表示这条边所属的 Rigid Cluster 编号。该数据来源于 ContrastColor 的 Counter 属性。

5.2.4. Color

Operation_Group 中的 Graph.Nodes.Color 表示这条边所属 Rigid Cluster 的 RGB 颜色。该数据来源于 ContrastColor 的 ColorRGB 属性。用于在可视化过程中为边染色。

6. Functions

如需利用本工具箱判断某图的 generally rigidity 属性，仅需要利用几个函数即可。详情请见 **Examples and How To** 一章。也可以利用本工具箱学习 Pebble Game 的算法，详细观察 Pebble 的移动，甚至定制出自己的 Pebble Game。此时就需要手动调用这些函数。在函数的设计上，力求可靠和安全。如果不满足某种执行条件，则会出错退出而不至于破坏一般意义上的 Pebble Game 规则，比如某一结点 Free Pebble 的个数多余两个，或者丢失泡泡。但是，仍按有些函数无法做到“绝对安全”，作为提示，这些“危险”的函数名称后面有一个“_”。

6.1. Construction

6.1.1. PebbleGamePlayStage

- Syntax

  obj = PebbleGamePlayStage(varargin)

- Input and Output Arguments

  初始化 PebbleGamePlayStage 与 MATLAB 内置方法 graph 完全相同，请参阅 MATLAB 相关文档。

- Description

  本工具箱不会在 MATLAB 的工作区内产生任何附加变量，用户通过操纵 PebbleGamePlayStage 实例的属性和方法完成所有操作。PebbleGamePlayStage 看成是容纳整个 Pebble Game 的工具集合。

- Examples

  初始化 PebbleGamePlayStage 与 MATLAB 内置方法 graph 完全相同。可以按节点初始化：

```
s = [1 1 1 2 2 2 2 3 3 4 4 5]
t = [2 4 6 3 4 5 6 4 6 5 6 6];
PebbleGame = PebbleGamePlayStage(s,t);
```

或者按照边初始化：

```
T1 = [1 2; 1 10; 2 3; 2 4; 2 10; 3 4; 4 5; 4 6; 5 6; 6 7; 6 8; 7 8 ; 8 9 ; 8 10; 9 10];
T = [T1];
EdgeTable = table(T,'VariableNames',{'EndNodes'});
PebbleGame = PebbleGamePlayStage(EdgeTable);
```

6.1.2. UserInPut_Group

- Syntax

  obj = UserInPut_Group(varargin)

- Input and Output Arguments

  初始化 UserInPut_Group 与 MATLAB 内置方法 graph 完全相同，请参阅 MATLAB 相关文档。

- Description

  如仅利用本工具箱判断某图的 generally rigidity 属性，则没有必要专门创建此对象。它会被 PebbleGamePlayStage 创建，并将句柄放在 UserInPut 属性上。请见 Examples and How To 一章。

- Examples

6.1.3. Operation_Group

- Syntax

  obj = Operation_Group(names,xdata,ydata,edgereadyforadd)

- Input and Output Arguments

  names：节点名称

  xdata,ydata：节点在上一次绘图的位置

  edgereadyforadd：等待增加的边集合，表

- Description

  完成 Pebble Game 并显示实时过程。萃取结果并显示。

  如仅利用本工具箱判断某图的 generally rigidity 属性，则没有必要专门创建此对象。它会被 PebbleGamePlayStage 创建，并将句柄放在 Operation 属性上。请见 Examples and How To 一章。

- Examples

  Operation = Operation_Group(UserInPut.Graph.Nodes, UserInPut.GraphPlot.XData, UserInPut.GraphPlot.YData, UserInPut.Graph.Edges);

6.1.4. ZipNumList

- Syntax
  obj = ZipNumList(initialCapacity)

- Input and Output Arguments
  initialCapacity：列表的容量

- Description
  ZipNumList 是一个仅允许承接数组元素的列表。该列表带有压缩重复元素的功能。

- Examples
  ToVisitNodes = containers.ZipNumList(5);

6.1.5. ContrastColor

- Syntax
  obj = ContrastColor()

- Input and Output Arguments

- Description
  用来产生顺序计数和互补的 RGB 颜色。

- Examples
  RigidClusterEdgeColor = ContrastColor();

6.2. Manipulating Pebble Games

6.2.1. IDIntheRange

- Syntax

  [] = IDIntheRange(NodeID)

- Input and Output Arguments

NodeID：整数

- Description

  判断图上是否包含 NodeID 这个节点。

  a) 如果在范围内,则保持安静。

  b) 如果不在范围内,则立即出错退出。

- Examples

  IDIntheRange(5)

6.2.2. EdgeNodeMatch

- Syntax

[] = EdgeNodeMatch(nodeID,[VertexA, VertexB])

- Input and Output Arguments

  NodeID：整数

- Description

  检查 nodeIDs 是否是 VertexA 或 VertexB 其中之一。如果不是，出错退出。“错误，边和节点号不对应！”

- Examples

  EdgeNodeMatch(5, [4 5])

6.2.3. EdgeisCovered

- Syntax

[isCovered] = EdgeisCovered([VertexA, VertexB])

- Input and Output Arguments

- Description

  检查 Graph 的边[VertexA, VertexB]是否已经被覆盖了。注意边的顺序。

  a) 检查 0 == findedge(Graph, VertexA, VertexB)为真。

  b) 如果为真，说明没有这么一条边，isCovered = false

  c) 如果为假，说明没有这么一条边, isCovered = true

- Examples

  EdgeisCovered([3, 5])

6.2.4. ReadFreePebbleNumber

- Syntax

[FreePebbleNumber] = ReadFreePebbleNumber (NodeID)

- Input and Output Arguments

NodeID：节点编号。

- Description

  表查询，给出 NodeID 节点上的 FreePebble 个数。

- Examples

  PebbleNumber = ReadFreePebbleNumber (5)

6.2.5. WriteFreePebbleNumber_

- Syntax

[] = WriteFreePebbleNumber_ (NodeID, FreePebbleNumber)

- Input and Output Arguments

  NodeID：节点编号。

  FreePebbleNumber：需要写入的 Free Pebble 数量。

- Description

  a) 如果 FreePebbleNumber>2，出错退出。“写入了多余两个泡泡数值”。

  b) 表写入，改写 NodeID 节点上的 FreePebble 个数为 FreePebbleNumber。

- Examples

  WriteFreePebbleNumber_ (5, 1);

6.2.6. ReadPin

- Syntax
  [Pin Number] = ReadPin (NodeID)

- Input and Output Arguments
  NodeID：节点编号。

- Description
  表查询，给出 NodeID 节点上的 Pin 个数。

- Examples
  Pin = ReadPin (5);

6.2.7. WritePin_

- Syntax

[] = WritePin _ (NodeID, PinNumber)

- Input and Output Arguments

  NodeID：节点编号。

  PinNumber：需要写入的 Pin 数量。

- Description

  表写入，改写 NodeID 节点上的 Pin 个数为 Pin Number。

- Examples

  WritePin _ (5, 1)

6.2.8. PinPebble

- Syntax
  [] = PinPebble(NodeID, NumbertoPin)

- Input and Output Arguments
  NodeID：节点编号。
  NumbertoPin：需要别住的 Free Pebble 个数。

- Description
  暂时钉住 NumbertoPin 个泡泡不让它们移动.
  a) IDIntheRange(NodeID)
  b) 如果 ReadFreePebbleNumber (NodeID)< NumbertoPin。出错退出。“操作不能完成，没有那么多 FreePebble 可 Pin”。
  c) 调节 FreePebbleNumber：Temp = ReadFreePebbleNumber (NodeID) – NumbertoPin
  d) WriteFreePebbleNumber_ (NodeID, Temp)
  e) 调节 Pin：Temp = ReadPin (NodeID) + NumbertoPin
  f) WritePin _ (NodeID, Temp)

- Examples
  PinPebble(5, 1);

6.2.9. UnPinPebble

- Syntax

  [] = UnPinPebble(NodeID, NumbertoUnPin)

- Input and Output Arguments

  NodeID：节点编号。

  NumbertoPin：需要释放 Pebble 个数。

- Description

  将 NumbertoUnPin 个泡泡转换为 FreePebble。

  a) IDIntheRange(NodeID)

  b) 如果 ReadPin (NodeID)< NumbertoUnPin，出错退出。“没有那么多 Pin 泡泡可以变为 FreePebble”。

  c) = ReadFreePebbleNumber (NodeID) + NumbertoUnPin

  d) 如果 Temp >2，出错退出。“操作不能完成，会导致当前结点 FreePebble 太多”。

  e) 调节 FreePebbleNumber：

  f) WriteFreePebbleNumber_ (NodeID, Temp)

  g) 调节 Pin：Temp = ReadPin (NodeID) - NumbertoPin

  h) WritePin _ (NodeID, Temp)

- Examples

  UnPinPebble(5, 1);

6.2.10. AutoUnPinAPebble

- Syntax
  [] = AutoUnPinAPebble()
- Input and Output Arguments

- Description
  全图操作。如果有泡泡空位的话，自动从 Pin 的泡泡里面 UnPin 一个出来。
  a) 判断 HowManyPin()<=0。如果为真，出错退出。“全图没有任何被 Pin 住的泡泡。”
  b) 遍历每一个 NodeID。
  c) 如果：ReadPin (NodeID)>0
     1) try
     2) UnPinPebble(NodeID, 1)
     3) catch
     4) 空
     5) End
  d) 判断 HowManyPin()>0。给出提示。“仍然有被 Pin 住的泡泡。”

- Examples
  AutoUnPinAPebble();

6.2.11. HowManyFreePebble

- Syntax
  [TotalFreePebbleNumber] = HowManyFreePebble(obj)

- Input and Output Arguments

- Description
  计算当前全图共有多少个 FreePebble。

- Examples
  Total = HowManyFreePebble()

6.2.12. HowManyPin

- Syntax
  [TotalPinNumber] = HowManyPin(obj)

- Input and Output Arguments

- Description
  计算当前全图共有多少个 Pin。
- Examples
  Total = HowManyPin();

6.2.13. CovertheEdge_

- Syntax

  [] = CovertheEdge_([VertexA, VertexB])

- Input and Output Arguments

- Description

  利用 DonateVertex 上的 FreePebble 覆盖[VertexA, VertexB]。DonateVertex 一定是[VertexA, VertexB]相邻的。

  a) DonateVertex = VertexA
  b) EdgeisCovered([VertexA, VertexB]) == true，以检查当前的边是否被覆盖。如果为真，则出错退出.“这条边已经被覆盖了。”
  c) ReadFreePebbleNumber (DonateVertex)>0,以检查是否有足够的 FreePebble 以供覆盖。如果没有，出错退出。“DonateVertex”上没有 FreePebble 可供覆盖。
  d) Graph = Graph. addedge(DonateVertex, VertexB,1)，因为覆盖边就是增加边。增加的边应该离开 DonateVertex。
  e) Temp = ReadFreePebbleNumber (DonateVertex)
  f) WriteFreePebbleNumber_ (DonateVertex, Temp-1)，改写 DonateVertex 上的 FreePebble 数值。
  g) Show()，以更新 GraphPlot。

- Examples

  CovertheEdge_([3, 5]);

6.2.14. UnCovertheEdge_

- Syntax

  [] = UnCovertheEdge_([VertexA, VertexB])

- Input and Output Arguments

- Description

  解除对边的覆盖，并将泡泡归还给 VertexA。

  a) EdgeisCovered([VertexA, VertexB]) == false。说明这条边没有被覆盖，出错退出。“这条边没有被覆盖。”

  b) ReadFreePebbleNumber (VertexA)>=2。如果为真，说明没有地方释放 FreePebble。出错退出。“Source 节点上的 FreePebble 太多。”

  c) Graph = Graph. rmedge(VertexA, VertexB). 注意 Graph 是一个值类。

  d) Temp = ReadFreePebbleNumber (VertexA)

  e) WriteFreePebbleNumber_ (VertexA, Temp+1)，改写 VertexA 上的 FreePebble 数值。

  f) Show()，以更新 GraphPlot。

- Examples

  UnCovertheEdge_([3, 5]);

6.2.15. TryToAddaEdge

- Syntax

  [] = TryToAddaEdge(EdgeReadyForAddNumber, ‘database’)

- Input and Output Arguments

  EdgeReadyForAddNumber：是 EdgeReadyForAdd 表内某条边的行号。

- Description

  开始尝试将某一条边增加到 Graph 上面去。

  第二参数必须为‘database’，以限制用户误将不来不在 EdgeReadyForAdd 中的边添加到 Graph 中。

  a) 检测 WorkingEdge 是否为空，如果不为空，则出错退出。“上一次操作没有完成。如果这条边不能够被覆盖，建议执行承认失败功能。”
  b) 把这条边拷贝到 WorkingEdge。
  c) 删除 EdgeReadyForAdd 中的对应 EdgeReadyForAddNumber 项。（Table(EdgeReadyForAddNumber,:) = []）
  d) [] = Show() (千万不能操作 Graph)。
  e) 高亮待增加的边。并且使用 highlight(h,[1 3],'NodeColor','y','EdgeColor','y','LineWidth',1.5)将待增加的边变成黄色的。

- Examples

  TryToAddaEdge(1, ‘database’);

6.2.16. CoverWorkingEdge

- Syntax

  [] = CoverWorkingEdge()

- Input and Output Arguments

- Description

利用 WorkingEdge 相关联的两个节点中的任何一个上面的 FreePebble 来覆盖 WorkingEdge。

a) 提取与 WorkingEdge 相关联的 WorkingNodeA 和 WorkingNodeB
b) IDIntheRange(WorkingNodeA)
c) IDIntheRange(WorkingNodeB)
d) 判断 ReadFreePebbleNumber (WorkingNodeA)>0
   CovertheEdge_([WorkingNodeA，WorkingNodeB])
   WorkingNodeA =WorkingNodeB=0. WorkingEdge=0,
   Show()，以更新 GraphPlot。
   然后返回。
e) 判断 ReadFreePebbleNumber (WorkingNodeB)>0
   CovertheEdge_([WorkingNodeB，WorkingNodeA)
   WorkingNodeA =WorkingNodeB=0. WorkingEdge=0,
   Show()，以更新 GraphPlot。
   然后返回。
f) 出错。“WorkingEdge 相关联的节点上没有 FreePebble。尝试为其寻找 FreePebble”

- Examples

  CoverWorkingEdge();

6.2.17. UnabletoCoverWorkingEdge

- Syntax

  [] = UnabletoCoverWorkingEdge()

- Input and Output Arguments

- Description

  如果实在不能够找到 FreePebble 来覆盖 WorkingEdge，那么使用这个函数放弃。

  a) 检测 WorkingEdge ~= 0。如果是空的，则出错退出。“WorkingEdge 是空的。”
  b) EdgeUnableAdd{end+1,:}= WorkingEdge，以便把不成功加入的边保存到 EdgeUnableAdd 里面去。
  c) WorkingEdge = 0；
  d) Show()，以更新 GraphPlot。

- Examples

  UnabletoCoverWorkingEdge();

6.2.18. FindAPebble

- Syntax

  [PathtoFreePebble, BreadthFirstSearch] = FindAPebble(nodeID)

- Input and Output Arguments

  NodeID：节点编号。

  PathtoFreePebble：是从 NodeID 到具有 FreePebble 节点的路径。为一串节点号组成的数组。

  BreadthFirstSearch：为寻找 Free Pebble 的过程中，首先从 nodeID 发起的广度优先算法。

- Description

  从 nodeID 开始，沿着 donate 的方向，发现一个 FreePebble。查找 Free Pebble 的过程以及通向 Free Pebble 的路径，均以 MATLAB 内置颜色 yellow 显示。

  a) IDIntheRange(NodeID)
  b) Search = Graph.bfsearch(nodeID)，广度优先算法，找到所有可达节点。
  c) BreadthFirstSearch = Search. 当找不到 FreePebble 时，需要利用 BreadthFirstSearch 找到 Laman Subgraphs。
  d) GraphPlot.highlight(Search, 'NodeColor','y', 'EdgeColor','y','LineWidth',1.5), 显示广度优先算法信息。
  e) 遍历 Search，找到第一个 ReadFreePebbleNumber (NodeID)>0 的节点 FirstFreePebbleNode。
  f) 如果找到了，则计算最短路径。
  g) Show()，以更新 GraphPlot。
  h) PathtoFreePebble = Graph.shortestpath(nodeID, FirstFreePebbleNode)。然后高亮显示路径。GraphPlot.highlight(PathtoFreePebble, 'NodeColor','y', 'EdgeColor','y','LineWidth',1.5)
  i) 如果没找到，则返回 PathtoFreePebble=0

- Examples

  [PathtoFreePebble, BreadthFirstSearch] = FindAPebble(5);

6.2.19. RearrangePebble

- Syntax

  [] = RearrangePebble(PathtoFreePebble)

- Input and Output Arguments

  PathtoFreePebble：是到具有 FreePebble 节点的路径。为一串节点号组成的数组。

- Description

  按照 PathtoFreePebble 指示的方向，从路径尾节点上开始交换覆盖，从而使得头部得到一个 FreePebble。

  a) 检查 PathtoFreePebble == 0,如果是则出错退出。“无效的路径。”
  b) 判断 PathtoFreePebble 长度是否为 1，如果是，直接返回。因为不需要移动，当前路径只有一个节点。
  c) Show()，以更新 GraphPlot。
  d) 建立 Printer 指向最后一个 PathtoFreePebble 元素。循环执行 3 到 5 直到最后。
  e) UnCovertheEdge_([*(Printer-1), *(Printer)])
  f) CovertheEdge_([*(Printer), *(Printer-1)])
  g) Printer = Printer -1
  h) Show()，以更新 GraphPlot。

- Examples

  RearrangePebble(PathtoFreePebble);

6.2.20. IndependentEdge

- Syntax

  [Independent] = IndependentEdge(EdgeReadyForAddNumber)

- Input and Output Arguments

  EdgeReadyForAddNumber：是 EdgeReadyForAdd 表内某条边的行号。

- Description

  检测 EdgeReadyForAdd 中由 EdgeReadyForAddNumber 指定的边是否是独立边。

1) [] = TryToAddaEdge(EdgeReadyForAddNumber, 'database'). 将需要检查的边放在 WorkingEdge 上面。
2) 提取与 WorkingEdge 相关联的 WorkingNodeA 和 WorkingNodeB 备用。
3) CollectedFreePebbleNumber = 0; 记录这个过程一共收集到了多少个 FreePebble。
4) IDIntheRange(WorkingNodeA)。数值可靠性检查。
5) IDIntheRange(WorkingNodeB)。数值可靠性检查。

   （针对 WorkingNodeA 收集第一个泡泡。注意，即便 WorkingNodeA 本身就有 FreePebble 也是没有关系的。）
6) try 首先尝试在 WorkingNodeA 上收集第一个 FreePebble。除非找不到 FreePebble 才会出错。
   a) [PathtoFreePebble, BreadthFirstSearch] = FindAPebble(WorkingNodeA)
   b) [] = RearrangePebble(PathtoFreePebble)
7) catch ME
8) msg = ['需要详细研究。为什么在节点',num2str(WorkingNodeA),'上一个自由泡泡也收集不到！'];
9) causeException = MException('MATLAB:myCode:IndependentEdge',msg)
10) ME = addCause(ME,causeException);
11) Independent = -1. 第一个泡泡都找不到的条件，还没有研究过。所以是无效值。
12) rethrow(ME) . 出错退出了。
13) end
14) [] = PinPebble(WorkingNodeA, 1)。如果把 WorkingNodeA 上收集的这个泡泡 Pin 住。
15) CollectedFreePebbleNumber = CollectedFreePebbleNumber + 1. 成功收集到了泡泡。

    （针对 WorkingNodeB 收集第一个泡泡。注意，即便 WorkingNodeB 本身就有 FreePebble 也是没有关系的。）
16) try 首先尝试在 WorkingNodeB 上收集第一个 FreePebble。除非找不到 FreePebble 才会出错。
    a) [PathtoFreePebble, BreadthFirstSearch] = FindAPebble(WorkingNodeB)
    b) [] = RearrangePebble(PathtoFreePebble)
17) catch ME
18) msg = ['需要详细研究。为什么在节点',num2str(WorkingNodeB),'上一个自由泡泡也找不到！'];
19) causeException = MException('MATLAB:myCode:IndependentEdge',msg)

20) ME = addCause(ME,causeException);
21) Independent = -2. 第二个泡泡都找不到的条件，还没有研究过。所以是无效值。
22) rethrow(ME)
23) end
24) [] = PinPebble(WorkingNodeB, 1), 如果找到了，把 WorkingNodeB 上收集的第二个泡泡也 Pin 住。
25) CollectedFreePebbleNumber = CollectedFreePebbleNumber + 1 成功收集到了泡泡。

（下面就可能出现找不到 FreePebble 的情况了。）

26) BreadthFirstSearch_A = BreadthFirstSearch_B = 0 .新建两个宽度优先算法待存储变量。

（在 WorkingNodeA 上收集第三个 FreePebble。）

27) FailedA = false 在 A 上第二次收集到泡泡是否失败的标志。
28) try
    a) [PathtoFreePebble, BreadthFirstSearch] = FindAPebble(WorkingNodeA)
    b) [] = RearrangePebble(PathtoFreePebble)
29) catch ME 发现在 WorkingNodeA 上面不能发现第三个泡泡。保存宽度优先算法的结果以便生成 Laman Subgraphs.
30) BreadthFirstSearch_A = BreadthFirstSearch
31) FailedA = true
32) end
33) 判断 PathtoFreePebble ~= 0. 如果是，说明找到了 FreePebble。
34) [] = PinPebble(WorkingNodeA, 1), 把 WorkingNodeA 上收集的第三个泡泡也 Pin 住。
35) CollectedFreePebbleNumber = CollectedFreePebbleNumber + 1
36) FailedA = false
37) end

（在 WorkingNodeB 上收集第四个 FreePebble。）

38) FailedB = false
39) try
    a) [PathtoFreePebble, BreadthFirstSearch] = FindAPebble(WorkingNodeB)
    b) [] = RearrangePebble(PathtoFreePebble)
40) catch ME 发现在 WorkingNodeB 上面不能发现第四个泡泡。保存宽度优先算法的结果以便生成 Laman Subgraphs.
41) BreadthFirstSearch_B = BreadthFirstSearch
42) FailedB = true
43) end
44) 判断 PathtoFreePebble ~= 0. 如果是，说明找到了 FreePebble。
45) [] = PinPebble(WorkingNodeB, 1), 把 WorkingNodeB 上收集的第四个泡泡也 Pin 住。
46) CollectedFreePebbleNumber = CollectedFreePebbleNumber + 1
47) FailedB = false

48) end

49) CollectedFreePebbleNumber<3。判断是否只找到了两个 FreePebble
50) 出错退出。“同时找不到第三个和第四个泡泡，需要自己研究。”

51) 判断 CollectedFreePebbleNumber==4。如果是，则说明找到了第四个泡泡。
52) [] = UnPinPebble(WorkingNodeA, 1) 前面把 4 个泡泡都 Pin 住了，现在释放 WorkingNodeA 上面的一个。
53) [] = CoverWorkingEdge()
54) [] = UnPinPebble(WorkingNodeA, 1)
55) [] = UnPinPebble(WorkingNodeB, 2)
56) Independent = true;
57) Show()，以更新 GraphPlot。
58) return

59) 判断 CollectedFreePebbleNumber~=3。如果是，出错退出。“找到了 5 个甚至以上的 FreePebble，这是不可能的。”

60) 判断 FailedB == true. 如果是，则说明在 WorkingNodeB 上发生了找泡泡失败。
61) [] = UnabletoCoverWorkingEdge() 承认这条边不是独立边，无法覆盖。
62) [] = UnPinPebble(WorkingNodeA, 2)
63) [] = UnPinPebble(WorkingNodeB, 1)
64) LamanSubgraphs.appendElement(BreadthFirstSearch_B). 保存 Laman Subgraphs。
65) Independent = false;
66) Show()，以更新 GraphPlot。
67) return

68) 判断 FailedA == true. 如果是，则说明在 WorkingNodeA 上发生了找泡泡失败。
69) [] = UnabletoCoverWorkingEdge(),承认这条边不是独立边，无法覆盖。
70) [] = UnPinPebble(WorkingNodeA, 1)
71) [] = UnPinPebble(WorkingNodeB, 2)
72) LamanSubgraphs.appendElement(BreadthFirstSearch_A). 保存 Laman Subgraphs。
73) Independent = false;
74) Show()，以更新 GraphPlot。
75) return

- Examples
  IndependentEdge(1);

6.3. Exploring Rigid Clusters

6.3.1. StartIdentifyRigidCluster

- Syntax

  [] = StartIdentifyRigidCluster()

- Input and Output Arguments

- Description

  启动 RigidCluster 识别工作，主要为识别做数值准备。
  初始化 RigidCluster 标识和颜色类。
  为 Graph. Edges 表增加 Cluster 列。
  为 Graph. Edges 表增加 Color 列。
  为 Graph. Nodes 表增加 Visited 列。

  a) 创建 handle 着色类。obj.RigidClusterEdgeColor = color.ContrastColor(). 其中含有 RigidCluster 的编号，对应边的颜色。
  b) n= height (Graph.Edges).测出边的个数。
  c) Graph.Edges.Cluster = ones(n,1)*(-1)。为 Graph. Edges 表准备好 Cluster 列。
     初始化为-1 标示无效。
     不应该出现 0 值。
     其他整数为对应的 Cluster 归属。
  d) Graph.Edges.Color = zeros(n,3)。为 Graph. Edges 表准备好 Color 列用于保存颜色。

  e) n= height (Graph.Nodes).测出节点的个数。
  f) Graph.Nodes.Visited = zeros(n,1)
     (如果这里用 false(n,1)，以后这个表就不能写入其他非逻辑数值)
     False（0）表示还没有被访问过。
     True（1）表示已经被访问过，且为 rigid.
     2(double)表示已经被访问过，且为 floppy.

- Examples

  StartIdentifyRigidCluster();

6.3.2. ReadCluster

- Syntax
  [ClusterNumber] = ReadCluster(Edge)

- Input and Output Arguments
  Edge 一定是数值性的横向数组。

- Description
  表读取，读取边的 RigidCluster 归属信息。

  1) 提取 Edge 的两个点 WorkingNodeA，WorkingNodeB。
  2) IDIntheRange(WorkingNodeA)。数值可靠性检查。
  3) IDIntheRange(WorkingNodeB)。数值可靠性检查。
  4) edgeID = Graph.findedge(WorkingNodeA, WorkingNodeB)
  5) 判断 edgeID==0。如果是，则出错退出。“没有这么一条边。”
  6) ClusterNumber = Graph.Edges.Cluster(edgeID)

- Examples
  [ClusterNumber] = ReadCluster([3 5]);

6.3.3. ReadColour

- Syntax
  [ColourInformation] = ReadColour(Edge)

- Input and Output Arguments
  Edge 一定是数值性的横向数组

- Description
  表读取，读取边的 RigidCluster 相关联的颜色信息。

  1) 提取 edge 的两个点 WorkingNodeA，WorkingNodeB。
  2) IDIntheRange(WorkingNodeA)。数值可靠性检查。
  3) IDIntheRange(WorkingNodeB)。数值可靠性检查。
  4) edgeID = Graph.findedge(WorkingNodeA, WorkingNodeB)
  5) 判断 edgeID==0。如果是，则出错退出。“没有这么一条边。”
  6) ColourInformation = Graph.Edges.Color (edgeID,:)

- Examples
  Colour = ReadColour([3 5]);

6.3.4. WriteAllCluster_

- Syntax

[] = WriteAllCluster_(NodeIDs, ClusterNumber)

- Input and Output Arguments

  NodeIDs: 一串节点号，代表一个 rigid cluster。

  ClusterNumber：要写入的 rigid cluster 编号。

- Description

将 NodeIDs 中所包含的所有边的 Cluster 均设定为 ClusterNumber。用于处理同时需要设定很多值的情况。

1) AEdge = Graph.Edges.EndNodes(i,:)。遍历 EndNodes，设为 AEdge（不要对表使用类似 for each 的语句）
   a) WorkingNodeA= AEdge(1,1); WorkingNodeB= AEdge(1,2);提取 Edge 的两个点。当前数据为 cell。
   b) edgeID = Graph.findedge(WorkingNodeA, WorkingNodeB)。找到 Edge 的 ID 号。
   c) [WorkingNodeA, WorkingNodeB] = Graph.findedge(edgeID)。将 cell 结构的节点号转换为数字类型的节点号。

   a) 如果 isequal(intersect(NodeIDs, WorkingNodeA),WorkingNodeA)，
      同时 isequal(intersect(NodeIDs, WorkingNodeB),WorkingNodeB)，
      说明 Edge 这条边在 NodeIDs 中。不要用==判断，因为 intersect 会返回空矩阵。
   d) 则 Graph.Edges.Cluster(edgeID) = ClusterNumber。

- Examples

  WriteAllCluster_([2 4 6 8 10], 2);

6.3.5. WriteAllColour_

- Syntax

  []= WriteAllColour_(NodeIDs, ColourInformation)

- Input and Output Arguments

  NodeIDs: 一串节点号，代表一个 rigid cluster。

  ClusterNumber：要写入的 rigid cluster 编号。

- Description

将 NodeIDs 中所包含的所有边的 Graph.Edges.Color()均设定为 ColourInformation。用于处理同时需要设定很多值的情况。

1) AEdge = PebbleGame.Operation.Graph.Edges.EndNodes(i,:)。遍历 EndNodes，设为 AEdge（不要对表使用类似 for each 的语句）
   a) WorkingNodeA= AEdge(1,1); WorkingNodeB= AEdge(1,2);提取 Edge 的两个点。当前数据为 cell。
   b) edgeID = Graph.findedge(WorkingNodeA, WorkingNodeB)。找到 Edge 的 ID 号。
   c) [WorkingNodeA, WorkingNodeB] = Graph.findedge(edgeID)。将 cell 结构的节点号转换为数字类型的节点号。
   d) 如果 isequal(intersect(NodeIDs, WorkingNodeA),WorkingNodeA)，
      同时 isequal(intersect(NodeIDs, WorkingNodeB),WorkingNodeB)，
      说明 Edge 这条边在 NodeIDs 中。不要用==判断，因为 intersect 会返回空矩阵。
   a) Graph.Edges.Colour (edgeID,:) = ColourInformation。

- Examples

  WriteAllColour_([3 4 6 8 10], [1 0 0]);

6.3.6. AllNeighbour

- Syntax

  [Neighbours] = AllNeighbour(NodeID)

- Input and Output Arguments

  NodeID：节点编号。

  Neighbours：NodeID 的前向和后向邻居集合。

- Description

  返回 NodeID 的所有邻居。包括前向邻居和后向邻居。

  注意这个是有向图，需要找到 preIDs = predecessors(G, NodeID)，sucIDs = successors(G, NodeID)。然后合并在一起。

  1) P = Graph.predecessors(NodeID)
  2) S = Graph.successors(NodeID)
  3) Neighbours = [P ; S]

- Examples

  [Neighbours] = AllNeighbour(3);

6.3.7. WriteAllVisited_

- Syntax

  [] = WriteAllVisited_(NodeIDs, Logical)

- Input and Output Arguments

  NodeIDs: 一串节点号。

  Logical：要写入至表 Graph.Nodes. Visited 的数值。

- Description

  遍历 NodeIDs 中的所有节点，将每一个节点的 Visited 均设定为 Logical。用于处理同时需要设定很多值的情况。

  False（0）表示还没有被访问过。

  True（1）表示已经被访问过，且为 rigid.

  2(double)表示已经被访问过，且为 floppy.

  1) 遍历 NodeIDs，设为 NodeID。注意使用 NodeIDs(i)语法。
  2) Graph.Nodes. Visited = Logical

- Examples

  WriteAllVisited_([2 4 6 8 10], true);

6.3.8. IsVisited

- Syntax
  [Logical] = IsVisited(NodeID)

- Input and Output Arguments
  NodeID：节点编号。

- Description
  返回节点 NodeID 的 Visited 状态。
  1) Logical = Graph.Nodes. Visited(NodeID)

- Examples
  Logical = Graph.Nodes. Visited(3);

6.3.9. ExploreARigidClusterNode

- Syntax

  [Explore, NodeIDs] = ExploreARigidClusterNode(Edge, ExploreNode)

- Input and Output Arguments

  Edge:已经被归属为某个 rigid cluster 的边。

  ExploreNode：需要判断与 rigid cluster 关系的节点号。

  Explore：判断的结果是否导致原有的 rigid cluster 需要扩充。

  NodeIDs：需要扩充原有 rigid cluster 的节点号列表。

- Description

  判断 ExploreNode 相对于 edge 的性质，**暂时不管边的事情**。

  Explore=true，发现了 Rigid 节点，则 NodeIDs 里面包含这些节点，也包含 ExploreNode。

  Explore=false，发现了 floppy 节点，则 NodeIDs 里面包含这些节点，也包含 ExploreNode。

  edge 是已经被标记过属于现存的某一个 RigidCluster 的边。

1) 提取 edge 的两个点 WorkingNodeA，WorkingNodeB。

   (输入有效性检查)
2) IDIntheRange(WorkingNodeA)。数值可靠性检查。
3) IDIntheRange(WorkingNodeB)。数值可靠性检查。
4) IDIntheRange(ExploreNode)。数值可靠性检查。
5) EdgeisCovered([WorkingNodeA, WorkingNodeB]) == false。出错退出。“输入的 edge 在图上并不存在。”

   (记录这个过程一共收集到了多少个 FreePebble。)
   CollectedFreePebbleNumber = 0;

   （开始第一和第二泡泡收集过程）
   （针对 WorkingNodeA 收集第一个泡泡。注意，即便 WorkingNodeA 本身就有 FreePebble 也是没有关系的。）
6) try 首先尝试在 WorkingNodeA 上收集第一个 FreePebble。除非找不到 FreePebble 才会出错。
   a) [PathtoFreePebble, BreadthFirstSearch] = FindAPebble(WorkingNodeA)
   b) [] = RearrangePebble(PathtoFreePebble)
7) catch ME
8) msg = ['需要详细研究。为什么在节点',num2str(WorkingNodeA),'上一个自由泡泡也收集不到！'];
9) causeException = MException('MATLAB:myCode:IndependentEdge',msg)

10) ME = addCause(ME,causeException);
11) Explore = -1. 第一个泡泡都找不到的条件，还没有研究过。所以是无效值。
12) rethrow(ME) . 出错退出了。
13) end
14) [] = PinPebble(WorkingNodeA, 1)。如果能收集到，把 WorkingNodeA 上收集的这个泡泡 Pin 住。
15) CollectedFreePebbleNumber = CollectedFreePebbleNumber+1；

（针对 WorkingNodeB 收集第一个泡泡。注意，即便 ExploreNode 本身就有 FreePebble 也是没有关系的。）
16) try 首先尝试在 WorkingNodeB 上收集第一个 FreePebble。除非找不到 FreePebble 才会出错。
    a) [PathtoFreePebble, BreadthFirstSearch] = FindAPebble(WorkingNodeB)
    b) [] = RearrangePebble(PathtoFreePebble)
17) catch ME
18) msg = ['需要详细研究。为什么在节点',num2str(WorkingNodeB),'上一个自由泡泡也找不到！'];
19) causeException = MException('MATLAB:myCode:IndependentEdge',msg)
20) ME = addCause(ME,causeException);
21) Explore = -2. 第二个泡泡都找不到的条件，还没有研究过。所以是无效值。
22) rethrow(ME)
23) end
24) [] = PinPebble(WorkingNodeB, 1), 如果找到了，把 WorkingNodeB 上收集的第二个泡泡也 Pin 住。
25) CollectedFreePebbleNumber = CollectedFreePebbleNumber+1；

（下面就可能出现找不到 FreePebble 的情况了。）
（在 WorkingNodeA 上第二次收集 FreePebble。）
26) FailedA = false 在 WorkingNodeA 上第二次收集到泡泡是否失败的标志。
27) try
    a) [PathtoFreePebble, BreadthFirstSearch] = FindAPebble(WorkingNodeA)
    b) [] = RearrangePebble(PathtoFreePebble)
28) catch ME 发现在 WorkingNodeA 上面不能发现泡泡。
29) FailedA = true
30) end
31) 判断 FailedA==false. 如果是，说明找到了 FreePebble。
32) [] = PinPebble(WorkingNodeA, 1), 把 WorkingNodeA 上收集的第三个泡泡也 Pin 住。
33) CollectedFreePebbleNumber = CollectedFreePebbleNumber+1；
34) end

（在 WorkingNodeB 上第二次收集 FreePebble。）
35) FailedB = false 在 WorkingNodeB 上第二次收集到泡泡是否失败的标志。
36) try
    a) [PathtoFreePebble, BreadthFirstSearch] = FindAPebble(WorkingNodeB)

b) [] = RearrangePebble(PathtoFreePebble)

37) catch ME 发现在 WorkingNodeB 上面不能发现泡泡。
38) FailedB = true
39) end
40) 判断 FailedB==false. 如果是，说明找到了 FreePebble。
41) [] = PinPebble(WorkingNodeB, 1), 把 WorkingNodeB 上收集的泡泡也 Pin 住。
42) CollectedFreePebbleNumber = CollectedFreePebbleNumber+1；
43) end

44) 判断 CollectedFreePebbleNumber ~= 3。如果为真出错退出。
45) Explore = -3
46) “收集到了 num2str(CollectedFreePebbleNumber)个泡泡，真是奇怪。”

(下面开始在 ExploreNode 上找第四个 FreePebble)
FailedExploreNode = false

47) try
c) [PathtoFreePebble, BreadthFirstSearch] = FindAPebble(ExploreNode)
d) [] = RearrangePebble(PathtoFreePebble)
48) catch ME 发现在 ExploreNode 上面不能发现泡泡。
49) FailedExploreNode = true
50) end

（下面开始判断并输出结果）

51) 判断 FailedExploreNode == true
52) NodeIDs = BreadthFirstSearch 失败的时候是返回广度优先算法的结果
53) Explore = true
54) WriteAllVisited_(NodeIDs, true(1,1))
55) WriteAllVisited_(ExploreNode, true(1,1))

56) 判断 FailedExploreNode == false
57) NodeIDs = PathtoFreePebble 成功的时候是返回泡泡移动的过程。
58) Explore = false
59) WriteAllVisited_(NodeIDs, 2)
60) WriteAllVisited_(ExploreNode, 2)

(释放 Pin 住的泡泡)

61) 判断 FailedA == true
62) UnPinPebble(WorkingNodeA, 1)
63) UnPinPebble(WorkingNodeB, 2)

64) 判断 FailedB == true
65) UnPinPebble(WorkingNodeA, 2)
66) UnPinPebble(WorkingNodeB, 1)

- Examples
  [Explore, NodeIDs] = ExploreARigidClusterNode([3 5], 7);

6.3.10. IdentifyARigidCluster

- Syntax

  [IdentifyFinished] = IdentifyARigidCluster()

- Input and Output Arguments

  IdentifyFinished：是否还可能存在更多的 Rigid Cluster 需要识别。

- Description

  完全识别出一个 RigidCluster。这个函数运行的前提是认为前面一个标记的 RigidCluster 完全探索完毕了。如果发现所有的边均被标记了，就会返回 true，否则返回 false.

1) n = height(Graph.Nodes)。得到节点个数。
2) ToVisitNodes = containers.ZipNumList(n). 生成待分析节点列表。
3) WriteAllVisited_([1:1:n], false)。标记所有的节点均为没有访问过。
4) e= height(Graph.Edges)。得到边数。
5) IdentifyFinished = false. 生成 RigidCluster
6) 遍历 Graph.Edges.Cluster(i).
   a) 判断。Graph.Edges.Cluster == -1
   b) 如果找到了，就 break. 得到 WorkingEdgesID = i。
   c) IdentifyFinished = false
   d) 否则的话，IdentifyFinished = true
7) 判断。IdentifyFinished == true. 如果是，return。所有的边都判断完毕，没有必要继续运行。
8) Graph.Edges.
9) [WorkingNodeA, WorkingNodeB]=PebbleGame.Operation.Graph.findedge(WorkingEdgesID). 提取与 WorkingEdge 相关联的 WorkingNodeA 和 WorkingNodeB。

   (为这条边标记 RigidCluster 数据)
10) RigidClusterEdgeColor.Flash()。生成一个新的编号和颜色。
11) WriteAllCluster_([WorkingNodeA ; WorkingNodeB], RigidClusterEdgeColor.Counter)
12) WriteAllColour_([WorkingNodeA ; WorkingNodeB], RigidClusterEdgeColor.ColorRGB)
13) WriteAllVisited_([WorkingNodeA ; WorkingNodeB],true);

   (提取 WorkingNodeA 邻居)
14) Neighbours WorkingNodeA = AllNeighbour(WorkingNodeA)
15) 遍历 Neighbours WorkingNodeA(i)。
   a) 判断。IsVisited(NodeID)==false
   b) 如果是。ToVisitNodes.appendElement(Neighbours WorkingNodeA(i))

   (提取 WorkingNodeB 邻居)
16) Neighbours WorkingNodeB = AllNeighbour(WorkingNodeB)
17) 遍历 Neighbours WorkingNodeB(i)。

a) 判断。IsVisited(NodeID)==false
b) 如果是。ToVisitNodes.appendElement(Neighbours WorkingNodeB(i))

(精简化 ToVisitNodes)

18) ToVisitNodes.ZipList()

(判断是否根本就没有邻居)

19) 判断 ToVisitNodes. Count == 0
20) 如果是。IdentifyFinished = true. return.

(下面开始做一个超级迭代)

21) While(ToVisitNodes. Count ~=0)
    a) ExploreNode = ToVisitNodes.removeLast(). 首先弹出来一个。
    b) [Explore, NodeIDs] = ExploreARigidClusterNode([WorkingNodeA WorkingNodeB], ExploreNode)。
    c) 判断 Explore==true
        i. 遍历 NodeIDs(i), ANode
        ii. NeighboursANode = AllNeighbour(NodeIDs(i))
            1. 遍历 NeighboursANode(i)
            2. 判断。IsVisited(ANeighboursANode)==false
            3. 如果是。ToVisitNodes.appendElement(ANeighboursANode (i))
    d) ToVisitNodes. ZipList()。化简 ToVisitNodes

(最后一步，开始标记)

22) 遍历 IsVisited(ANeighboursANode)==true 的节点，生成节点数组 RigidNodes。
23) [] = WriteAllCluster_(RigidNodes, RigidClusterEdgeColor.Counter)
24) []= WriteAllColour_(RigidNodes, RigidClusterEdgeColor.ColorRGB)

- Examples
  [IdentifyFinished] = IdentifyARigidCluster();

6.4. Visualization

6.4.1. ShowRigidCluster

- Syntax
  ShowRigidCluster(obj)

- Input and Output Arguments

- Description
  显示一张 Rigid Cluster 的图，图上已经被识别的 Rigid Cluster 带有不同的颜色。
- Examples
  ShowRigidCluster();

6.4.2. Show

- Syntax
  [] = Show()

- Input and Output Arguments

- Description
  生成新的生成新的 GraphPlot 对象。
  节点根据拥有的 Free Pebble 个数可以改变颜色。采用 Matlab 内置的三种颜色：
  blue 有 2 个 Free 泡泡。
  green 有 1 个 Free 泡泡。
  black 没有 Free 泡泡。
  red 表示某一个节点自由泡泡多余两个。
  a) GraphPlot = Graph.plot('XData',obj.XData,'YData',obj.YData,' EdgeColor', 'k');
  b) 遍历每一个节点
  c) 表查询 ReadFreePebbleNumber (NodeID)==2，则高亮这个节点 highlight(GraphPlot,nodeIDs, 'NodeColor','b')
  d) 表查询 ReadFreePebbleNumber (NodeID)==1，则高亮这个节点 highlight(GraphPlot,nodeIDs, 'NodeColor','g')
  e) 表查询 ReadFreePebbleNumber (NodeID)==0，则高亮这个节点 highlight(GraphPlot,nodeIDs, 'NodeColor','k')
  f) 表查询 ReadFreePebbleNumber (NodeID)>2，出错退出! “出现了拥有过多泡泡的节点”
  g) 表查询 ReadFreePebbleNumber (NodeID)<0，出错退出 !“部分节点出现了负值泡泡 !”

- Examples
  Show();

6.4.3. Flash

- Syntax

  Flash(obj)

- Input and Output Arguments

- Description

  执行这个方法后，ContrastColor 内的 Counter 属性加 1。ColorRGB 属性上得到新的一种颜色的 RGB 数值。

- Examples

  Flash();

6.5. Containers

6.5.1. appendElement

- Syntax

  appendElement(obj,Arrays,varargin)

- Input and Output Arguments

  Arrays: 任意维度的数组。

  varargin：为空时，将 Arrays 中的元素逐个加入列表。

  为'rows'时，将 Arrays 中的元素按行加入列表。

- Description

  将 Arrays 中的元素逐个加入 ZipList。

- Examples

  appendElement([1 2 3 4 5]);

  appendElement([1 2 3 4 5 ; 5 6 7 8 9 10], 'rows');

6.5.2. ZipList

- Syntax

  ZipList(obj,varargin)

- Input and Output Arguments

  varargin：为空时，查找 ZipList 中的重复元素并将其剔除。

  为'rows'时，按行查找 ZipList 中的重复元素并将其剔除。

- Description

- Examples

  ZipList();

  ZipList('rows');

7. Acknowledgments

- The toolbox uses a class **ArrayList** from an excellent MATLAB-based robot software Sim.I.am [3].
- The suggestion of professor Jacobs, which is to build an easy-to-use interface, gave us confidence about the work.
- Professor Hongxia Wang gave attention and encouragement during the software development.
- Professor Li Yu provided financial support when the software was registered.
- This work is supported by the Key Programs of the Natural Science Foundation of Zhejiang Province under Grant LZ15F030003.

8. References

[1] Wikipedia contributors. Pebble game [Internet]. Wikipedia, The Free Encyclopedia; 2015 Jul 27, 10:50 UTC [cited 2016 Jan 25]. Available from: https://en.wikipedia.org/w/index.php?title=Pebble_game&oldid=673286933.

[2] D. J. Jacobs and B. Hendrickson, "An algorithm for two-dimensional rigidity percolation: the pebble game," *Journal of Computational Physics,* vol. 137, pp. 346-365, 1997.

[3] J. P. d. l. Croix. (2013). *Sim.I.am*. Available: http://jpdelacroix.com/simiam/