Finding Perfect Matchings and Completing Latin Rectangles

Nicholas Dibble-Kahn

University of California Santa Barbara

April 3, 2014

Outline

- Definitions

Outline

- Definitions
- The Problem

Outline

- Definitions
- The Problem
- The Set up

Outline

- Definitions
- The Problem
- The Set up
- The Process

Outline

- Definitions
- The Problem
- The Set up
- The Process
- The Proof

Outline

- Definitions
- The Problem
- The Set up
- The Process
- The Proof
- Latin Rectangles

Graph Terminology

- A **Graph** is a collection of vertices, and edges between them

Graph Terminology

- A **Graph** is a collection of vertices, and edges between them
- Edges are defined as the unique connection between two different vertices

Graph Terminology

- A **Graph** is a collection of vertices, and edges between them
- Edges are defined as the unique connection between two different vertices
- The degree of a vertex in a graph is the number of edges that vertex is connected to

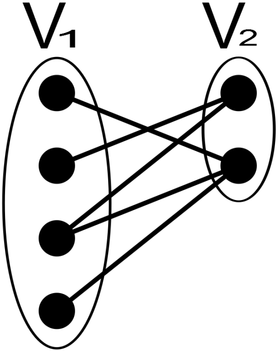
Bipartite Graph

Definition: A Bipartite Graph is a graph such that every vertex can be put into one of two groups, V_1 and V_2, with the property that all edges connect vertices in V_1 to vertices in V_2 (no edges go between vertices in the same group)

Bipartite Graph

Definition: A Bipartite Graph is a graph such that every vertex can be put into one of two groups, V_1 and V_2, with the property that all edges connect vertices in V_1 to vertices in V_2 (no edges go between vertices in the same group)

Example:



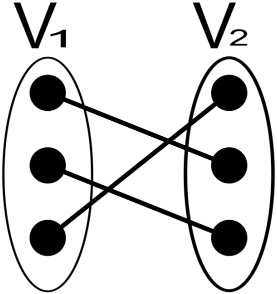
Perfect Matching

Definition: A Perfect Matching is a bipartite graph such that there is exactly one edge connected to each vertex

Perfect Matching

Definition: A Perfect Matching is a bipartite graph such that there is exactly one edge connected to each vertex

Example:



The Problem

Problem: We are given a **Bipartite Graph**, G, with n vertices of degree d in sets V_1 and V_2

We are asked:

The Problem

Problem: We are given a **Bipartite Graph**, G, with n vertices of degree d in sets V_1 and V_2

We are asked:

- Can we find d **Perfect Matchings** of G?

The Problem

Problem: We are given a **Bipartite Graph**, G, with n vertices of degree d in sets V_1 and V_2

We are asked:

- Can we find d **Perfect Matchings** of G?
- How can we find these Perfect Matchings?

The Problem

- Given: G is is composed of two sets of vertices: V_1 and V_2 that each have n elements

The Problem

- Given: G is is composed of two sets of vertices: V_1 and V_2 that each have n elements
- We shall label the vertices in V_1 the numbers 1', 2',...,(n-1)', n'

The Problem

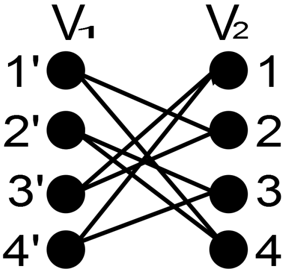
- Given: G is is composed of two sets of vertices: V_1 and V_2 that each have n elements
- We shall label the vertices in V_1 the numbers 1', 2',...,(n-1)', n'
- And label the vertices in V_2 the numbers 1, 2,...,(n-1), n

The Problem

This may seem like a trivial task but let us consider a simple case:

The Problem

This may seem like a trivial task but let us consider a simple case:



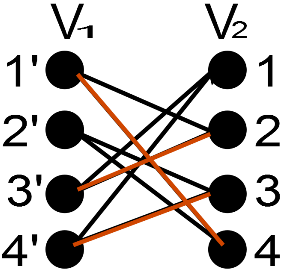
The Problem

If we randomly pick edges we could get the following:

The Problem

If we randomly pick edges we could get the following:

- This is notably not a perfect matching, and we cannot easily make it one because there does not exist an edge in our initial graph from 2' to 1



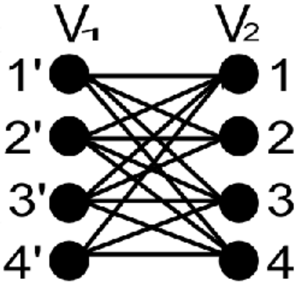
The Set Up

Let us consider the case when d=n; there is an edge from every vertex in V_1 to every vertex in V_2, and we can model this as a table:

The Set Up

Let us consider the case when d=n; there is an edge from every vertex in V_1 to every vertex in V_2, and we can model this as a table:

Example: let d=n=4

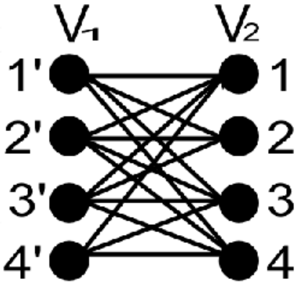


$1'$	$2'$	$3'$	$4'$
1	1	1	1
2	2	2	2
3	3	3	3
4	4	4	4

The Set Up

Let us consider the case when d=n; there is an edge from every vertex in V_1 to every vertex in V_2, and we can model this as a table:

Example: let d=n=4



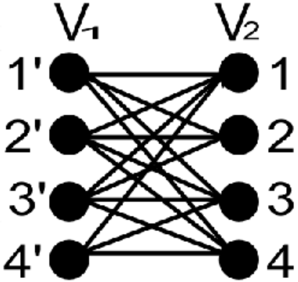
$1'$	$2'$	$3'$	$4'$
1	1	1	1
2	2	2	2
3	3	3	3
4	4	4	4

- Each column is headed by a vertex in V_1

The Set Up

Let us consider the case when d=n; there is an edge from every vertex in V_1 to every vertex in V_2, and we can model this as a table:

Example: let d=n=4



$1'$	$2'$	$3'$	$4'$
1	1	1	1
2	2	2	2
3	3	3	3
4	4	4	4

- Each column is headed by a vertex in V_1
- Each other number in the column is a vertex in V_2 such that there is an edge between the head of a column and each element in its column

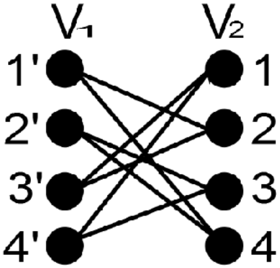
The Set Up

If instead of having our table represent d=n we wanted some other d less than n, we can do this by simply crossing off edges that are not present

The Set Up

If instead of having our table represent d=n we wanted some other d less than n, we can do this by simply crossing off edges that are not present

Example: n=4, d=2

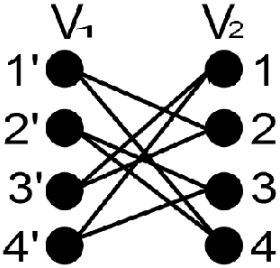


$1'$	$2'$	$3'$	$4'$
X_1	X_1	1	1
2	X_2	2	X_2
X_3	3	X_3	3
4	4	X_4	X_4

The Set Up

If instead of having our table represent d=n we wanted some other d less than n, we can do this by simply crossing off edges that are not present

Example: n=4, d=2



$1'$	$2'$	$3'$	$4'$
X_1	X_1	1	1
2	X_2	2	X_2
X_3	3	X_3	3
4	4	X_4	X_4

For edges that we removed from the d=n graph, we put an X in the array to indicate that there is not an edge

The Process

- We can take a bipartite graph with d edges and construct a table for it such that each column has d elements that are not-X's

The Process

- We can take a bipartite graph with d edges and construct a table for it such that each column has d elements that are not-X's
- From here we shall start picking elements, in a specific fashion, to find a single perfect matching

The Process

- We can take a bipartite graph with d edges and construct a table for it such that each column has d elements that are not-X's
- From here we shall start picking elements, in a specific fashion, to find a single perfect matching
- Once we find a perfect matching we can delete it from our graph to reduce d by 1,

The Process

- We can take a bipartite graph with d edges and construct a table for it such that each column has d elements that are not-X's
- From here we shall start picking elements, in a specific fashion, to find a single perfect matching
- Once we find a perfect matching we can delete it from our graph to reduce d by 1, from which we can repeat the process d times until the entire initial graph becomes partitioned into d perfect matchings

The Process

- We can take a bipartite graph with d edges and construct a table for it such that each column has d elements that are not-X's
- From here we shall start picking elements, in a specific fashion, to find a single perfect matching
- Once we find a perfect matching we can delete it from our graph to reduce d by 1, from which we can repeat the process d times until the entire initial graph becomes partitioned into d perfect matchings
- When we pick a vertex of V_2 to draw an edge to a specific vertex in V_1,

The Process

- We can take a bipartite graph with d edges and construct a table for it such that each column has d elements that are not-X's
- From here we shall start picking elements, in a specific fashion, to find a single perfect matching
- Once we find a perfect matching we can delete it from our graph to reduce d by 1, from which we can repeat the process d times until the entire initial graph becomes partitioned into d perfect matchings
- When we pick a vertex of V_2 to draw an edge to a specific vertex in V_1, we cannot not pick another edge that shares either of these two vertices

The Process

- We can take a bipartite graph with d edges and construct a table for it such that each column has d elements that are not-X's
- From here we shall start picking elements, in a specific fashion, to find a single perfect matching
- Once we find a perfect matching we can delete it from our graph to reduce d by 1, from which we can repeat the process d times until the entire initial graph becomes partitioned into d perfect matchings
- When we pick a vertex of V_2 to draw an edge to a specific vertex in V_1, we cannot not pick another edge that shares either of these two vertices
- On the table this corresponds to picking an element, E, in a column, C', and then not picking another element E in a different column

The Process

- We can take a bipartite graph with d edges and construct a table for it such that each column has d elements that are not-X's
- From here we shall start picking elements, in a specific fashion, to find a single perfect matching
- Once we find a perfect matching we can delete it from our graph to reduce d by 1, from which we can repeat the process d times until the entire initial graph becomes partitioned into d perfect matchings
- When we pick a vertex of V_2 to draw an edge to a specific vertex in V_1, we cannot not pick another edge that shares either of these two vertices
- On the table this corresponds to picking an element, E, in a column, C', and then not picking another element E in a different column and likewise not pick another element from column C'

The Process

- The last, and most important step, is determining which points to pick

The Process

- The last, and most important step, is determining which points to pick
- To answer this we shall define a K_c to be the number of elements in a column C' that are not X,

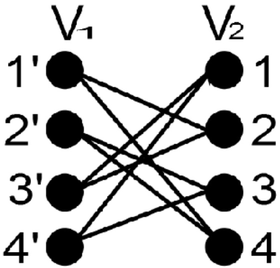
The Process

- The last, and most important step, is determining which points to pick
- To answer this we shall define a K_c to be the number of elements in a column C' that are not X,
- If we consider all of the columns, then we will pick any element from the column with the smallest value for K_c

The Process

- The last, and most important step, is determining which points to pick
- To answer this we shall define a K_c to be the number of elements in a column C' that are not X,
- If we consider all of the columns, then we will pick any element from the column with the smallest value for K_c

To best understand this process, let us find a perfect matching of the previous example:



$\Longrightarrow$

$1'$	$2'$	$3'$	$4'$
X_1	X_1	1	1
2	X_2	2	X_2
X_3	3	X_3	3
4	4	X_4	X_4

The Process

To best understand this process, let us find a perfect matching of the previous example:

$1'$	$2'$	$3'$	$4'$
X_1	X_1	1	1
2	X_2	2	X_2
X_3	3	X_3	3
4	4	X_4	X_4

As $K_c = 2$ for all the columns we can pick a random element from any of the columns, let us then arbitrarily pick column 2' element 3

The Process

Thus having picked (2',3) we shall make all other elements in column 2' and row 3 X:

$1'$	$2'$	$3'$	$4'$
X_1	X_1	1	1
2	X_2	2	X_2
X_3	3	X_3	3
4	4	X_4	X_4

The Process

Thus having picked (2',3) we shall make all other elements in column 2' and row 3 X:

$1'$	$2'$	$3'$	$4'$
X_1	X_1	1	1
2	X_2	2	X_2
X_3	3	X_3	X_3
4	X_4	X_4	X_4

The Process

There are now three rows left. $K_1 = K_3 = 2$ while $K_4 = 1$ thus we must pick from column 4' the element 1

$1'$	$2'$	$3'$	$4'$
X_1	X_1	1	1
2	X_2	2	X_2
X_3	3	X_3	X_3
4	X_4	X_4	X_4

The Process

Having picked (4',1) we must now X out all other elements in Column 4' and row 1:

$1'$	$2'$	$3'$	$4'$
X_1	X_1	1	1
2	X_2	2	X_2
X_3	3	X_3	X_3
4	X_4	X_4	X_4

The Process

Having picked (4',1) we must now X out all other elements in Column 4' and row 1:

$1'$	$2'$	$3'$	$4'$
X_1	X_1	X_1	1
2	X_2	2	X_2
X_3	3	X_3	X_3
4	X_4	X_4	X_4

The Process

We now have that $K_1 = 2$ and $K_3 = 1$ so we must pick from column 3' element 2:

$1'$	$2'$	$3'$	$4'$
X_1	X_1	X_1	1
2	X_2	2	X_2
X_3	3	X_3	X_3
4	X_4	X_4	X_4

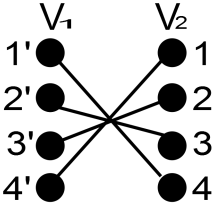
The Process

Putting X's in the appropriate places yields:

$1'$	$2'$	$3'$	$4'$
X_1	X_1	X_1	1
2	X_2	2	X_2
X_3	3	X_3	X_3
4	X_4	X_4	X_4

$1'$	$2'$	$3'$	$4'$
X_1	X_1	X_1	1
X_2	X_2	2	X_2
X_3	3	X_3	X_3
4	X_4	X_4	X_4

Leaving only element (1',4) to pick. The edges defined by (1',4), (2',3), (3', 2), and (4',1) thus form a perfect matching:



The Proof

Claim: That the process detailed above will give a perfect matching

The Proof

Claim: That the process detailed above will give a perfect matching

We will prove this by contradiction

The Proof

Claim: That the process detailed above will give a perfect matching
We will prove this by contradiction

- This process will always give some matching, but if this matching wasn't perfect, there would be some vertices without edges

The Proof

Claim: That the process detailed above will give a perfect matching
We will prove this by contradiction

- This process will always give some matching, but if this matching wasn't perfect, there would be some vertices without edges
- The only way this would happen in the process is if some K_{A_0} were to become 0 (such that we did not yet pick an element from column A_0')

The Proof

Claim: That the process detailed above will give a perfect matching
We will prove this by contradiction

- This process will always give some matching, but if this matching wasn't perfect, there would be some vertices without edges
- The only way this would happen in the process is if some K_{A_0} were to become 0 (such that we did not yet pick an element from column A'_0)
- If we look at the step in the process before $K_{A_0} = 0$ then column A_0 must have had an element, e_0

The Proof

Claim: That the process detailed above will give a perfect matching
We will prove this by contradiction

- This process will always give some matching, but if this matching wasn't perfect, there would be some vertices without edges
- The only way this would happen in the process is if some K_{A_0} were to become 0 (such that we did not yet pick an element from column A_0')
- If we look at the step in the process before $K_{A_0} = 0$ then column A_0 must have had an element, e_0
- Because we must pick from the column with the fewest elements to pick from,

The Proof

Claim: That the process detailed above will give a perfect matching
We will prove this by contradiction

- This process will always give some matching, but if this matching wasn't perfect, there would be some vertices without edges
- The only way this would happen in the process is if some K_{A_0} were to become 0 (such that we did not yet pick an element from column A_0')
- If we look at the step in the process before $K_{A_0} = 0$ then column A_0 must have had an element, e_0
- Because we must pick from the column with the fewest elements to pick from, the only reason we would not have select (A_0', e_0) would be if there were another column, A_1', that also only had one element, e_1, left to pick

The Proof

Claim: That the process detailed above will give a perfect matching
We will prove this by contradiction

- This process will always give some matching, but if this matching wasn't perfect, there would be some vertices without edges
- The only way this would happen in the process is if some K_{A_0} were to become 0 (such that we did not yet pick an element from column A'_0)
- If we look at the step in the process before $K_{A_0} = 0$ then column A_0 must have had an element, e_0
- Because we must pick from the column with the fewest elements to pick from, the only reason we would not have select (A'_0, e_0) would be if there were another column, A'_1, that also only had one element, e_1, left to pick
- And specifically the only reason (A'_1, e_1) would prevent (A'_0, e_0) from getting chosen is if $e_1 = e_0$ so as to then change e_0 to an X

The Proof

- If we now look at the pick that lead to $K_{A_0} = K_{A_1} = 1$ then the columns A_0' and A_1' must've had a second element in them.

The Proof

- If we now look at the pick that lead to $K_{A_0} = K_{A_1} = 1$ then the columns A'_0 and A'_1 must've had a second element in them.
- We get that the only reason we would not have selected either (A'_0, a_0) or (A'_1, a_1) is if there was a third column, A'_2 that had also had at most two elements left.

The Proof

- If we now look at the pick that lead to $K_{A_0} = K_{A_1} = 1$ then the columns A_0' and A_1' must've had a second element in them.
- We get that the only reason we would not have selected either (A_0', a_0) or (A_1', a_1) is if there was a third column, A_2' that had also had at most two elements left.
- For A_2' to be selected and leave behind the situation we had above (going one step back from $K_{A_1} = 0$) we get that all three columns must've shared an element that was selected in column A_2', and we shall call this element e_2

The Proof

- If we continue this process going P picks back from $K_{A_1} = 0$ we are considering P+1 columns, $A_0, A_1, ...A_p$, each of which has P elements to choose from

The Proof

- If we continue this process going P picks back from $K_{A_1} = 0$ we are considering P+1 columns, $A_0, A_1, ...A_p$, each of which has P elements to choose from
- We also get that these P+1 rows must share at least one element in common so as to lead to the scenario of P-1 picks away from $K_{A_1} = 0$

The Proof

- If we continue this process going P picks back from $K_{A_1} = 0$ we are considering P+1 columns, $A_0, A_1, ...A_p$, each of which has P elements to choose from
- We also get that these P+1 rows must share at least one element in common so as to lead to the scenario of P-1 picks away from $K_{A_1} = 0$
- If we let P=n-d then we arrive at the fact that there are (n-d)+1 rows that have at least 1 element, E, in common from which to choose

The Proof

- If we continue this process going P picks back from $K_{A_1} = 0$ we are considering P+1 columns, $A_0, A_1, ...A_p$, each of which has P elements to choose from
- We also get that these P+1 rows must share at least one element in common so as to lead to the scenario of P-1 picks away from $K_{A_1} = 0$
- If we let P=n-d then we arrive at the fact that there are (n-d)+1 rows that have at least 1 element, E, in common from which to choose
- However we have only removed d edges from our the base graph d=n

The Proof

- If we continue this process going P picks back from $K_{A_1} = 0$ we are considering P+1 columns, $A_0, A_1, ...A_p$, each of which has P elements to choose from
- We also get that these P+1 rows must share at least one element in common so as to lead to the scenario of P-1 picks away from $K_{A_1} = 0$
- If we let P=n-d then we arrive at the fact that there are (n-d)+1 rows that have at least 1 element, E, in common from which to choose
- However we have only removed d edges from our the base graph d=n which means that each element is removed d times across all of the columns, thus each row has an element appear n-d times,

The Proof

- If we continue this process going P picks back from $K_{A_1} = 0$ we are considering P+1 columns, $A_0, A_1, ...A_p$, each of which has P elements to choose from
- We also get that these P+1 rows must share at least one element in common so as to lead to the scenario of P-1 picks away from $K_{A_1} = 0$
- If we let P=n-d then we arrive at the fact that there are (n-d)+1 rows that have at least 1 element, E, in common from which to choose
- However we have only removed d edges from our the base graph d=n which means that each element is removed d times across all of the columns, thus each row has an element appear n-d times, but we now have a row, E, can be chosen by n-d+1 columns, thus

The Proof

- If we continue this process going P picks back from $K_{A_1} = 0$ we are considering P+1 columns, $A_0, A_1, ...A_p$, each of which has P elements to choose from
- We also get that these P+1 rows must share at least one element in common so as to lead to the scenario of P-1 picks away from $K_{A_1} = 0$
- If we let P=n-d then we arrive at the fact that there are (n-d)+1 rows that have at least 1 element, E, in common from which to choose
- However we have only removed d edges from our the base graph d=n which means that each element is removed d times across all of the columns, thus each row has an element appear n-d times, but we now have a row, E, can be chosen by n-d+1 columns, thus WE HAVE A CONTRADICTION!

The Proof

- If we continue this process going P picks back from $K_{A_1} = 0$ we are considering P+1 columns, $A_0, A_1, ...A_p$, each of which has P elements to choose from
- We also get that these P+1 rows must share at least one element in common so as to lead to the scenario of P-1 picks away from $K_{A_1} = 0$
- If we let P=n-d then we arrive at the fact that there are (n-d)+1 rows that have at least 1 element, E, in common from which to choose
- However we have only removed d edges from our the base graph d=n which means that each element is removed d times across all of the columns, thus each row has an element appear n-d times, but we now have a row, E, can be chosen by n-d+1 columns, thus WE HAVE A CONTRADICTION!
- This means that the process detailed in this presentation will always yield a perfect matching

Applications

Definition: A Latin Square is an n by n array that has each of n symbols appear exactly once in each row and each column

Applications

Definition: A Latin Square is an n by n array that has each of n symbols appear exactly once in each row and each column

Example of a 5 by 5 Latin Square:

1	2	3	4	5
2	4	5	1	3
3	5	4	2	1
4	3	1	5	2
5	1	2	3	4

Applications

Definition: A Latin Square is an n by n array that has each of n symbols appear exactly once in each row and each column
Example of a 5 by 5 Latin Square:

1	2	3	4	5
2	4	5	1	3
3	5	4	2	1
4	3	1	5	2
5	1	2	3	4

Definition: A Latin Rectangle is a partial Latin Square that has the first D rows given and the rest blank

Applications

Definition: A Latin Square is an n by n array that has each of n symbols appear exactly once in each row and each column

Example of a 5 by 5 Latin Square:

1	2	3	4	5
2	4	5	1	3
3	5	4	2	1
4	3	1	5	2
5	1	2	3	4

Definition: A Latin Rectangle is a partial Latin Square that has the first D rows given and the rest blank

Example of a 2 by 4 Latin Rectangle:

1	2	3	4
2	4	5	1

Application

We can complete a Latin Rectangle, by application of the process used to find perfect matchings, in the follow way:

Application

We can complete a Latin Rectangle, by application of the process used to find perfect matchings, in the follow way:

Bipartite Graph Features	Analogous Features of Latin Squares

Application

We can complete a Latin Rectangle, by application of the process used to find perfect matchings, in the follow way:

Bipartite Graph Features	Analogous Features of Latin Squares
Each vertex in V_1	One specific Column

Application

We can complete a Latin Rectangle, by application of the process used to find perfect matchings, in the follow way:

Bipartite Graph Features	Analogous Features of Latin Squares
Each vertex in V_1	One specific Column
Each vertex in V_2	One specific symbol

Application

We can complete a Latin Rectangle, by application of the process used to find perfect matchings, in the follow way:

Bipartite Graph Features	Analogous Features of Latin Squares
Each vertex in V_1	One specific Column
Each vertex in V_2	One specific symbol
Each perfect matching	One specific row

Application

We can complete a Latin Rectangle, by application of the process used to find perfect matchings, in the follow way:

Bipartite Graph Features	Analogous Features of Latin Squares
Each vertex in V_1	One specific Column
Each vertex in V_2	One specific symbol
Each perfect matching	One specific row
The common degree of all vertices	The number of blank rows

Application

We can complete a Latin Rectangle, by application of the process used to find perfect matchings, in the follow way:

Bipartite Graph Features	Analogous Features of Latin Squares
Each vertex in V_1	One specific Column
Each vertex in V_2	One specific symbol
Each perfect matching	One specific row
The common degree of all vertices	The number of blank rows

As we can create d perfect matchings of a Bipartite Graph, we can complete the Latin Rectangle

Thank you

Thank You all for listening to this presentation
Questions?