Filtering Algorithms based on Graph Theory

Jean-Charles REGIN
ILOG, Sophia Antipolis
regin@ilog.fr

Plan

- CP
- Graph Theory: Flows
- Global cardinality constraints: based on flow algorithms
- Global cardinality with costs: based on minimum cost flow
- Alldiff constraint: matching
- Symmetric alldiff constraint: symmetric matching
- Conclusion

Plan

- **CP**
- Graph Theory: Flows
- Global cardinality constraints: based on flow algorithms
- Global cardinality with costs: based on minimum cost flow
- Alldiff constraint: matching
- Symmetric alldiff constraint: symmetric matching
- Conclusion

Constraint Programming

- In CP a problem is defined from:
  - variables with possible values (domain)
  - constraints
- Domain can be discrete or continuous, symbolic values or numerical values
- Constraints express properties that have to be satisfied

Problem = conjunction of sub-problems

- In CP a problem can be viewed as a conjunction of sub-problems that we are able to solve
- A sub-problem can be trivial: $x < y$ or complex: search for a feasible flow
- A sub-problem = a constraint

Constraints

- Predefined constraints: arithmetic ($x < y$, $x = y +z$, $|x-y| > k$, alldiff, cardinality, sequence …
- Constraints given in extension by the list of allowed (or forbidden) combinations of values
- user-defined constraints: any algorithm can be encapsulated
- Logical combination of constraints using OR, AND, NOT, XOR operators. Sometimes called meta-constraints

Filtering

- We are able to solve a sub-problem: a method is available
- CP uses this method to remove values from domain that do not belong to a solution of this sub-problem: **filtering or domain-reduction**
- E.g: x < y and D(x)=[10,20], D(y)=[5,15]
  => D(x)=[10,14], D(y)=[11,15]

Filtering

- A filtering algorithm is associated with each constraint (sub-problem).
- Can be simple ($x < y$) or complex (alldiff)
- Theoretical basics: arc consistency, remove all the values that do not belong to a solution of the underlined sub-problem.

Arc consistency

- All the values which do not belong to any solution of the constraint are deleted.
- Example: Alldiff({x,y,z}) with
  D(x)=D(y)={0,1}, D(z)={0,1,2}
  the two variables x and y take the values 0 and 1, thus z cannot take these values.
  FA by AC => 0 and 1 are removed from D(z)

Propagation

- Domain Reduction due to one constraint can lead to new domain reduction of other variables
- When a domain is modified all the constraints involving this variable are studied and so on ...

Why Propagation?

- Idea: problem = conjunction of easy sub-problems.
- Sub-problems: local point of view. Problem: global point of view. Propagation tries to obtain a global point of view from independent local point of view
- The conjunction is stronger that the union of independent resolution

Search

- Backtrack algorithm with strategies:
  try to successively assign variables with values. If a dead-end occurs then backtrack and try another value for the variable
- Strategy: define which variable and which value will be chosen.
- After each domain reduction (i.e assignment) filtering and propagation are triggered

Constraint Programming

- 3 notions:
  - constraint network: variables, domains constraints
  + filtering (domain reduction)
  - propagation
  - search procedure (assignments + backtrack)

Plan

- CP
- **Graph Theory: Flows**
- Global cardinality constraints: based on flow algorithms
- Global cardinality with costs: based on minimum cost flow
- Alldiff constraint: matching
- Symmetric alldiff constraint: symmetric matching
- Conclusion

Graph Theory

- Directed graph or digraph G=(X,U), X set of nodes, U set of arcs
- A path from v1 to vk is a set of nodes [v1,…,vk] such that (vi,vi+1) is an arc for every i in [1,..k-1]
- A path is simple if all its nodes are distinct
- A path is a cycle iff k >1 and v1=vk
- Length(p) is the sum of the costs of the arcs contained in p
- A shortest path from s to t is a path from s to t whose length is minimum
- There is a simple shortest path

Flows

- Let G be a graph in which every arc (i,j) is associated with 2 integers:
  - l(i,j) the lower bound capacity of the arc
  - u(i,j) the upper bound capacity of the arc
- A flow is a function f satisfying:
  - For any arc (i,j), f(i,j) represents the amount of some commodity that can ``flow'' through the arc.
    Such a flow is permitted only in the indicated direction of the arc, i.e., from i to j.
    For convenience, we assume f(i,j)=0 if (i,j) is not an arc.
  - A conservation law is observed at each node:
    for every node j: $\sum f(i,j) = \sum f(j,k)$.

Flows

- The feasible flow problem:
  - Does there exist a flow in G that satisfies the capacity constraints?
    That is find f such that
    for every arc (i,j) in U(G): $l(i,j) \leq f(i,j) \leq u(i,j)$.
- The problem of the maximum flow for an arc (i,j):
  - Find a feasible flow in G for which the value of f(i,j) is maximum.

Flows

- Without loss of generality, and to overcome notational difficulties, we will consider that:
  - if (i,j) is an arc of G then (j,i) is not an arc of G.
  - all boundaries of capacities are nonnegative integers.
- If all the upper bounds and all the lower bounds are integers and if there exists a feasible flow, then for any arc (i,j) there exists a maximum flow from j to i which is integral on every arc in G

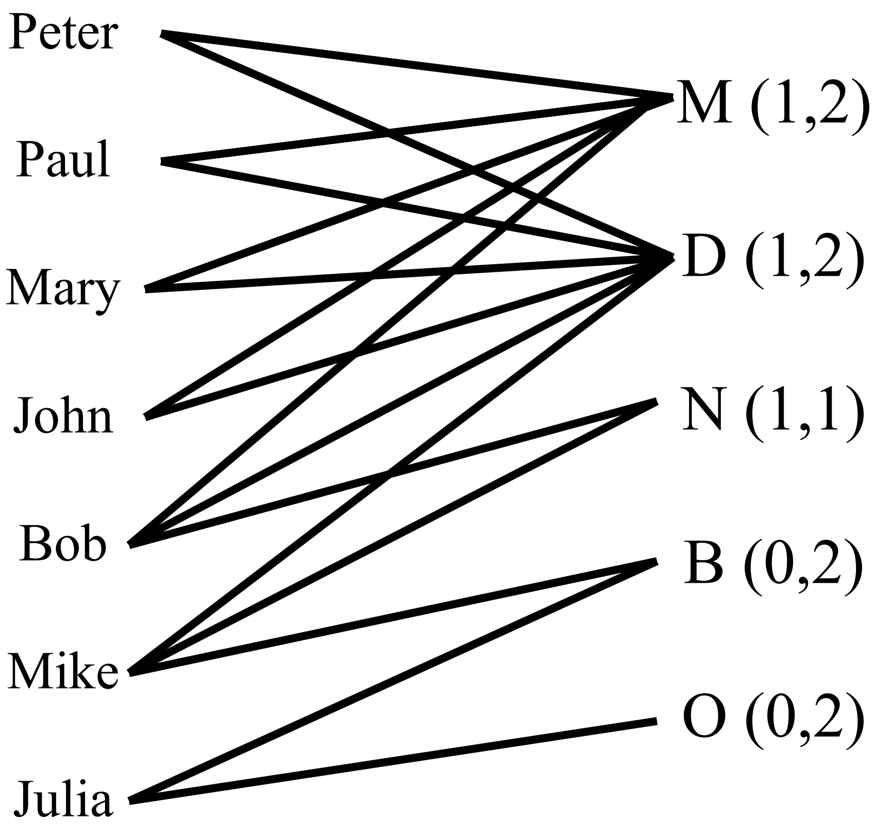
Plan

- CP
- Graph Theory: Flows
- **Global cardinality constraints: based on flow algorithms**
- Global cardinality with costs: based on minimum cost flow
- Alldiff constraint: matching
- Symmetric alldiff constraint: symmetric matching
- Conclusion

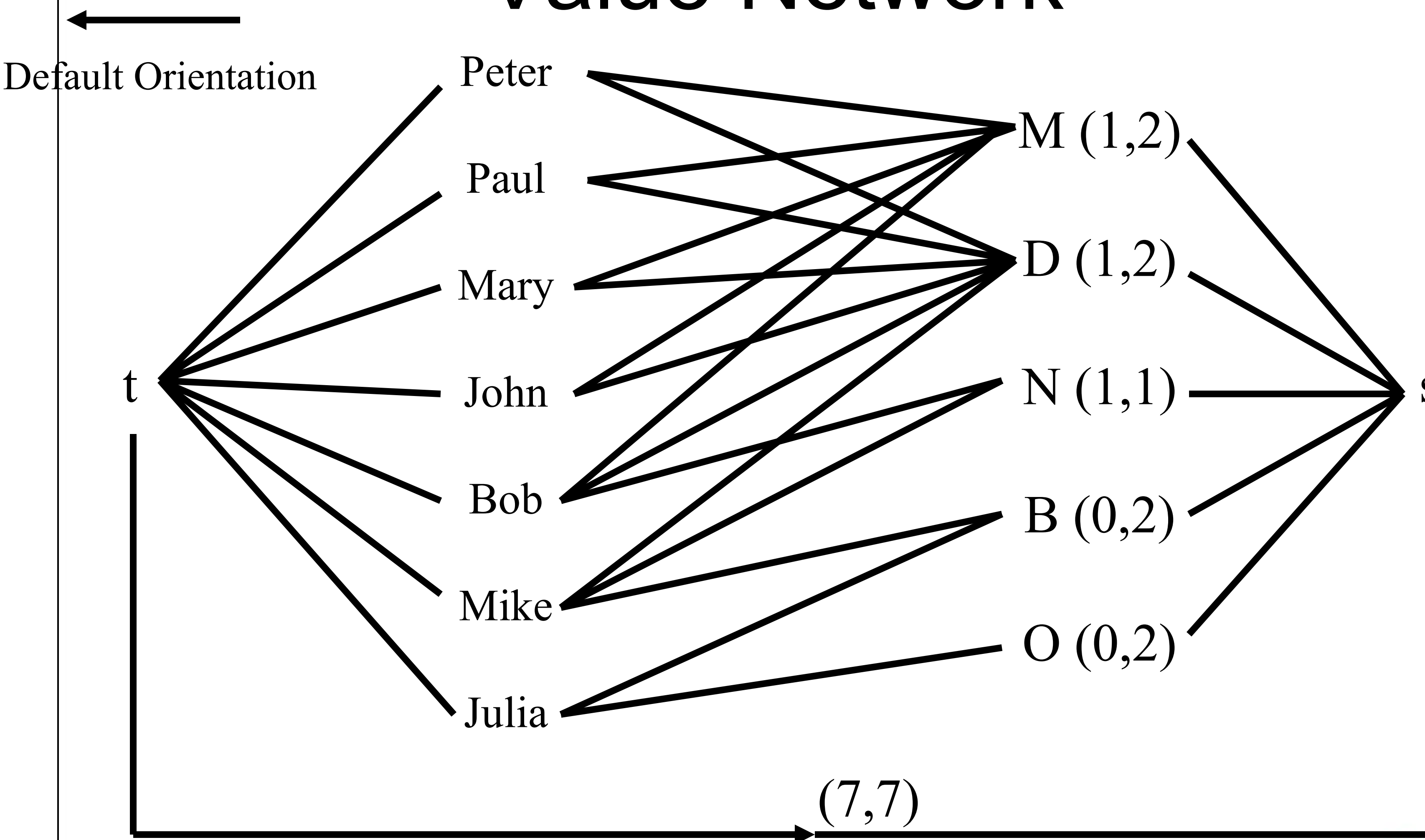
Global Cardinality Constraint

- GCC(X,$\{l_i\}$,$\{u_i\}$)
- Defined on a set X of variables, the number of times each value v_i can be taken must be in a given interval $[l_i, u_i]$
- Example: D(x1)={a,b,c,d}, D(x2)={a,b,c,d}, D(x3)={b,c},D(x4)={c,d}. Values b and c must be taken at most 2, values a and d must be taken at least 1.

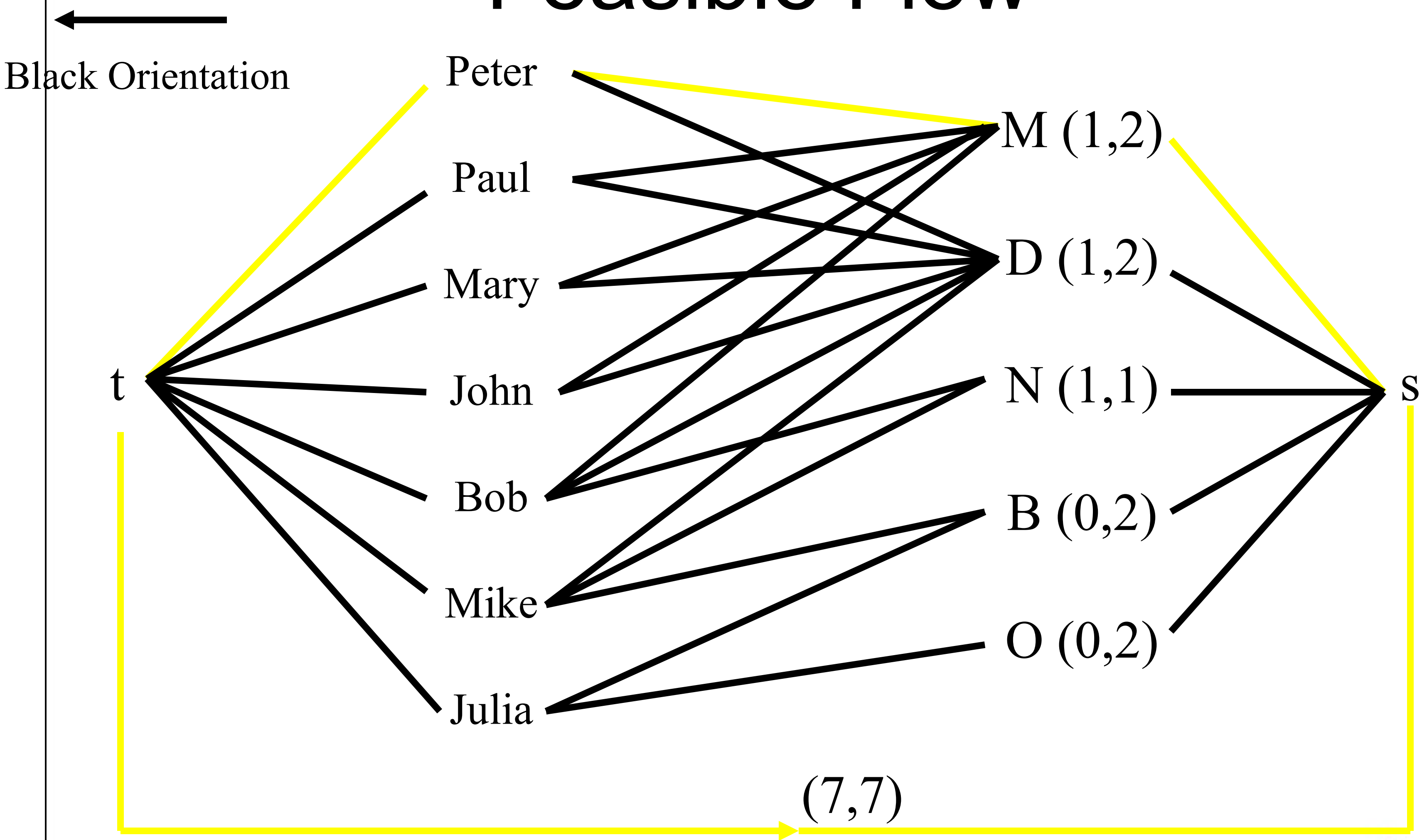
GCC



Value Network



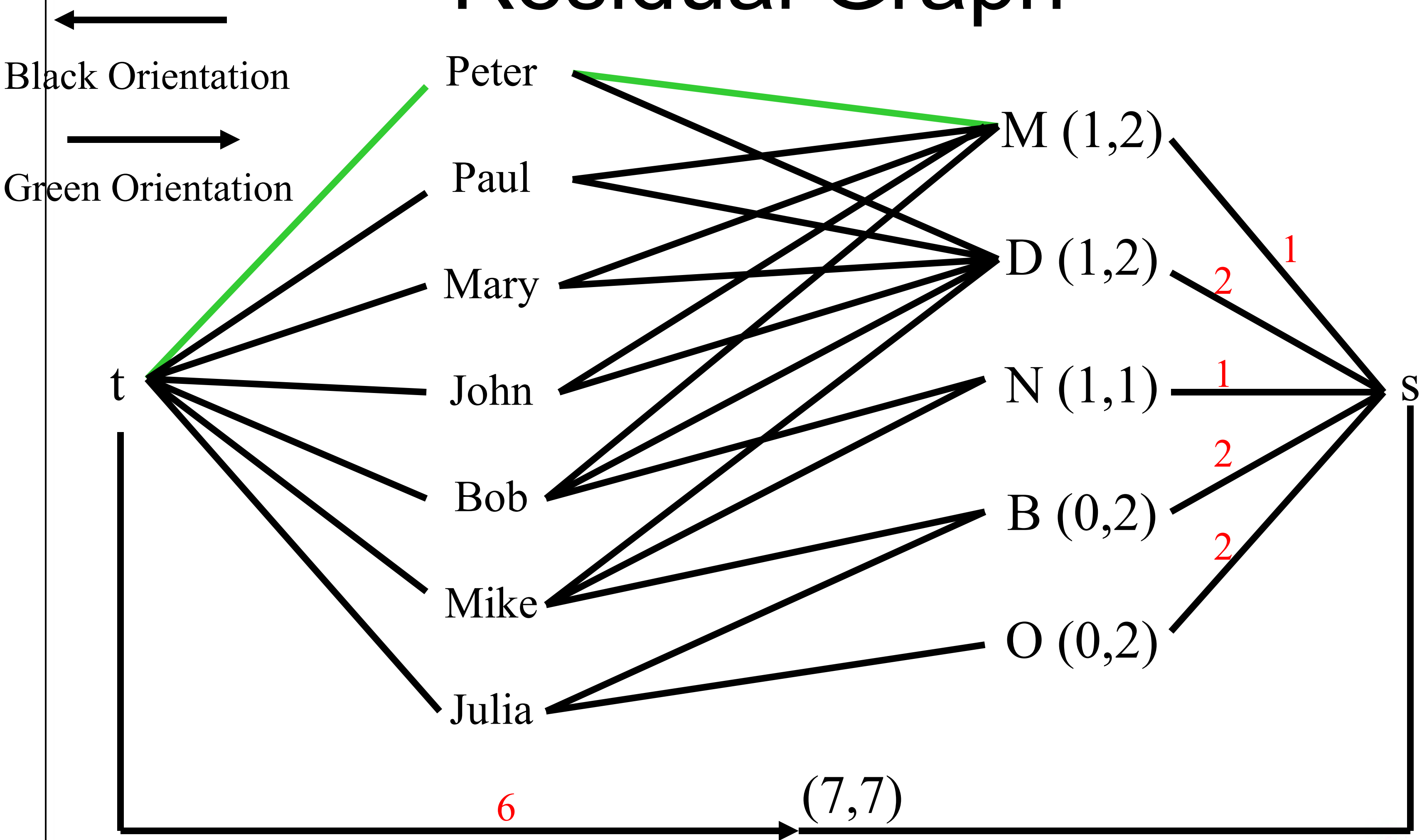
Feasible Flow



Successive augmentation

- Successive augmentation are computed in a particular graph:
  The **residual graph**
- The residual graph has **no lower bounds**
- In our case this algorithm is equivalent to the best ones.

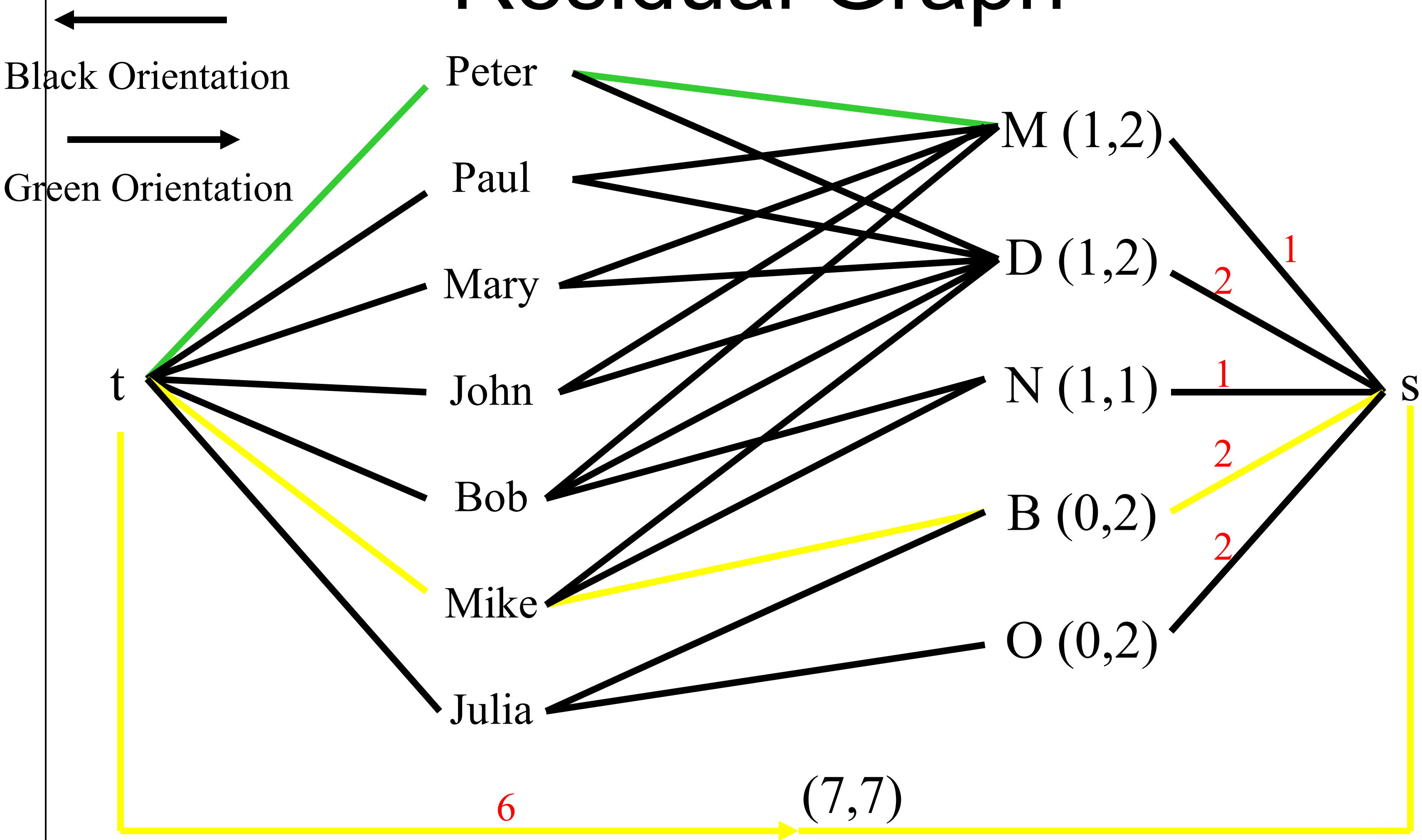
Residual Graph



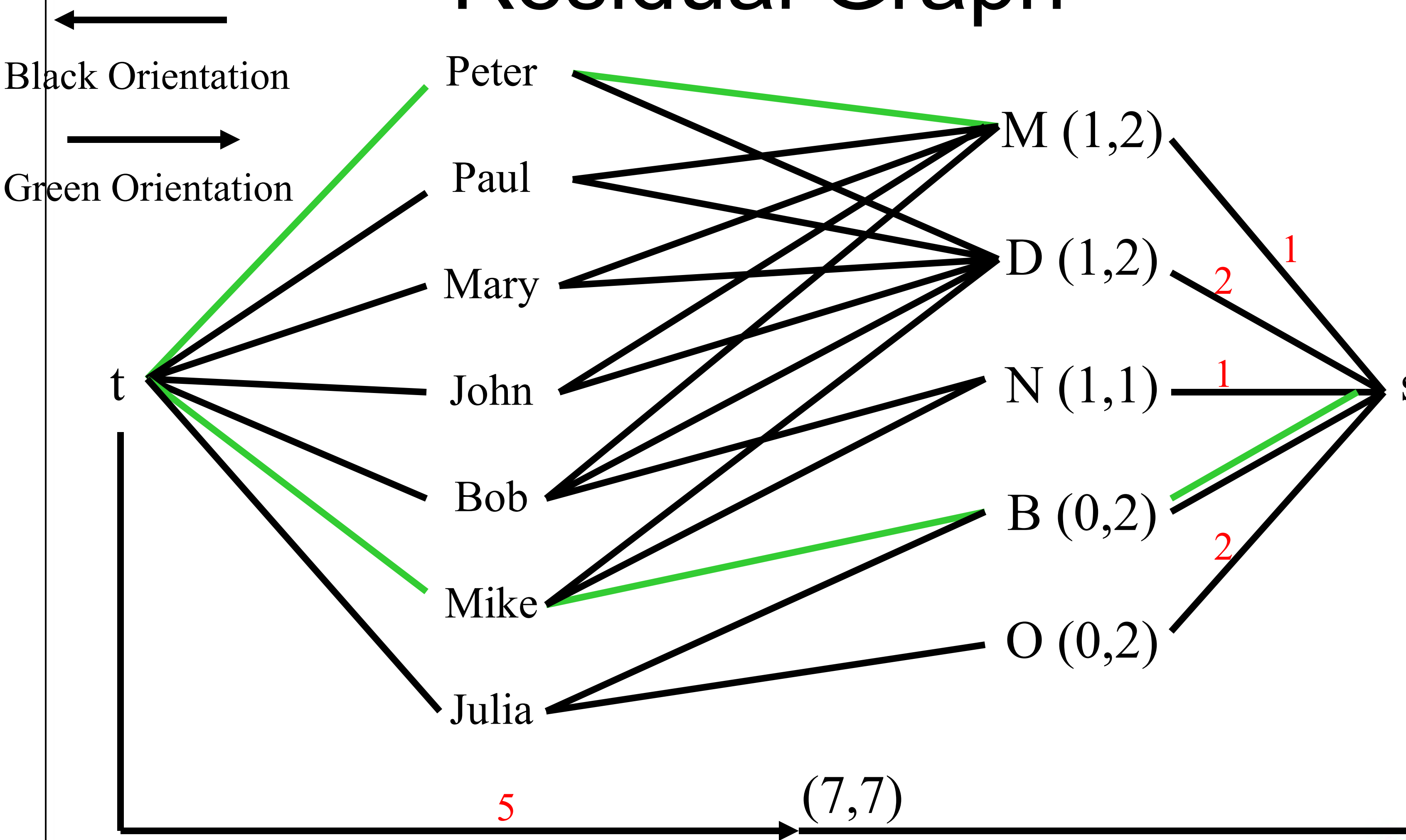
If f(i,j) < u(i,j) then (i,j) and r(i,j)= u(i,j) – f(i,j)

If f(i,j) > l(i,j) then (j,i) and r(j,i)= f(i,j) – l(i,j)

Residual Graph



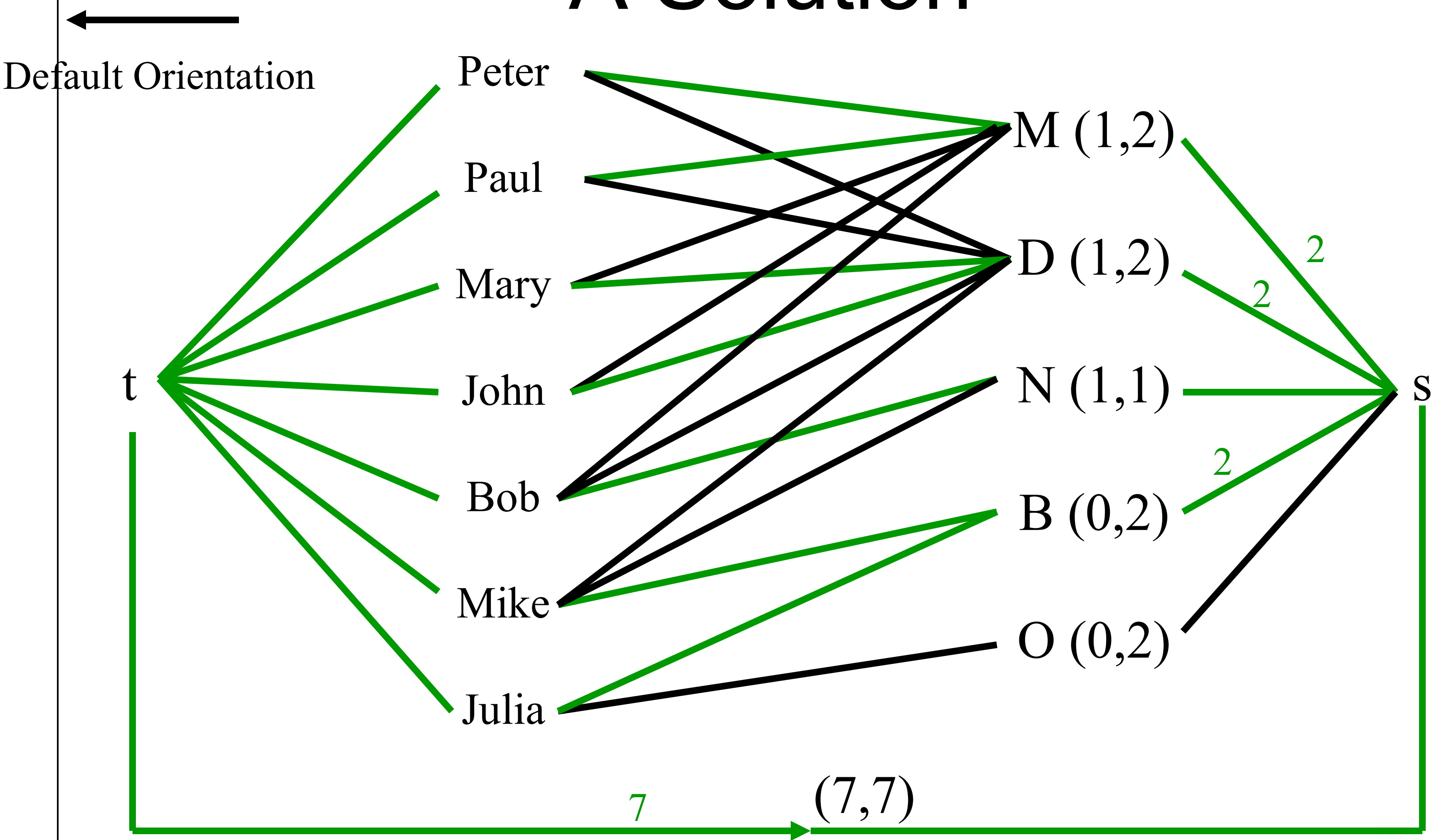
Residual Graph



If f(i,j) < u(i,j) then (i,j) and r(i,j)= u(i,j) – f(i,j)

If f(i,j) > l(i,j) then (j,i) and r(j,i)= f(i,j) – l(i,j)

A Solution

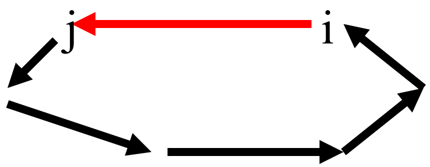


7 flow value

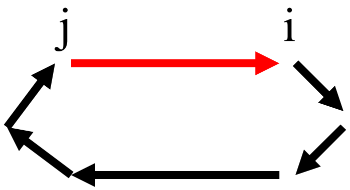
Sum = 7

Properties

- The flow value xij of (i,j) can be increased iff there is a path from j to i in R - {(j,i)}



- The flow value xij of (i,j) can be decreased iff there is a path from i to j in R - {(i,j)}



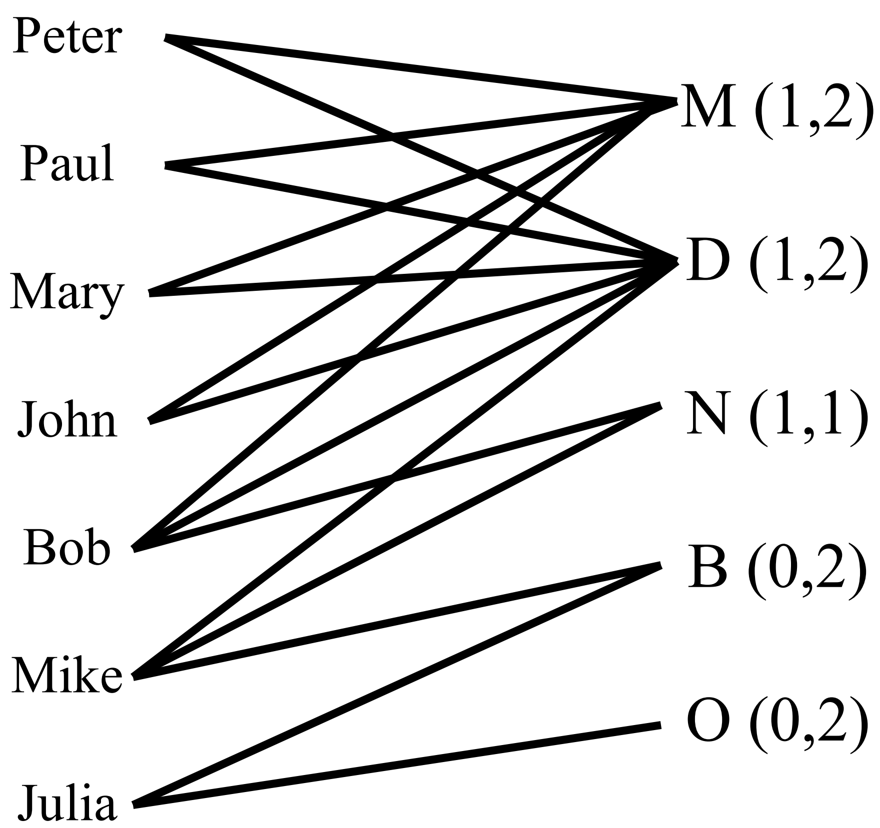
Arc consistency

- The flow value of an arc is constant iff the arc does not belong to a directed cycle of the residual graph
- Definition of strongly connected components

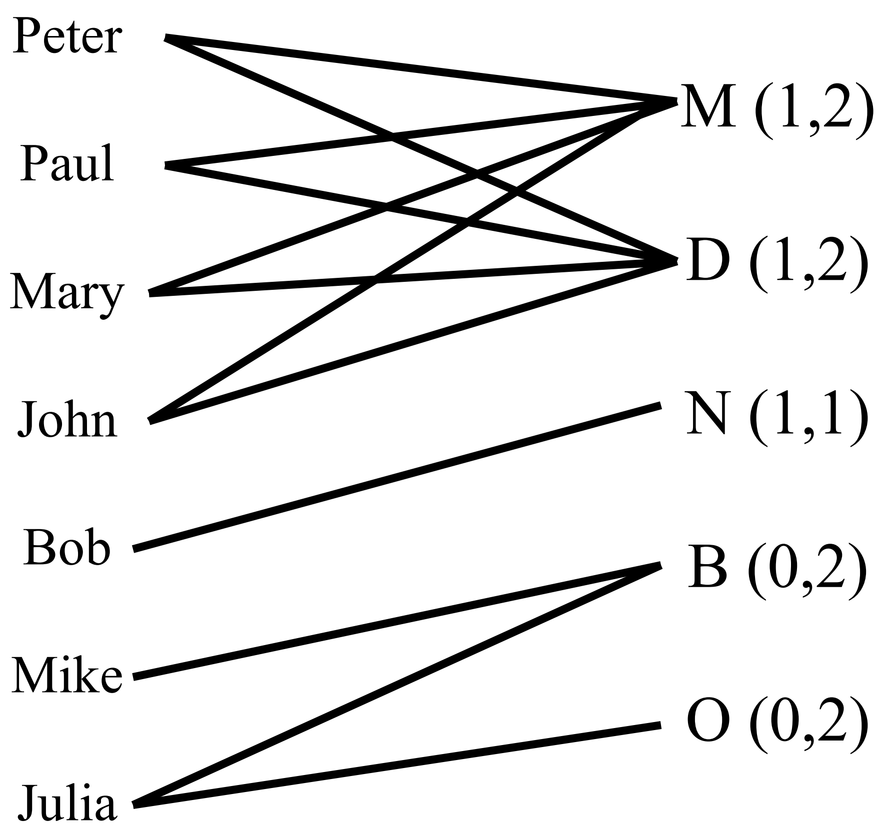
Filtering algorithm for GCC

- Compute a feasible flow
- Compute the strongly connected components
- Remove every arc with a zero flow value for which the ends belong to two different components
- Linear algorithm achieving arc consistency
- work well due to (0,1) arcs

GCC



GCC after AC



Plan

- CP
- Graph Theory: Flows
- Global cardinality constraints: based on flow algorithms
- **Global cardinality with costs: based on minimum cost flow**
- Alldiff constraint: matching
- Symmetric alldiff constraint: symmetric matching
- Conclusion

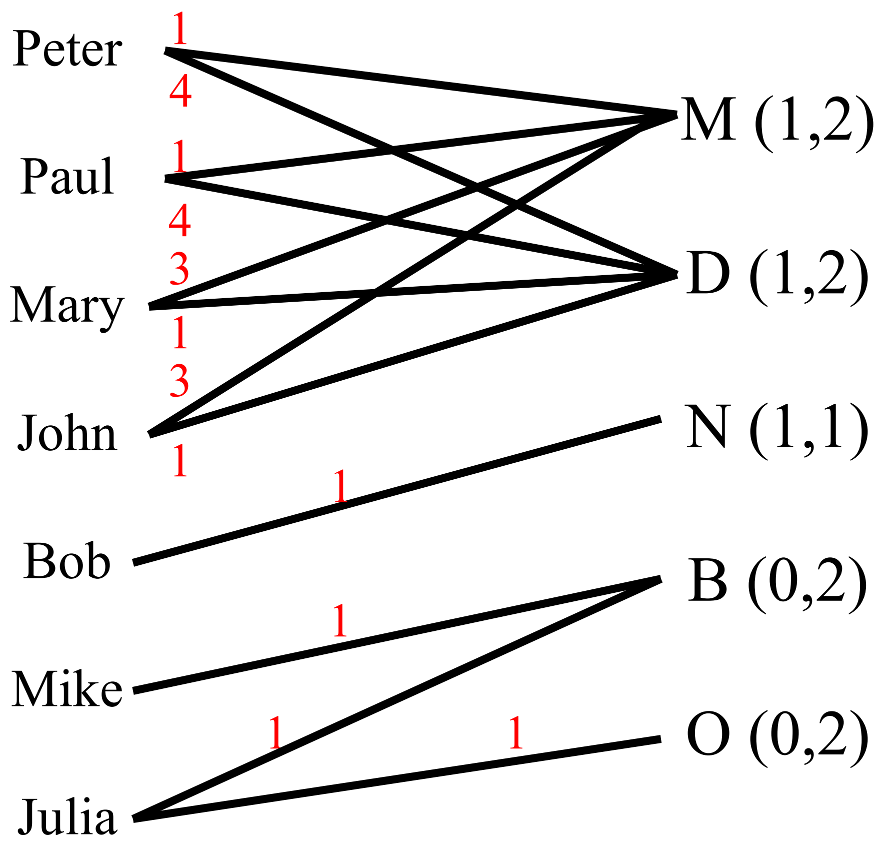
GCC with costs

- GCC with costs =
  Global cardinality constraint
  + Sum constraint on the assignment costs
- AC algorithm which takes into account the globality of the constraint + the costs

Minimum cost Flows

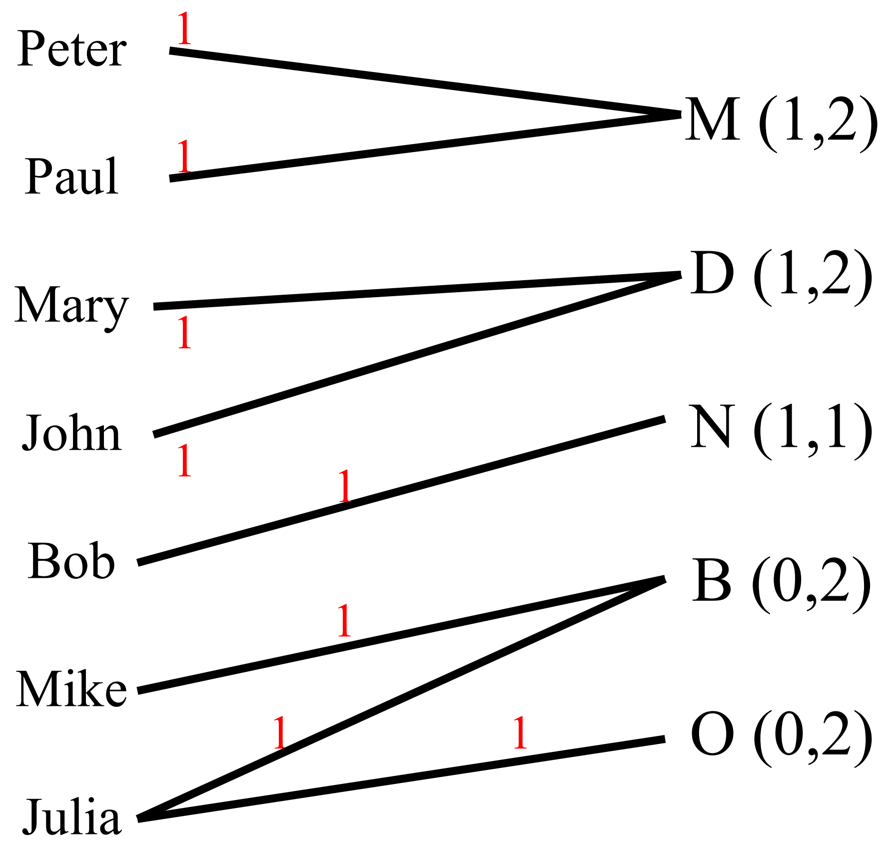
- Let G be a graph in which every arc (i,j) is associated with 3 integers:
  - l(i,j) the lower bound capacity of the arc
  - u(i,j) the upper bound capacity of the arc
  - c(i,j) the cost of carrying one unit of flow
- The cost of a flow f is cost(f)=$\sum$ f(i,j) c(i,j)
- The minimum cost flow problem:
  - If there exists a feasible flow, find a feasible flow f such that cost(f) is minimum.

GCC with costs



Sum < 12

Arc consistency



Sum < 12

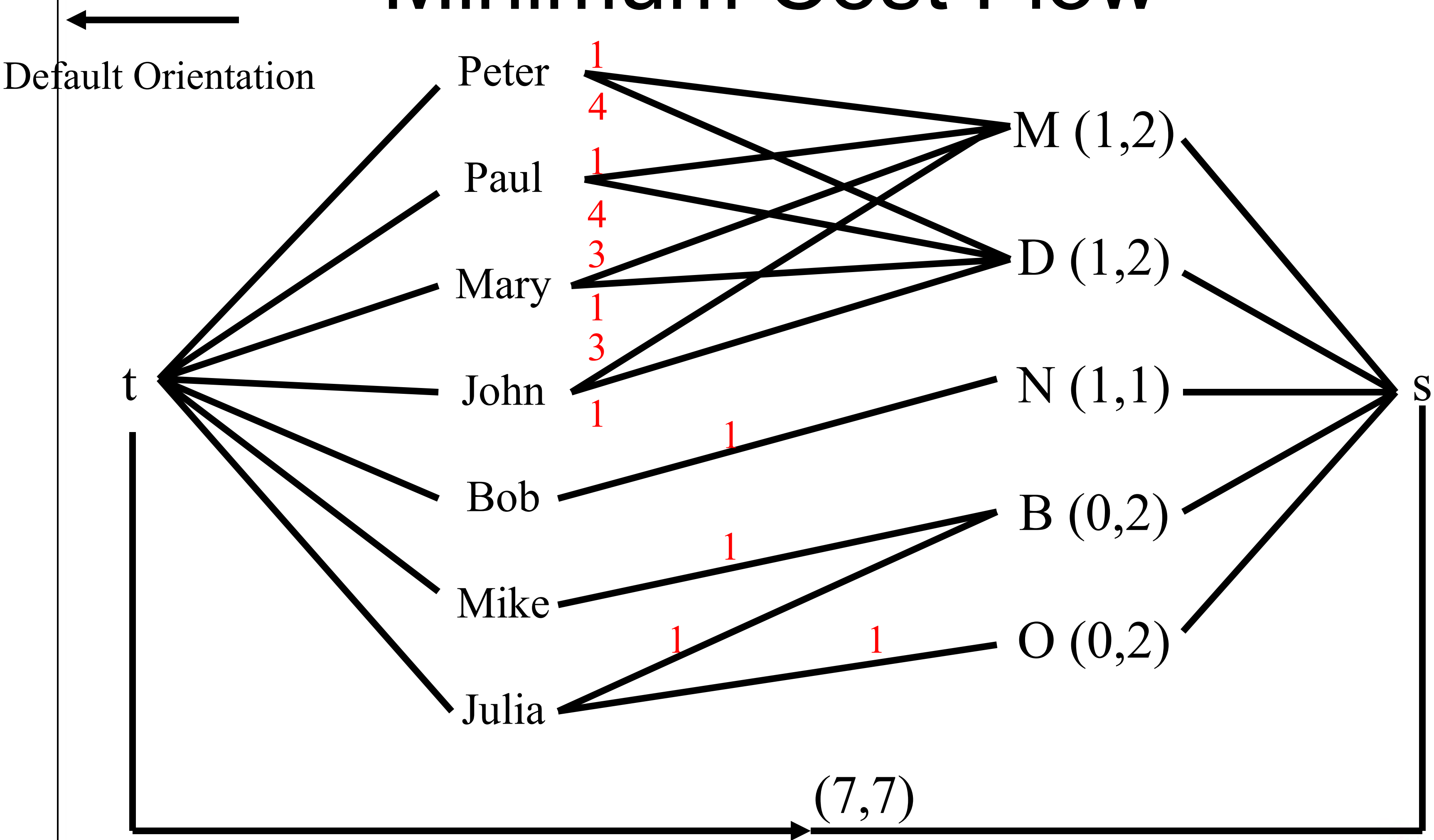
GCC with costs

- Consistency can be computed by searching for a minimum cost flow
- Arc consistency can be computed by searching for shortest paths in a special graph.

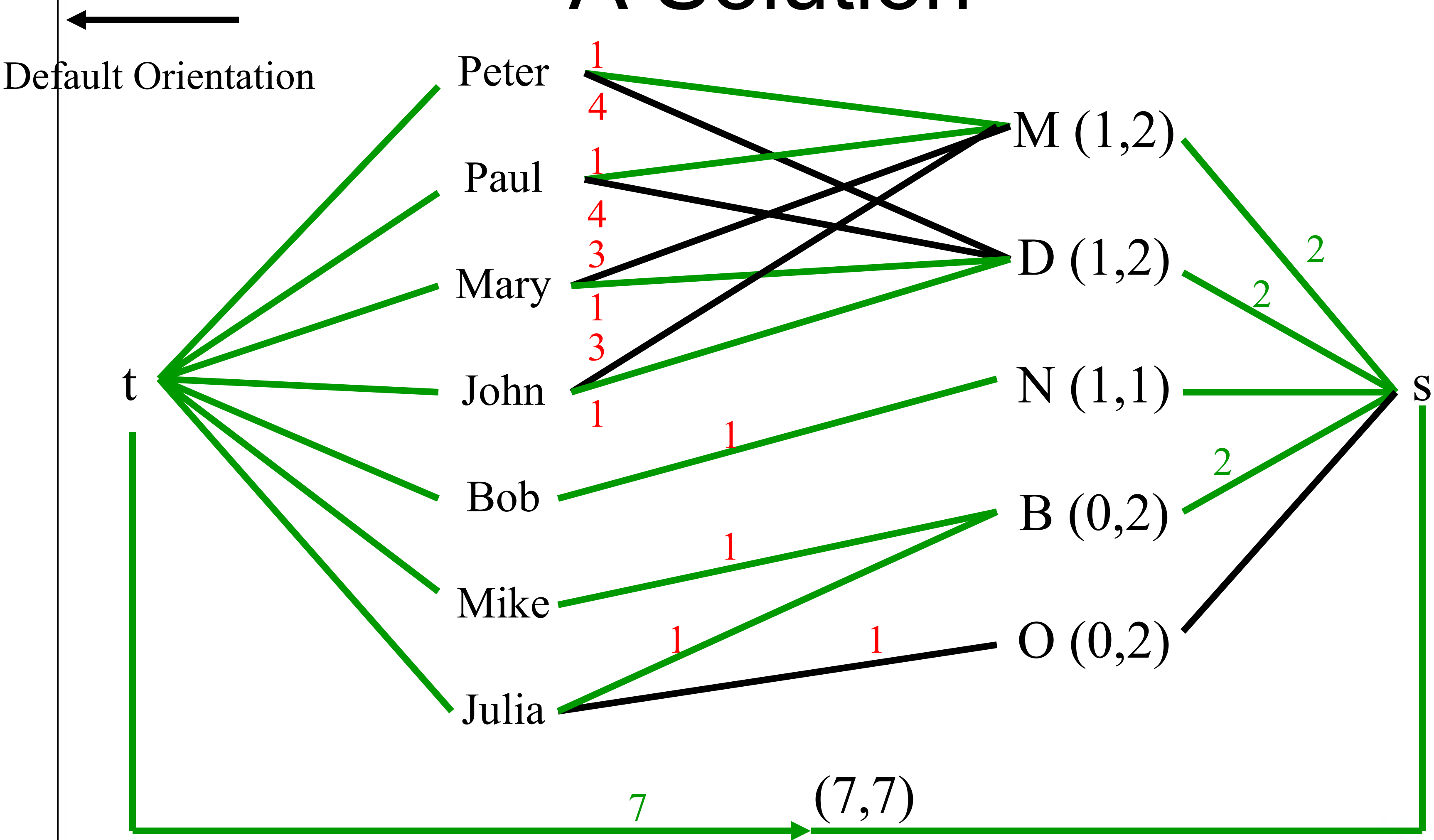
Minimum Cost flow

- Similar to feasible flow except that: **shortest** paths are considered.
- length of an arc = **reduced cost** of the arc
- Reduced costs are used to work with nonnegative value (useful for shortest paths algorithms), but the principles remains the same with **residual costs**.
- We will consider here only the residual costs

Minimum Cost Flow



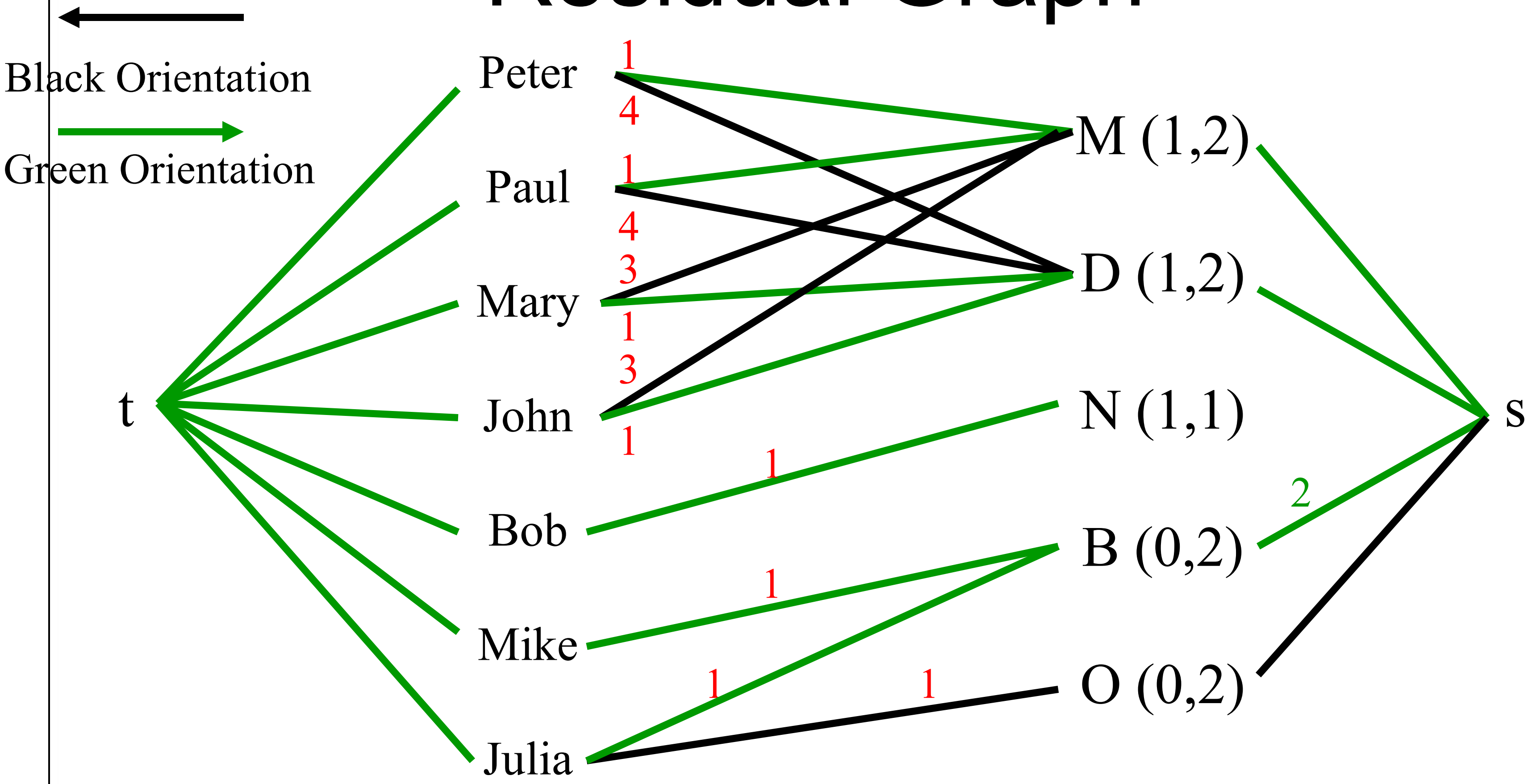
A Solution



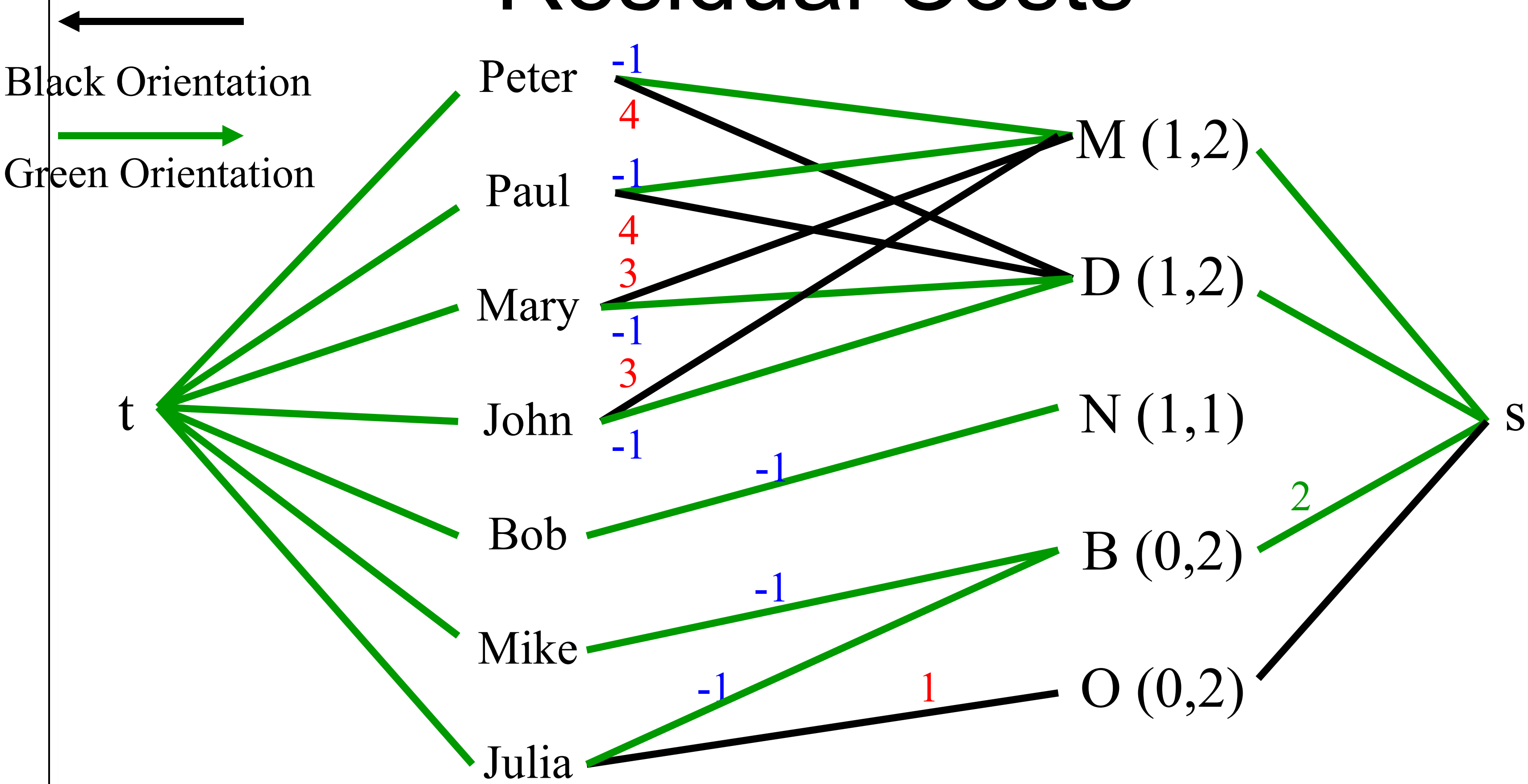
7 flow value

Sum = 7

Residual Graph

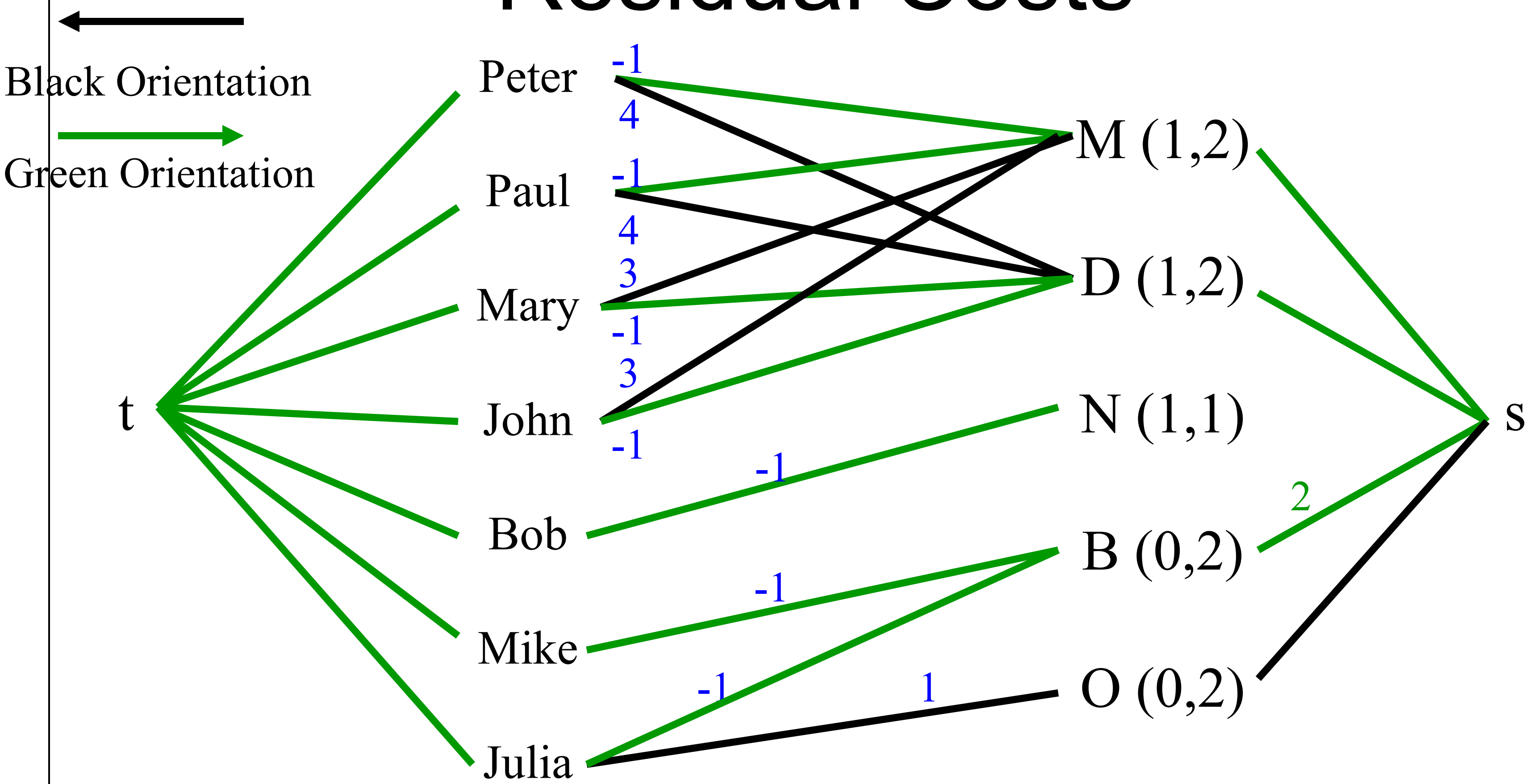


Residual Costs



-1 residual cost = - cost if opposite arc

Residual Costs



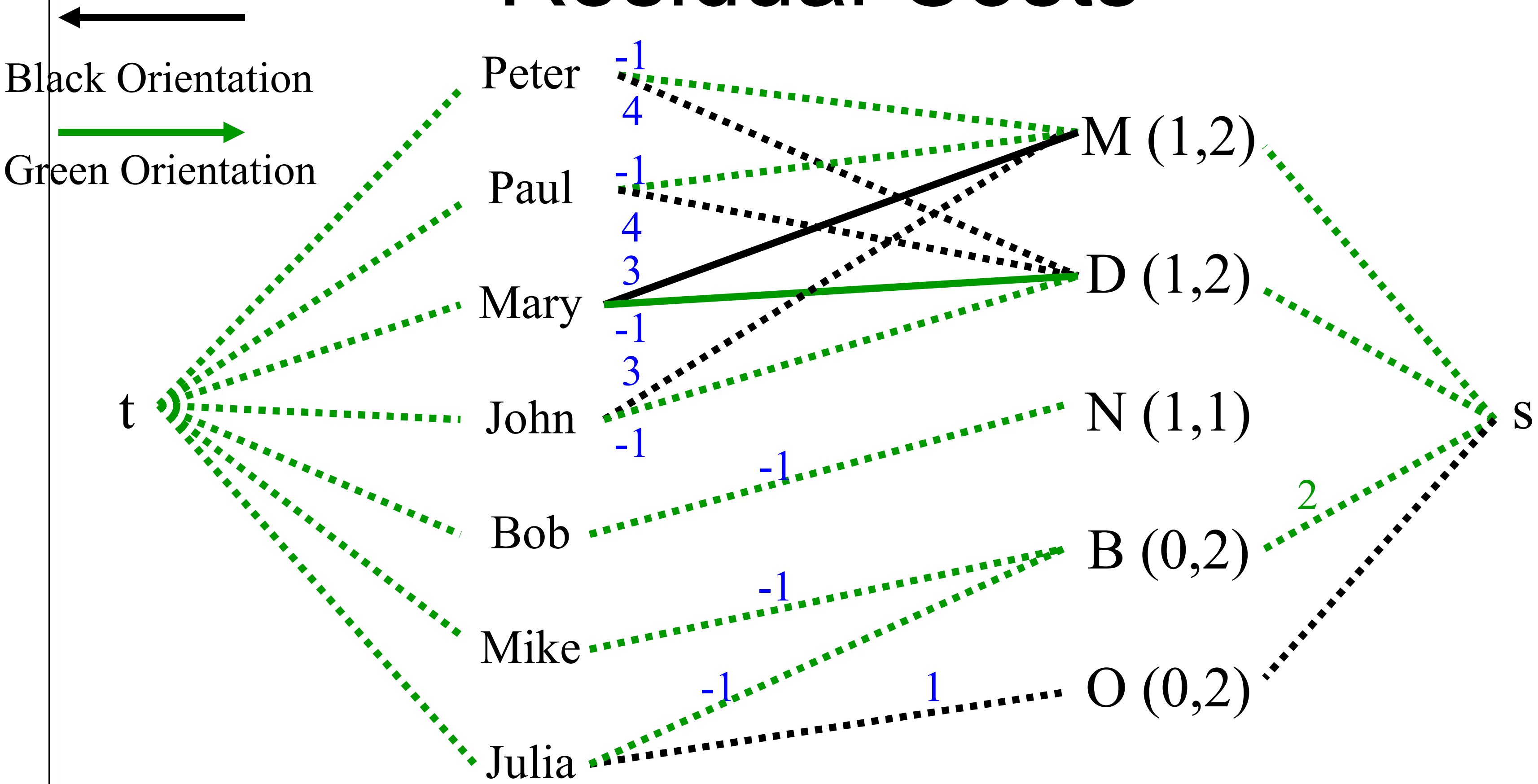
-1 residual cost = - cost if opposite arc

1 residual cost = cost if arc

Shortest path

- d(i,j) = length of the shortest path which does not use (i,j) in the residual graph. The length is the sum of the residual costs of the arc contained in the path.

Residual Costs



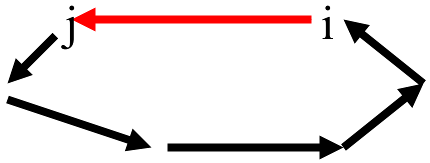
d(M,D)=3 + (-1) = 2

Minimum cost flow

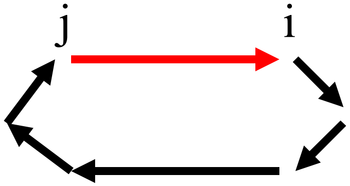
- If the feasible flow is computed by augmenting the flow along shortest paths then the solution is optimal.
- Complexity O(n S(n,m,χ)) where χ is the maximum cost value.

Arc consistency

- The flow value xij of (i,j) can be increased iff there is a path from j to i in R - {(j,i)}



- The flow value xij of (i,j) can be decreased iff there is a path from i to j in R - {(i,j)}



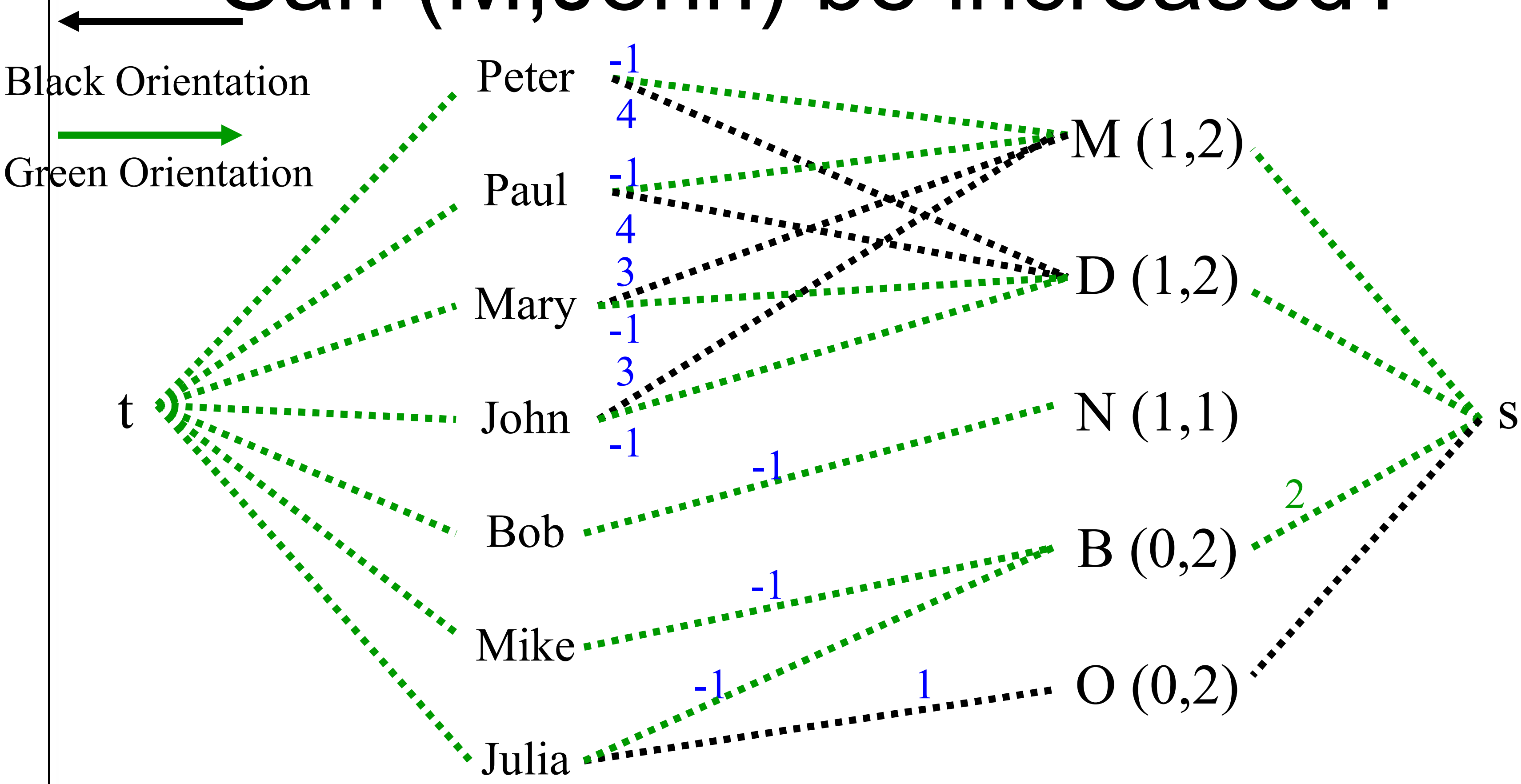
Arc consistency

- Let optcost be the value of the minimum cost flow, and H be the maximum value of the assignments.
- The flow value of an arc (i,j) can be increased if and only if:

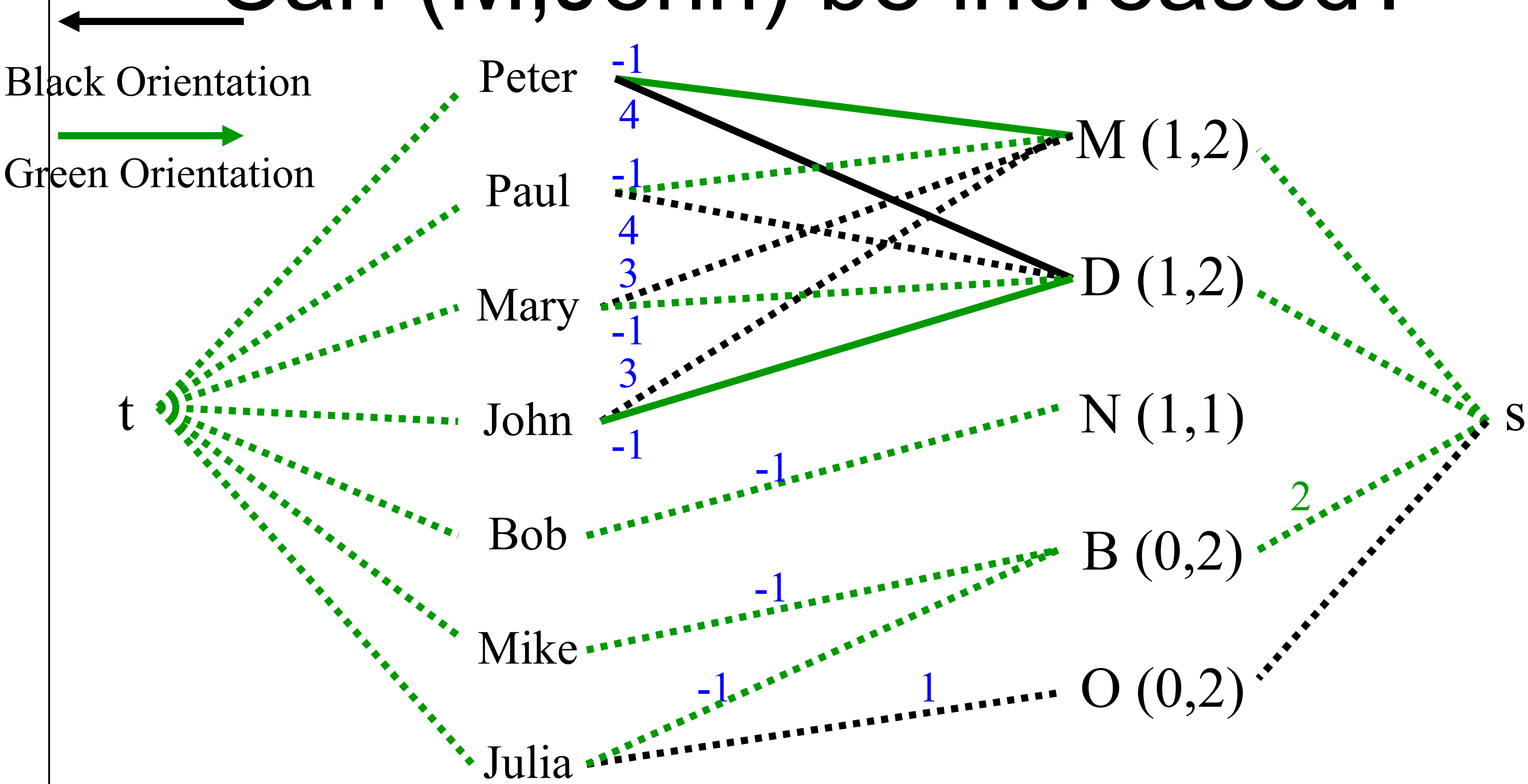
  $$rcij + d(j,i) + optcost < H$$

  The cost of the directed cycle is computed, that is the cost of rerouting the flow.

Can (M,John) be increased?



Can (M,John) be increased?



rc(M,John) + d(John,M) + optcost = 3 + (-1+4+(-1)) + 7 = 12 > 11: NO

Arc consistency

- Similar reasoning for decreasing the flow value.
- Complexity O(m S(n,m,χ))
- **can be improved!**

AC Improvement

- Problem: shortest paths from j to i cannot contain (j,i).
- How the computations can be grouped, since the graph changes for each computation?

AC Improvement

- Problem: shortest paths from j to i cannot contain (j,i).
- How the computations can be grouped, since the graph changes for each computation?
- **The graph does not change for (0,1) arcs!**

AC Improvement

- Between variables and values there are only (0,1) arcs.
- If we search for increasing the flow value of (i,j) then xij=0 and (j,i) does not exist in R
- If we search for decreasing the flow value of (i,j) then xij=1 and (i,j) does not exist in R

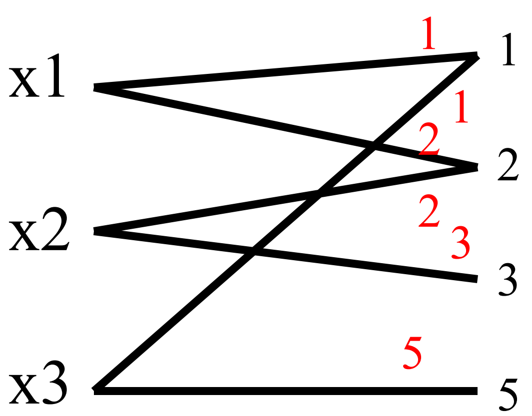
AC Improvement

- The computation can be grouped:
  For each variable, the shortest paths to all the values are computed
- Complexity $O(n\ S(n,m,\chi))$.
- Can be improved by searching for shortest path from the values that are assigned.
- Reduced costs can be used instead of residual cost to have only nonnegative costs and to improve the search for shortest paths.

Sum of all different var

- Constraint:
  $x1+x2+x3 < H$ and alldiff(x1,x2,x3).
- Can be represented as a gcc with costs.

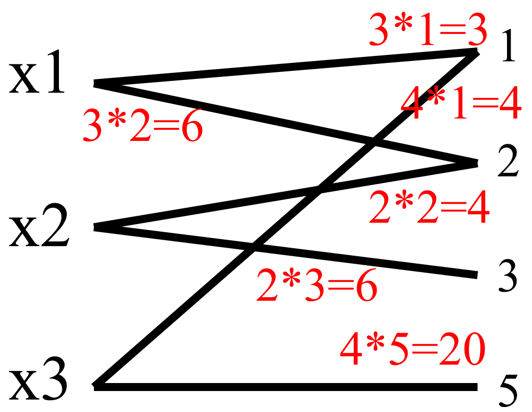
Sum of all different var



The cost of an arc involving a value is equal to this value

Can be generalized to cardinality constraints

Scalar Product of all different var



3x1 + 2x2 + 4x3 = s

The cost of an arc involving a value is equal to the coefficient of the variable multiplied by the value

Can be generalized to cardinality constraints

Plan

- CP
- Graph Theory: Flows
- Global cardinality constraints: based on flow algorithms
- Global cardinality with costs: based on minimum cost flow
- **Alldiff constraint: matching**
- Symmetric alldiff constraint: symmetric matching
- Conclusion

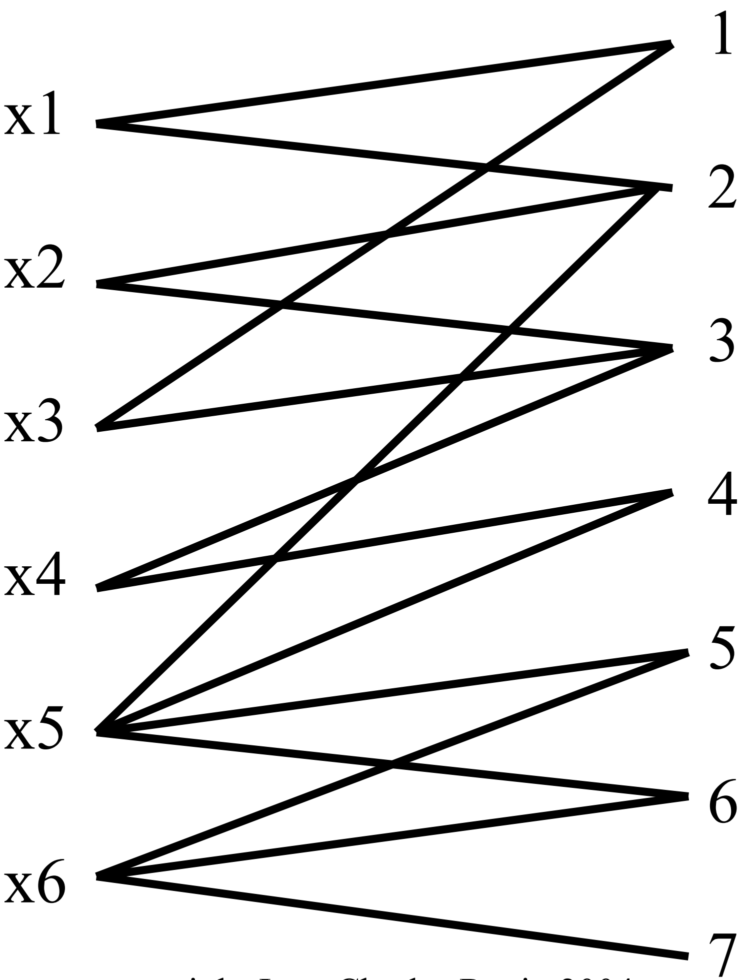
Matching

- A matching is a set of edges no two of which have a common endpoint.
- A matching M covers X if all the nodes is an endpoint of M

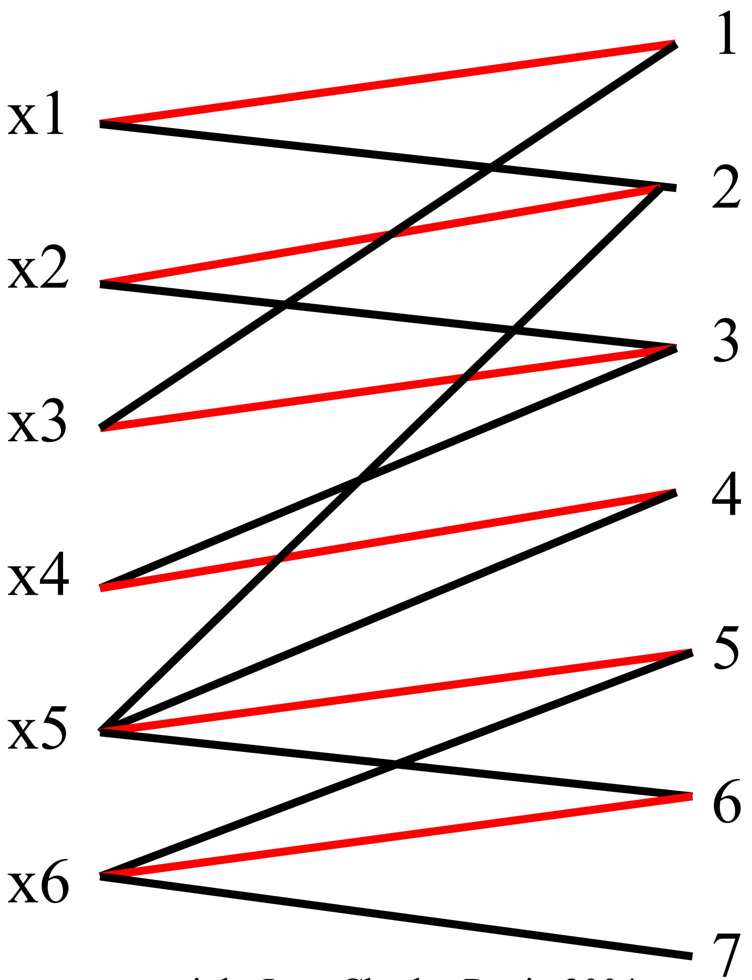
Alldiff constraint

The value graph:

D(x1)={1,2}
D(x2)={2,3}
D(x3)={1,3}
D(x4)={3,4}
D(x5)={2,4,5,6}
D(x6)={5,6,7}



Matching

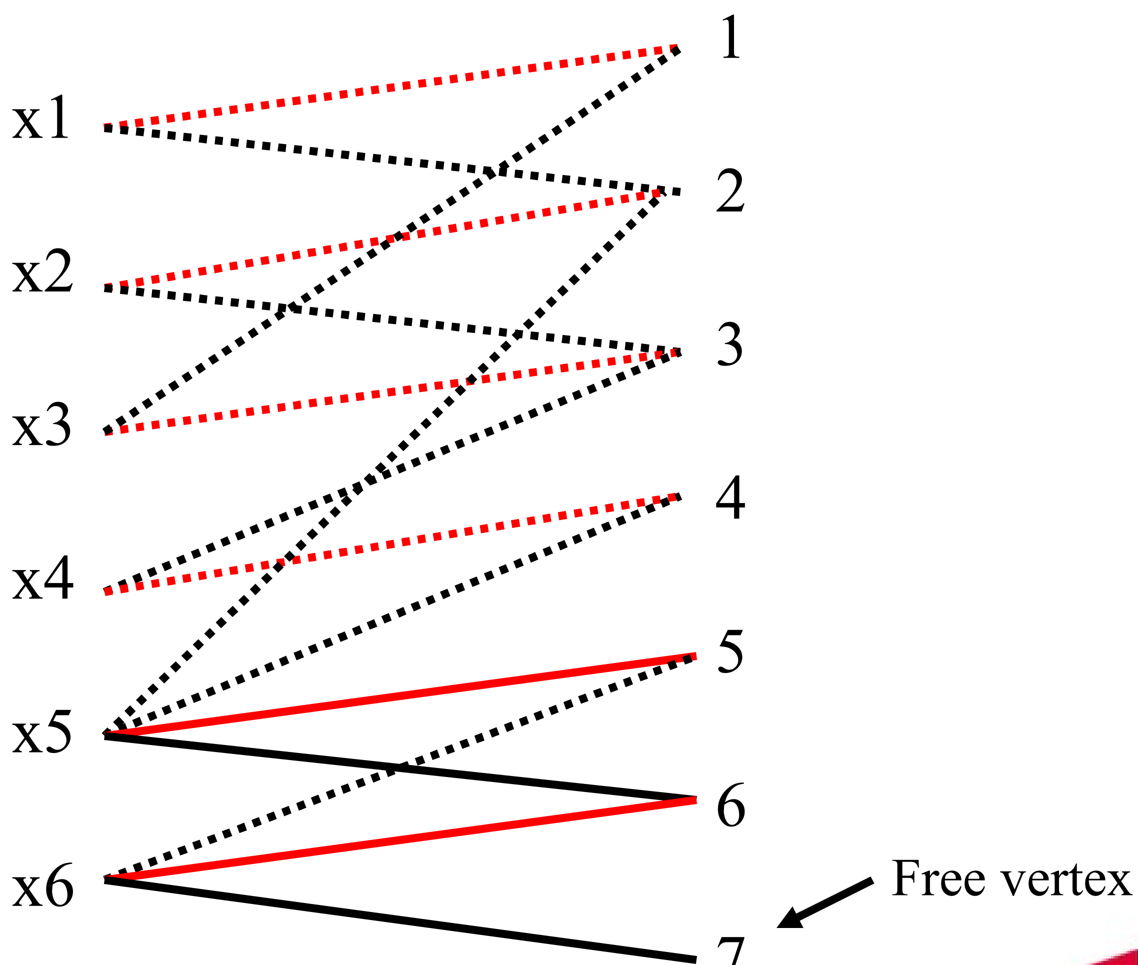


Arc consistency

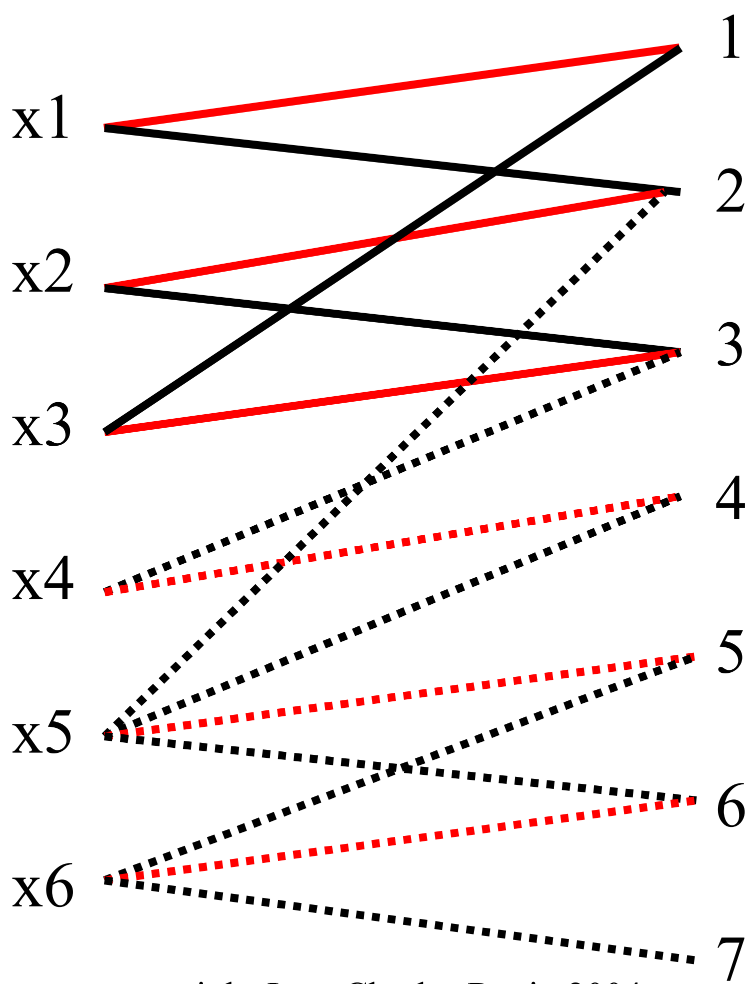
Berge's theorem:

An edge belong to some but not all maximum matching, iff, for an arbitrary matching it belongs to either an even alternating path which begins at a free vertex, or an even alternating cycle.

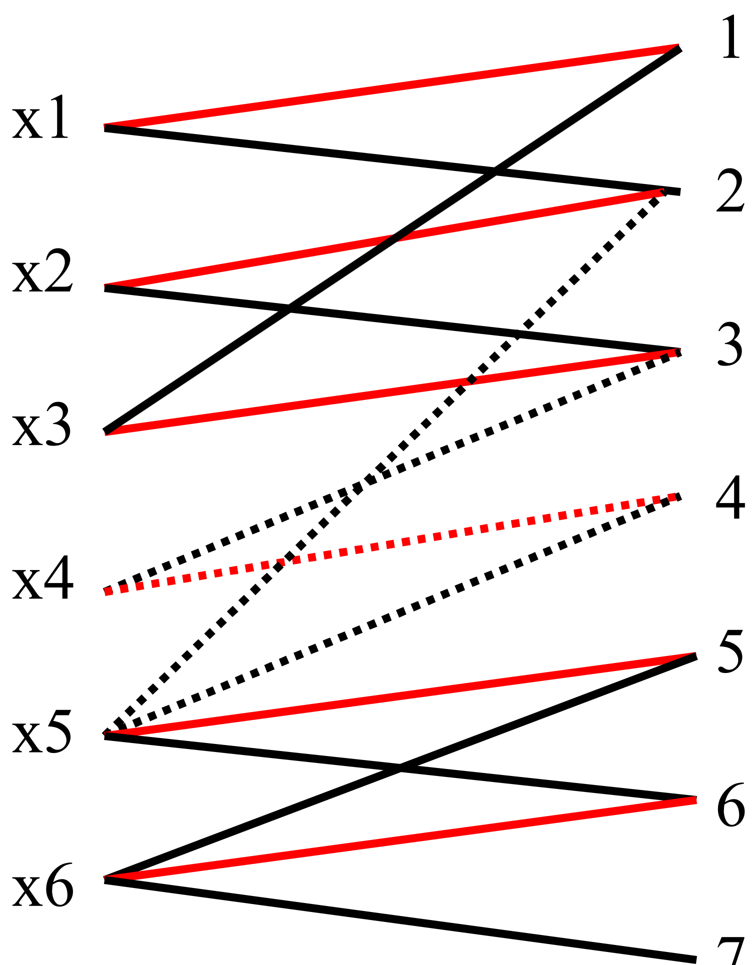
Alternating path



Alternating cycle



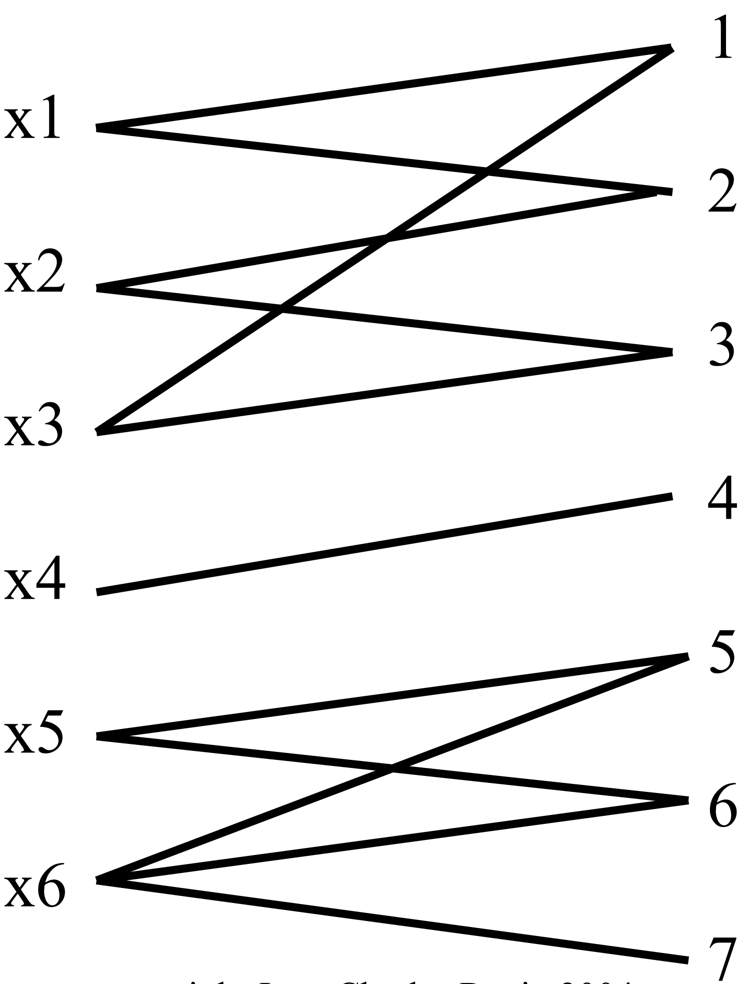
Alldiff constraint



Arc consistency

The value graph:

D(x1)={1,2}
D(x2)={2,3}
D(x3)={1,3}
D(x4)={4}
D(x5)={5,6}
D(x6)={5,6,7}



Alldiff constraint

- Compute a matching which covers X
- Compute the strongly connected components
- Remove every unmatched arc for which the ends belong to two different components
- Consistency: $O(n^{1/2}m)=O(n^{3/2}d)$ from scratch
  $O(knd)$ incremental
- **Linear algorithm** achieving arc consistency $O(m)=O(nd)$

Plan

- CP
- Graph Theory: Flows
- Global cardinality constraints: based on flow algorithms
- Global cardinality with costs: based on minimum cost flow
- Alldiff constraint: matching
- **Symmetric alldiff constraint: symmetric matching**
- Conclusion

The symmetric alldiff

- Goal: group entities by pair
- Example: aircraft pilots, nurses …

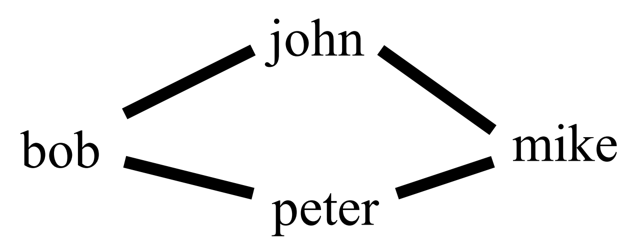
  List of compatibility:

  bob can work with john and peter

  john can work with bob and mike

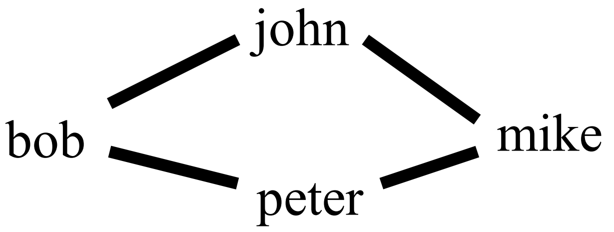
  mike can work with peter and john

  peter can work with bob and mike



Possible model

- **One variable per person**
  values = variables
  D(Vjohn)={bob,mike}
  D(Vbob)={john,peter}



- **Constraints**:
  AllDiff(Vjohn,Vbob,Vpeter,Vmike)
  + $\forall$i,j: (Vi=j <=> Vj=i)
  If bob works with john then john works with bob

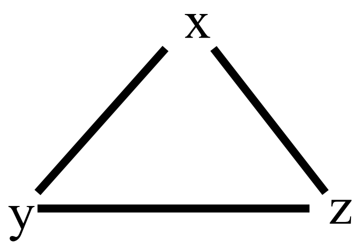
Symmetric Alldiff

- A Symmetric Alldiff Constraint takes into account SIMULTANEOUSLY:

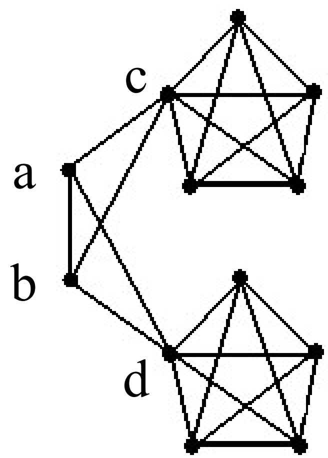
  AllDiff(Vjohn,Vbob,Vpeter,Vmike)
  + $\forall$i,j: (Vi=j <=> Vj=i)

Symmetric Alldiff

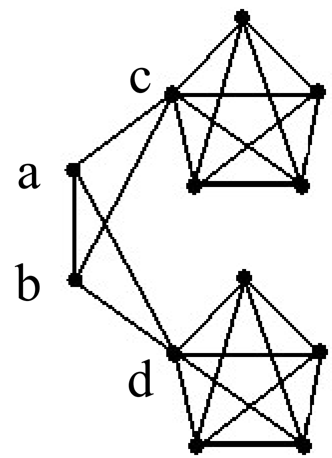
- Why is it interesting?
  D(x)={y,z}, D(y)={x,z}
  D(z)={x,y}
  Nothing is deduced
  There is no solution!

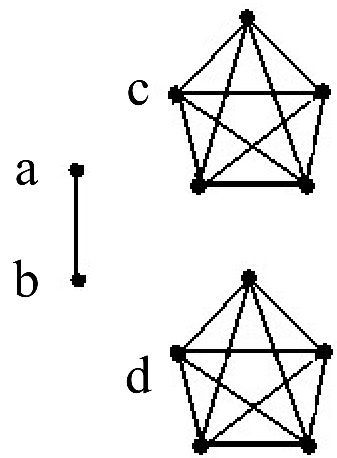


Why is it important?

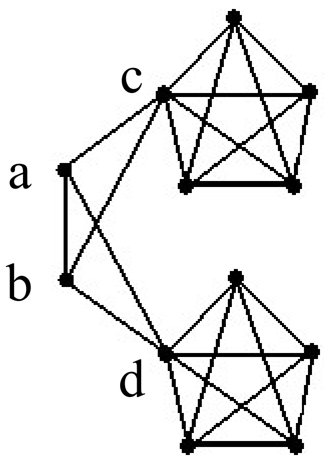


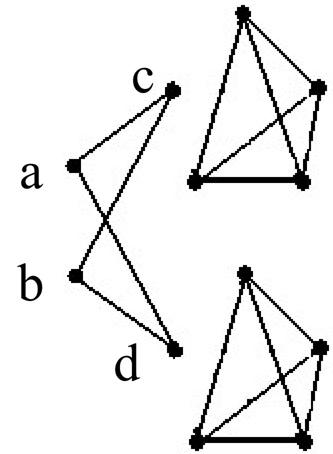
Why is it important?



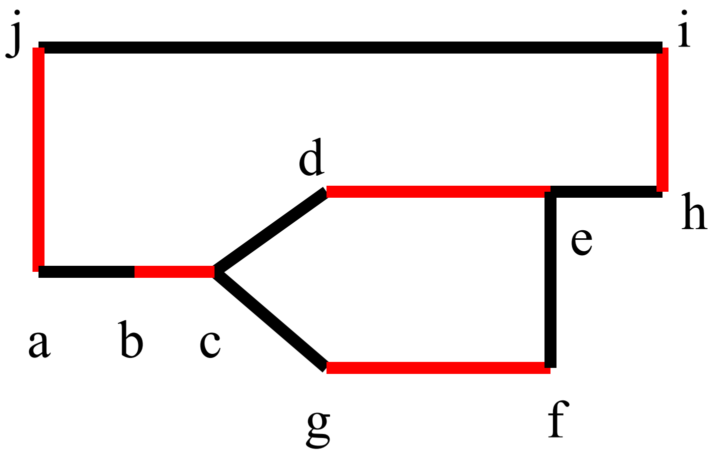


Why is it important?

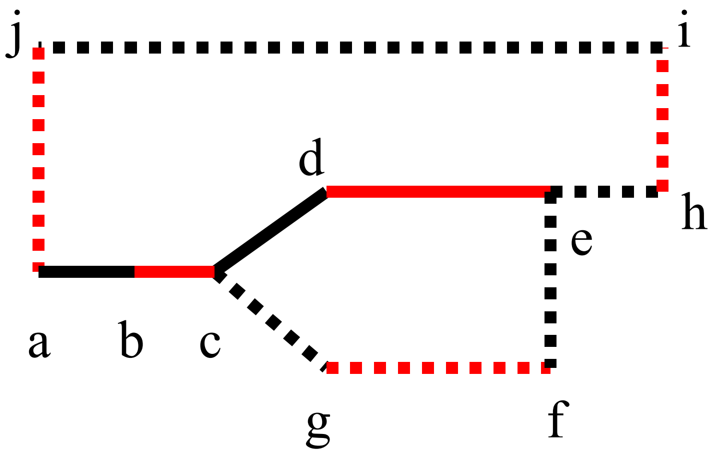




Non bipartite matching

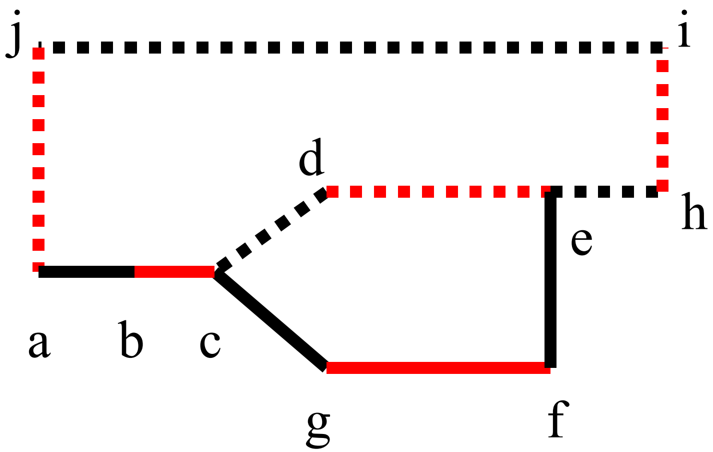


Non bipartite matching



e is mark even

Non bipartite matching



e is mark odd

Non bipartite matching

- In bipartite graph the edge can be oriented (from one set to another set)
- In non bipartite graph this is impossible!
- We loose the efficient algorithm for the alldiff constraint

Non bipartite matching

- Idea: Edmond's algorithm
- Improvement by Tarjan and other
- Complexity $O(nm\alpha(n,m))$ easy to implement
  $O(nm)$ not to hard
  $O(n^{1/2}m)$: 42 pages of non intuitive demonstration

Arc consistency

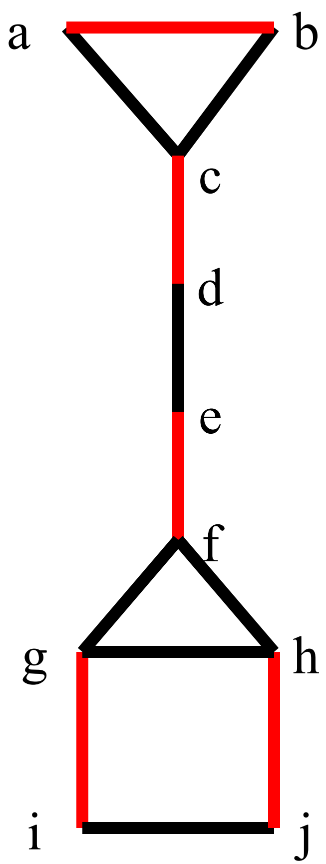
- Pb: find all edges that do not belong to any matching which covers X
- First solution:
  for each edge in turn:
  remove the two extremities, search for a matching which covers X-{e1,e2}
  Complexity : $mO(m)=O(m^2)$ (because incremental algorithm)

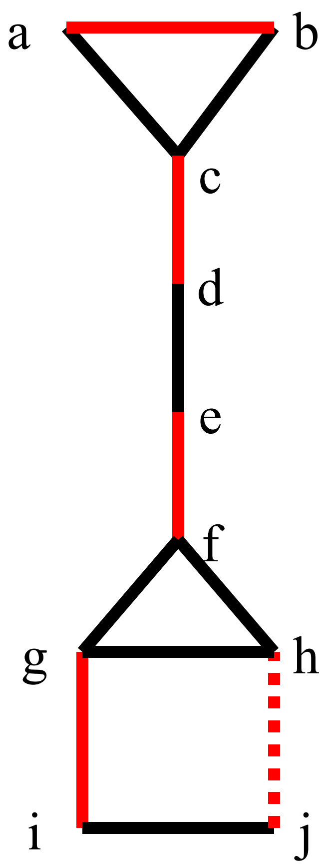
Arc consistency

- We propose an algorithm in nO(m)=O(nm)
- Berge's theorem:
An edge belong to some but not all maximum matching, iff, for an arbitrary matching it belongs to either an even alternating path which begins at a free vertex, or an even alternating cycle.

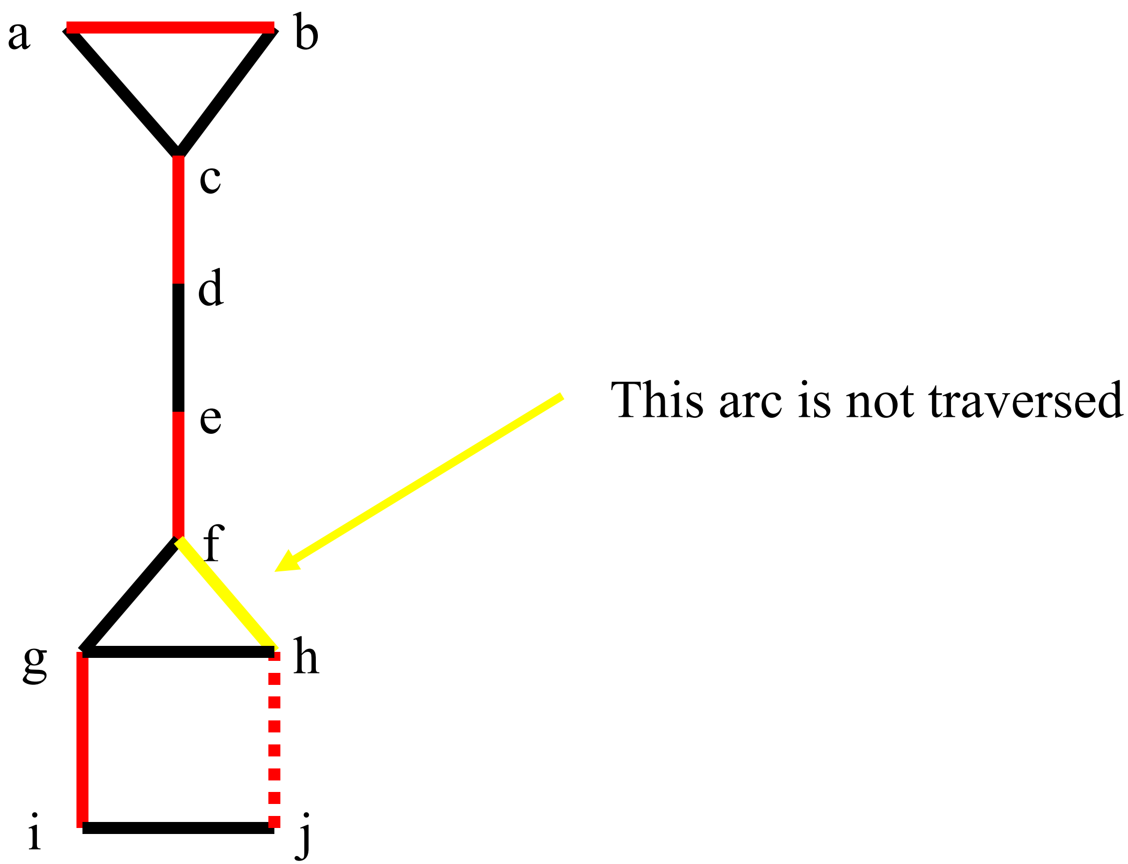
Arc consistency

- No free vertex. Thus only even alternating cycles have to be identified
- **Idea**:
  alternating cycle = a matching edge {u,v}
  + alternating path from v to u
- **Algorithm**: for each matching edge {u,v} in turn we identify the edge {w,u} that can form a cycle with an alternating path from v
  (idem from u)


a
b
c
d
e
f
g
h
i
j



The search starts from j


a
b
c
d
e
f
g
h
i
j
This arc is not traversed

Arc consistency

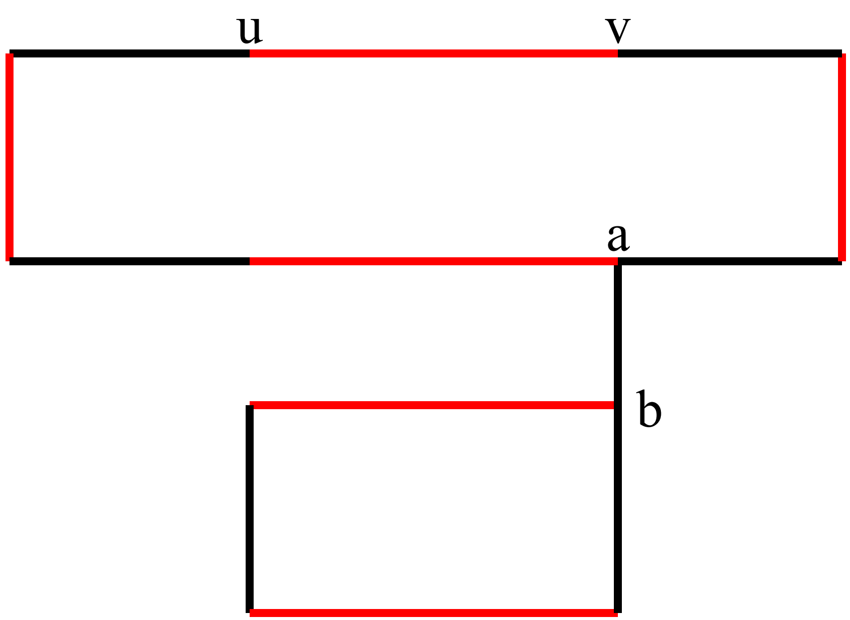
- Algorithm in $O(nm)=O(n^2d)$
- Problems:
  - non incremental algorithm
  - complexity too high for certain applications
- Solution: a filtering algorithm with a lower complexity

Filtering algorithm

- New Proposition:

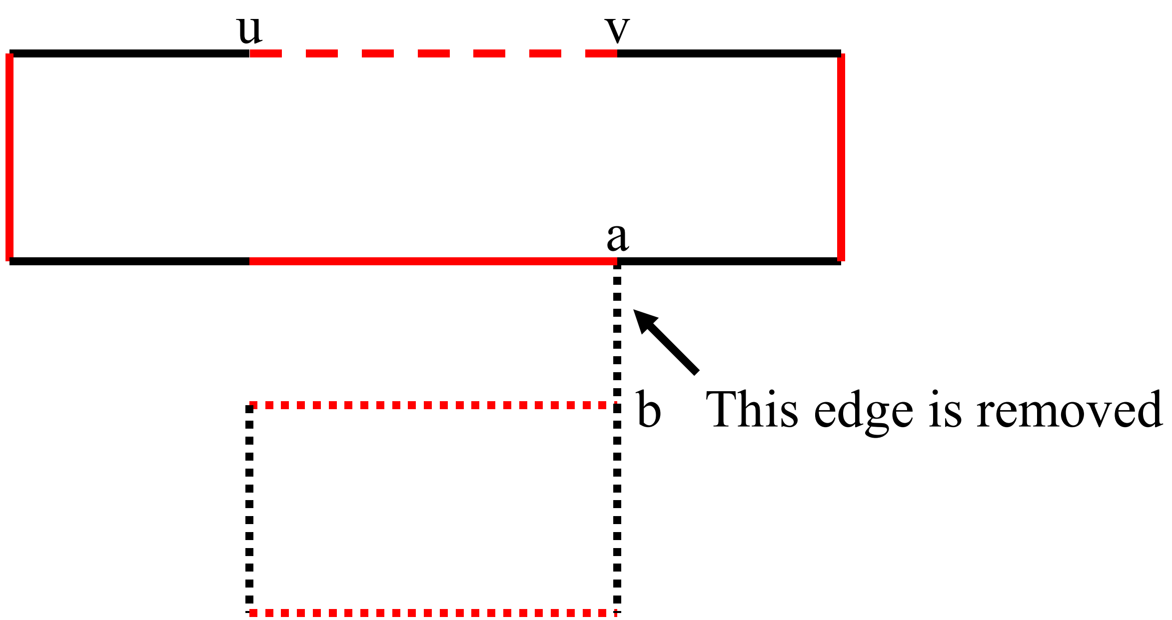
  During the search for an alternating path: If one of the extremities of an edge is reached and if the edge is not traversed then the edge does not belong to any maximum matching

Filtering algorithm



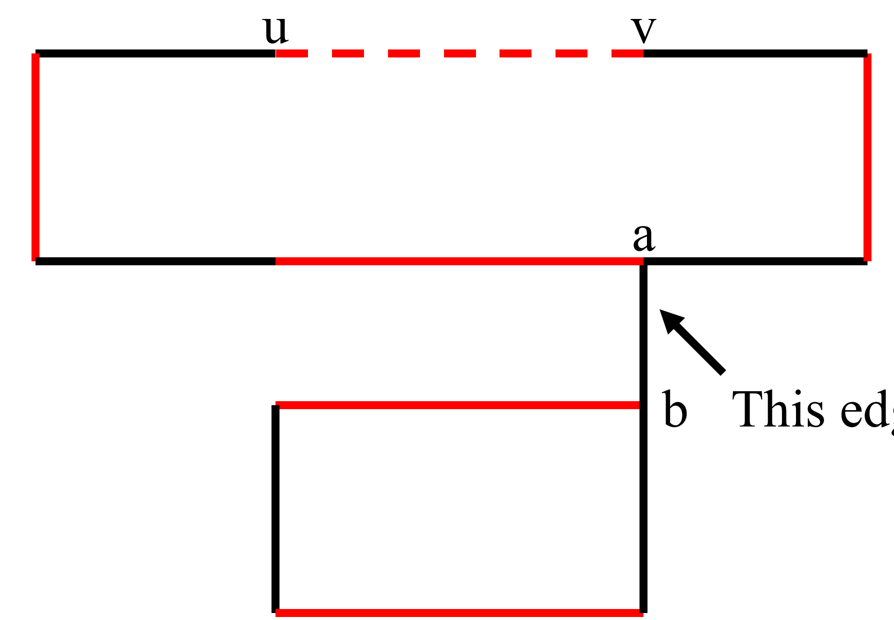
Filtering algorithm

We start from v



Filtering algorithm

We start from u



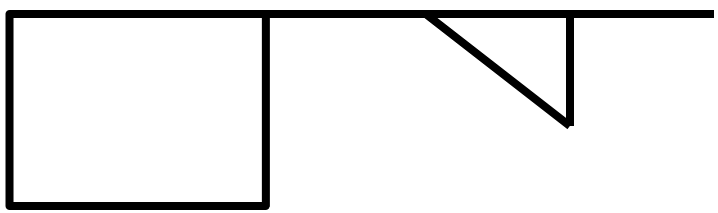
This edge is NOT removed

Filtering algorithm

- 1) An edge is arbitrary chosen
- 2) Search for an alternating path
- 3) If no deletion occurs then stop
  Else goto 1)
- Complexity O(m) per deletion
- We can also use credit/debit

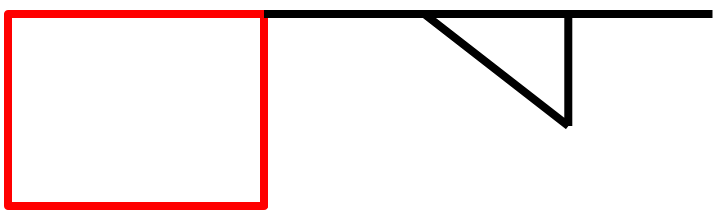
Improvements

- Use the classical alldiff constraint
- Search for 2-connected components and cutpoints



Improvements

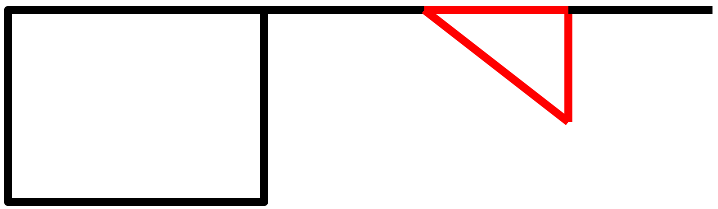
- Use the classical alldiff constraint
- Search for 2-connected components and cutpoints



2-connected component with an EVEN number of nodes

Improvements

- Use the classical alldiff constraint
- Search for 2-connected components and cutpoints



2-connected component with an ODD number of nodes

Results

- Best compromise is:
  - maintain the consistency by maintaining a maximum matching in a non bipartite graph
  - use the filtering algorithm we propose
  - use the classical alldiff AC algorithm
  - use the improvements we proposed

Symmetric alldiff

- Consistency for a symmetric alldiff constraint: O(nm), incremental algorithm (similar to alldiff)
- Arc consistency for a symmetric alldiff constraint: O(nm) (alldiff O(m))
- Filtering algorithm: O(m) per deletion

References

- Graph Theory: books of Tarjan, Lawler, Berge, Golumbic, Gondran & Minoux
- Flows: books of Ahuja & Magnanti & Orlin, Ford & Fulkerson
- Integration of OR in CP: book of Milano.

Conclusion

- Filtering algorithms are quite important
- Global constraints are quite important
- Flows for integers and matchings are powerful
- Integration of flow algorithms or matching algorithm in filtering algorithm dramatically improve CP
- A lot of work can be done on this integration!