Fundamental Algorithms for Bioinformatics

Algorithm Design

The Consecutive Ones Property

The Consecutive Ones Property

CONSECUTIVE ONES PROPERTY (C1P)
Input: A Boolean matrix ***A***
Output: A permutation of the columns of ***A*** such that in the the resulting matrix in each row the 1's appear in consecutive positions; Or "NO", meaning that no permutation of the columns results in a matrix where all rows have the ones in consecutive positions

Example 1: A matrix that has the C1P (namely, it can be appropriately permuted)

	c1	c2	c3	c4	c5
r1	0	1	0	1	1
r2	1	1	0	0	1
r3	0	1	0	1	0
r4	1	0	1	0	0

Choosing the permutation of the columns:
c3, c1, c5, c2, c4
The resulting matrix has C1P

	c3	c1	c5	c2	c4
r1	0	0	1	1	1
r2	0	1	1	1	0
r3	0	0	0	1	1
r4	1	1	0	0	0

The Consecutive Ones Property

CONSECUTIVE ONES PROPERTY (C1P)
Input: A Boolean matrix ***A***
Output: A permutation of the columns of ***A*** such that in the the resulting matrix in each row the 1's appear in consecutive positions; Or "NO", meaning that no permutation of the columns results in a matrix where all rows have the ones in consecutive positions

Example 2: A matrix that does not have the C1P

	c1	c2	c3	c4
r1	0	1	1	0
r2	0	1	0	1
r3	1	1	0	0

Because of r1 and r2, if we want the C1P we must have c2 between c3 and c4. There are two possibilities

	c1	c3	c2	c4	c1
r1	0	1	1	0	0
r2	0	0	1	1	0
r3	1	0	1	0	1

	c1	c4	c2	c3	c1
r1	0	0	1	1	0
r2	0	1	1	0	0
r3	1	0	1	0	1

In either case, wherever we put c1 results in a matrix where the ones in the row 3 are not consecutive!

The Consecutive Ones Property

CONSECUTIVE ONES PROPERTY (C1P)
Input: A Boolean matrix ***A***
Output: A permutation of the columns of ***A*** such that in the the resulting matrix in each row the 1's appear in consecutive positions; Or "NO", meaning that no permutation of the columns results in a matrix where all rows have the ones in consecutive positions

Example 2: A matrix that has more than one solution

	c1	c2	c3	c4	c5
r1	0	1	1	1	0
r2	1	1	0	1	0
r3	1	0	0	0	1

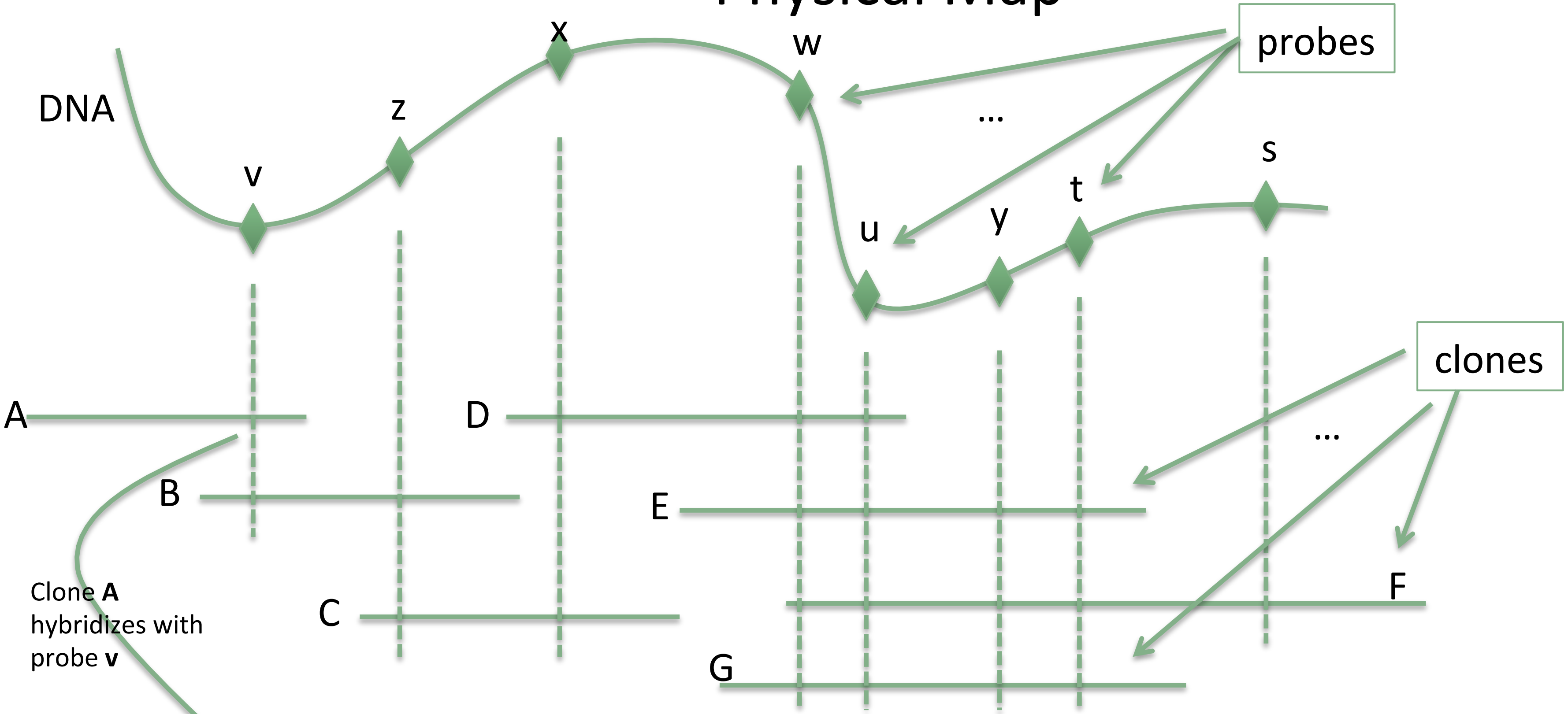
We can put c2 and c4 in any order

In addition for any of the solutions shown reversing the order of the columns gives another solution

	c3	c2	c4	c1	c5
r1	1	1	1	0	0
r2	0	1	1	1	0
r3	0	0	0	1	1

	c3	c4	c2	c1	c5
r1	1	1	1	0	0
r2	0	1	1	1	0
r3	0	0	0	1	1

Physical Map



	t	s	u	v	w	x	y	z
A	0	0	0	1	0	0	0	0
B	0	0	0	1	0	0	0	1
C	0	0	0	0	0	1	0	1
D	0	0	1	0	1	1	0	0
E	1	0	1	0	1	0	1	0
F	1	1	1	0	1	0	1	0
G	1	0	1	0	1	0	1	0

Permuting the columns so to check the CP1 gives an order for the probes, representing the physical map

	v	z	x	w	u	y	t	s
A	1	0	0	0	0	0	0	0
B	1	1	0	0	0	0	0	0
C	0	1	1	0	0	0	0	0
D	0	0	1	1	1	0	0	0
E	0	0	0	1	1	1	1	0
F	0	0	0	1	1	1	1	1
G	0	0	0	1	1	1	1	1

Physical Map vs. CP1 – different models

- Experiments (hybridizations) are fully reliable (correct)
  - The permutation corresponding to the correct physical map is among the solutions of the CP1 problem
  - It is helpful to have algorithms that produce all solutions of the CP1 problem
    - Biologists can then do additional tests in order to discriminate among the different solutions
- Experiments might contain errors
  - False positives are spurious 1’s in the matrix
  - False negatives are missing 1’s in the matrix
  - We’ll deal with errors later

The Consecutive Ones Property

CONSECUTIVE ONES PROPERTY (C1P)

Input: A Boolean matrix ***A***

Output: A permutation of the columns of ***A*** such that in the the resulting matrix in each row the 1's appear in consecutive positions; Or "NO", meaning that no permutation of the columns results in a matrix where all rows have the ones in consecutive positions

Theorem. The C1P problem can be solved in polynomial time

	c1	c2	c3	c4	c5
r1	0	1	0	1	1
r2	1	1	0	0	1
r3	0	1	0	1	0
r4	1	0	1	0	0

Let A be the n x m input matrix

For each i=1, 2, …, we associate row r_i with the set S_i of the columns in which the 1's of row r_i are.

Example:

$r_1 \rightarrow S_1 = \{2, 4, 5\}$

$r_2 \rightarrow S_2 = \{1,2, 5\}$

$r_3 \rightarrow S_3 = \{1, 3\}$

Modeling via overlap graph

Theorem. The C1P problem can be solved in polynomial time

	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9
r_1	1	1	0	1	1	0	1	0	1
r_2	0	1	1	1	1	1	1	1	1
r_3	0	1	0	1	1	0	1	0	1
r_4	0	0	1	0	0	0	0	1	0
r_5	0	0	1	0	0	1	0	0	0
r_6	0	0	0	1	0	0	1	0	0
r_7	0	1	0	0	0	0	1	0	0
r_8	0	0	0	1	1	0	0	0	1

Let A be the n x m input matrix
For each i=1, 2, …, we associate row r_i with the set S_i of the columns in which the 1's of row r_i are.
Example: A is the 8 x 9 matrix on the right

$r_1 \rightarrow S_1 = \{1, 2, 4, 5, 7, 9\}$
$r_2 \rightarrow S_2 = \{2, 3, 4, 5, 6, 7, 8, 9\}$
$r_3 \rightarrow S_3 = \{2, 4, 5, 7, 9\}$
$r_4 \rightarrow S_4 = \{3, 8\}$
$r_5 \rightarrow S_5 = \{3, 6\}$
$r_6 \rightarrow S_6 = \{4, 7\}$
$r_7 \rightarrow S_7 = \{2, 7\}$
$r_8 \rightarrow S_8 = \{4, 5, 9\}$

Definition [***overlap***].
We say that two rows r_i and r_j ***overlap*** if the corresponding sets intersect but none of the two contains the other:

- $S_i \cap S_j \neq \varnothing$ they intersect
- $S_i \setminus S_j \neq \varnothing$ S_j does not contain S_i
- $S_j \setminus S_i \neq \varnothing$ S_i does not contain S_j

Example [***overlap***].

r_1 ***overlaps*** with r_2
r_3 ***overlaps*** with no other row
r_6 ***overlaps*** with r_8 and r_7
r_7 does not ***overlap*** with r_6

Modeling via overlap graph

***Theorem*.** The C1P problem can be solved in polynomial time

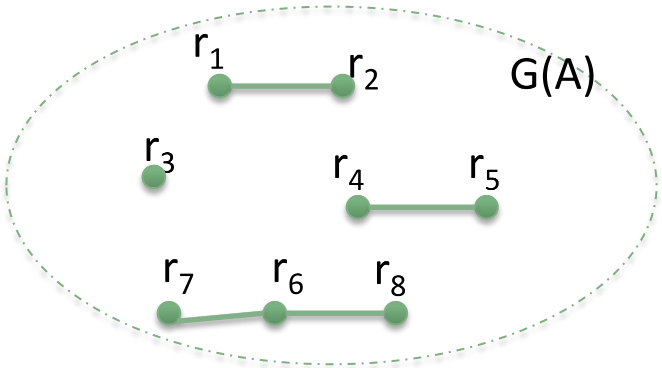
	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9
r_1	1	1	0	1	1	0	1	0	1
r_2	0	1	1	1	1	1	1	1	1
r_3	0	1	0	1	1	0	1	0	1
r_4	0	0	1	0	0	0	0	1	0
r_5	0	0	1	0	0	1	0	0	0
r_6	0	0	0	1	0	0	1	0	0
r_7	0	1	0	0	0	0	1	0	0
r_8	0	0	0	1	1	0	0	0	1

Let A be the n x m input matrix
For each i=1, 2, …, we associate row r_i with the set S_i o the columns in which the 1's of row r_i are.
Example: A is the 8 x 9 matrix on the right

$r_1 \rightarrow S_1 = \{1, 2, 4, 5, 7, 9\}$
$r_2 \rightarrow S_2 = \{2, 3, 4, 5, 6, 7, 8, 9\}$
$r_3 \rightarrow S_3 = \{2, 4, 5, 7, 9\}$
$r_4 \rightarrow S_4 = \{3, 8\}$
$r_5 \rightarrow S_5 = \{3, 6\}$
$r_6 \rightarrow S_6 = \{4, 7\}$
$r_7 \rightarrow S_7 = \{2, 7\}$
$r_8 \rightarrow S_8 = \{4, 5, 9\}$

Definition [***Overlap Graph***].
Given the matrix A, the overlap graph of A, denoted G(A), is an ***undirected*** graph whose vertices represent the columns of A and whose edges represent the overlap between two rows

Example [***overlap graph of A***]

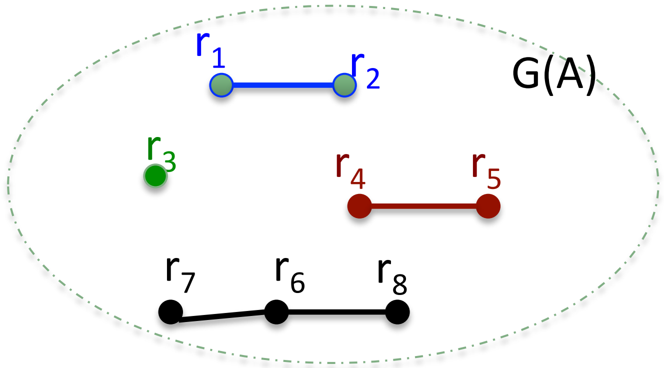


Modeling via overlap graph

Theorem. The C1P problem can be solved in polynomial time

	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	
r_1	1	1	0	1	1	0	1	0	1	A_1
r_2	0	1	1	1	1	1	1	1	1	
r_3	0	1	0	1	1	0	1	0	1	A_2
r_4	0	0	1	0	0	0	0	1	0	A_3
r_5	0	0	1	0	0	1	0	0	0	
r_6	0	0	0	1	0	0	1	0	0	A_4
r_7	0	1	0	0	0	0	1	0	0	
r_8	0	0	0	1	1	0	0	0	1	

Example [***overlap graph of A***]



Each connected component of the ***Overlap Graph*** represents a submatrix A' of A such that for each row r' in A' there is another row r'' in A' that overlaps with r'.

Lemma.

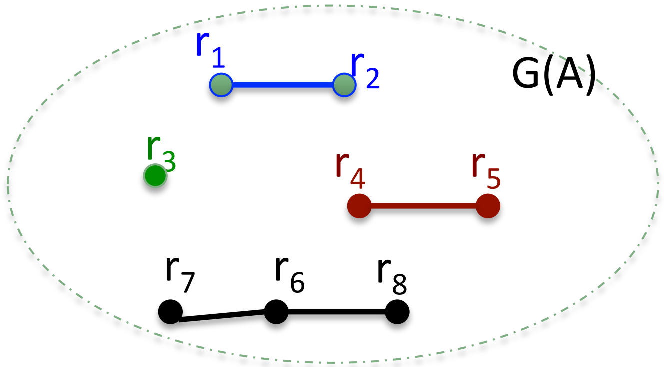
Let A_1, …, A_k, be the submatrices (subsets of rows) of A corresponding to the connected components of G(A). Then A has the C1P if and only if for each i=1,…,k, A_i has the C1P

Modeling via overlap graph

Theorem. The C1P problem can be solved in polynomial time

	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	
r_1	1	1	0	1	1	0	1	0	1	A_1
r_2	0	1	1	1	1	1	1	1	1	
r_3	0	1	0	1	1	0	1	0	1	A_2
r_4	0	0	1	0	0	0	0	1	0	A_3
r_5	0	0	1	0	0	1	0	0	0	
r_6	0	0	0	1	0	0	1	0	0	A_4
r_7	0	1	0	0	0	0	1	0	0	
r_8	0	0	0	1	1	0	0	0	1	

Example [***overlap graph of A***]



Lemma.

Let A_1, …, A_k, be the submatrices (subsets of rows) of A corresponding to the connected components of G(A). Then A has the C1P if and only if for each i=1,…,k, A_i has the C1P

Proof: Clearly if A has C1P then each A_i has the property: simply choose the same permutation of columns showing the property for A.

Therefore if we find one A_i not having C1P it follows that A doesn't have it.

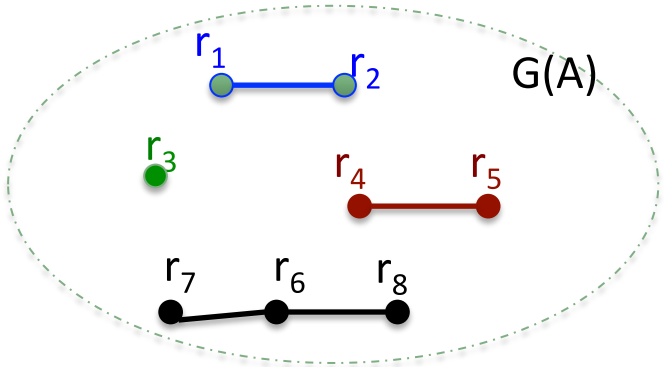
Modeling via overlap graph

Theorem. The C1P problem can be solved in polynomial time

	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	
r_1	1	1	0	1	1	0	1	0	1	A_1
r_2	0	1	1	1	1	1	1	1	1	
r_3	0	1	0	1	1	0	1	0	1	A_2
r_4	0	0	1	0	0	0	0	1	0	A_3
r_5	0	0	1	0	0	1	0	0	0	
r_6	0	0	0	1	0	0	1	0	0	A_4
r_7	0	1	0	0	0	0	1	0	0	
r_8	0	0	0	1	1	0	0	0	1	

1. We will first look at how to verify, independently, that the submatrices (components) A_1, A_2, A_3, A_4, have the C1P

Example [***overlap graph of A***]



Lemma.
Let A_1, …, A_k, be the submatrices (subsets of rows) of A corresponding to the connected components of G(A). Then A has the C1P if and only if for each i=1,…,k, A_i has the C1P

Proof: Clearly if A has C1P then each A_i has the property: simply choose the same permutation of columns showing the property for A.
Therefore if we find one A_i not having C1P it follows that A doesn't have it.

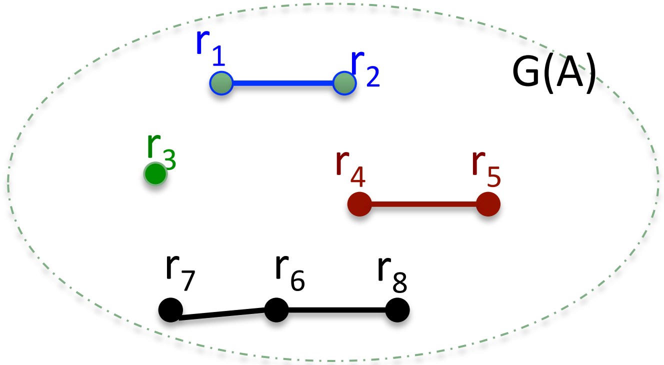
Modeling via overlap graph

Theorem. The C1P problem can be solved in polynomial time

		c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9
A_1	r_1	1	1	0	1	1	0	1	0	1
	r_2	0	1	1	1	1	1	1	1	1
A_2	r_3	0	1	0	1	1	0	1	0	1
A_3	r_4	0	0	1	0	0	0	0	1	0
	r_5	0	0	1	0	0	1	0	0	0
A_4	r_6	0	0	0	1	0	0	1	0	0
	r_7	0	1	0	0	0	0	1	0	0
	r_8	0	0	0	1	1	0	0	0	1

1. We will first look at how to verify, independently, that the submatrices (components) A_1, A_2, A_3, A_4, have the C1P
2. Then we will prove that if A_1, A_2, …, A_k, have the C1P, we can combine the permutations into one showing that A has C1P

Example [***overlap graph of A***]



Lemma.
Let A_1, …, A_k, be the submatrices (subsets of rows) of A corresponding to the connected components of G(A). Then A has the C1P if and only if for each i=1,…,k, A_i has the C1P

Proof: Clearly if A has C1P then each A_i has the property: simply choose the same permutation of columns showing the property for A.
Therefore if we find one A_i not having C1P it follows that A doesn't have it.

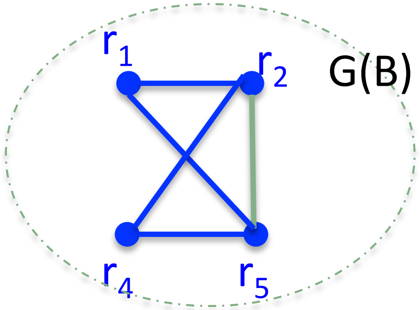
Solving the C1P for one component

Let us consider as an example a new matrix B with only one component

	c_1	c_2	c_3	c_4	c_5	c_6
r_1	1	1	0	1	0	1
r_2	0	1	1	0	1	1
r_3	0	1	0	1	1	1
r_4	1	0	0	1	0	1

$S_1=\{1,2,4,6\}$
$S_2=\{2,3,5,6\}$
$S_3=\{2,4,5,6\}$
$S_4=\{1,4,6\}$

Here is the ***overlap graph of B***



We check the C1P one row at a time

Step 1 – the first row

Trivially one row can always be accommodated by putting all the ones in consecutive positions

We start with r_1, put c_1,c_2,c_4,c_6 consecutively and in order to indicate that every permutation is equally good, we consider it one (multi)-column labeled by the set {c1,c2,c4,c6}

		$\{c_1, c_2, c_4, c_6\}$				
r_1		1	1	1	1	

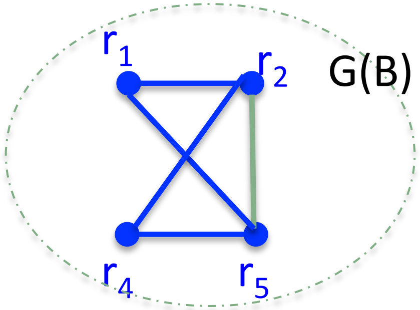
Solving the C1P for one component

Let us consider as an example a new matrix B with only one component

	c_1	c_2	c_3	c_4	c_5	c_6
r_1	1	1	0	1	0	1
r_2	0	1	1	0	1	1
r_3	0	1	0	1	1	1
r_4	1	0	0	1	0	1

$S_1=\{1,2,4,6\}$
$S_2=\{2,3,5,6\}$
$S_3=\{2,4,5,6\}$
$S_4=\{1,4,6\}$

Here is the ***overlap graph of B***



We check the C1P one row at a time

Step 2 – the second row

Also a second row can always be accommodated.
There is a row in the component that overlaps with the (first) row already taken care of.
For instance, in this case we could take r_2
We modify the matrix we have for r_1, so to have the columns in $S_1 \cap S_2$ ***= $\{c_2,c_6\}$ consecutively, and on one side the columns of*** $S_1 \setminus S_2$ = ***$\{c_1,c_4\}$ and on the other side the columns of*** $S_2 \setminus S_1$ = ***$\{c_3,c_5\}$***
Again the set notation indicates that any permutation of those columns will do

		$\{c_1, c_2, c_4, c_6\}$				
r_1		1	1	1	1	

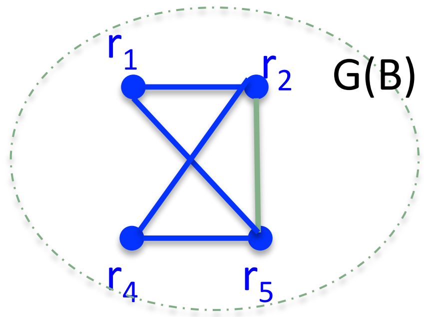
	$\{c_1, c_4\}$		$\{c_2, c_6\}$		$\{c_3, c_5\}$	
r_1	1	1	1	1	0	0
r_2	0	0	1	1	1	1

Solving the C1P for one component

Let us consider as an example a new matrix B with only one component

	c_1	c_2	c_3	c_4	c_5	c_6	
r_1	1	1	0	1	0	1	$S_1=\{1,2,4,6\}$
r_2	0	1	1	0	1	1	$S_2=\{2,3,5,6\}$
r_3	0	1	0	1	1	1	$S_3=\{2,4,5,6\}$
r_4	1	0	0	1	0	1	$S_4=\{1,4,6\}$

Here is the ***overlap graph of B***



We check the C1P one row at a time

Step i > 2 – the third and following rows

We select a new row a that overlaps with one of the rows already processed, b, and another already processed, c, which overlaps with b. We try to find out on which side of ***b***, the row ***a*** should go, by considering the constraints imposed by the overlaps of ***a*** with ***b***, of ***a*** with ***c*** (if any) and of ***b*** with ***c***.

		$\{c_1, c_2, c_4, c_6\}$				
r_1		1	1	1	1	

↓

	$\{c_1, c_4\}$		$\{c_2, c_6\}$		$\{c_3, c_5\}$	
r_1	1	1	1	1	0	0
r_2	0	0	1	1	1	1

For instance in this example, when we try to add row r_3, we can choose to compare with r_2 which overlaps with r_3 and with r_1 which overlaps with r_2

Solving the C1P for one component

Let us consider as an example a new matrix B with only one component

	c_1	c_2	c_3	c_4	c_5	c_6
r_1	1	1	0	1	0	1
r_2	0	1	1	0	1	1
r_3	0	1	0	1	1	1
r_4	1	0	0	1	0	1

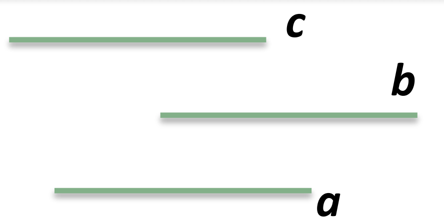
$S_1=\{1,2,4,6\}$
$S_2=\{2,3,5,6\}$
$S_3=\{2,4,5,6\}$
$S_4=\{1,4,6\}$

Assume the C1P holds, then there is permutation of columns such that ***a***, ***b***, ***c*** have ones consecutively. W.l.o.g. ***c*** sticks out to the left of ***b*** then either ***a*** also sticks out to the left of ***b*** (same side as ***c***) or to the right of ***b*** (opposite side **w**ith **r**ispect **t**o (w.r.t) ***c***)

We check the C1P one row at a time

Step i > 2 – the third and following rows

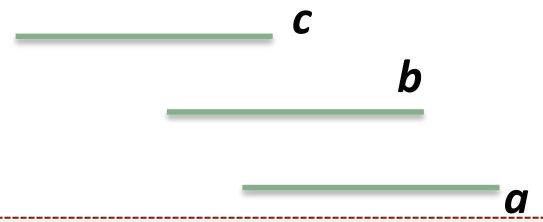
We select a new row a that overlaps with one of the rows already processed, b, and another already processed, c, which overlaps with b.



Case1: ***a*** w.r.t ***b*** to the same side as ***c***

Case1 if and only if: $a \cdot c \geq \min\{a \cdot b, b \cdot c\}$

Case2 if and only if: $a \cdot c < \min\{a \cdot b, b \cdot c\}$



Case2: ***a*** w.r.t ***b*** to the opposite side w.r.t. ***c***

Definition

For rows r_i and r_j let $r_i \cdot r_j$ denote $|S_i \cap S_j|$

Solving the C1P for one component

Let us consider as an example a new matrix B with only one component

	c_1	c_2	c_3	c_4	c_5	c_6	
r_1	1	1	0	1	0	1	$S_1=\{1,2,4,6\}$
r_2	0	1	1	0	1	1	$S_2=\{2,3,5,6\}$
r_3	0	1	0	1	1	1	$S_3=\{2,4,5,6\}$
r_4	1	0	0	1	0	1	$S_4=\{1,4,6\}$

	$\{c_1, c_4\}$		$\{c_2,c_6\}$		$\{c_3, c_5\}$	
r_1	1	1	1	1	0	0
r_2	0	0	1	1	1	1

We put r_3 1's to the side of r_2 according to Case1 and in order to satisfy the intersection size with r_2

	$\{c_1, c_4\}$		$\{c_2,c_6\}$		$\{c_3, c_5\}$	
r_1	1	1	1	1	0	0
r_2	0	0	1	1	1	1
r_3		1	1	1	1	

We redefine columns order according to 1's in r_3 . ***If wasn't possible then C1P wouldn't hold***

	c_1	c_4	$\{c_2,c_6\}$		c_5	c_3
r_1	1	1	1	1	0	0
r_2	0	0	1	1	1	1
r_3	0	1	1	1	1	0

We check the C1P one row at a time

Step i > 2 – the third and following rows

We select a new row a that overlaps with one of the rows already processed, b, and another already processed, c, which overlaps with b.

For instance in the example, when we try to add row r_3 , comparing with r_2 which overlaps with r_3 and with r_1 which overlaps with r_2

- $r_3 \cdot r_1 = 3, \quad r_3 \cdot r_2 = 3, r_2 \cdot r_1 = 2$ then
- $r_3 \cdot r_1 \geq \min\{r_3 \cdot r_2, r_2 \cdot r_1\}$, ***i.e we have Case1***
- ***Then*** row r_3 must be on the same side as r_1 w.r.t. r_2

Solving the C1P for one component

The matrix B

	c_1	c_2	c_3	c_4	c_5	c_6	
r_1	1	1	0	1	0	1	$S_1=\{1,2,4,6\}$
r_2	0	1	1	0	1	1	$S_2=\{2,3,5,6\}$
r_3	0	1	0	1	1	1	$S_3=\{2,4,5,6\}$
r_4	1	0	0	1	0	1	$S_4=\{1,4,6\}$

We check the C1P one row at a time

.

In the example, we finally try to add row $\boldsymbol{r_4}$, comparing with $\boldsymbol{r_3}$ which overlaps with $\boldsymbol{r_4}$ and with $\boldsymbol{r_2}$ which overlaps with $\boldsymbol{r_3}$

- $r_4 \cdot r_2 = 1,\quad r_4 \cdot r_3 = 2,\ r_2 \cdot r_3 = 2$ then
- $r_4 \cdot r_1 < \min\{r_4 \cdot r_3,\ r_2 \cdot r_3\}$, *i.e we have* ***Case2***
- ***Then*** row $\boldsymbol{r_4}$ must be on the opposite side of r_2 w.r.t. $\boldsymbol{r_3}$

	c_1	c_4	$\{c_2,c_6\}$		c_5	c_3
r_1	1	1	1	1	0	0
r_2	0	0	1	1	1	1
r_3	0	1	1	1	1	0

We put the 1's of r_4 to the side of r_3 according to Case2 and in order to satisfy the intersection size with r_3

	c_1	c_4	$\{c_2,c_6\}$		c_5	c_3
r_1	1	1	1	1	0	0
r_2	0	0	1	1	1	1
r_3	0	1	1	1	1	0
r_4	1	1	1			

We redefine columns order (for c_2,c_6) according to 1's in r_3.

	c_1	c_4	c_6	c_2	c_5	c_3
r_1	1	1	1	1	0	0
r_2	0	0	1	1	1	1
r_3	0	1	1	1	1	0
r_4	1	1	1	0	0	0

Solving the C1P for one component

The matrix B

	c_1	c_2	c_3	c_4	c_5	c_6	
r_1	1	1	0	1	0	1	$S_1=\{1,2,4,6\}$
r_2	0	1	1	0	1	1	$S_2=\{2,3,5,6\}$
r_3	0	1	0	1	1	1	$S_3=\{2,4,5,6\}$
r_4	1	0	0	1	0	1	$S_4=\{1,4,6\}$

We check the C1P one row at a time

.

In the example, we finally try to add row $\boldsymbol{r_4}$, comparing with $\boldsymbol{r_3}$ which overlaps with $\boldsymbol{r_4}$ and with $\boldsymbol{r_2}$ which overlaps with $\boldsymbol{r_3}$

- $\boldsymbol{r_4 \cdot r_2 = 1, \quad r_4 \cdot r_3 = 2, r_2 \cdot r_3 = 2}$ then
- $\boldsymbol{r_4 \cdot r_1 < \min\{r_4 \cdot r_3, r_2 \cdot r_3\}}$, ***i.e we have Case2***
- ***Then*** row $\boldsymbol{r_4}$ must be on the opposite side of r_2 w.r.t $\boldsymbol{r_3}$

	c_1	c_4	$\{c_2,c_6\}$		c_5	c_3
r_1	1	1	1	1	0	0
r_2	0	0	1	1	1	1
r_3	0	1	1	1	1	0

We put r_4 1's to the side of r_3 according to Case2 and in order to satisfy the intersection size with r_3

	c_1	c_4	$\{c_2,c_6\}$		c_5	c_3
r_1	1	1	1	1	0	0
r_2	0	0	1	1	1	1
r_3	0	1	1	1	1	0
r_4	1	1	1			

We redefine columns order (for c_2,c_6) according to 1's in r_3 .

	c_1	c_4	c_6	c_2	c_5	c_3
r_1	1	1	1	1	0	0
r_2	0	0	1	1	1	1
r_3	0	1	1	1	1	0
r_4	1	1	1	0	0	0

We have found a permutation of the columns of B, namely: c1, c4, c6, c2, c5, c3 Showing that the C1P holds.

...**back to our theorem** (and initial example)

	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	
r_1	1	1	0	1	1	0	1	0	1	A_1
r_2	0	1	1	1	1	1	1	1	1	
r_3	0	1	0	1	1	0	1	0	1	A_2
r_4	0	0	1	0	0	0	0	1	0	A_3
r_5	0	0	1	0	0	1	0	0	0	
r_6	0	0	0	1	0	0	1	0	0	A_4
r_7	0	1	0	0	0	0	1	0	0	
r_8	0	0	0	1	1	0	0	0	1	

1. We will first look at how to verify, independently, that the submatrices (components) A_1, A_2, A_3, A_4, have the C1P

By applying the method of the previous slide to the components A_1, A_2, A_3, A_4, we find that they all have the C1P and get the following permutations (for each submatrix we accommodated we are reporting only the columns with 1's)

A_1

	c_1	$\{c_2, c_4, c_5, c_7, c_9\}$					$\{c_3, c_6, c_8\}$		
r_1	1	1	1	1	1	1	0	0	0
r_2	0	1	1	1	1	1	1	1	1

A_2

		$\{c_2, c_4, c_5, c_7, c_9\}$					
r_3		1	1	1	1	1	

A_1

	c_6	c_3	c_8
r_4	0	1	1
r_2	1	1	0

A_4

	c_2	c_7	c_4	$\{c_5, c_9\}$	
r_6	0	1	1	0	0
r_7	1	1	0	0	0
r_8	0	0	1	1	1

…back to our initial example

	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	
r_1	1	1	0	1	1	0	1	0	1	A_1
r_2	0	1	1	1	1	1	1	1	1	
r_3	0	1	0	1	1	0	1	0	1	A_2
r_4	0	0	1	0	0	0	0	1	0	A_3
r_5	0	0	1	0	0	1	0	0	0	
r_6	0	0	0	1	0	0	1	0	0	A_4
r_7	0	1	0	0	0	0	1	0	0	
r_8	0	0	0	1	1	0	0	0	1	

A_1

	c_1	$\{c_2, c_4, c_5, c_7, c_9\}$					$\{c_3, c_6, c_8\}$		
r_1	1	1	1	1	1	1	0	0	0
r_2	0	1	1	1	1	1	1	1	1

A_2

		$\{c_2, c_4, c_5, c_7, c_9\}$					
r_3		1	1	1	1	1	

A_3

	c_6	c_3	c_8
r_4	0	1	1
r_2	1	1	0

A_4

	c_2	c_7	c_4	$\{c_5, c_9\}$	
r_6	0	1	1	0	0
r_7	1	1	0	0	0
r_8	0	0	1	1	1

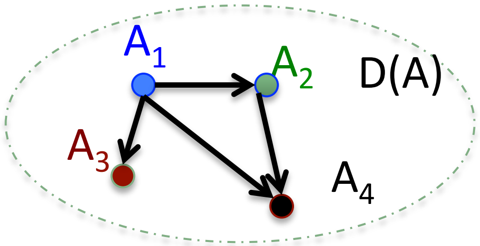
2. Then we will prove that if $A_1, A_2, \ldots, A_k$, have the C1P, we can combine the permutations into one showing that A has C1P

Definition.

The Containment graph of A, D(A) is the ***directed*** graph defined by:

- the components of G(A) are the vertices of D(A)
- there is an edge $A_i \longrightarrow A_j$ if in A_j there is a row that is contained in some row of A_i

Example [***containment graph of A***]



…back to our initial example

	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	
r_1	1	1	0	1	1	0	1	0	1	A_1
r_2	0	1	1	1	1	1	1	1	1	
r_3	0	1	0	1	1	0	1	0	1	A_2
r_4	0	0	1	0	0	0	0	1	0	A_3
r_5	0	0	1	0	0	1	0	0	0	
r_6	0	0	0	1	0	0	1	0	0	A_4
r_7	0	1	0	0	0	0	1	0	0	
r_8	0	0	0	1	1	0	0	0	1	

2. Then we will prove that if $A_1, A_2, \ldots, A_k$, have the C1P, we can combine the permutations into one showing that A has C1P

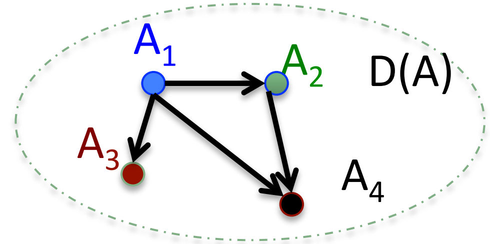
***Lemma*.**
Let $A_i \longrightarrow A_j$ be an edge of the containment graph. For each row r of A_j denote by R(r) the set of rows in A_i that contain r.
For any r, r' in A_j it holds that R(r) = R(r').

Corollary
The lemma says that the rows of A_j are all contained in exactly the same rows of A_i. Thus, all these rows of A_i must have 1's in the columns where rows of A_j have a 1. Therefore, we can find a block of columns in the permutation of A_i where we can insert A_j

Definition.
The Containment graph of A, D(A) is the ***directed*** graph defined by:

- the components of G(A) are the vertices of D(A)
- there is an edge $A_i \longrightarrow A_j$ if in A_j there is a row that is contained in some row of A_i

Example [***containment graph of A***]



…Putting together the component-permutations

A_1

	c_1	$\{c_2, c_4, c_5, c_7, c_9\}$					$\{c_3, c_6, c_8\}$		
r_1	1	1	1	1	1	1	0	0	0
r_2	0	1	1	1	1	1	1	1	1

A_2

		$\{c_2, c_4, c_5, c_7, c_9\}$					
r_3		1	1	1	1	1	

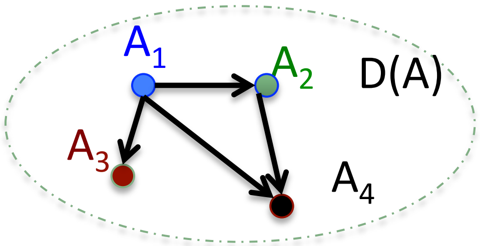
A_3

	c_6	c_3	c_8
r_4	0	1	1
r_2	1	1	0

A_4

	c_2	c_7	c_4	$\{c_5, c_9\}$	
r_6	0	1	1	0	0
r_7	1	1	0	0	0
r_8	0	0	1	1	1

Containment graph of A



A Topological Sorting of ***D(A)***:
A_1, A_2, A_4, A_3

By the ***Corollary***, we can combine permutations for the component-submatrices, following a *topological sorting* of D(A). Note that D(A) is a DAG (there cannot be cycles): it can be *topologically sorted*. In the example, we can choose A_1, A_2, A_4, A_3

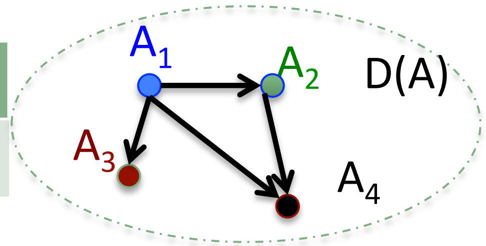
…Putting together the component-permutations

A_1

	c_1	$\{c_2, c_4, c_5, c_7, c_9\}$					$\{c_3, c_6, c_8\}$		
r_1	1	1	1	1	1	1	0	0	0
r_2	0	1	1	1	1	1	1	1	1

A_2

		$\{c_2, c_4, c_5, c_7, c_9\}$					
r_3		1	1	1	1	1	



Containment graph of A

A Topological Sorting of ***D(A)***:

A_1, A_2, A_4, A_3

Let's start with A_1 and A_2

$A_1 \longrightarrow A_2$ implies that the rows of A_1 containing all the rows of A_2 have a block of columns with all 1's (see above).

No other row in A_1 has a 1 in these columns. Thus these columns can be permuted according to A_2 permutation.

Then we get the new matrix ***$B = A_1 \oplus A_2$***

	c_1	$\{c_2, c_4, c_5, c_7, c_9\}$					$\{c_3, c_6, c_8\}$		
r_1	1	1	1	1	1	1	0	0	0
r_2	0	1	1	1	1	1	1	1	1
r_3	0	1	1	1	1	1	0	0	0

Following the topologica order, we are now going to merge ***B*** with ***A_4***

Note that B inherits the containment properties of ***A_1*** and ***A_2*** with respect to the remaining components/submatrices. If we substituted ***A_1*** and ***A_2*** in ***D(A)*** with ***B***, we would have ***$B \longrightarrow A_4$***

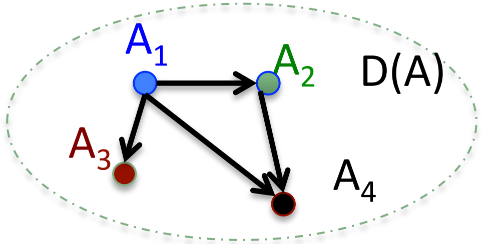
…Putting together the component-permutations

$B = A_1 \oplus A_2$

	c_1	$\{c_2, c_4, c_5, c_7, c_9\}$					$\{c_3, c_6, c_8\}$		
r_1	1	1	1	1	1	1	0	0	0
r_2	0	1	1	1	1	1	1	1	1
r_3	0	1	1	1	1	1	0	0	0

A_4

	c_2	c_7	c_4	$\{c_5, c_9\}$	
r_6	0	1	1	0	0
r_7	1	1	0	0	0
r_8	0	0	1	1	1



A Topological Sorting of ***D(A)***:
A_1, A_2, A_4, A_3

Let's merge B and A_4

$\boldsymbol{B \longrightarrow A_4}$ implies that the rows of ***B*** containing all the rows of $\boldsymbol{A_4}$ have a block of columns with all 1's (see above). No other row in ***B*** ---in this case there aren't any--- can have a 1 in these columns. Thus these columns can be permuted according to $\boldsymbol{A_4}$ permutation. Then we get the new matrix $\boldsymbol{C = B \oplus A_4 = A_1 \oplus A_2 \oplus A_4}$

	c_1	$\{c_2, c_4, c_5, c_7, c_9\}$					$\{c_3, c_6, c_8\}$		
r_1	1	1	1	1	1	1	0	0	0
r_2	0	1	1	1	1	1	1	1	1
r_3	0	1	1	1	1	1	0	0	0

	c_2	c_7	c_4	$\{c_5, c_9\}$	
r_6	0	1	1	0	0
r_7	1	1	0	0	0
r_8	0	0	1	1	1

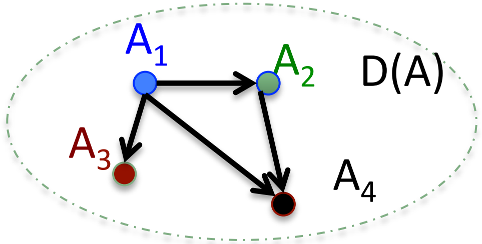
$\boldsymbol{C = B \oplus A_4 = A_1 \oplus A_2 \oplus A_4}$

	c_1	c_2	c_7	c_4	$\{c_5, c_9\}$		$\{c_3, c_6, c_8\}$		
r_1	1	1	1	1	1	1	0	0	0
r_2	0	1	1	1	1	1	1	1	1
r_3	0	1	1	1	1	1	0	0	0
r_6	0	0	1	1	0	0	0	0	0
r_7	0	1	1	0	0	0	0	0	0
r_8	0	0	0	1	1	1	0	0	0

…Putting together the component-permutations

Let's merge C = $A_1 \oplus A_2 \oplus A_4$ ***and*** A_3

$A_1 \longrightarrow A_3$ ***implies*** that the rows of C containing all the rows of A_3 (coinciding with those originally in A_1) have a block of columns with all 1's (see below). No other row in C can have a 1 in these columns. Thus these columns can be permuted according to A_3 permutation.



A Topological Sorting of ***D(A)***:
A_1, A_2, A_4, A_3

	c_1	c_2	c_7	c_4	$\{c_5, c_9\}$		$\{c_3, c_6, c_8\}$		
r_1	1	1	1	1	1	1	0	0	0
r_2	0	1	1	1	1	1	1	1	1
r_3	0	1	1	1	1	1	0	0	0
r_6	0	0	1	1	0	0	0	0	0
r_7	0	1	1	0	0	0	0	0	0
r_8	0	0	0	1	1	1	0	0	0

A_3

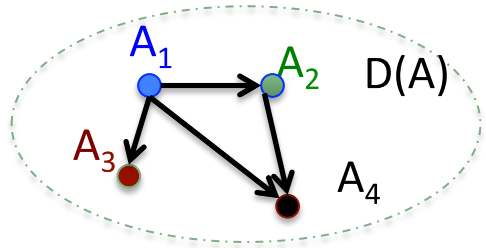
	c_6	c_3	c_8
r_4	0	1	1
r_2	1	1	0

	c_1	c_2	c_7	c_4	$\{c_5, c_9\}$		c_6	c_3	c_8
r_1	1	1	1	1	1	1	0	0	0
r_2	0	1	1	1	1	1	1	1	1
r_3	0	1	1	1	1	1	0	0	0
r_6	0	0	1	1	0	0	0	0	0
r_7	0	1	1	0	0	0	0	0	0
r_8	0	0	0	1	1	1	0	0	0
r_4	0	0	0	0	0	0	0	1	1
r_2	0	0	0	0	0	0	1	1	0

This is the final matrix showing C1P

Final Observations

	c_1	c_2	c_7	c_4	$\{c_5, c_9\}$		c_6	c_3	c_8
r_1	1	1	1	1	1	1	0	0	0
r_2	0	1	1	1	1	1	1	1	1
r_3	0	1	1	1	1	1	0	0	0
r_6	0	0	1	1	0	0	0	0	0
r_7	0	1	1	0	0	0	0	0	0
r_8	0	0	0	1	1	1	0	0	0
r_4	0	0	0	0	0	0	0	1	1
r_2	0	0	0	0	0	0	1	1	0



A Topological Sorting of ***D(A)***:
A_1, A_2, A_4, A_3

This is the final matrix showing C1P

Observations

- We have permuted the rows in the process, but if needed, we can repermute them in the original order, keeping the C1P.
- The columns c_5, c_9 are together indicating that we have found no constraint about their relative order and they can be permuted in any order (giving more solutions)
- More solutions can also arise from different topological sortings.

Complexity of the Algorithm

A. Overlaps and Containments is $O(n^2 m)$

- For a pair of rows r_i, r_j, we can compute the size of their intersection $|S_i \cap S_j|$ in time $O(m)$: while scanning the two rows, count the columns where they both have a 1
  1. If $0 < |S_i \cap S_j| < \min\{|S_i|, |S_j|\}$ then r_i and r_j overlap
  2. If $|S_i \cap S_j| = 0$ r_i and r_j are disjoint
  3. If $|S_i \cap S_j| = |S_i|$ then r_i is contained in r_j
- We can compute $|S_i \cap S_j|$ *for all pairs* i, j in $O(n^2 m)$ and from this we can build the graphs G(A) and D(A) in time $O(n^2)$ by using (1-3)

Complexity of the Algorithm

B. Checking the C1P in the components of the overlap graph G(A) is also $O(n^2 m)$

- After building G(A), we can find all the components in $O(n^2)$ by BFS, since this graph has n vertices and at most n^2 edges
- In each connected component A_i we find the permutation of the columns by processing one row at a time and checking its overlap against 3 other rows. Since we already have computed the overlap for any pair of rows (see above) processing each rows takes
  - constant time (or at most O(m) if we want to recompute the overlaps with the two rows we compare it to) to verify that the new row can be added preserving the C1P
- In total processing each connected components takes O(#row in the component) and in total O(n) (times m if we recompute the overlaps)

Complexity of the Algorithm

C. Merging the components is also doable in $O(n^2\ m)$

1. Topological sorting on D(A) costs $O(n^2)$ as this graph has n vertices and at most n^2 edges
2. If for each component A_i we store the first column with a one, we can merge it to the matrix B containing the merge so far in O(#rows of $A_i \cdot m$)
   - We have to reorder the labels of the columns of B which are going to coincide with the columns of A_i that contain 1's
   - We have to copy in B the rows of A_i
3. Therefore merging all the components takes $O(n \cdot m)$
   - Since summing the cost of 2. over all components we have n (the total number of rows, times m.

Complexity of the Algorithm - Summary

A. Computing Overlaps and Containments is done in $O(n^2\ m)$

B. Checking the C1P in the components of the overlap graph G(A) is also $O(n^2\ m)$

C. Merging the components is also doable in $O(n^2\ m)$

Therefore the whole algorithm takes $O(n^2\ m)$

- It returns a matrix storing information on several solutions (if more are possible)
- If the C1P does not hold we can return the triple of rows that show that no permutation can do!

C1P Algorithm – computing intersections

Overlaps_and_containment (A)

Input: a Boolean Matrix A[nxm] // n rows and m columns

Output: an nxn matrix I[nxn] where I[i,j] = $|S_i \cap S_j|$

```
For i = 1 to n
        For  j = 1 to n
            I[i,j] ← 0
            For  k = 1 to m
                        I[i,j] ← I[i,j] + A[i, k]*A[j,k]   // add 1 iff  A[i, k]=A[j,k]=1
return  I
```

Overlap(i,j)

Output: true or false. True iff r_i overlaps r_j

```
If  I[i,j] > 0 and I[i,j] < min{I[i,i], I[j, j]}
        return true
return false
```

Containment(i,j)

Output: true or false. True iff r_i contains r_j

```
If  I[i,j] = I[j, j]}
        return true
return false
```

C1P Algorithm – building G(A)

Build G (A)

Input: a Boolean Matrix A[nxm] // n rows and m columns

Output: The lists of adjacencies of the overlap graph G(A)

I ← ***Overlaps_and_containment*** (A) /computes the intersections of all pairs of rows

Create n empty lists L[1], …, L[n] // L[i] is the list of adjacencies for vertex r_i

For *i = 1 to n*

    For *j = 1 to n*

        if Overlap(i,j)

            Add i to the list L[j] and j to the list L[i]

return L[1], …, L[n]

Find Components(L[1], …, L[n])

Input: *the graph G(A) in the form of the lists of adjacencies of its vertices*

Output: the number of connected components of G(A) and for each one the set of rows in it

For *i = 1 to n*

    marked[i] ← false

k ← 0,

For *i = 1 to n*

    if *marked[i] = false*

        k ← k+1 // k counts the connected components

        Set A[k] ← ∅ // A[k] is the set of rows in k-th component

        perform ***BFS*** *on G(A) starting from r_i , mark each traversed vertex and add it to A[k]*

return k, A[1], …, A[k]

C1P Algorithm – C1P in Components of G(A)

Check_C1P_in_component (A, I, B, L[1], ..,L[n])

Input: a Boolean Matrix A[n×m] , the matrix of row-intersections I, a component B of G(A), G(A) as lists of adj.

Output: A permutation of the columns where at least one row in B has a 1; NO if C1P does not hold for B

If B has only one row r

    return the list of columns where r has 1's

If B has at least two rows r_i and r_j

    T1 ⇽ columns in $S_i \setminus S_j$; T2 ⇽ columns in $S_i \cap S_j$; T3 ⇽ columns in $S_j \setminus S_i$; **T** = (T1, T2, T3)

    mark the two rows as ***processed***

    store that r_i is on left of r_j // could use an additional matrix for this

    store in an array **First** the list in **T** with the first column of r_i and that with the first column of r_j

    if |B| = 2

        return the list of lists **T** = T1, T2, T3

    For *each remaining row **a** in B*

        Using the adjacency lists, find rows ***b*** and ***c*** marked as ***processed*** and

        *such that **a** overlaps with **b** and **b** overlaps with **c***

        if a.c < min{a.b, b.c} // use matrix I to check this condition in O(1)

            split list in **T** in order to have columns in $S_a \setminus S_b$ separated from $S_a \cap S_b$

            and on the opposite side w.r.t. columns in $S_c \setminus S_b$ *// With array First can do this in O(m)*

            if this is not possible return NO *// i.e. columns of a cannot be in consecutive lists of **T***

        else **//** a.c >= min{a.b, b.c}

            split lists in **T** in order to have columns in $S_a \setminus S_b$ separated from $S_a \cap S_b$

            and on the same side w.r.t. columns in $S_c \setminus S_b$ *// With Array First can do this in O(m)*

            if this is not possible return NO *// i.e. columns of a cannot be in consecutive lists of **T***

return T

C1P Algorithm – Merging Components of G(A)

To be completed…