Alternating Signed Bipartite Graph Colourings

Cian O'Brien
Rachel Quinlan and Kevin Jennings

Postgraduate Modelling Research Group
National University of Ireland, Galway

c.obrien40@nuigalway.ie

April 12th, 2019

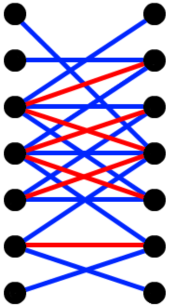
Alternating Signed Bipartite Graphs

An *Alternating Signed Bipartite Graph (ASBG)* is a graphical representation of Alternating Sign Matrices introduced by Richard Brualdi et al. (2013) [1].

Alternating Signed Bipartite Graphs

An *Alternating Signed Bipartite Graph (ASBG)* is a graphical representation of Alternating Sign Matrices introduced by Richard Brualdi et al. (2013) [1].

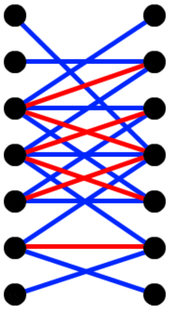
A bipartite graph G with edges coloured red and blue is an ASBG if there exists an ordering of the vertices in each part of the bipartition of G for which the edges incident with any vertex alternate in colour, starting and ending with blue.



Alternating Signed Bipartite Graphs

An *Alternating Signed Bipartite Graph (ASBG)* is a graphical representation of Alternating Sign Matrices introduced by Richard Brualdi et al. (2013) [1].

A bipartite graph G with edges coloured red and blue is an ASBG if there exists an ordering of the vertices in each part of the bipartition of G for which the edges incident with any vertex alternate in colour, starting and ending with blue.



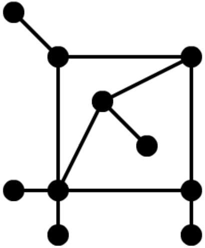
We say that G is *configurable*.

ASBG-Colourings

Question: Given a graph G without coloured edges, how can we tell if its edges have a red/blue colouring c such that G^c is an ASBG?

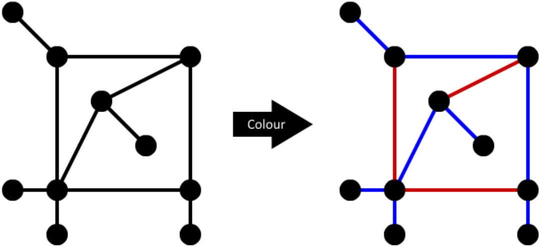
Question: Given a graph G without coloured edges, how can we tell if its edges have a red/blue colouring c such that G^c is an ASBG?

- We break the problem into two steps:



Question: Given a graph G without coloured edges, how can we tell if its edges have a red/blue colouring c such that G^c is an ASBG?

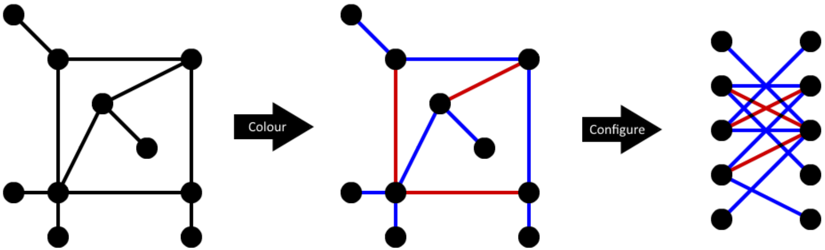
- We break the problem into two steps:



ASBG-Colourings

Question: Given a graph G without coloured edges, how can we tell if its edges have a red/blue colouring c such that G^c is an ASBG?

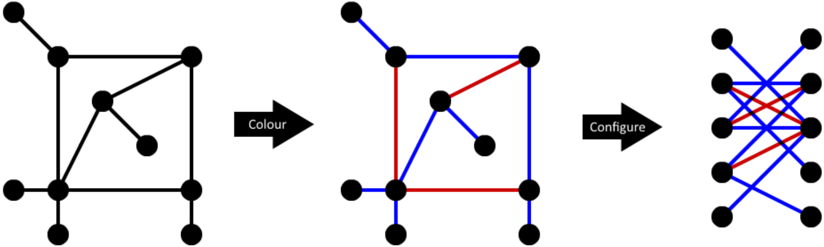
- We break the problem into two steps:



ASBG-Colourings

Question: Given a graph G without coloured edges, how can we tell if its edges have a red/blue colouring c such that G^c is an ASBG?

- We break the problem into two steps:



- We say a graph G has a *feasible colouring* c of its edges if $deg^B(v) - deg^R(v) = 1, \forall v \in V(G)$.

Leaves and Twigs

- A *leaf* is a vertex of degree 1.

Leaves and Twigs

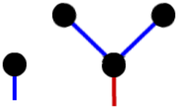
- A *leaf* is a vertex of degree 1.
- A *twig* is a configuration of three vertices; two leaves which are both attached to a third vertex of degree 3.

Leaves and Twigs

- A *leaf* is a vertex of degree 1.
- A *twig* is a configuration of three vertices; two leaves which are both attached to a third vertex of degree 3.

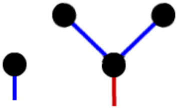
Leaves and Twigs

- A *leaf* is a vertex of degree 1.
- A *twig* is a configuration of three vertices; two leaves which are both attached to a third vertex of degree 3.



Leaves and Twigs

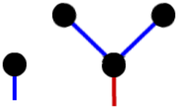
- A *leaf* is a vertex of degree 1.
- A *twig* is a configuration of three vertices; two leaves which are both attached to a third vertex of degree 3.



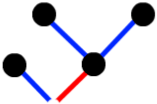
- A *leaf-twig configuration* at a vertex v is a leaf and a twig both attached to the same vertex v.

Leaves and Twigs

- A *leaf* is a vertex of degree 1.
- A *twig* is a configuration of three vertices; two leaves which are both attached to a third vertex of degree 3.

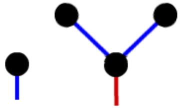


- A *leaf-twig configuration* at a vertex v is a leaf and a twig both attached to the same vertex v.

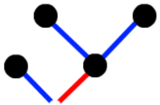


Leaves and Twigs

- A *leaf* is a vertex of degree 1.
- A *twig* is a configuration of three vertices; two leaves which are both attached to a third vertex of degree 3.



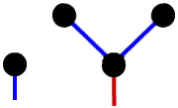
- A *leaf-twig configuration* at a vertex v is a leaf and a twig both attached to the same vertex v.



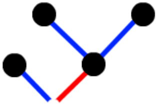
- We will refer to the following graph as *the trivial asbg*:

Leaves and Twigs

- A *leaf* is a vertex of degree 1.
- A *twig* is a configuration of three vertices; two leaves which are both attached to a third vertex of degree 3.



- A *leaf-twig configuration* at a vertex v is a leaf and a twig both attached to the same vertex v.



- We will refer to the following graph as *the trivial asbg*:

Trees

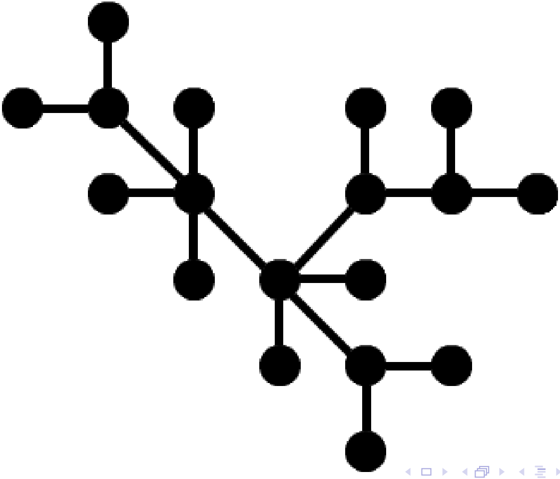
Theorem

A tree T has a feasible colouring if and only if leaf-twig configurations can be removed until the trivial ASBG remains.

Trees

Theorem

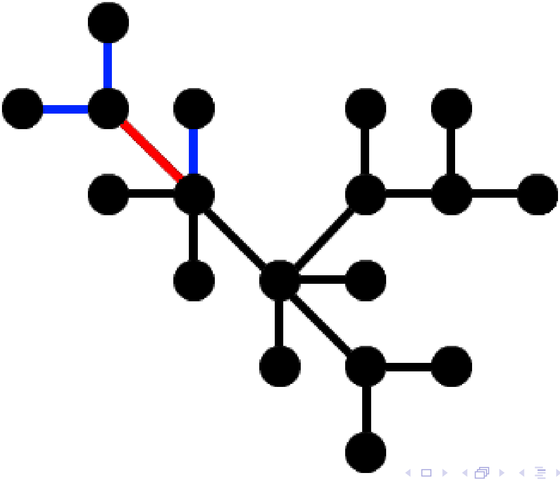
A tree T has a feasible colouring if and only if leaf-twig configurations can be removed until the trivial ASBG remains.



Trees

Theorem

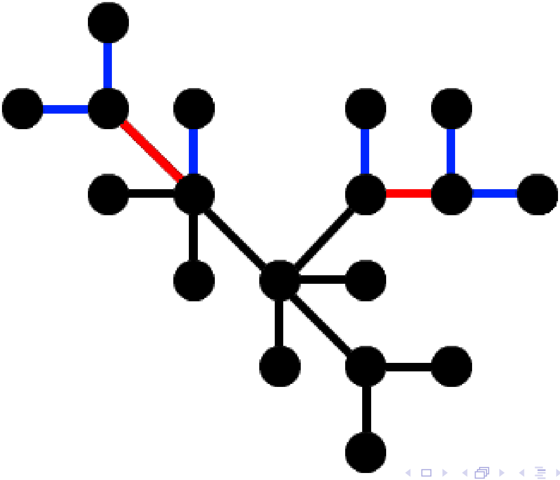
A tree T has a feasible colouring if and only if leaf-twig configurations can be removed until the trivial ASBG remains.



Trees

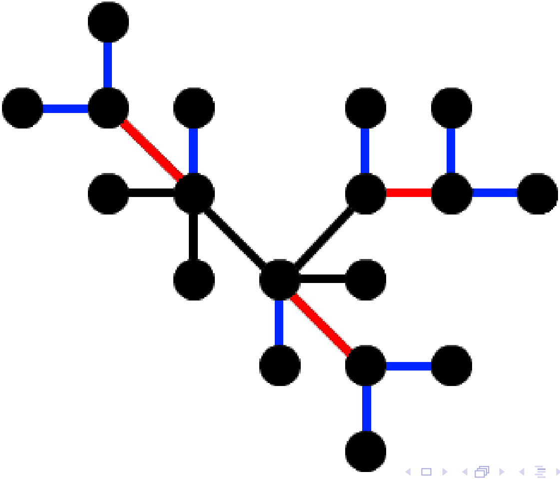
Theorem

A tree T has a feasible colouring if and only if leaf-twig configurations can be removed until the trivial ASBG remains.



Theorem

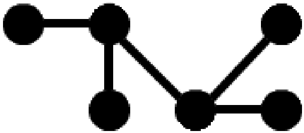
A tree T has a feasible colouring if and only if leaf-twig configurations can be removed until the trivial ASBG remains.



Trees

Theorem

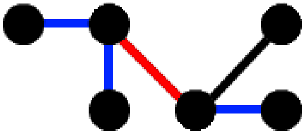
A tree T has a feasible colouring if and only if leaf-twig configurations can be removed until the trivial ASBG remains.



Trees

Theorem

A tree T has a feasible colouring if and only if leaf-twig configurations can be removed until the trivial ASBG remains.



Trees

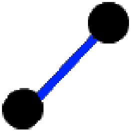
Theorem

A tree T has a feasible colouring if and only if leaf-twig configurations can be removed until the trivial ASBG remains.

Trees

Theorem

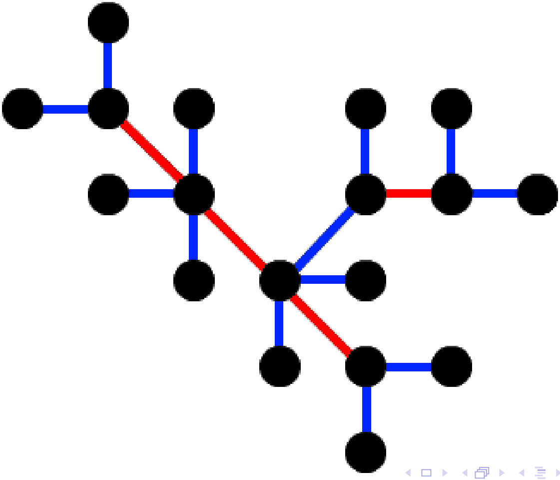
A tree T has a feasible colouring if and only if leaf-twig configurations can be removed until the trivial ASBG remains.



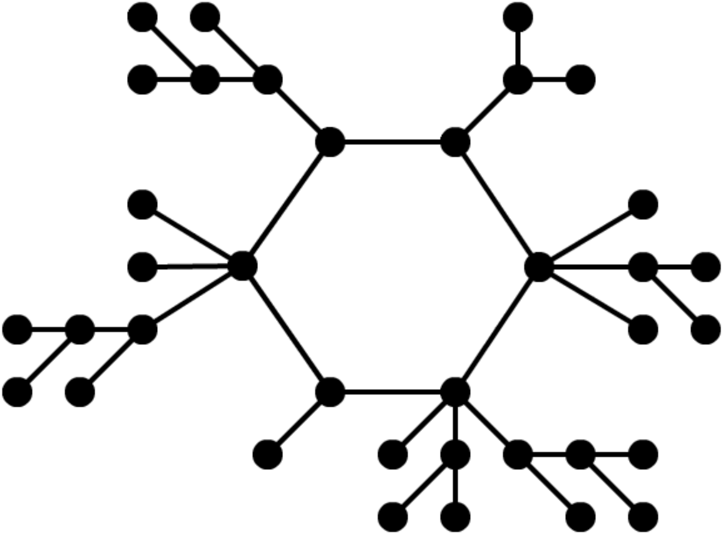
Trees

Theorem

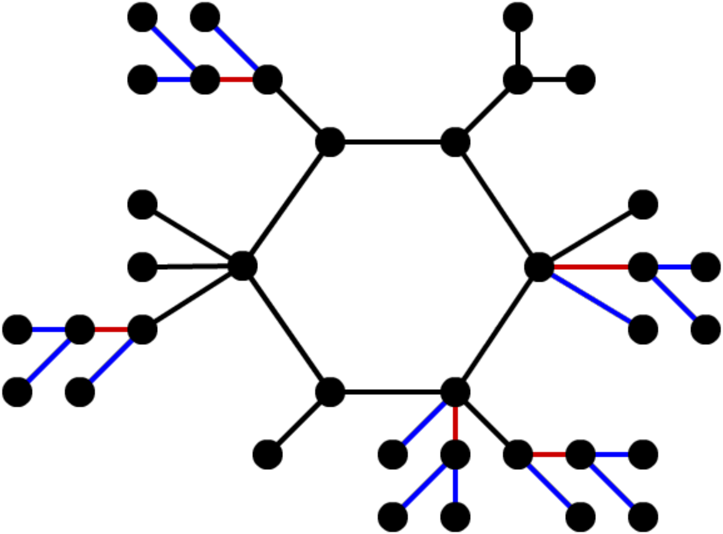
A tree T has a feasible colouring if and only if leaf-twig configurations can be removed until the trivial ASBG remains.



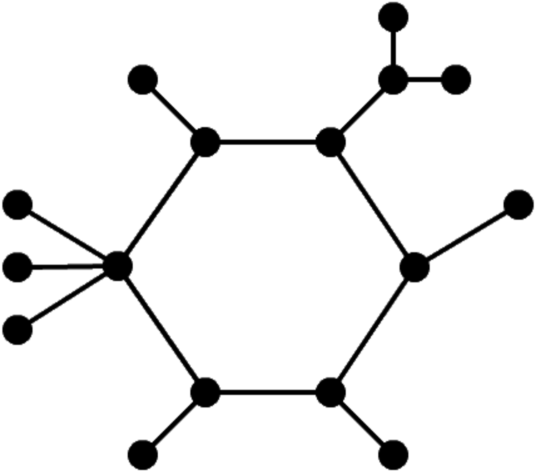
Unicyclic Graphs



Unicyclic Graphs



Unicyclic Graphs

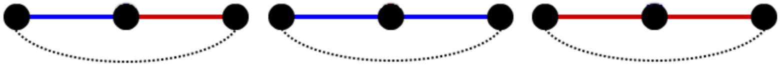


Leaf, Twig, and Triple-Type Vertices

There are three types of vertices that can appear in the cycle of a unicyclic graph: leaf-type, twig-type, and triple-type:

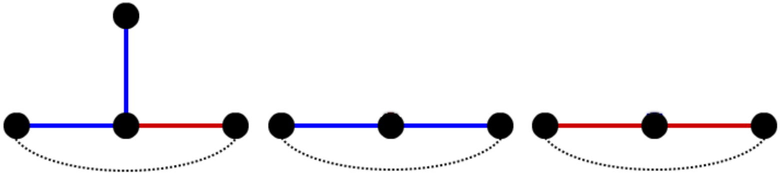
Leaf, Twig, and Triple-Type Vertices

There are three types of vertices that can appear in the cycle of a unicyclic graph: leaf-type, twig-type, and triple-type:



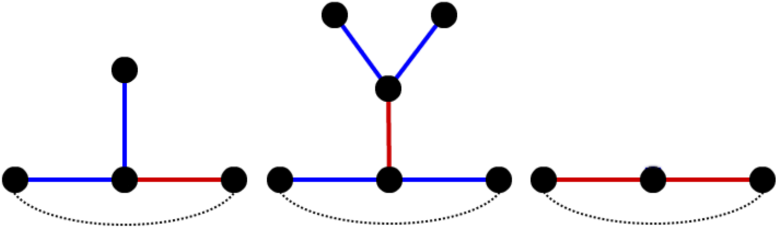
Leaf, Twig, and Triple-Type Vertices

There are three types of vertices that can appear in the cycle of a unicyclic graph: leaf-type, twig-type, and triple-type:



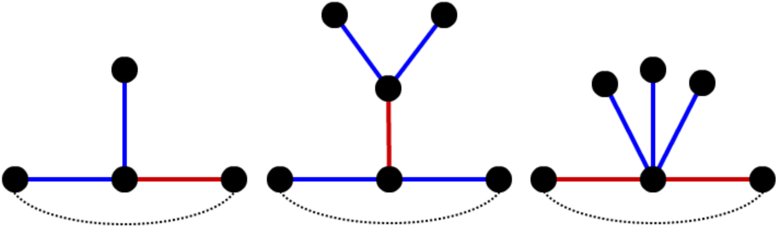
Leaf, Twig, and Triple-Type Vertices

There are three types of vertices that can appear in the cycle of a unicyclic graph: leaf-type, twig-type, and triple-type:

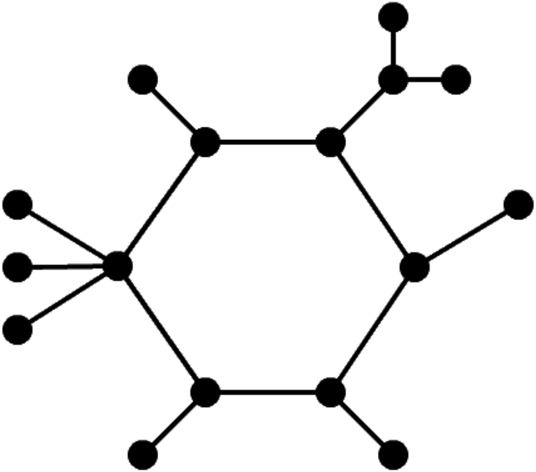


Leaf, Twig, and Triple-Type Vertices

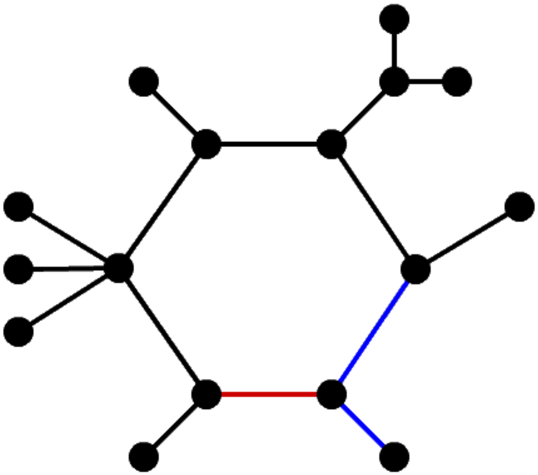
There are three types of vertices that can appear in the cycle of a unicyclic graph: leaf-type, twig-type, and triple-type:



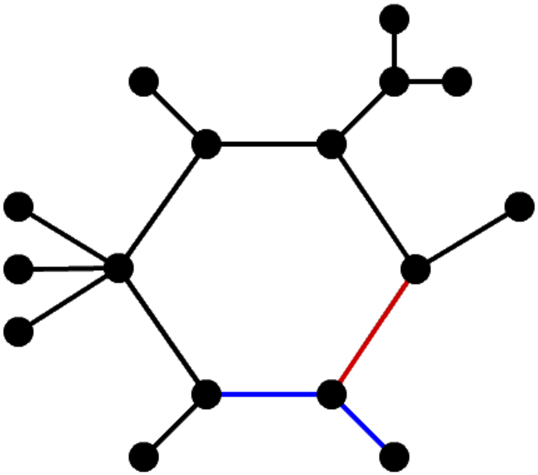
Unicyclic Graphs



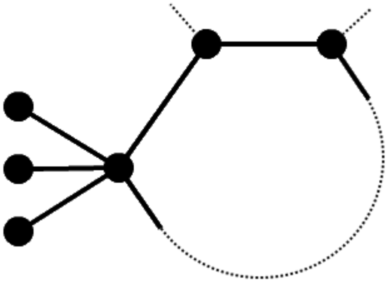
Unicyclic Graphs



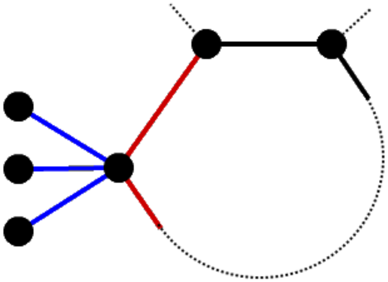
Unicyclic Graphs



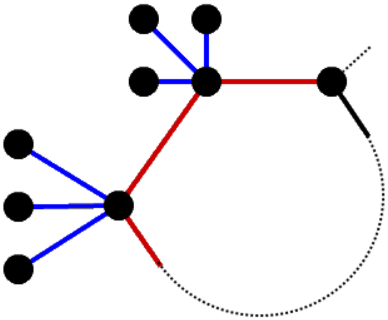
Unicyclic Graphs



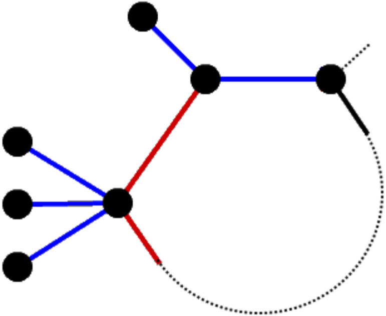
Unicyclic Graphs



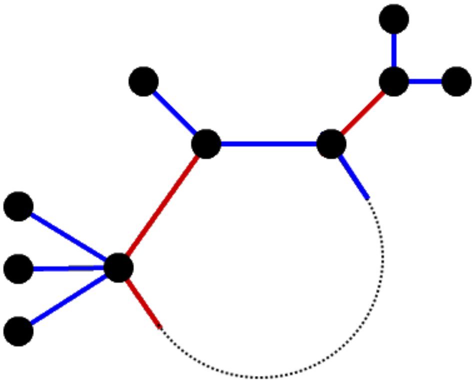
Unicyclic Graphs



Unicyclic Graphs



Unicyclic Graphs



Unicyclic Graphs

Theorem

A unicyclic bipartite graph G has a feasible colouring if and only if:

Unicyclic Graphs

Theorem

A unicyclic bipartite graph G has a feasible colouring if and only if:

- *Every vertex of the cycle of G is either leaf-type, twig-type, or triple-type;*

Unicyclic Graphs

Theorem

A unicyclic bipartite graph G has a feasible colouring if and only if:

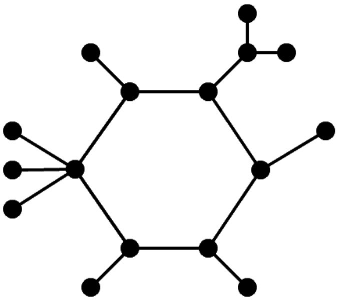
- *Every vertex of the cycle of G is either leaf-type, twig-type, or triple-type;*
- *Any twig/triple-type vertex must be an odd (even) distance away from the next twig/triple-type vertex if it is of the same (opposite) type.*

Unicyclic Graphs

Theorem

A unicyclic bipartite graph G has a feasible colouring if and only if:

- *Every vertex of the cycle of G is either leaf-type, twig-type, or triple-type;*
- *Any twig/triple-type vertex must be an odd (even) distance away from the next twig/triple-type vertex if it is of the same (opposite) type.*

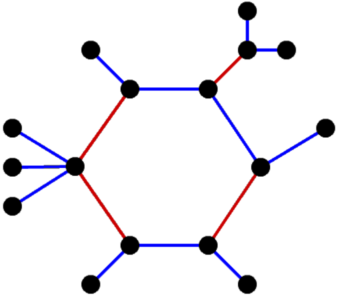


Unicyclic Graphs

Theorem

A unicyclic bipartite graph G has a feasible colouring if and only if:

- *Every vertex of the cycle of G is either leaf-type, twig-type, or triple-type;*
- *Any twig/triple-type vertex must be an odd (even) distance away from the next twig/triple-type vertex if it is of the same (opposite) type.*

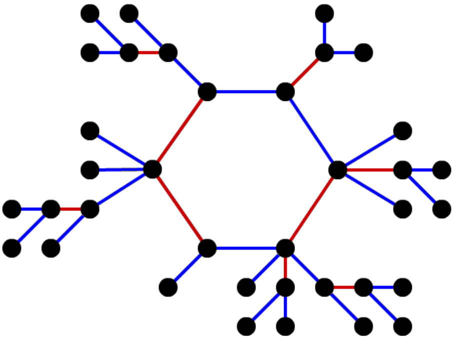


Unicyclic Graphs

Theorem

A unicyclic bipartite graph G has a feasible colouring if and only if:

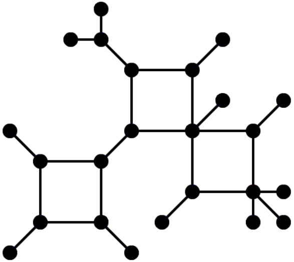
- *Every vertex of the cycle of G is either leaf-type, twig-type, or triple-type;*
- *Any twig/triple-type vertex must be an odd (even) distance away from the next twig/triple-type vertex if it is of the same (opposite) type.*



We define the *skeleton of a graph* $Sk(G)$ to be the graph that remains after all leaves have been repeatedly removed.

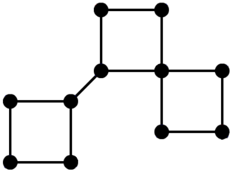
Skeletons and Junctions

We define the *skeleton of a graph* $Sk(G)$ to be the graph that remains after all leaves have been repeatedly removed.



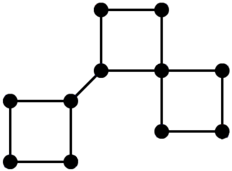
Skeletons and Junctions

We define the *skeleton of a graph* $Sk(G)$ to be the graph that remains after all leaves have been repeatedly removed.



Skeletons and Junctions

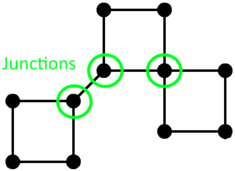
We define the *skeleton of a graph* $Sk(G)$ to be the graph that remains after all leaves have been repeatedly removed.



As well as leaf, twig, and triple-type vertices, we define one other type of vertex: a junction, which is a vertex with $deg_{Sk(G)} > 2$.

Skeletons and Junctions

We define the *skeleton of a graph* $Sk(G)$ to be the graph that remains after all leaves have been repeatedly removed.



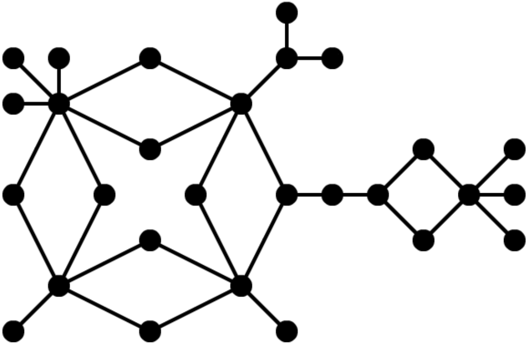
As well as leaf, twig, and triple-type vertices, we define one other type of vertex: a junction, which is a vertex with $deg_{Sk(G)} > 2$.

Surplus Weights

We have an algorithm that tries to assign values of 1 or -1 to all edges incident with each junction in a graph:

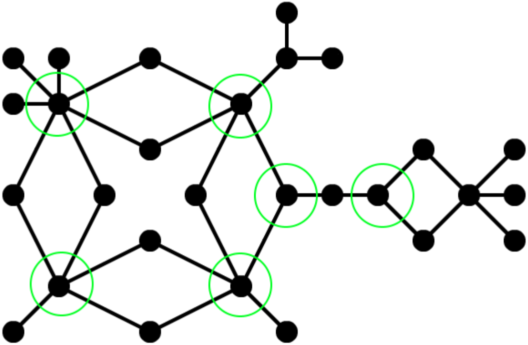
Surplus Weights

We have an algorithm that tries to assign values of 1 or -1 to all edges incident with each junction in a graph:



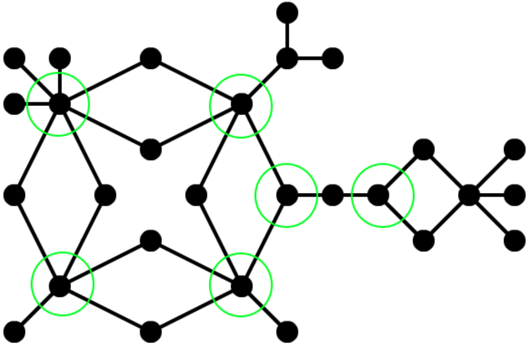
Surplus Weights

We have an algorithm that tries to assign values of 1 or -1 to all edges incident with each junction in a graph:



Surplus Weights

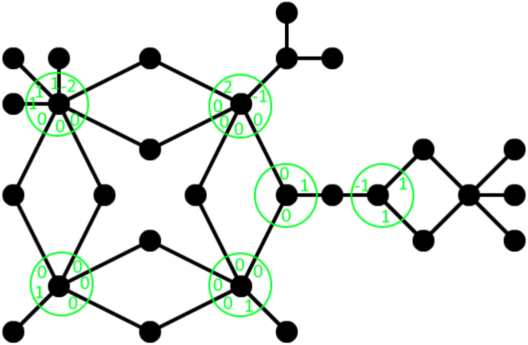
We have an algorithm that tries to assign values of 1 or -1 to all edges incident with each junction in a graph:



We don't want the algorithm to make any choices - this can lead to *surplus weights*.

Surplus Weights

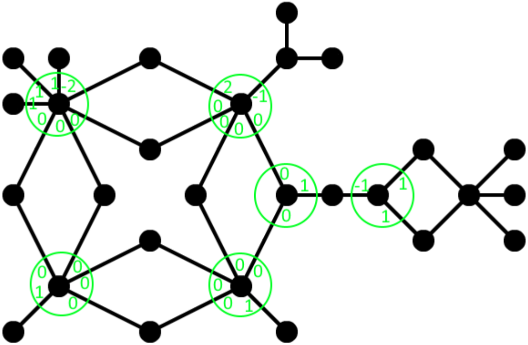
We have an algorithm that tries to assign values of 1 or -1 to all edges incident with each junction in a graph:



We don't want the algorithm to make any choices - this can lead to *surplus weights*.

Surplus Weights

We have an algorithm that tries to assign values of 1 or -1 to all edges incident with each junction in a graph:

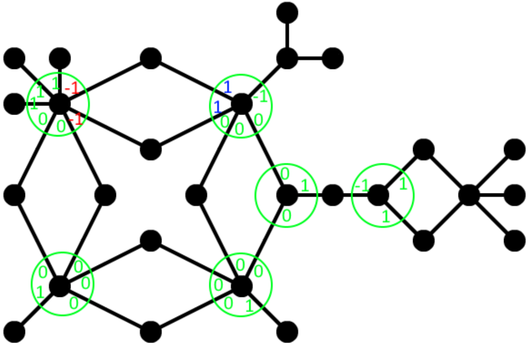


We don't want the algorithm to make any choices - this can lead to *surplus weights*.

We need to figure out if the surplus weights are *redistributable*.

Surplus Weights

We have an algorithm that tries to assign values of 1 or -1 to all edges incident with each junction in a graph:

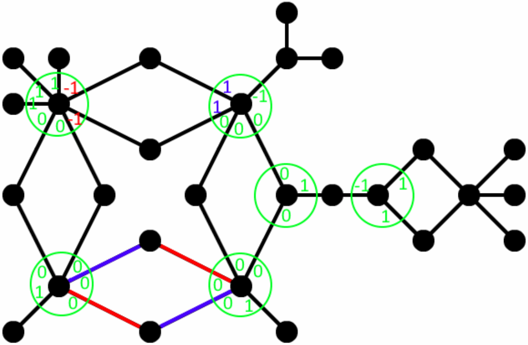


We don't want the algorithm to make any choices - this can lead to *surplus weights*.

We need to figure out if the surplus weights are *redistributable*.

Surplus Weights

We have an algorithm that tries to assign values of 1 or -1 to all edges incident with each junction in a graph:

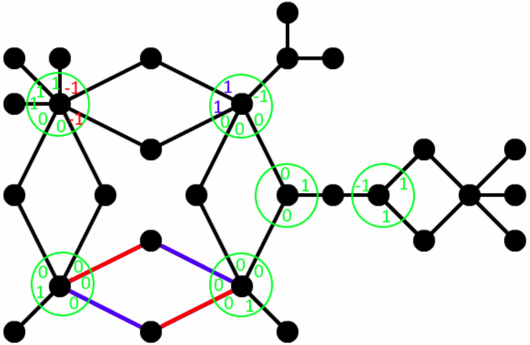


We don't want the algorithm to make any choices - this can lead to *surplus weights*.

We need to figure out if the surplus weights are *redistributable*.

Surplus Weights

We have an algorithm that tries to assign values of 1 or -1 to all edges incident with each junction in a graph:

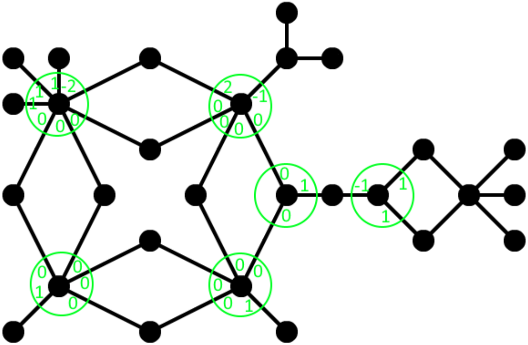


We don't want the algorithm to make any choices - this can lead to *surplus weights*.

We need to figure out if the surplus weights are *redistributable*.

Surplus Weights

We have an algorithm that tries to assign values of 1 or -1 to all edges incident with each junction in a graph:

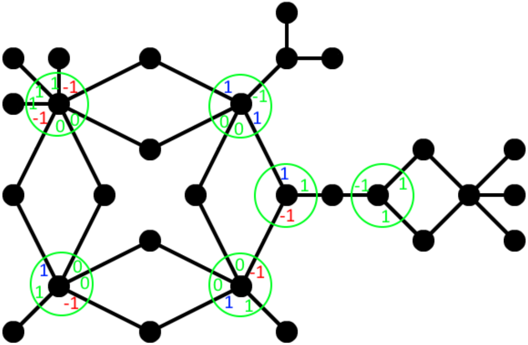


We don't want the algorithm to make any choices - this can lead to *surplus weights*.

We need to figure out if the surplus weights are *redistributable*.

Surplus Weights

We have an algorithm that tries to assign values of 1 or -1 to all edges incident with each junction in a graph:



We don't want the algorithm to make any choices - this can lead to *surplus weights*.

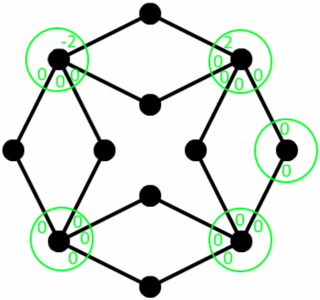
We need to figure out if the surplus weights are *redistributable*.

Redistributability

We can reframe the problem of redistributing surplus weights as finding a subgraph H of $Sk(G)$ where every vertex of H has some degree dictated by the surplus weight of that vertex.

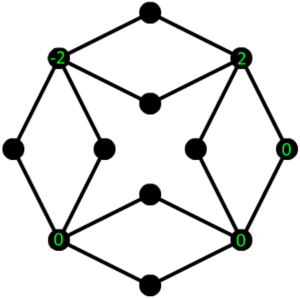
Redistributability

We can reframe the problem of redistributing surplus weights as finding a subgraph H of $Sk(G)$ where every vertex of H has some degree dictated by the surplus weight of that vertex.



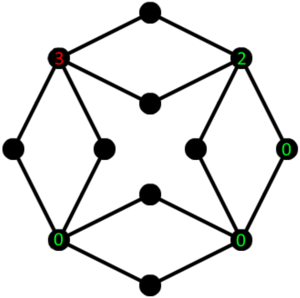
Redistributability

We can reframe the problem of redistributing surplus weights as finding a subgraph H of $Sk(G)$ where every vertex of H has some degree dictated by the surplus weight of that vertex.



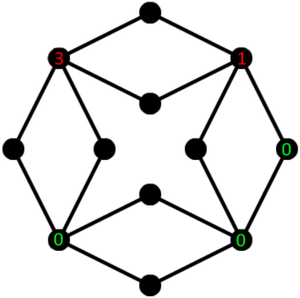
Redistributability

We can reframe the problem of redistributing surplus weights as finding a subgraph H of $Sk(G)$ where every vertex of H has some degree dictated by the surplus weight of that vertex.



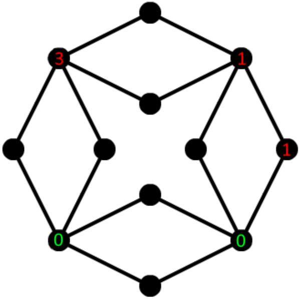
Redistributability

We can reframe the problem of redistributing surplus weights as finding a subgraph H of $Sk(G)$ where every vertex of H has some degree dictated by the surplus weight of that vertex.



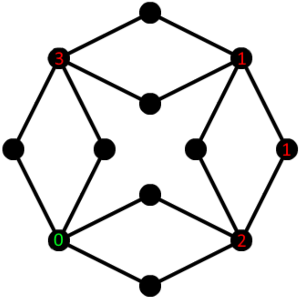
Redistributability

We can reframe the problem of redistributing surplus weights as finding a subgraph H of $Sk(G)$ where every vertex of H has some degree dictated by the surplus weight of that vertex.



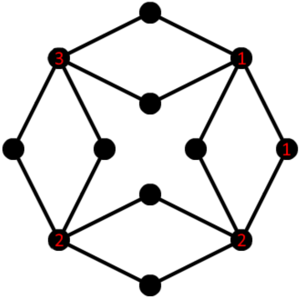
Redistributability

We can reframe the problem of redistributing surplus weights as finding a subgraph H of $Sk(G)$ where every vertex of H has some degree dictated by the surplus weight of that vertex.



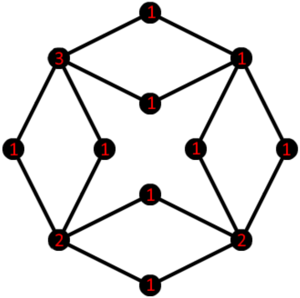
Redistributability

We can reframe the problem of redistributing surplus weights as finding a subgraph H of $Sk(G)$ where every vertex of H has some degree dictated by the surplus weight of that vertex.



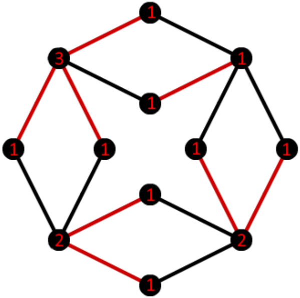
Redistributability

We can reframe the problem of redistributing surplus weights as finding a subgraph H of $Sk(G)$ where every vertex of H has some degree dictated by the surplus weight of that vertex.



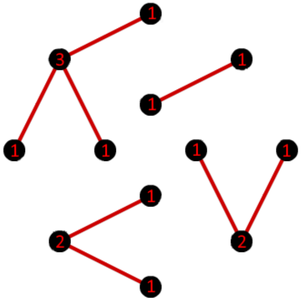
Redistributability

We can reframe the problem of redistributing surplus weights as finding a subgraph H of $Sk(G)$ where every vertex of H has some degree dictated by the surplus weight of that vertex.



Redistributability

We can reframe the problem of redistributing surplus weights as finding a subgraph H of $Sk(G)$ where every vertex of H has some degree dictated by the surplus weight of that vertex.



Redistributability

We have the following condition for when it is possible to find a subgraph H of a graph G where the degree of each vertex v is some required value $r(v)$:

Redistributability

We have the following condition for when it is possible to find a subgraph H of a graph G where the degree of each vertex v is some required value $r(v)$:

Theorem

Let G be a bipartite graph with bipartition P_1 and P_2, and let each vertex v of G be assigned an integer value $r(v)$ in the range 0 to $deg(v)$. Then G has a subgraph H with $deg_H(v) = r(v)$ for all vertices v if and only if every $S \subset P_1$ in G satisfies

$$\sum_{v \in S} r(v) \leq \sum_{n \in \Gamma(S)} \min\{r(n), |\Gamma(n) \cap S|\},$$

where $\Gamma(S)$ is the set of neighbours of S in G.

The General Case

Theorem

A bipartite graph G has a feasible colouring if and only if:

The General Case

Theorem

A bipartite graph G has a feasible colouring if and only if:

- *Each vertex in $Sk(G)$ is either leaf-type, twig-type, triple-type, or a junction which has been assigned a weight of 1 by our algorithm;*

The General Case

Theorem

A bipartite graph G has a feasible colouring if and only if:

- *Each vertex in $Sk(G)$ is either leaf-type, twig-type, triple-type, or a junction which has been assigned a weight of 1 by our algorithm;*
- *Any twig/triple-type vertex must be an odd (even) distance away from the next twig/triple-type vertex if it is of the same (opposite) type;*

The General Case

Theorem

A bipartite graph G has a feasible colouring if and only if:

- *Each vertex in $Sk(G)$ is either leaf-type, twig-type, triple-type, or a junction which has been assigned a weight of 1 by our algorithm;*
- *Any twig/triple-type vertex must be an odd (even) distance away from the next twig/triple-type vertex if it is of the same (opposite) type;*
- *All surplus weights in G are redistributable.*

Unique Colourings and Configurability

- There are classes of graphs for which all feasible colourings are unique and configurable.

Unique Colourings and Configurability

- There are classes of graphs for which all feasible colourings are unique and configurable.
- We define the class of *cycle-edge-disjoint graphs* to be any graph in which no pair of cycles contain a common edge.

Unique Colourings and Configurability

- There are classes of graphs for which all feasible colourings are unique and configurable.
- We define the class of *cycle-edge-disjoint graphs* to be any graph in which no pair of cycles contain a common edge.

Theorem

If a cycle-edge-disjoint graph G has a feasible colouring c, then c is the unique feasible colouring of G and G^c is configurable.

Unique Colourings and Configurability

- There are classes of graphs for which all feasible colourings are unique and configurable.
- We define the class of *cycle-edge-disjoint graphs* to be any graph in which no pair of cycles contain a common edge.

Theorem

If a cycle-edge-disjoint graph G has a feasible colouring c, then c is the unique feasible colouring of G and G^c is configurable.

Theorem

If a graph G has two distinct feasible colouring c and d, then c and d differ only by alternating cycle rotations.

Unique Colourings and Configurability

- There are classes of graphs for which all feasible colourings are unique and configurable.
- We define the class of *cycle-edge-disjoint graphs* to be any graph in which no pair of cycles contain a common edge.

Theorem

If a cycle-edge-disjoint graph G has a feasible colouring c, then c is the unique feasible colouring of G and G^c is configurable.

Theorem

If a graph G has two distinct feasible colouring c and d, then c and d differ only by alternating cycle rotations.

- We are currently working on conditions for when a coloured graph G^c is configurable in general.

Richard A. Brualdi, Kathleen P. Kiernan, Seth A. Meyer, Michael W. Schroeder , *Patterns of Alternating Sign Matrices*, Department of Mathematics University of Wisconsin, 2011