Tradeoffs in routing reconfiguration problems

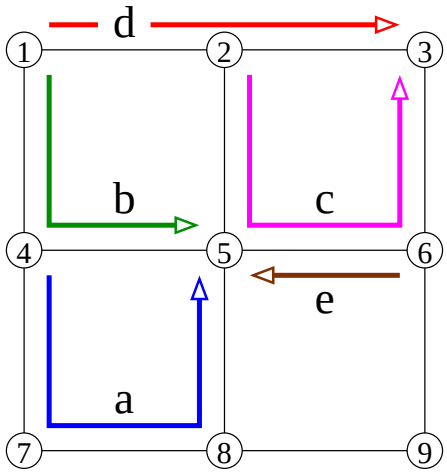
N. Cohen, D. Coudert, D. Mazauric, N. Nepomuceno, N. Nisse

MASCOTTE, INRIA, I3S, CNRS, Univ. Nice Sophia Antipolis
Sophia Antipolis, France

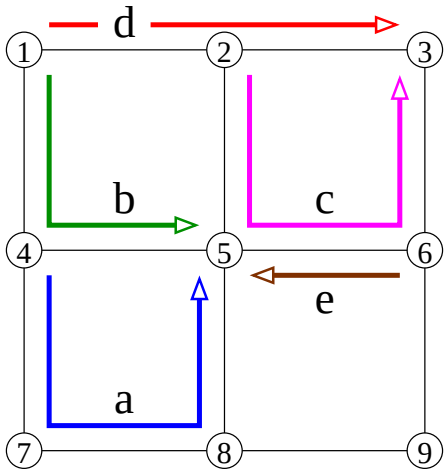
AlgoTel'10, Belle Dune

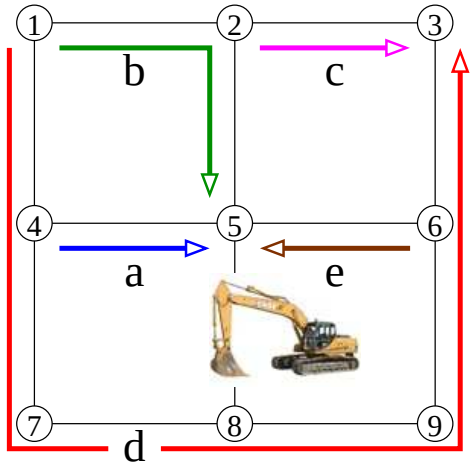
June 3rd 2010

How to change the routing ?



How to change the routing ?





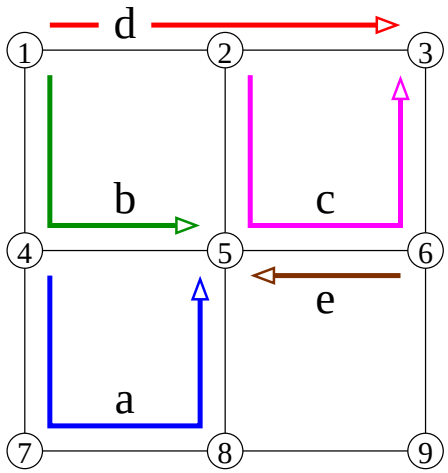
Motivation

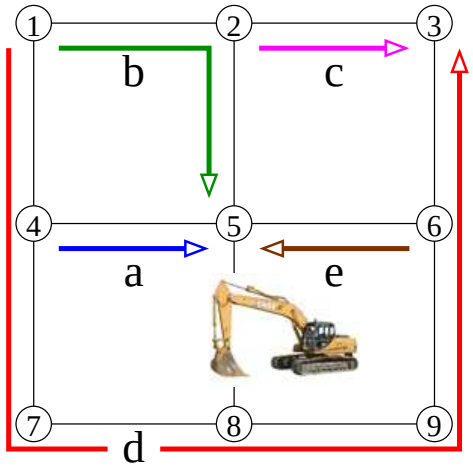
- Maintenance operations
- Optimize usage of resources (reduce blocking probability)
- Fault tolerance

Context

- Telephone: call repacking (70's)
- WDM, MPLS, ATM

Possible approaches





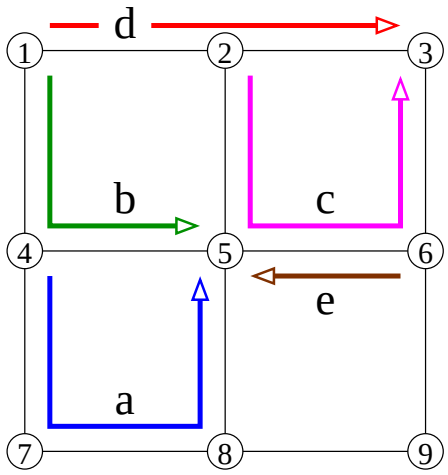
Move-To-Vacant

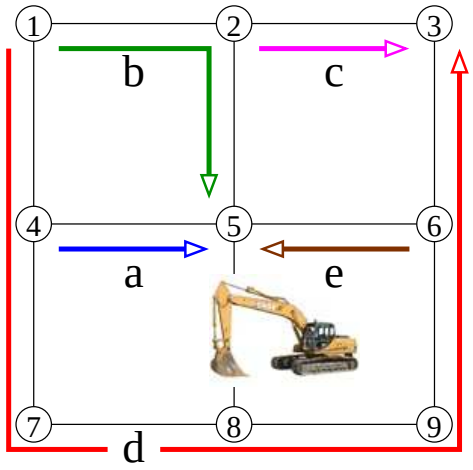
- Sequence of rerouting to converge to a valid routing
- ? Convergence time
- ? Existence

Compute the *closest* new *optimal* routing

- Minimizing number of changes (different routes)
- ? How to switch from current to new routing

The routing reconfiguration problem





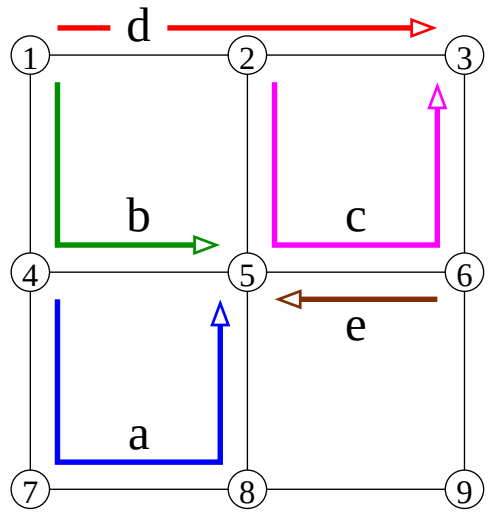
Inputs: Set of connection requests
+ current **and** new routing

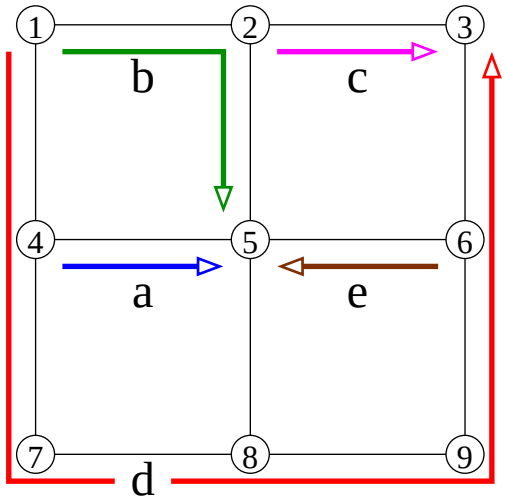
Output: Scheduling for rerouting connection requests from current to new routes

Constraint: A connection is switched only once

ObjectiveS: *detailed later*

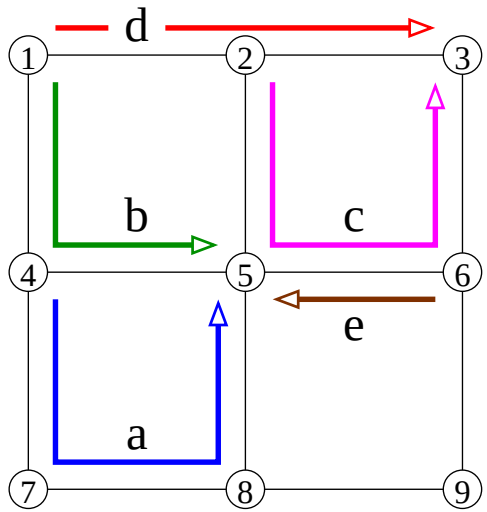
Dependency digraph

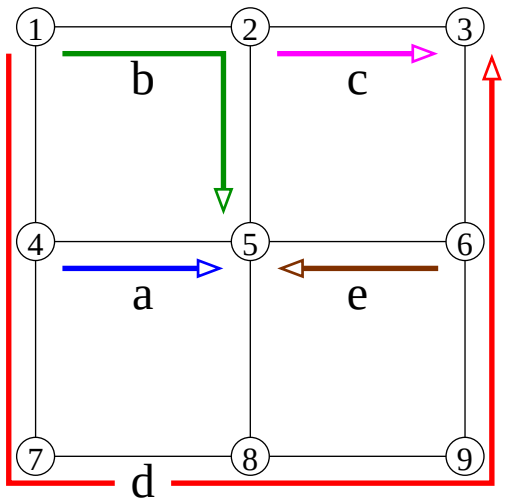




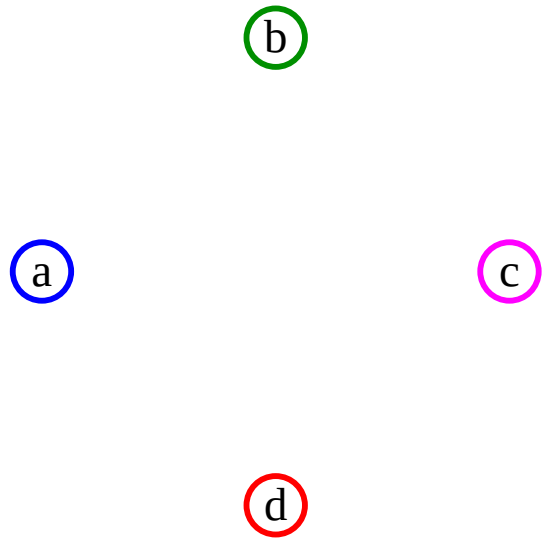
Arc from u to v if resources needed by u in $\mathcal{F}$ are used by v in $\mathcal{I}$

Dependency digraph

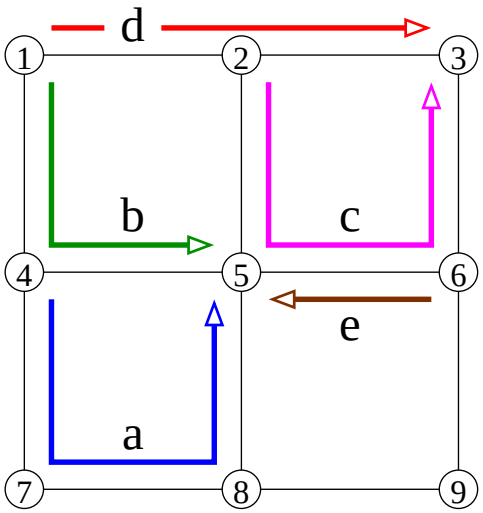


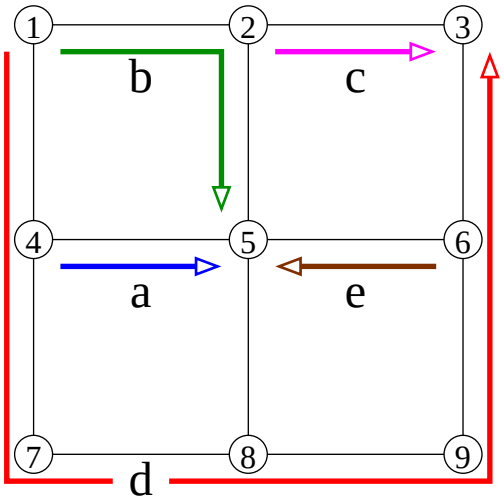


Arc from u to v if resources needed by u in $\mathcal{F}$ are used by v in $\mathcal{I}$

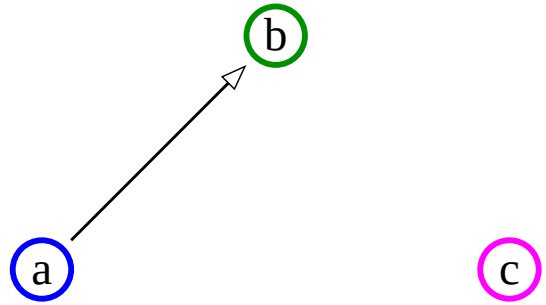


Dependency digraph



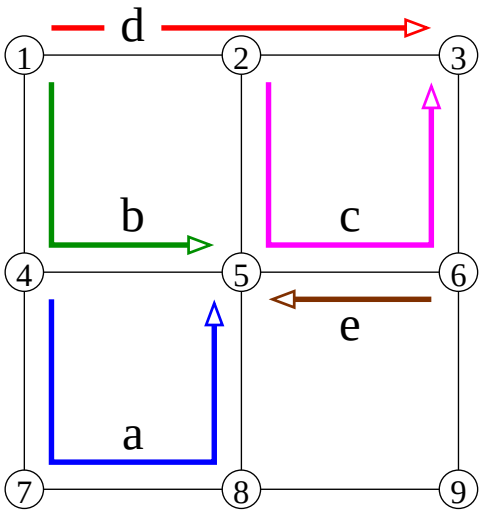


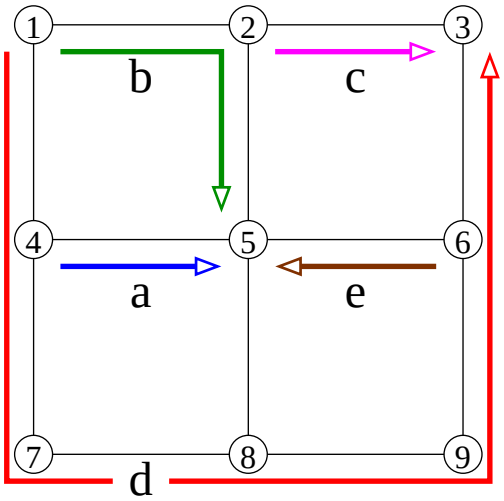
Arc from u to v if resources needed by u in $\mathcal{F}$ are used by v in $\mathcal{I}$



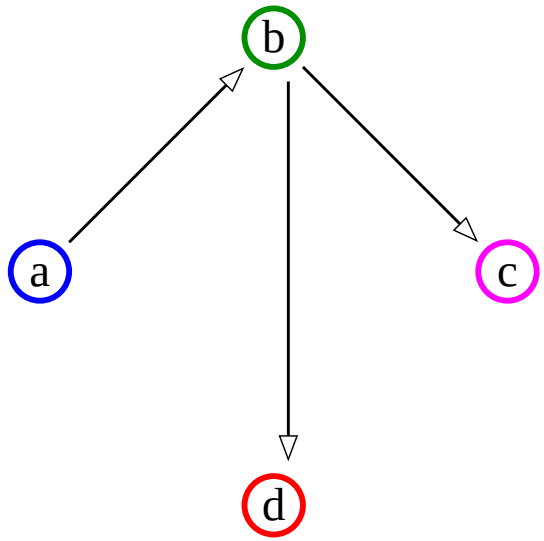
b before a

Dependency digraph



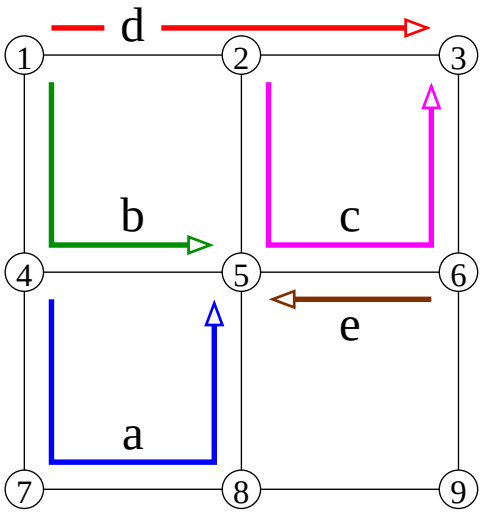


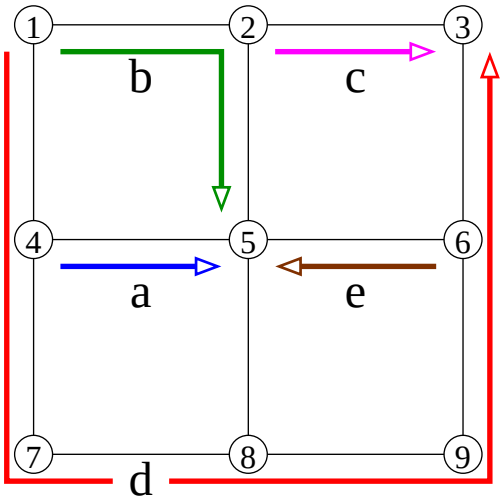
Arc from u to v if resources needed by u in $\mathcal{F}$ are used by v in $\mathcal{I}$



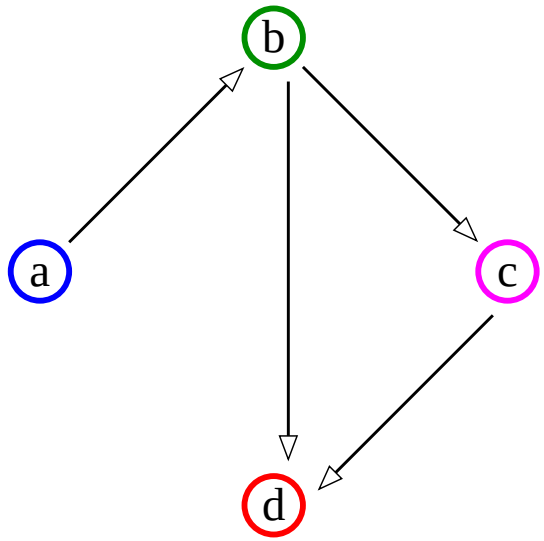
b before a
c and d before b

Dependency digraph



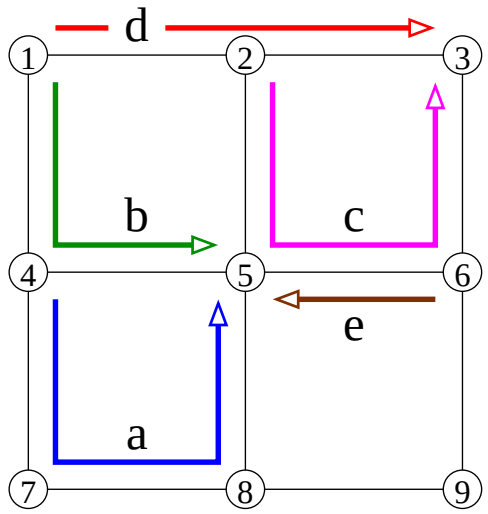


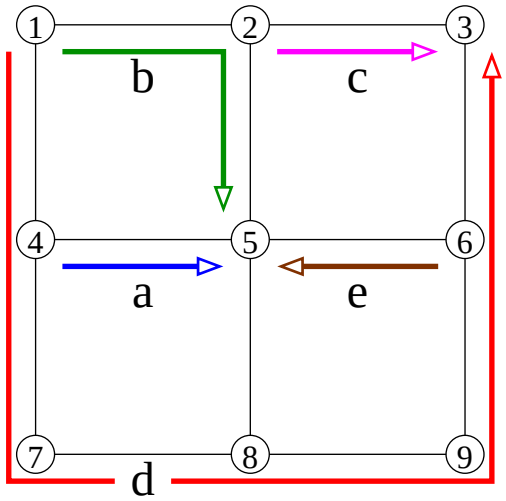
Arc from u to v if resources needed by u in $\mathcal{F}$ are used by v in $\mathcal{I}$



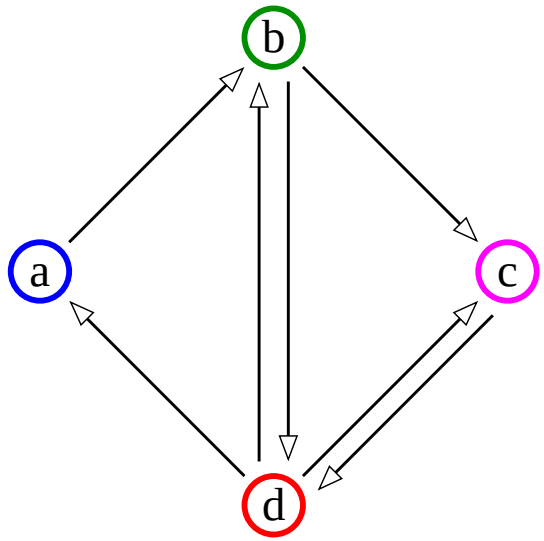
b before a
c and d before b
d before c

Dependency digraph



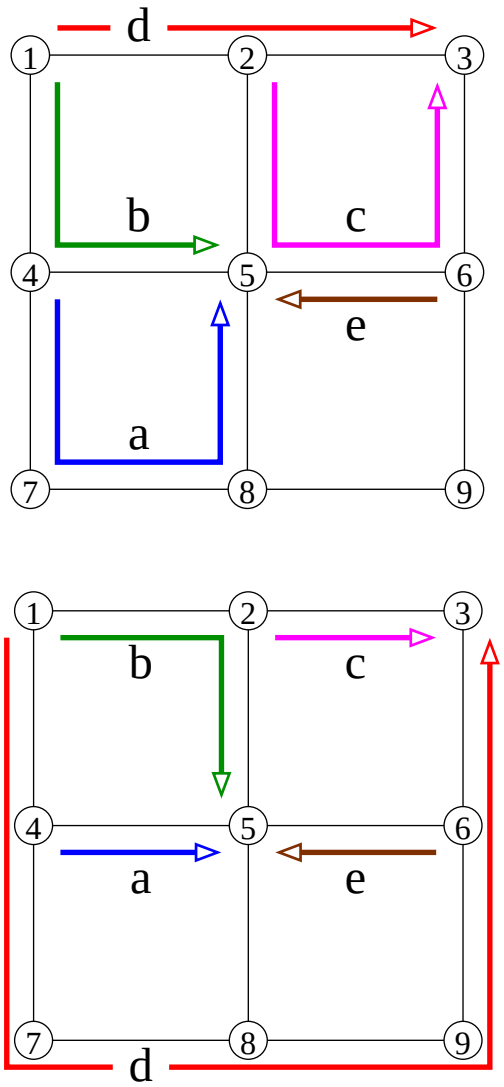


Arc from u to v if resources needed by u in $\mathcal{F}$ are used by v in $\mathcal{I}$

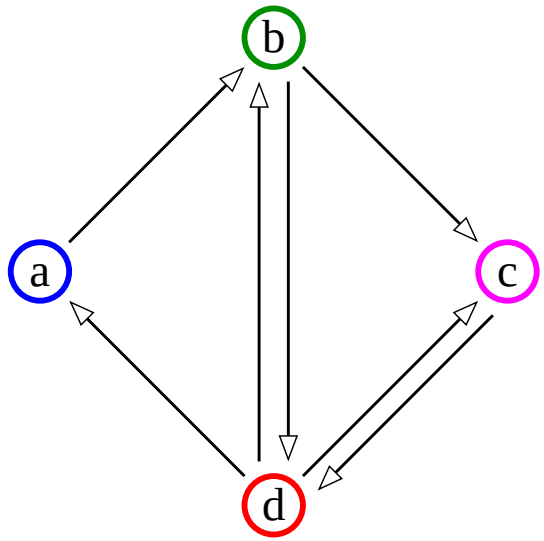


b before a
c and d before b
d before c
a, b, c before d

Dependency digraph



Arc from u to v if resources needed by u in $\mathcal{F}$ are used by v in $\mathcal{I}$



Cyclic dependencies
$\Rightarrow$ Interruptions required

Modeling, process strategy

A game with agents on the dependency digraph

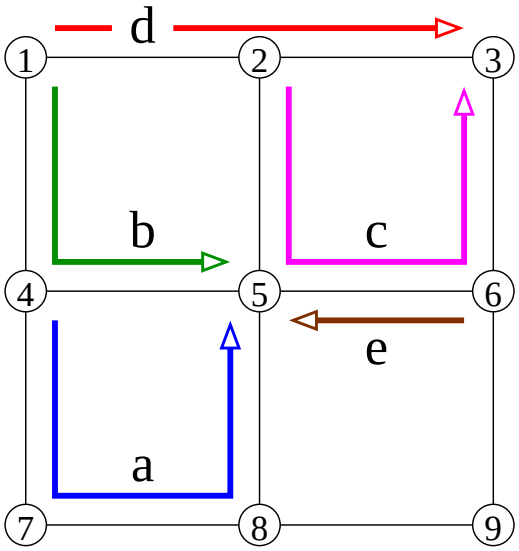
break-before-make

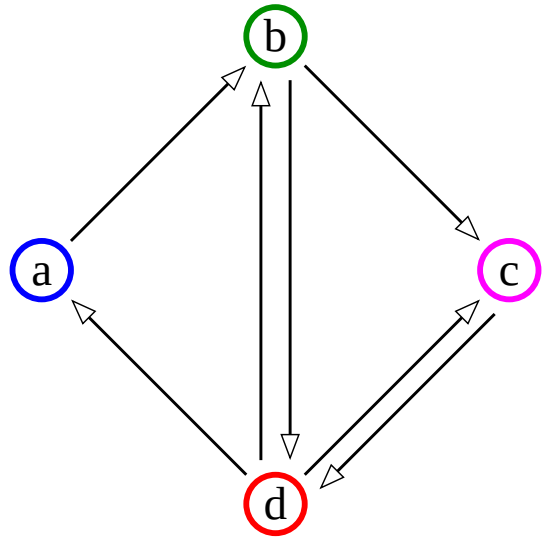
break a connection $\Longleftrightarrow$ place an agent on corresponding node
establish new path $\Longleftrightarrow$ remove agent from the node

Rules of the game

- R_1 Place an agent at a node = interrupt a connection
- R_2 Process a node if all its out-neighbors are either processed or occupied by an agent
  - = (Re)route a connection when final resources are available
- R_3 Remove an agent from a node, after having processed it

Example



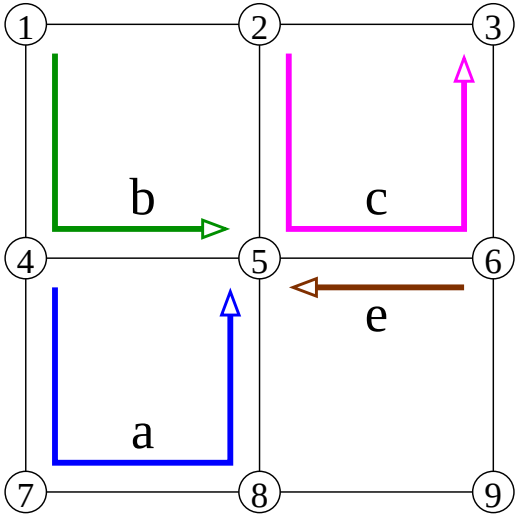


Cyclic dependencies
$\Rightarrow$ Interruption required

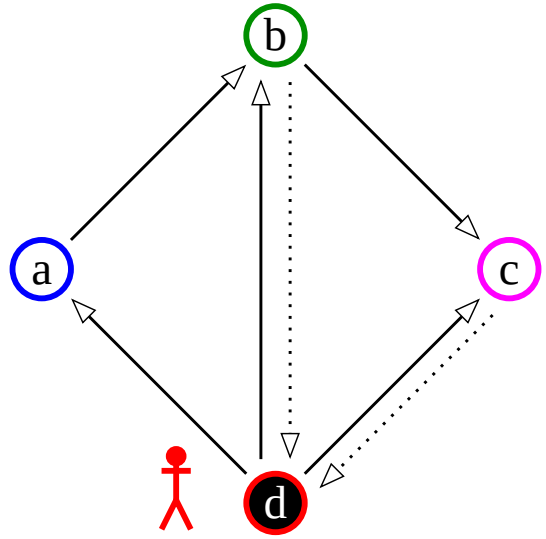
Rules

- R_1 Place an agent at a node = interrupt a connection
- R_2 Process a node if all its out-neighbors are either processed or occupied by an agent
  - = (Re)route a connection when final resources are available
- R_3 Remove an agent from a node, after having processed it

Example



break request d

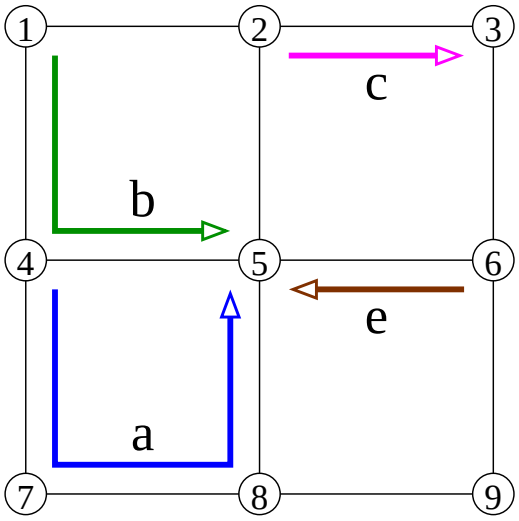


put an agent on node d

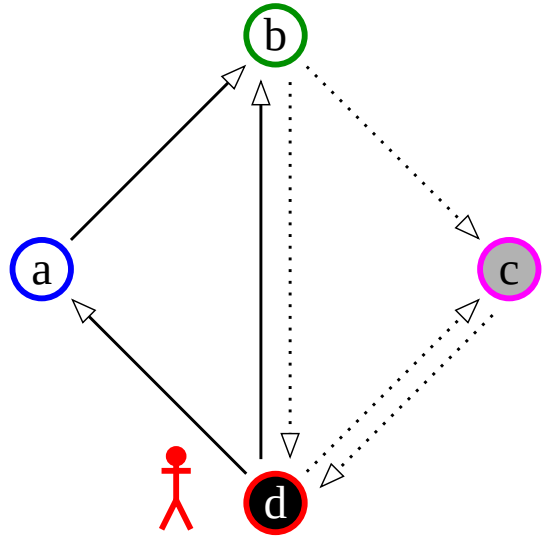
Rules

- R_1 Place an agent at a node = interrupt a connection
- R_2 Process a node if all its out-neighbors are either processed or occupied by an agent
  - = (Re)route a connection when final resources are available
- R_3 Remove an agent from a node, after having processed it

Example



reroute request c

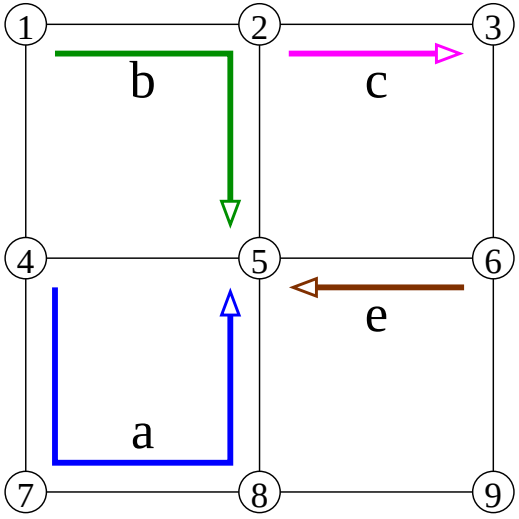


process node c

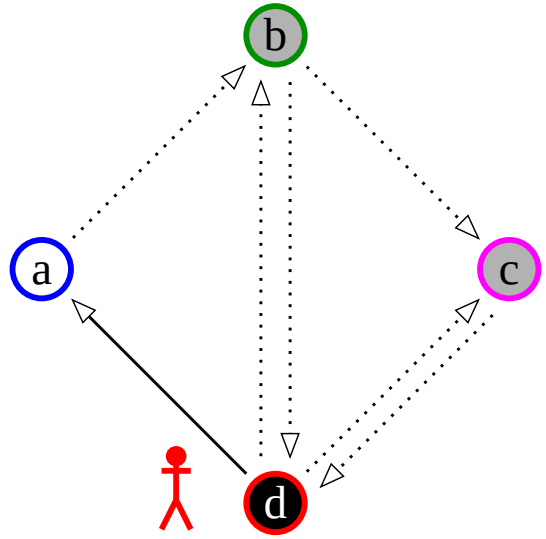
Rules

- R_1 Place an agent at a node = interrupt a connection
- R_2 Process a node if all its out-neighbors are either processed or occupied by an agent
  - = (Re)route a connection when final resources are available
- R_3 Remove an agent from a node, after having processed it

Example



reroute request b

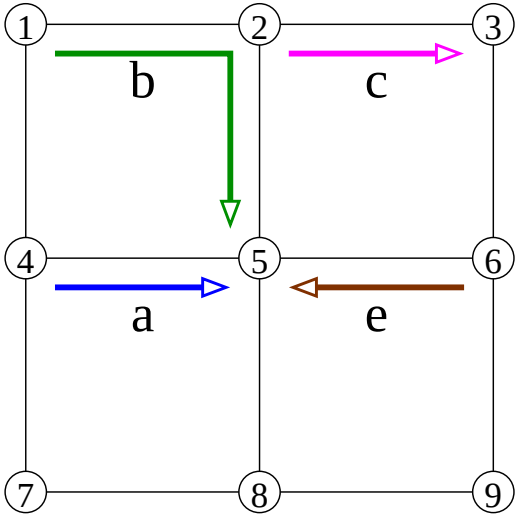


process node b

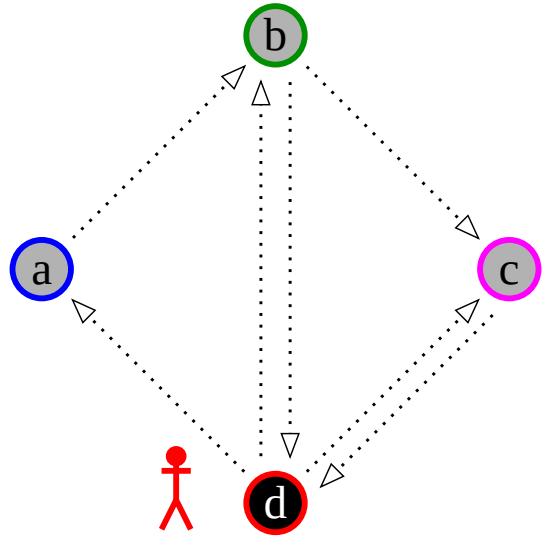
Rules

- R_1 Place an agent at a node = interrupt a connection
- R_2 Process a node if all its out-neighbors are either processed or occupied by an agent
  - = (Re)route a connection when final resources are available
- R_3 Remove an agent from a node, after having processed it

Example



reroute request *a*

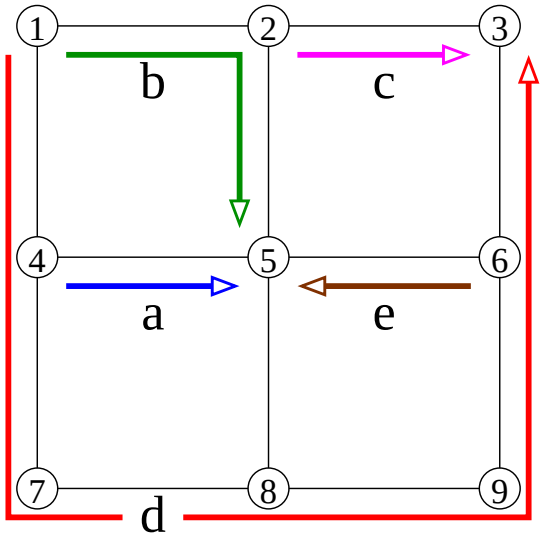


process node *a*

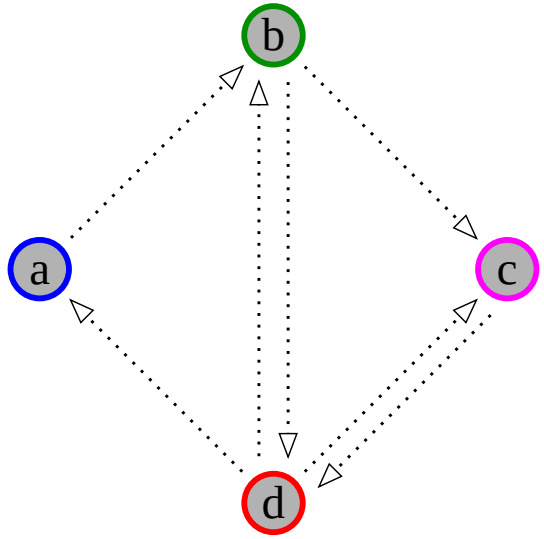
Rules

- R_1 Place an agent at a node = interrupt a connection
- R_2 Process a node if all its out-neighbors are either processed or occupied by an agent
  - = (Re)route a connection when final resources are available
- R_3 Remove an agent from a node, after having processed it

Example



route request d



process node d
and remove agent

Rules

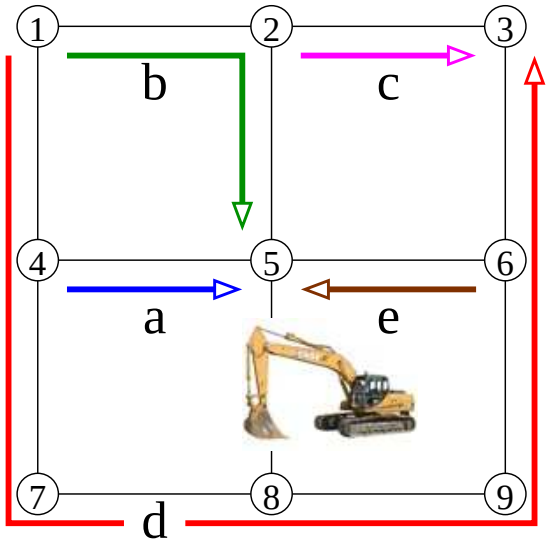
R_1 Place an agent at a node = interrupt a connection

R_2 Process a node if all its out-neighbors are either processed or occupied by an agent

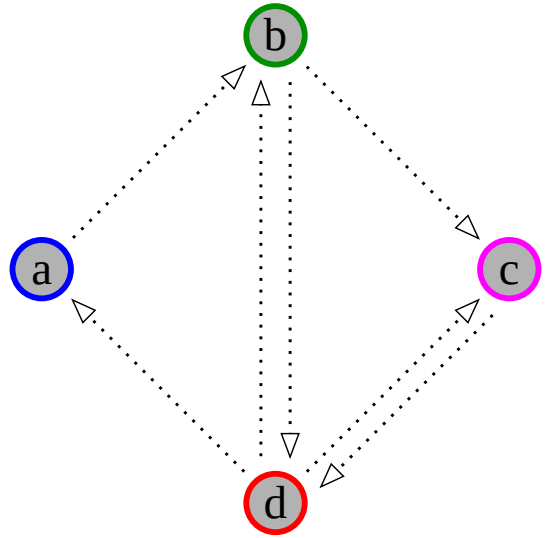
= (Re)route a connection when final resources are available

R_3 Remove an agent from a node, after having processed it

Example



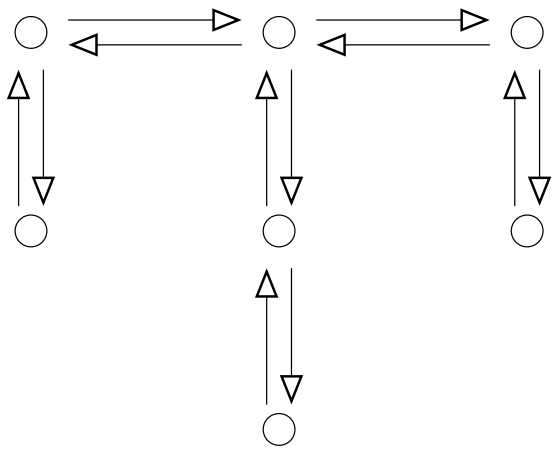
Ready for maintenance operation



Rules

- R_1 Place an agent at a node = interrupt a connection
- R_2 Process a node if all its out-neighbors are either processed or occupied by an agent
  - = (Re)route a connection when final resources are available
- R_3 Remove an agent from a node, after having processed it

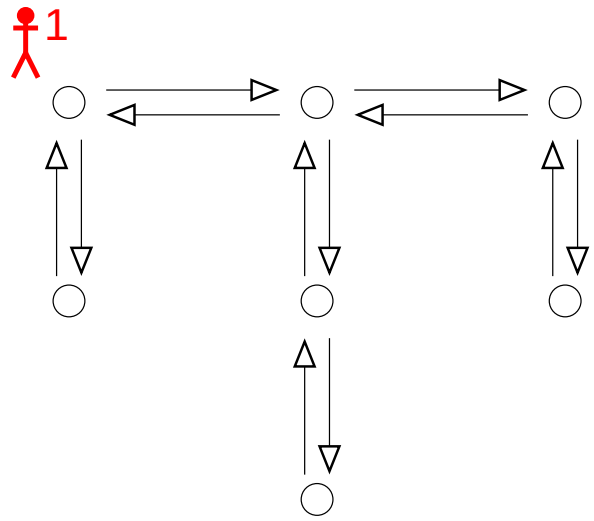
Example



Rules

- R_1 Place an agent at a node = interrupt a connection
- R_2 Process a node if all its out-neighbors are either processed or occupied by an agent
  - = (Re)route a connection when final resources are available
- R_3 Remove an agent from a node, after having processed it

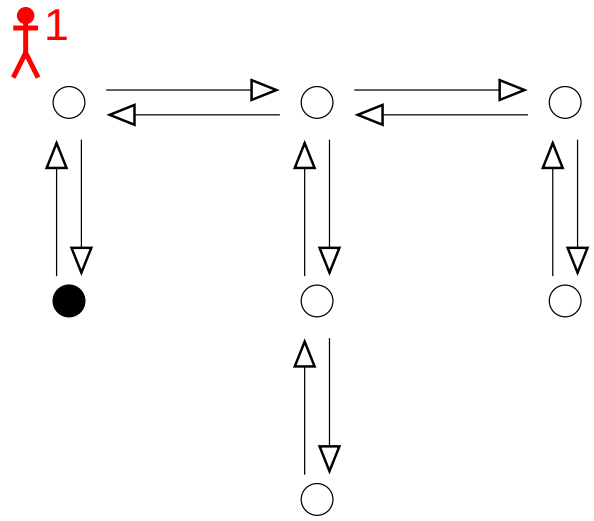
Example



Rules

- R_1 Place an agent at a node = interrupt a connection
- R_2 Process a node if all its out-neighbors are either processed or occupied by an agent
  - = (Re)route a connection when final resources are available
- R_3 Remove an agent from a node, after having processed it

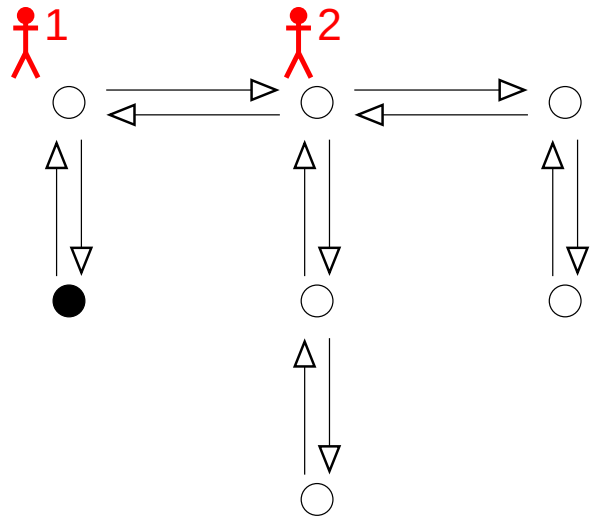
Example



Rules

- R_1 Place an agent at a node = interrupt a connection
- R_2 Process a node if all its out-neighbors are either processed or occupied by an agent
  - = (Re)route a connection when final resources are available
- R_3 Remove an agent from a node, after having processed it

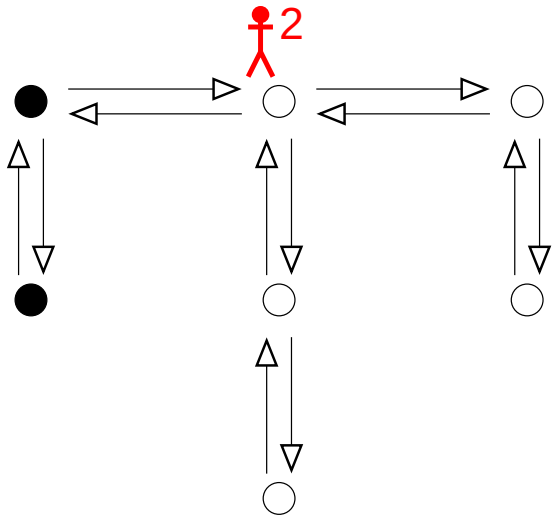
Example



Rules

- R_1 Place an agent at a node = interrupt a connection
- R_2 Process a node if all its out-neighbors are either processed or occupied by an agent
  - = (Re)route a connection when final resources are available
- R_3 Remove an agent from a node, after having processed it

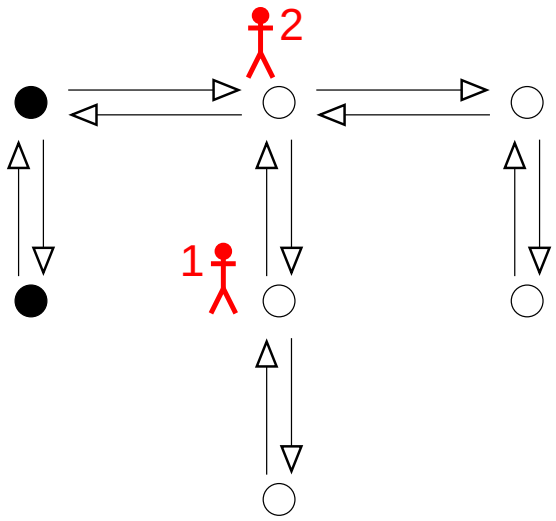
Example



Rules

- R_1 Place an agent at a node = interrupt a connection
- R_2 Process a node if all its out-neighbors are either processed or occupied by an agent
  - = (Re)route a connection when final resources are available
- R_3 Remove an agent from a node, after having processed it

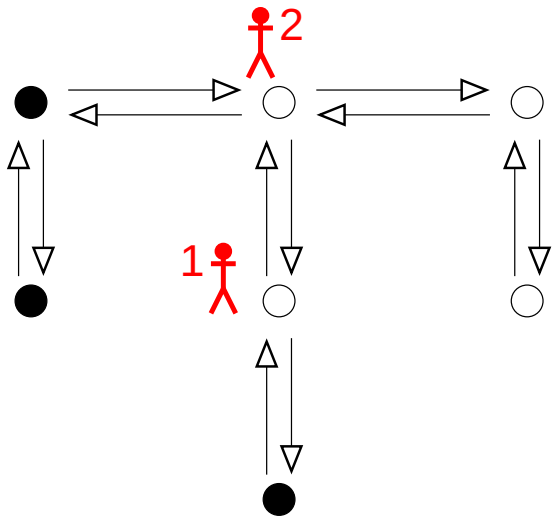
Example



Rules

- R_1 Place an agent at a node = interrupt a connection
- R_2 Process a node if all its out-neighbors are either processed or occupied by an agent
  - = (Re)route a connection when final resources are available
- R_3 Remove an agent from a node, after having processed it

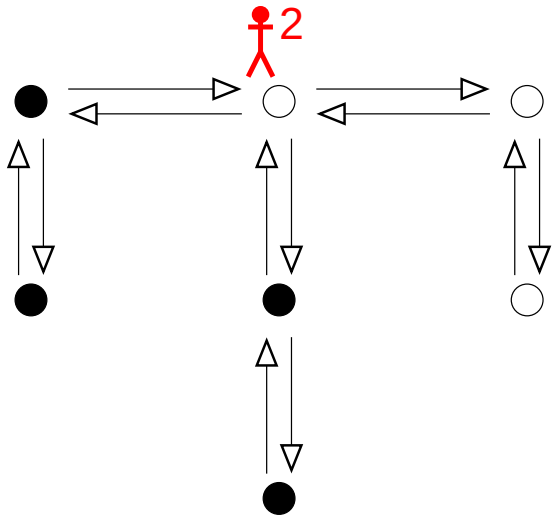
Example



Rules

- R_1 Place an agent at a node = interrupt a connection
- R_2 Process a node if all its out-neighbors are either processed or occupied by an agent
  - = (Re)route a connection when final resources are available
- R_3 Remove an agent from a node, after having processed it

Example



Rules

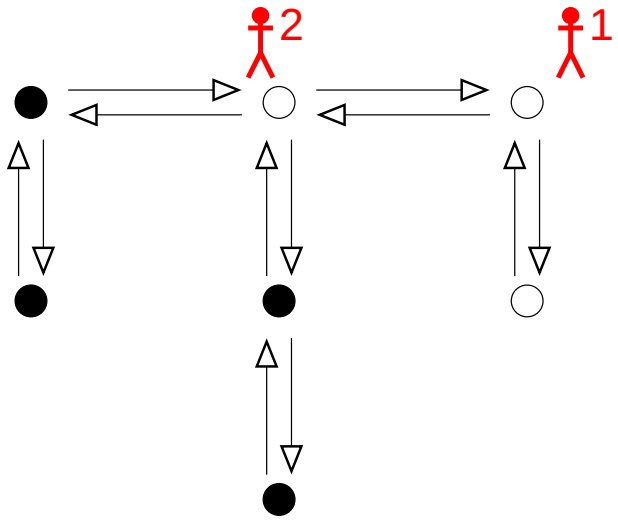
R_1 Place an agent at a node = interrupt a connection

R_2 Process a node if all its out-neighbors are either processed or occupied by an agent

= (Re)route a connection when final resources are available

R_3 Remove an agent from a node, after having processed it

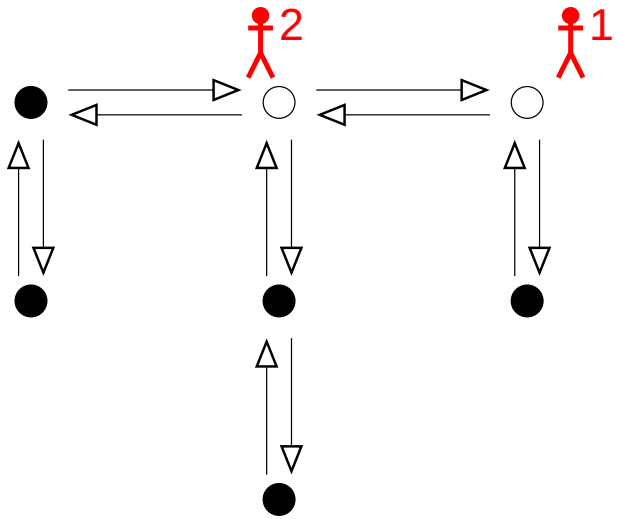
Example



Rules

- R_1 Place an agent at a node = interrupt a connection
- R_2 Process a node if all its out-neighbors are either processed or occupied by an agent
  - = (Re)route a connection when final resources are available
- R_3 Remove an agent from a node, after having processed it

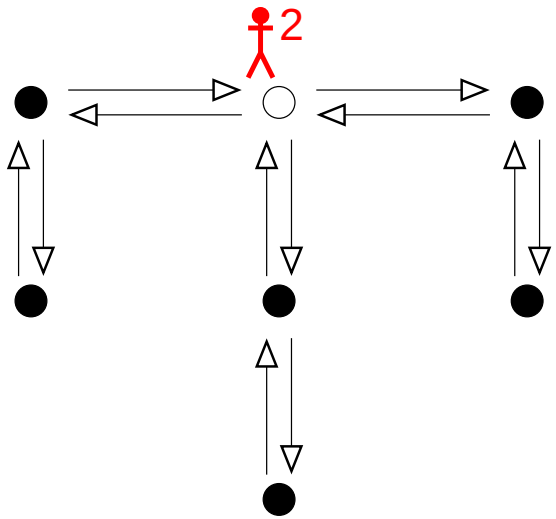
Example



Rules

- R_1 Place an agent at a node = interrupt a connection
- R_2 Process a node if all its out-neighbors are either processed or occupied by an agent
  - = (Re)route a connection when final resources are available
- R_3 Remove an agent from a node, after having processed it

Example



Rules

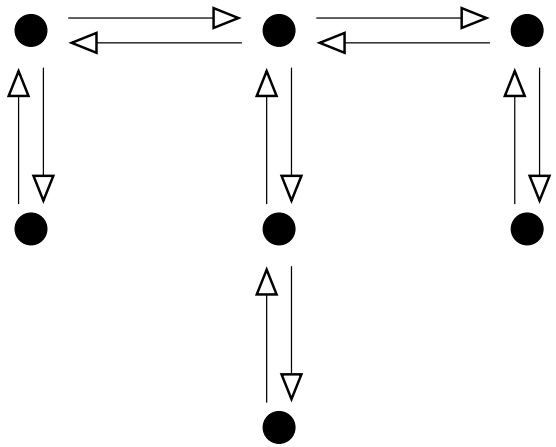
R_1 Place an agent at a node = interrupt a connection

R_2 Process a node if all its out-neighbors are either processed or occupied by an agent

= (Re)route a connection when final resources are available

R_3 Remove an agent from a node, after having processed it

Example

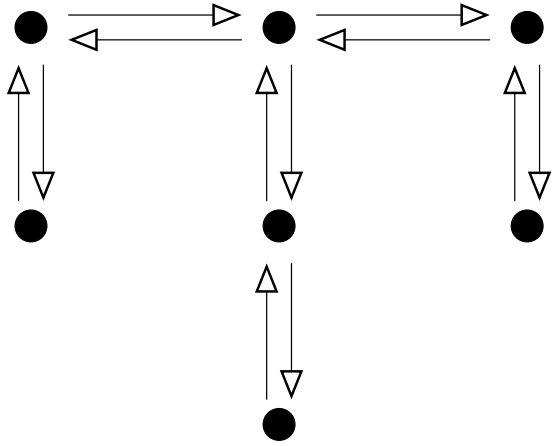


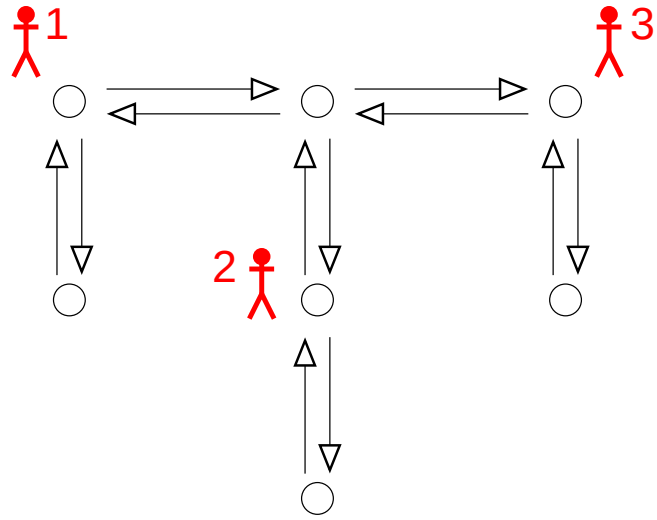
2 agents, 4 interruptions

Rules

- R_1 Place an agent at a node = interrupt a connection
- R_2 Process a node if all its out-neighbors are either processed or occupied by an agent
  - = (Re)route a connection when final resources are available
- R_3 Remove an agent from a node, after having processed it

Example



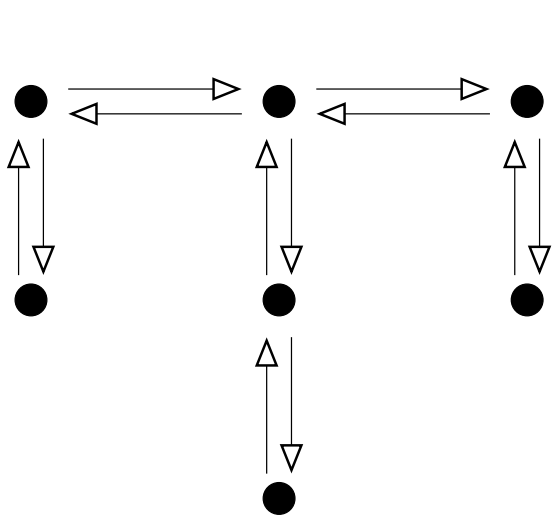


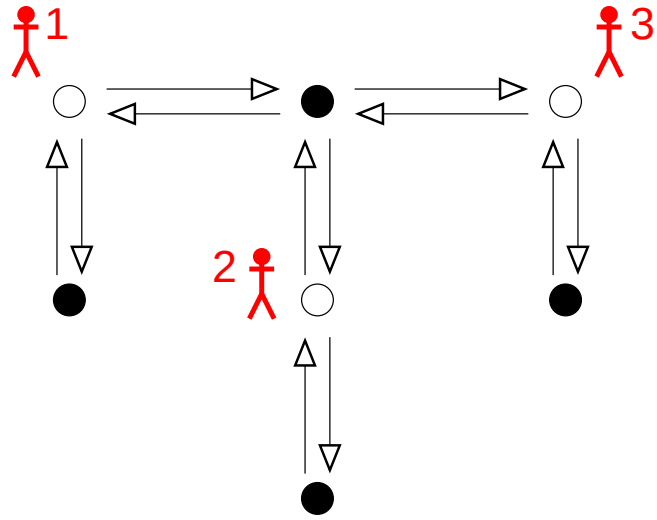
2 agents, 4 interruptions

Rules

- R_1 Place an agent at a node = interrupt a connection
- R_2 Process a node if all its out-neighbors are either processed or occupied by an agent
  - = (Re)route a connection when final resources are available
- R_3 Remove an agent from a node, after having processed it

Example



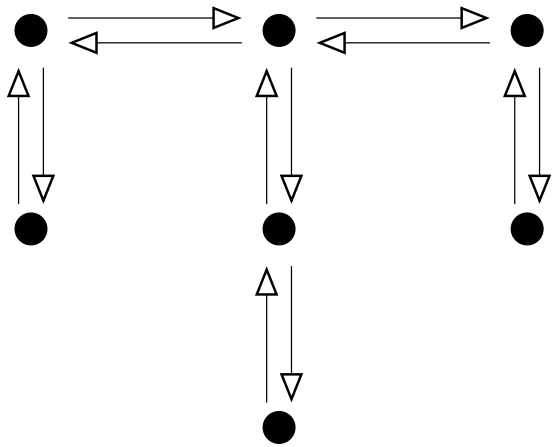


2 agents, 4 interruptions

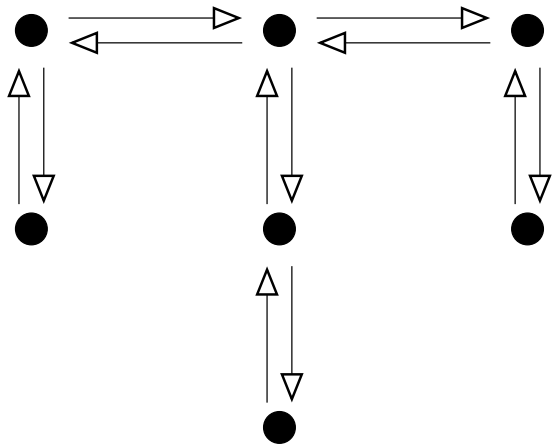
Rules

- R_1 Place an agent at a node = interrupt a connection
- R_2 Process a node if all its out-neighbors are either processed or occupied by an agent
  - = (Re)route a connection when final resources are available
- R_3 Remove an agent from a node, after having processed it

Example



2 agents, 4 interruptions



3 agents, 3 interruptions

Rules

- R_1 Place an agent at a node = interrupt a connection
- R_2 Process a node if all its out-neighbors are either processed or occupied by an agent
  - = (Re)route a connection when final resources are available
- R_3 Remove an agent from a node, after having processed it

Objectives (on the dependency digraph)

Minimize...

- Total number of interruptions = minimum feedback vertex set
  $\Leftrightarrow$ # **of occupied vertices**
- Number of simultaneous interruptions = process number, pn
  $\Leftrightarrow$ # **of agents used**
  - Vertex separation, vs $\qquad vs(D) \leq pn(D) \leq vs(D)+1$
  - Pathwidth, pw
  - Node search number, ns $\qquad pw(G) = vs(G) = ns(G)-1$

cops-and-robber games

Complexity

- All problems are NP-hard and "computing pn" is not in APX

Tradeoffs

- # agents given # of occupied vertices
- # of occupied vertices given # agents

Objectives (on the dependency digraph)

Minimize...

- Total number of interruptions = minimum feedback vertex set
  $\Leftrightarrow$ # **of occupied vertices**
- Number of simultaneous interruptions = process number, pn
  $\Leftrightarrow$ # **of agents used**
  - Vertex separation, vs $\qquad vs(D) \leq pn(D) \leq vs(D)+1$
  - Pathwidth, pw
  - Node search number, ns $\qquad pw(G) = vs(G) = ns(G)-1$

cops-and-robber games

Complexity

- All problems are NP-hard and "computing pn" is not in APX

Tradeoffs

- # agents given # of occupied vertices
- # of occupied vertices given # agents

State of the art

Total number of interruptions,

- Dependancy digraph, heuristics [Jose, Somani'03]

Simultaneous interruptions, Process strategy

- Modeling as a cops-and-robber like game
- Complexity and hardness [Coudert, Perennes, Pham, Sereni'05]
- Characterization of (di)graphs with process number 0, 1, 2 [Coudert, Sereni'07]
- Heuristic algorithms [Coudert, Huc, Mazauric, Nisse, Sereni'09]
- Generalized model allowing bandwidth sharing [Coudert,Mazauric, Nisse'09]

Tradeoffs

- Linear Programs, heuristic [Solano'09]

Tradeoff Parameters

Objectives

Min. number of simulteanous interruptions **given** total number of interruptions

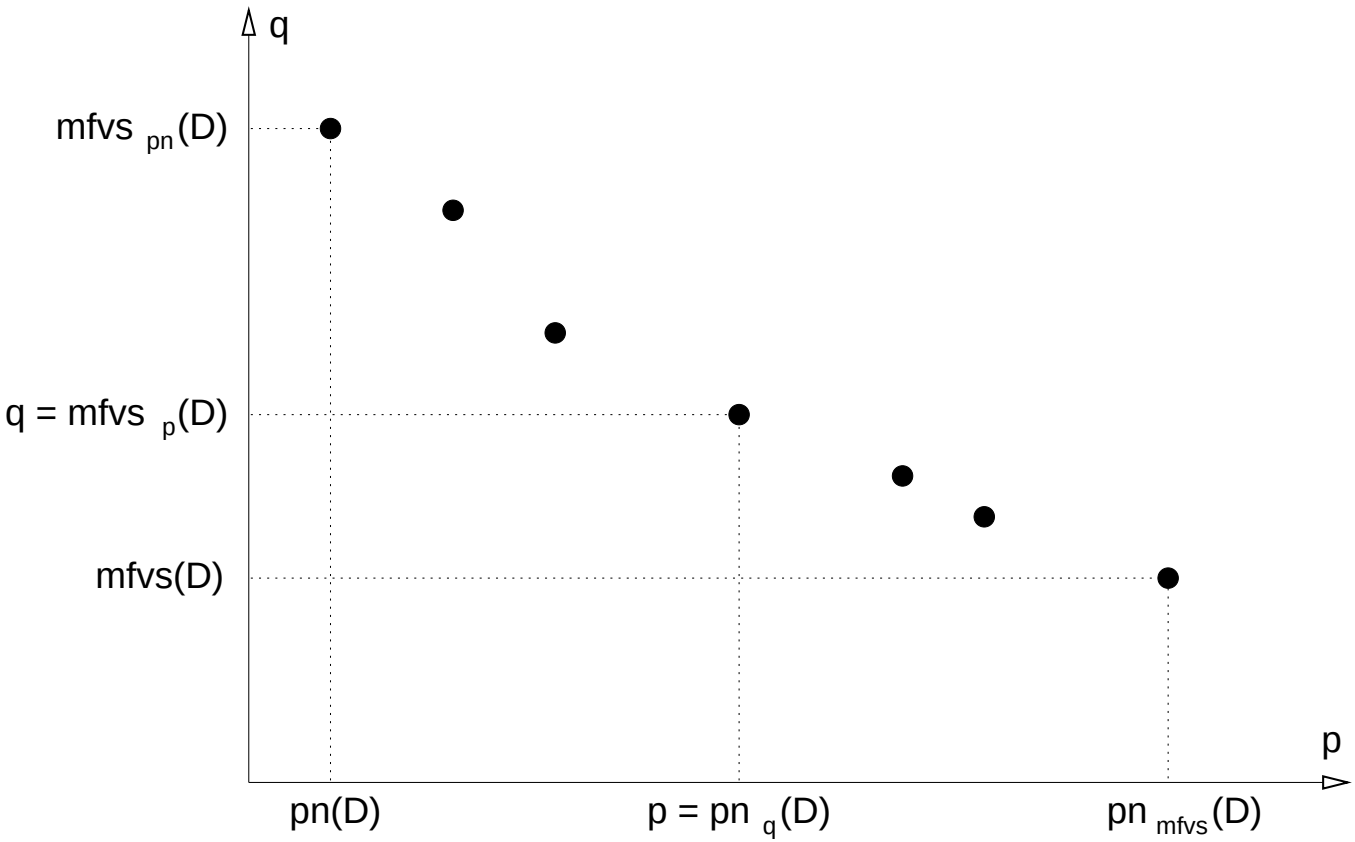
- $pn_m(D)$ = min. agents to occupy at most m nodes

Min. total number of interruptions **given** total maximum number of simulteanous interruptions

- $mfvs_p(D)$ = min. occupied nodes using p agents

Tradeoff Parameters

Representation of Minimal Values



Tradeoff Parameters

Complexity Results

Theorem

Let $\alpha \in [0,1]$. The problem that takes a digraph D as an input and aims at minimizing:

- $\alpha.pn_{mfvs}(D) + (1-\alpha)pn(D)$ *is not in APX,*
- $\alpha.mfvs_{pn}(D) + (1-\alpha)mfvs(D)$ *is APX-hard,*

Impossibility of Tradeoff

Unbounded Ratios

$\frac{pn_{mfvs+p}(D)}{pn(D)}$: how the number of agents increases if constrained to occupy "few" vertices? $(p \geq 0)$

$\frac{mfvs_{pn+p}(D)}{mfvs(D)}$: how the number of occupied vertices increases if constrained to occupy "few" agents? $(p \geq 0)$

Theorem

$\forall p \geq 0$, $\frac{pn_{mfvs+p}}{pn}$ *is not bounded* *(even in symmetric digraphs).*

Theorem

$\forall p \geq 0$, $\frac{mfvs_{pn+p}}{mfvs}$ *is not bounded.* *(even in class $pn(D) = 3$)*

Theorem

$\forall D$ *symmetric digraph,* $\frac{mfvs_{pn}(D)}{mfvs}(D) \leq pn(D)$ *(conj: ≤ 3?)*

Impossibility of Tradeoff

Unbounded Ratios

$\frac{pn_{mfvs+p}(D)}{pn(D)}$: how the number of agents increases if constrained to occupy "few" vertices? $(p \geq 0)$

$\frac{mfvs_{pn+p}(D)}{mfvs(D)}$: how the number of occupied vertices increases if constrained to occupy "few" agents? $(p \geq 0)$

Theorem

$\forall p \geq 0$, $\frac{pn_{mfvs+p}}{pn}$ *is not bounded* *(even in symmetric digraphs).*

Theorem

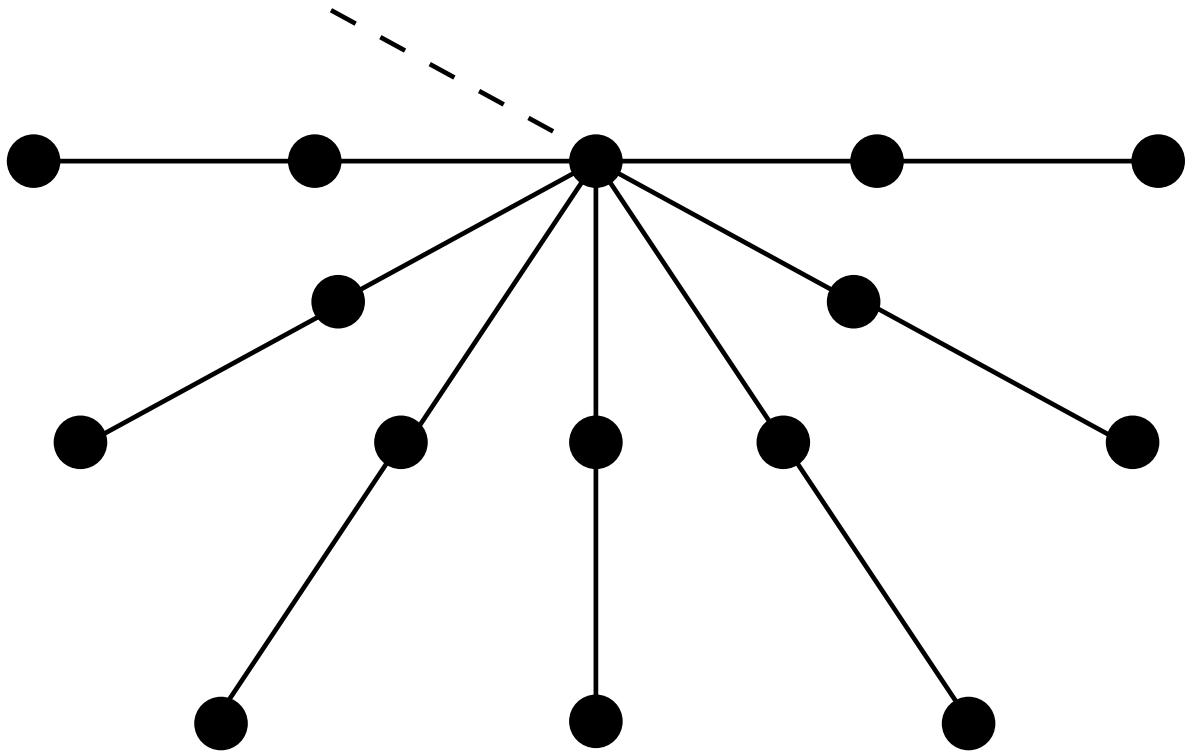
$\forall p \geq 0$, $\frac{mfvs_{pn+p}}{mfvs}$ *is not bounded.* *(even in class* $pn(D) = 3$*)*

Theorem

$\forall D$ *symmetric digraph,* $\frac{mfvs_{pn}(D)}{mfvs}(D) \leq pn(D)$ *(conj:* ≤ 3*?)*

agents for minimizing # occupied vertices

$\exists$ digraphs with arbitrary large ratio: $\frac{pn_{mfvs}(D)}{pn(D)}$.



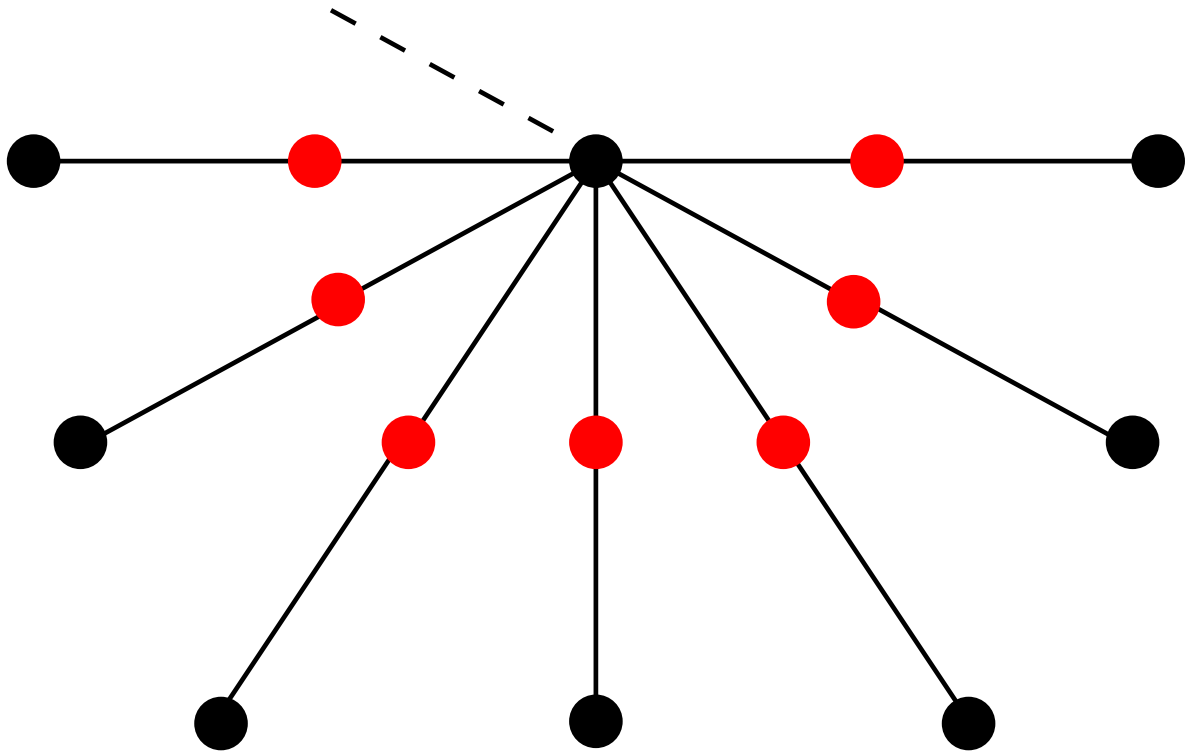
$mfvs(D) = n$

$pn(D) = 2$

$pn_{mfvs}(D) = n$

agents for minimizing # occupied vertices

$\exists$ digraphs with arbitrary large ratio: $\frac{pn_{mfvs}(D)}{pn(D)}$.



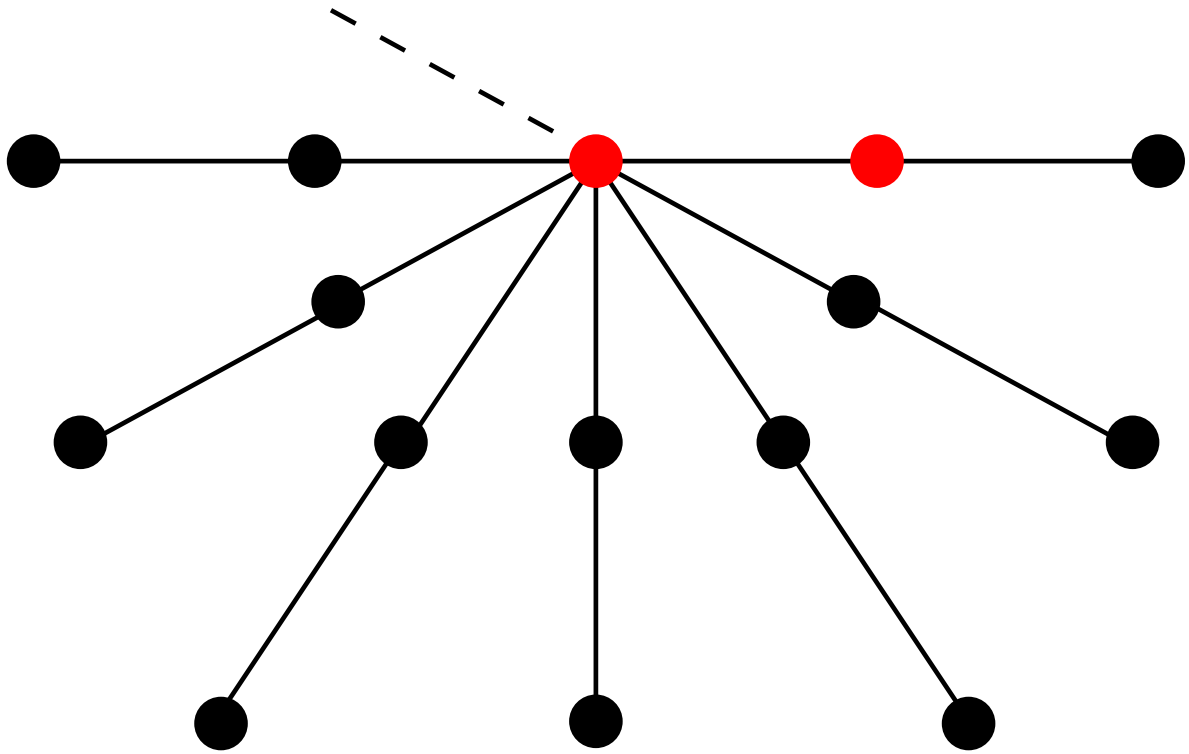
$mfvs(D) = n$

$pn(D) = 2$

$pn_{mfvs}(D) = n$

agents for minimizing # occupied vertices

$\exists$ digraphs with arbitrary large ratio: $\frac{pn_{mfvs}(D)}{pn(D)}$.



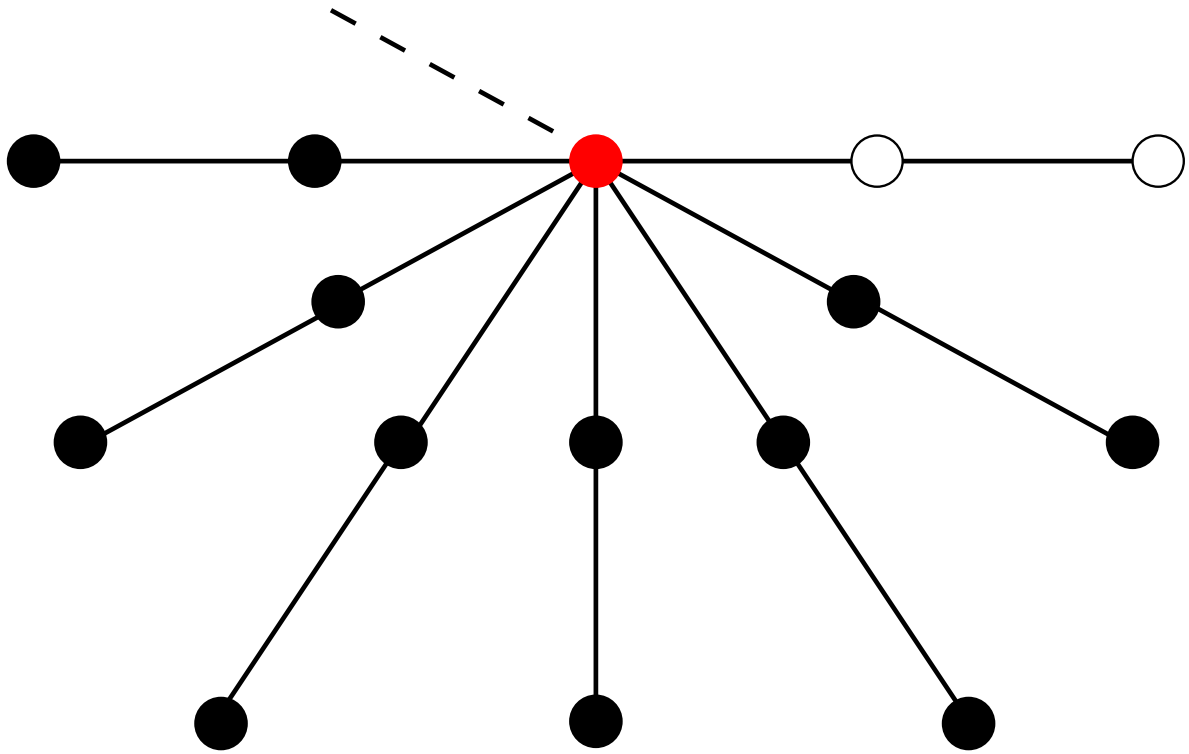
$mfvs(D) = n$

$pn(D) = 2$

$pn_{mfvs}(D) = n$

agents for minimizing # occupied vertices

$\exists$ digraphs with arbitrary large ratio: $\frac{pn_{mfvs}(D)}{pn(D)}$.



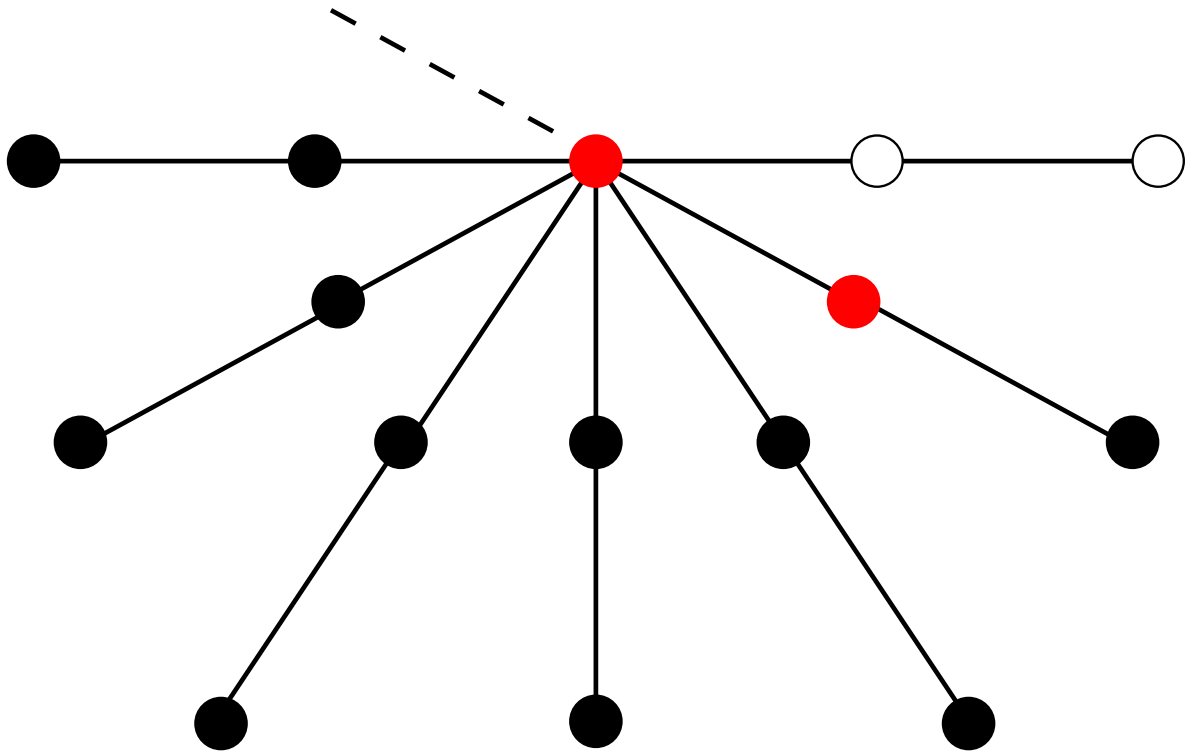
$mfvs(D) = n$

$pn(D) = 2$

$pn_{mfvs}(D) = n$

agents for minimizing # occupied vertices

$\exists$ digraphs with arbitrary large ratio: $\frac{pn_{mfvs}(D)}{pn(D)}$.



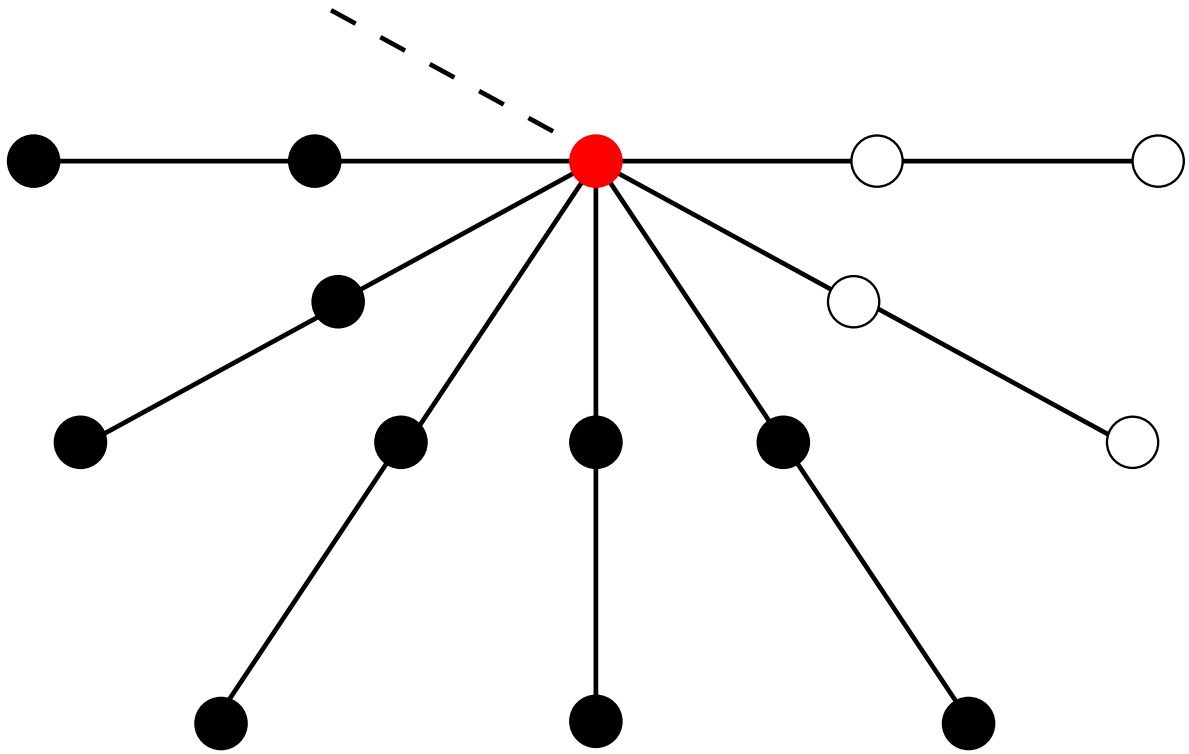
$mfvs(D) = n$

$pn(D) = 2$

$pn_{mfvs}(D) = n$

agents for minimizing # occupied vertices

$\exists$ digraphs with arbitrary large ratio: $\frac{pn_{mfvs}(D)}{pn(D)}$.



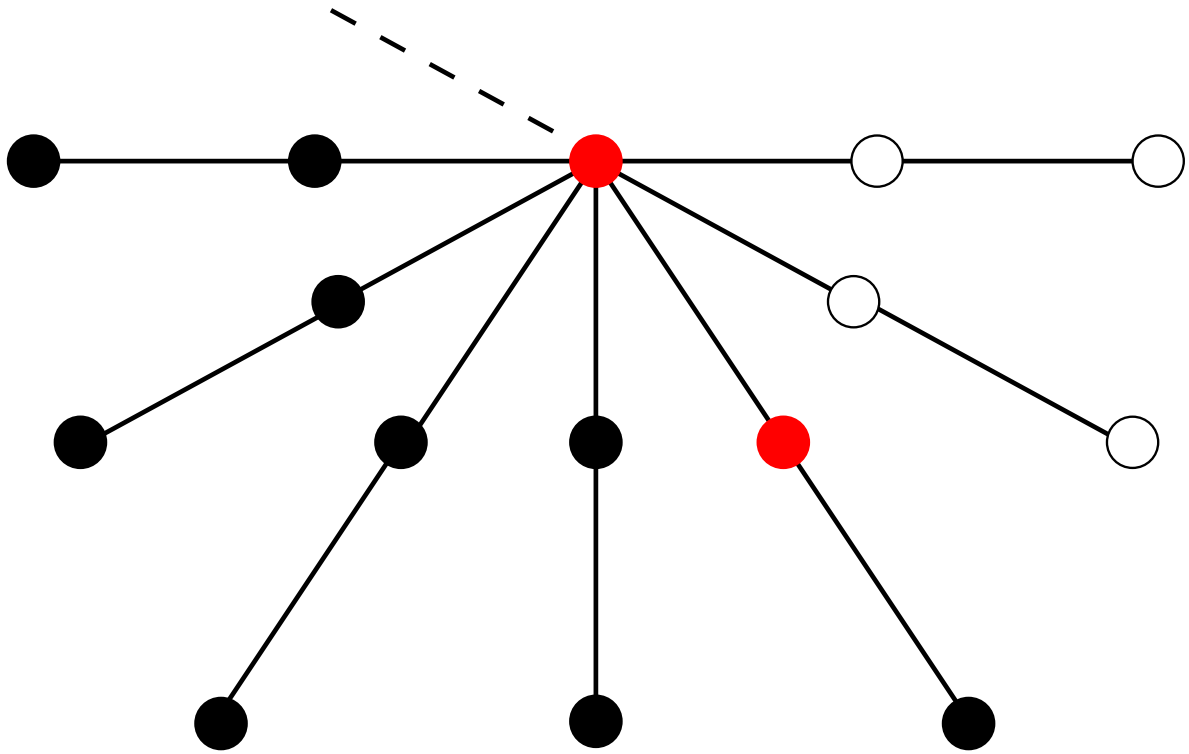
$mfvs(D) = n$

$pn(D) = 2$

$pn_{mfvs}(D) = n$

agents for minimizing # occupied vertices

$\exists$ digraphs with arbitrary large ratio: $\frac{pn_{mfvs}(D)}{pn(D)}$.



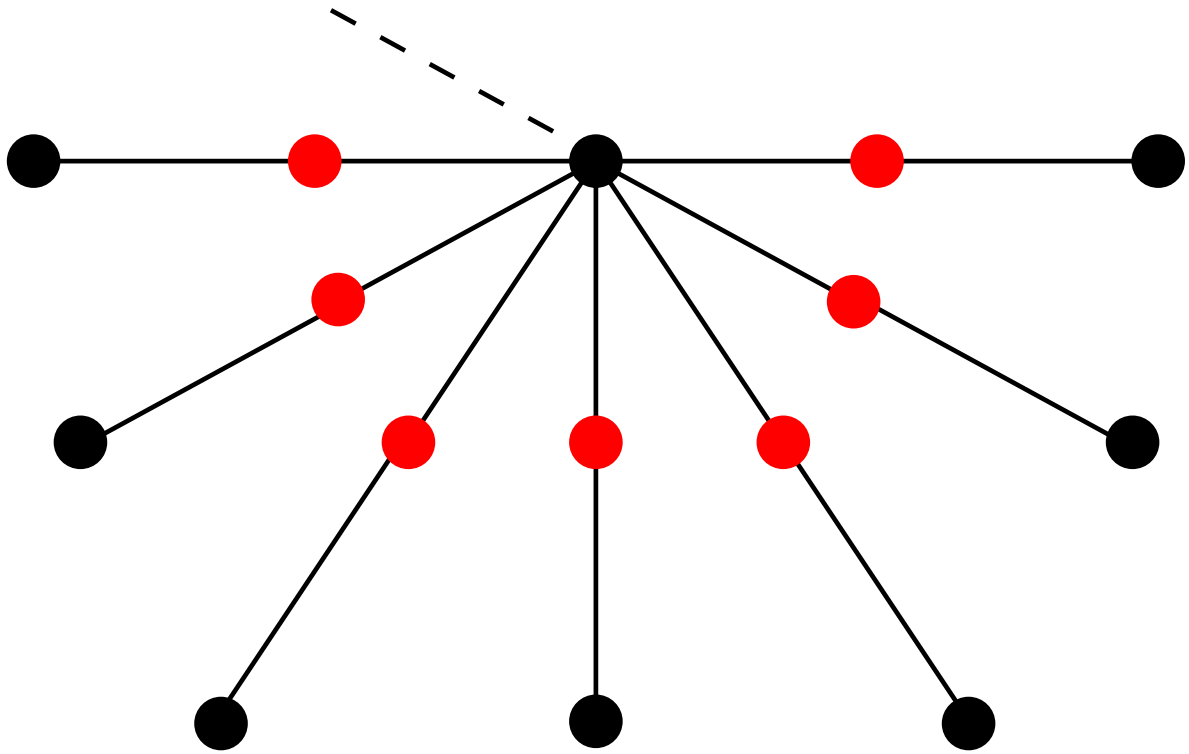
$mfvs(D) = n$

$pn(D) = 2$

$pn_{mfvs}(D) = n$

agents for minimizing # occupied vertices

$\exists$ digraphs with arbitrary large ratio: $\frac{pn_{mfvs}(D)}{pn(D)}$.



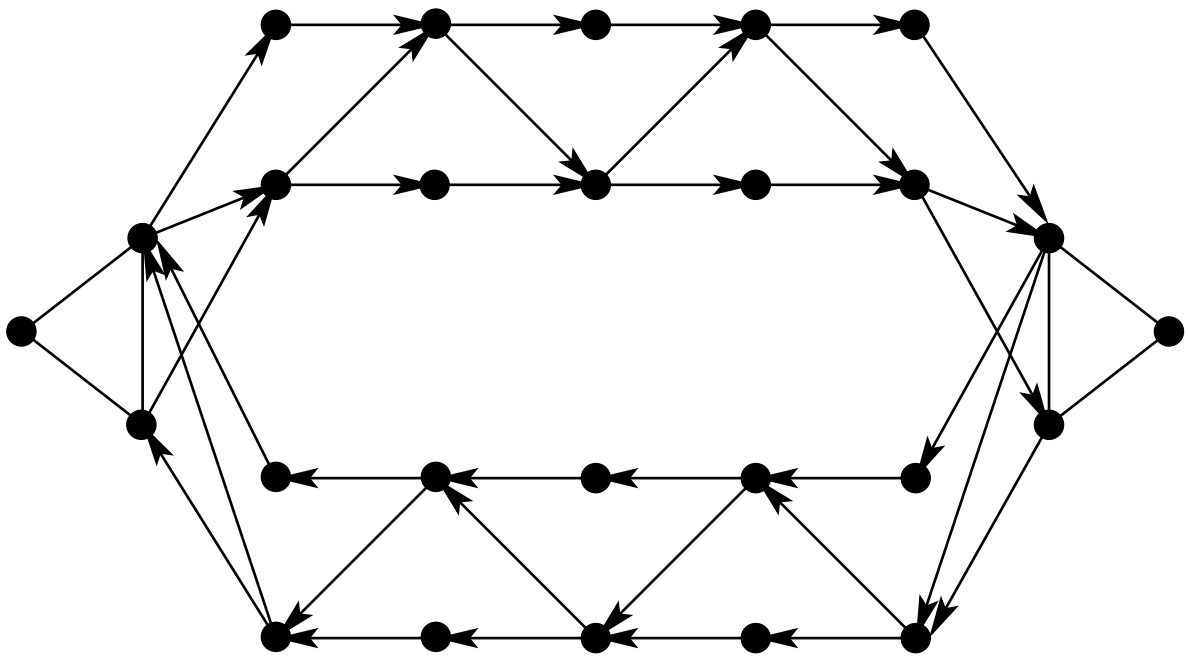
$mfvs(D) = n$

$pn(D) = 2$

$pn_{mfvs}(D) = n$

occupied vertices by the minimum # agents

$\exists$ digraphs with arbitrary large ratio: $\frac{mfvs_{pn}(D)}{mfvs(D)}$.



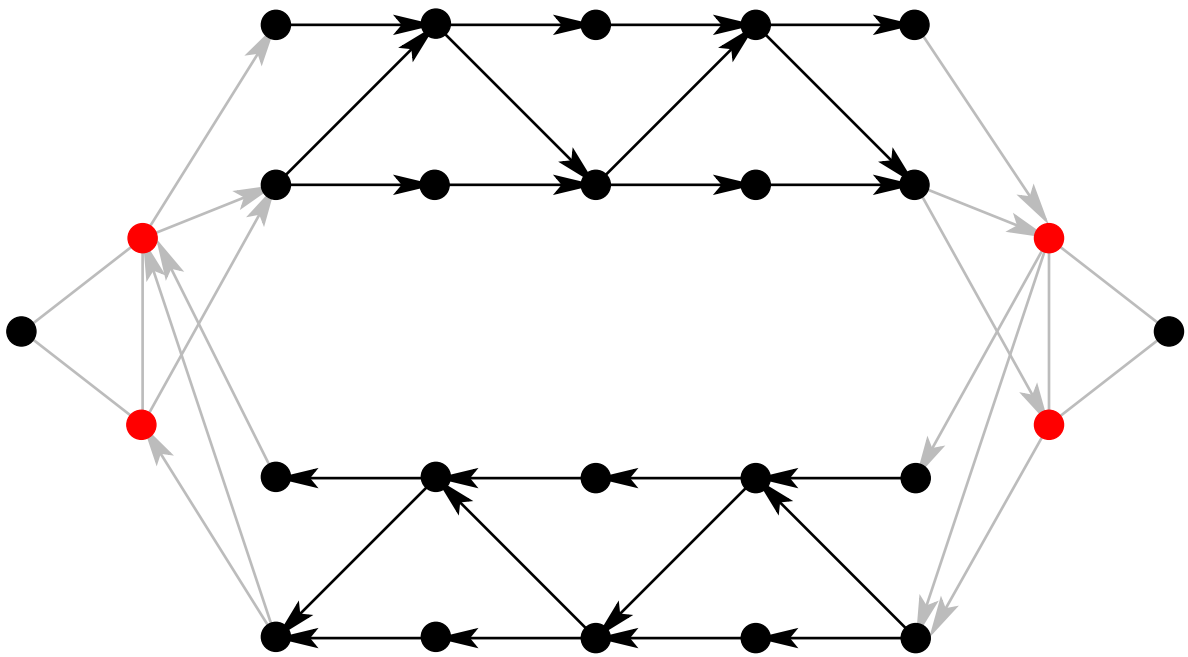
$mfvs(D) = 4$

$pn(D) = 3$

$mfvs_{pn}(D) = n + 4$

occupied vertices by the minimum # agents

$\exists$ digraphs with arbitrary large ratio: $\frac{mfvs_{pn}(D)}{mfvs(D)}$.



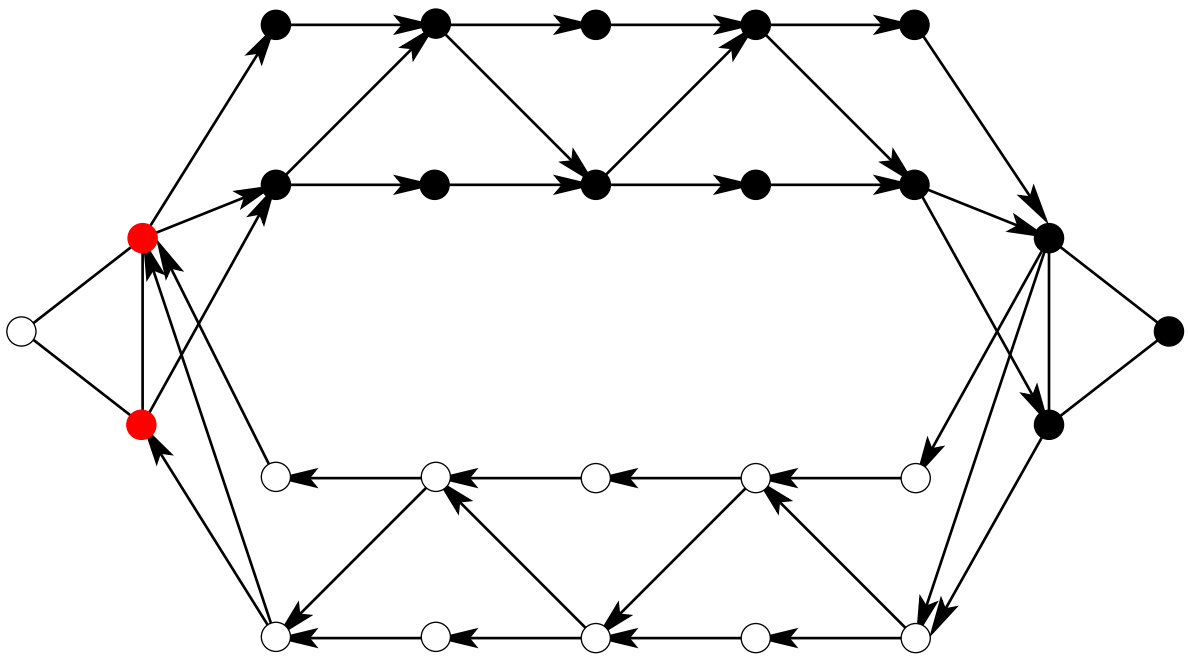
$mfvs(D) = 4$

$pn(D) = 3$

$mfvs_{pn}(D) = n + 4$

occupied vertices by the minimum # agents

$\exists$ digraphs with arbitrary large ratio: $\frac{mfvs_{pn}(D)}{mfvs(D)}$.



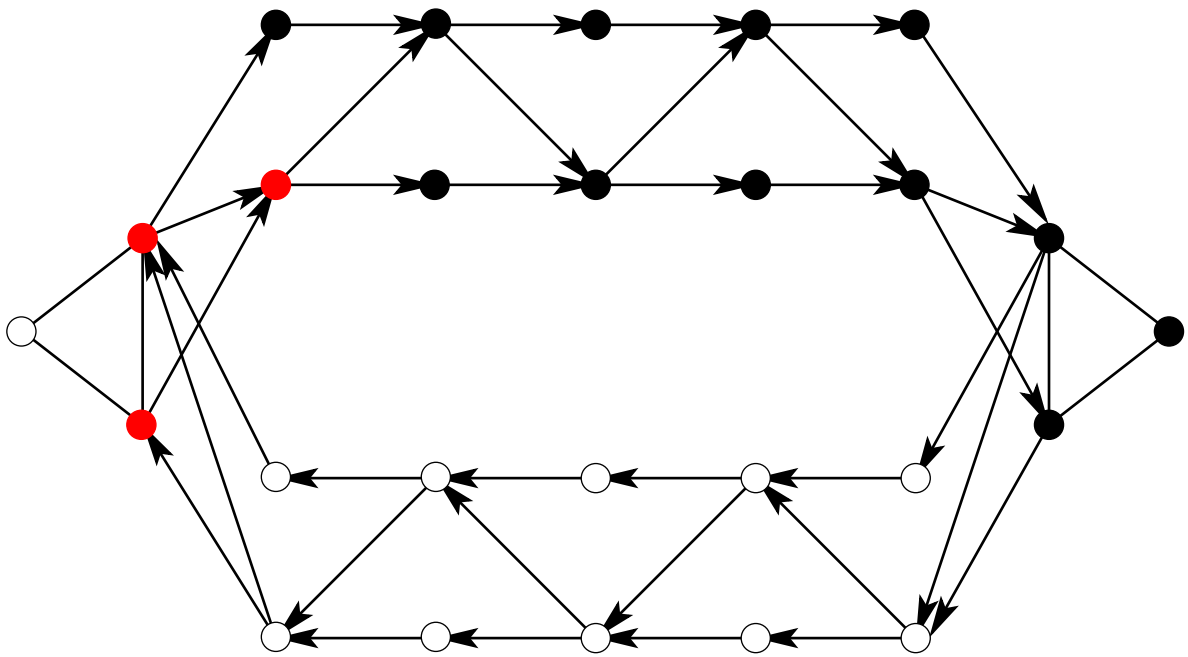
$mfvs(D) = 4$

$pn(D) = 3$

$mfvs_{pn}(D) = n + 4$

occupied vertices by the minimum # agents

$\exists$ digraphs with arbitrary large ratio: $\frac{mfvs_{pn}(D)}{mfvs(D)}$.



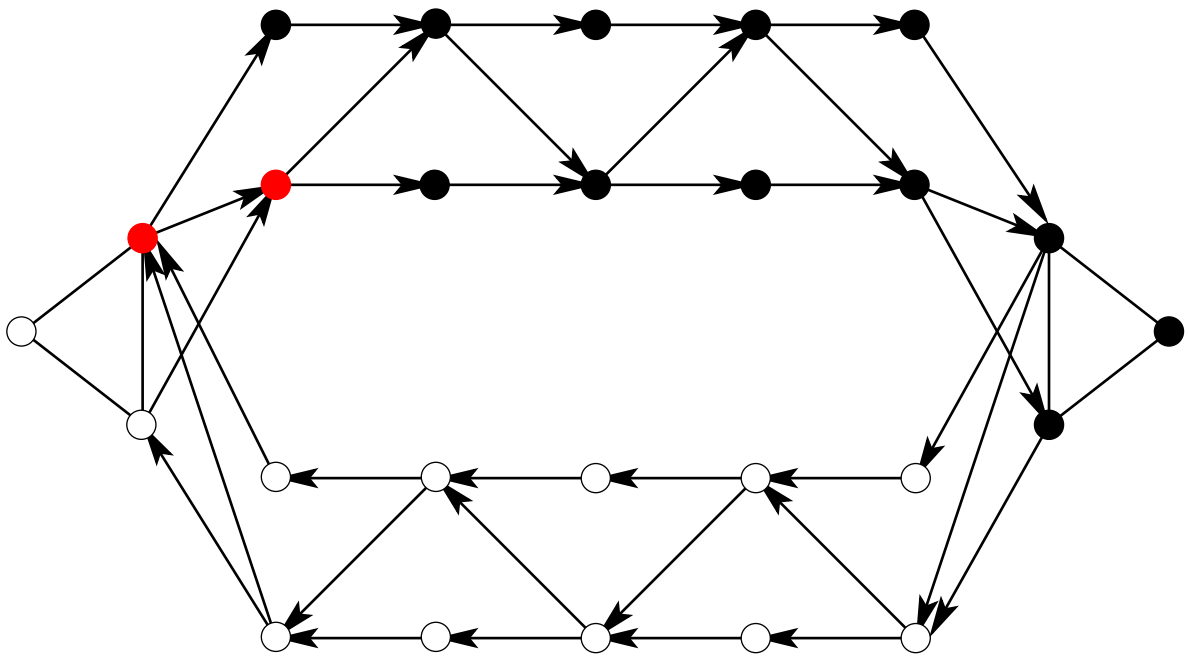
$mfvs(D) = 4$

$pn(D) = 3$

$mfvs_{pn}(D) = n + 4$

occupied vertices by the minimum # agents

$\exists$ digraphs with arbitrary large ratio: $\frac{mfvs_{pn}(D)}{mfvs(D)}$.



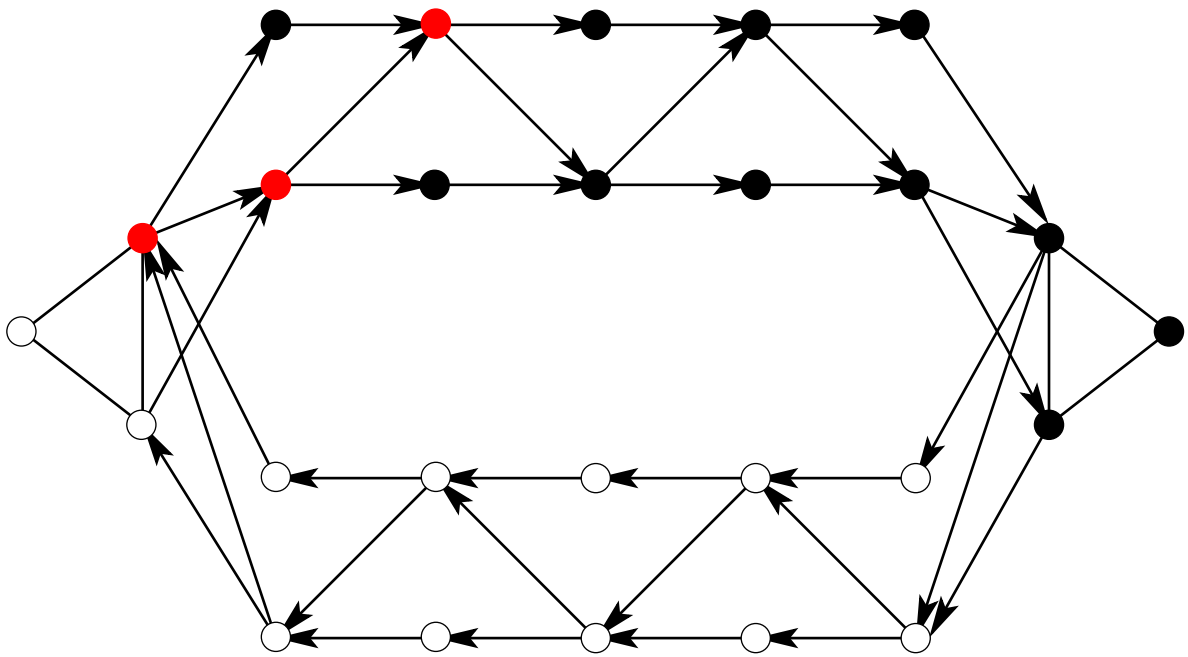
$mfvs(D) = 4$

$pn(D) = 3$

$mfvs_{pn}(D) = n + 4$

occupied vertices by the minimum # agents

$\exists$ digraphs with arbitrary large ratio: $\frac{mfvs_{pn}(D)}{mfvs(D)}$.



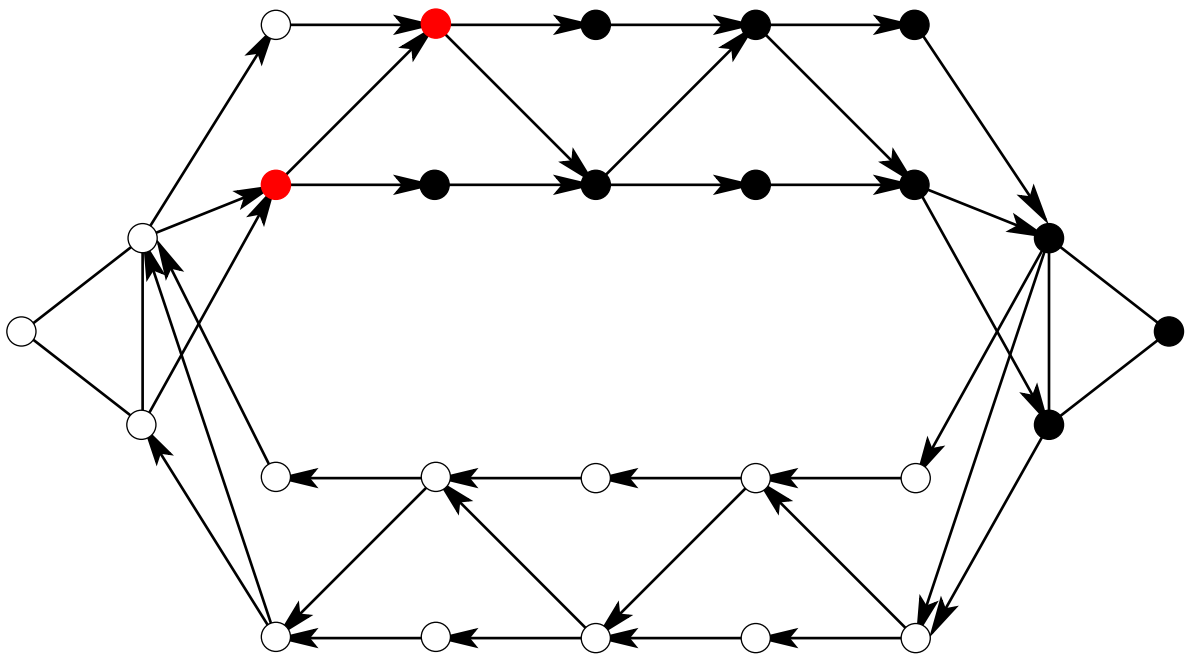
$mfvs(D) = 4$

$pn(D) = 3$

$mfvs_{pn}(D) = n + 4$

occupied vertices by the minimum # agents

$\exists$ digraphs with arbitrary large ratio: $\frac{mfvs_{pn}(D)}{mfvs(D)}$.



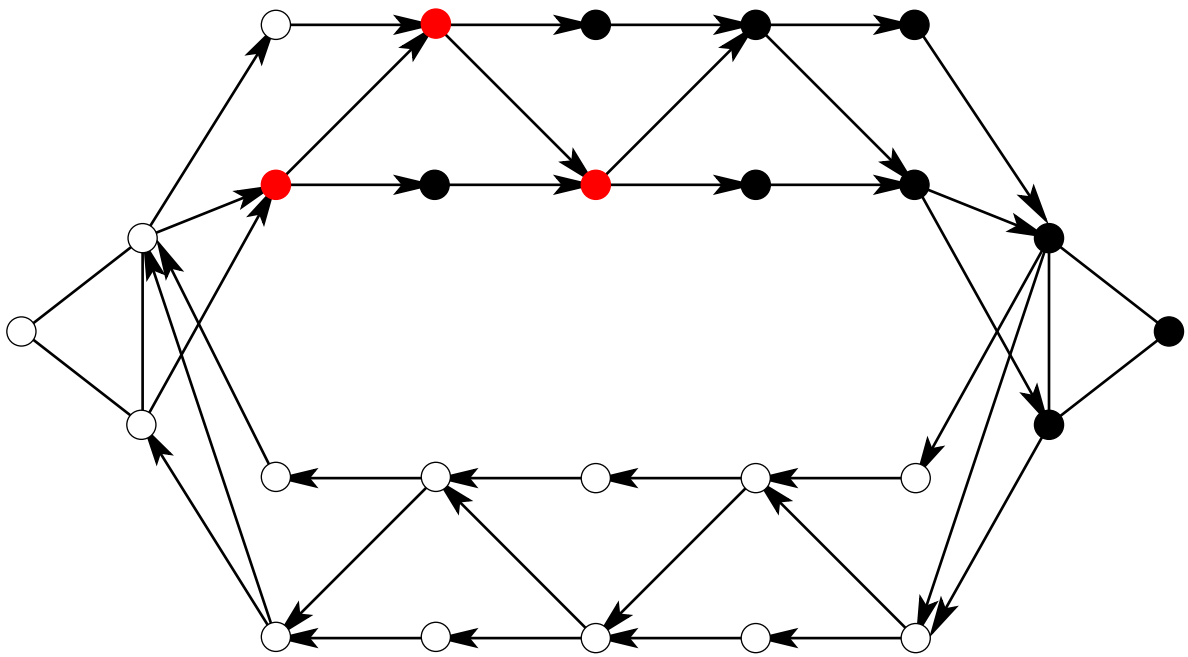
$mfvs(D) = 4$

$pn(D) = 3$

$mfvs_{pn}(D) = n + 4$

occupied vertices by the minimum # agents

$\exists$ digraphs with arbitrary large ratio: $\frac{mfvs_{pn}(D)}{mfvs(D)}$.



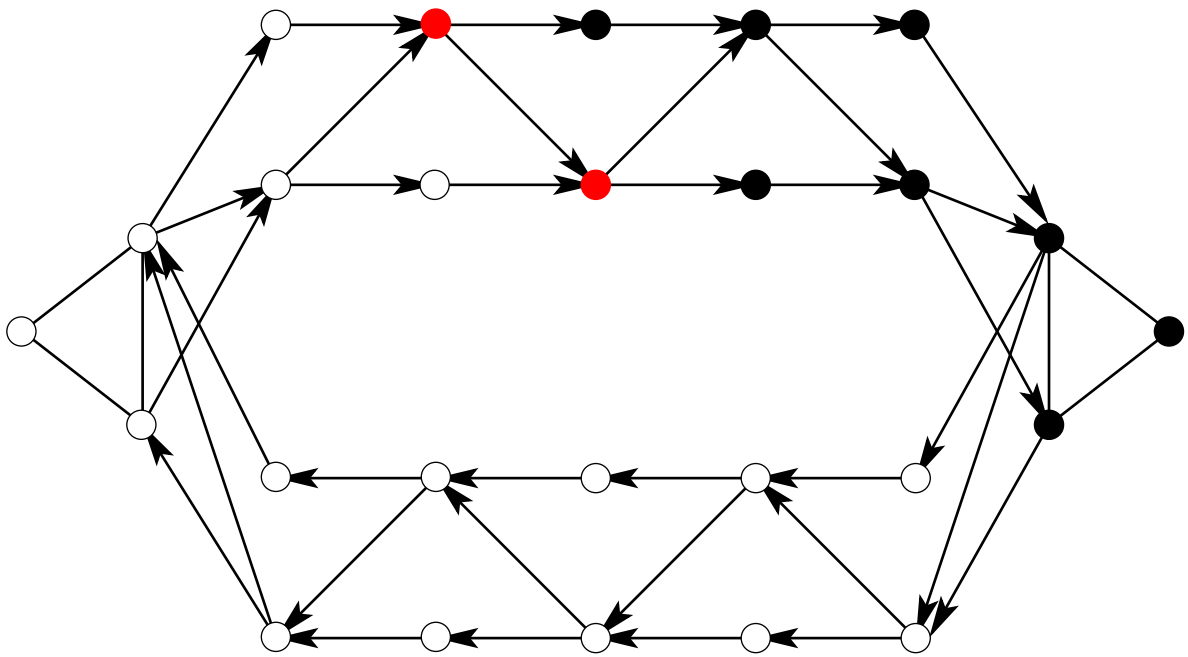
$mfvs(D) = 4$

$pn(D) = 3$

$mfvs_{pn}(D) = n + 4$

occupied vertices by the minimum # agents

$\exists$ digraphs with arbitrary large ratio: $\frac{mfvs_{pn}(D)}{mfvs(D)}$.



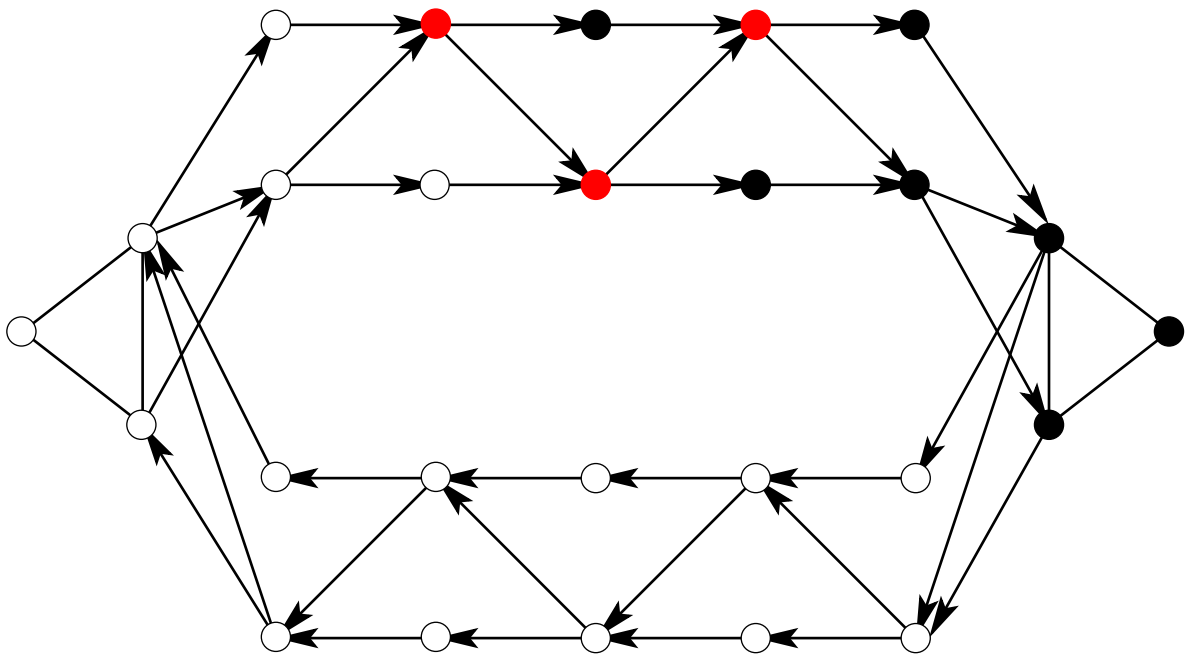
$mfvs(D) = 4$

$pn(D) = 3$

$mfvs_{pn}(D) = n + 4$

occupied vertices by the minimum # agents

$\exists$ digraphs with arbitrary large ratio: $\frac{mfvs_{pn}(D)}{mfvs(D)}$.



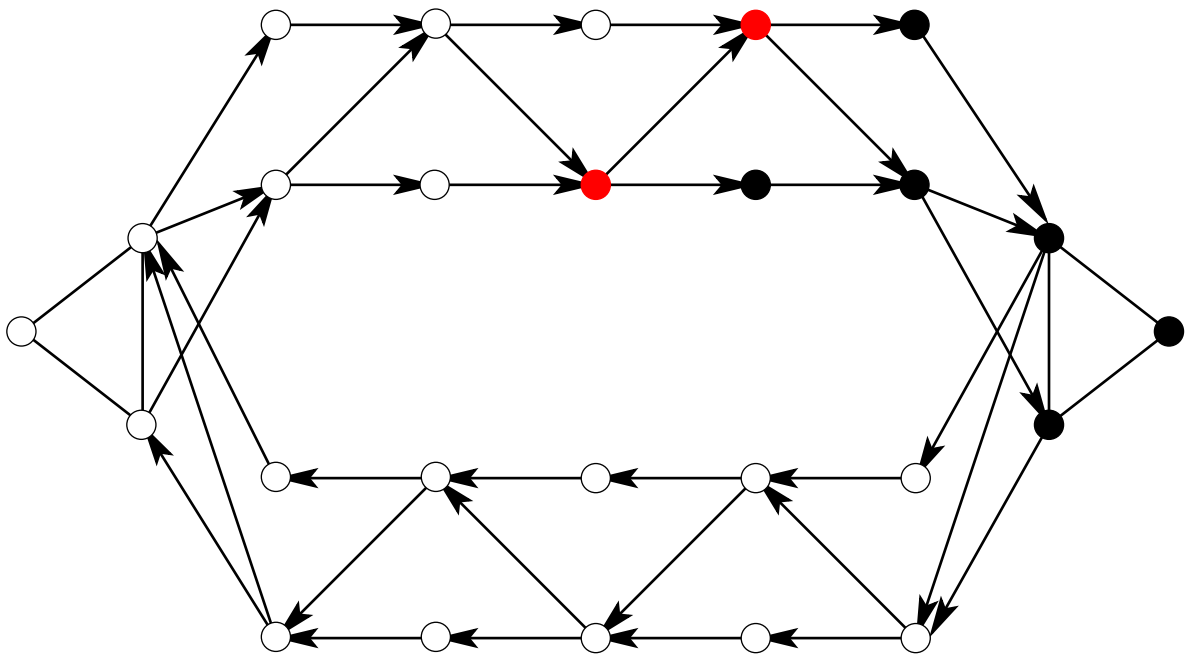
$mfvs(D) = 4$

$pn(D) = 3$

$mfvs_{pn}(D) = n + 4$

occupied vertices by the minimum # agents

$\exists$ digraphs with arbitrary large ratio: $\frac{mfvs_{pn}(D)}{mfvs(D)}$.



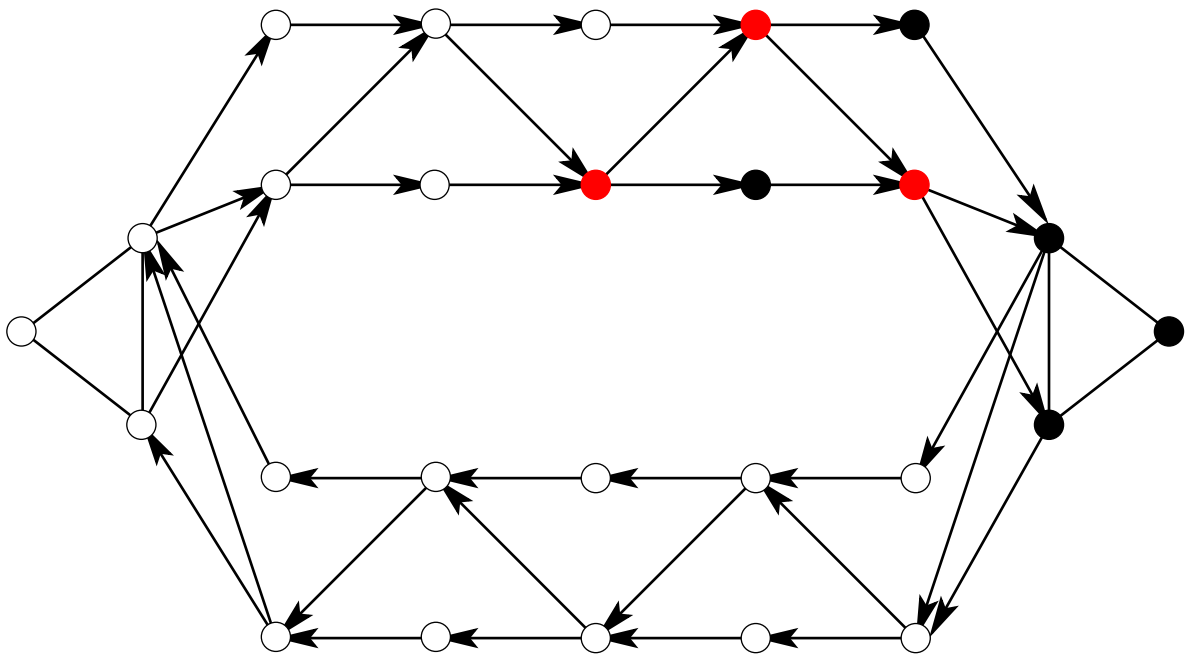
$mfvs(D) = 4$

$pn(D) = 3$

$mfvs_{pn}(D) = n + 4$

occupied vertices by the minimum # agents

$\exists$ digraphs with arbitrary large ratio: $\frac{mfvs_{pn}(D)}{mfvs(D)}$.



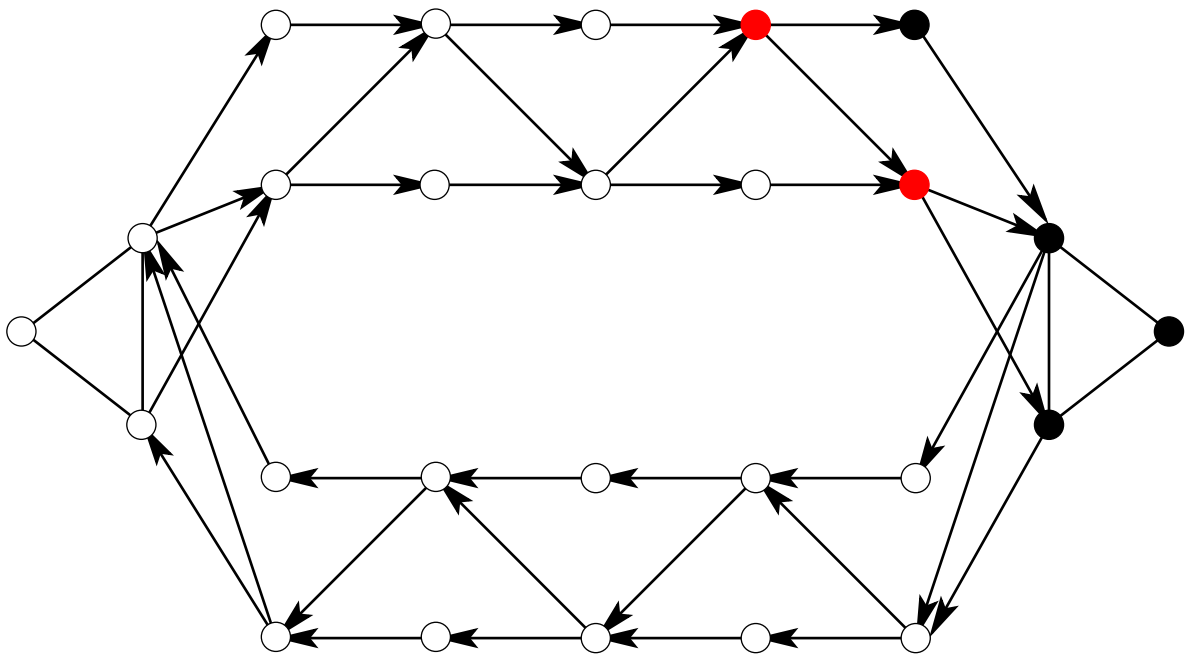
$mfvs(D) = 4$

$pn(D) = 3$

$mfvs_{pn}(D) = n + 4$

occupied vertices by the minimum # agents

$\exists$ digraphs with arbitrary large ratio: $\frac{mfvs_{pn}(D)}{mfvs(D)}$.



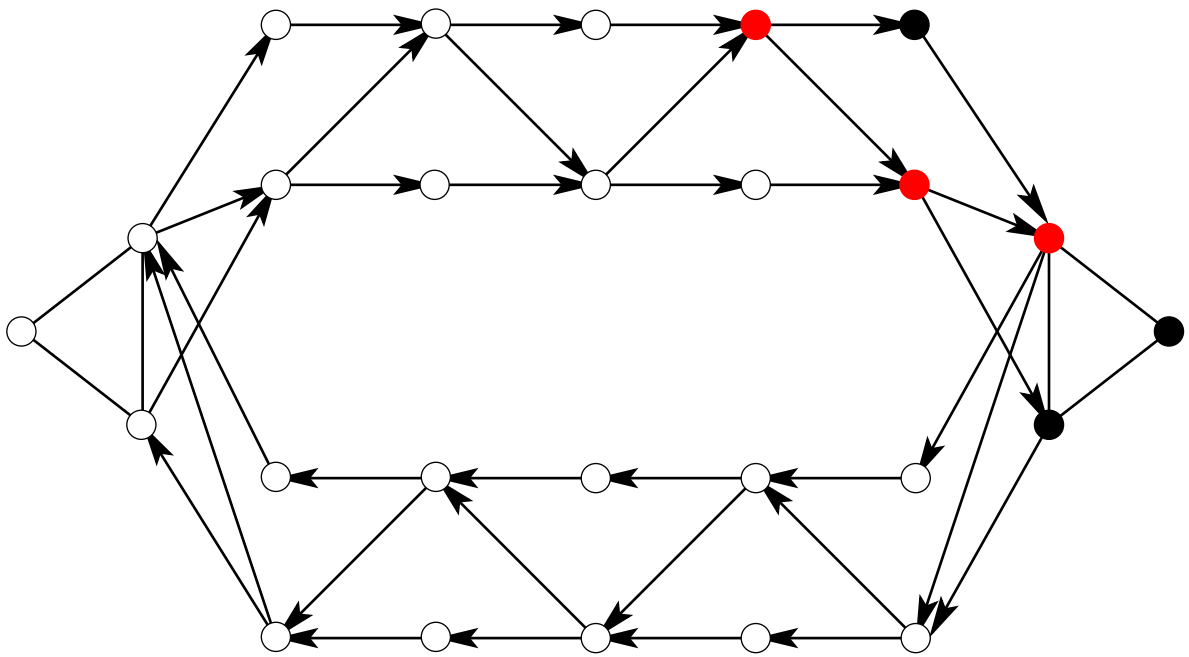
$mfvs(D) = 4$

$pn(D) = 3$

$mfvs_{pn}(D) = n + 4$

occupied vertices by the minimum # agents

$\exists$ digraphs with arbitrary large ratio: $\frac{mfvs_{pn}(D)}{mfvs(D)}$.



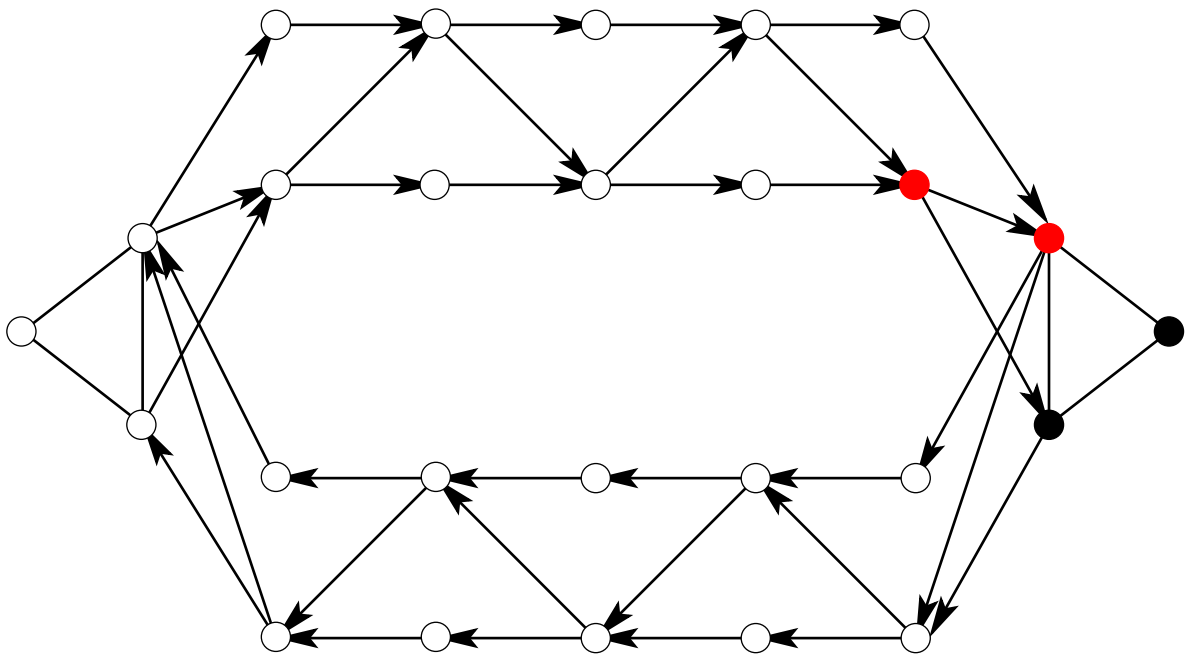
$mfvs(D) = 4$

$pn(D) = 3$

$mfvs_{pn}(D) = n + 4$

occupied vertices by the minimum # agents

$\exists$ digraphs with arbitrary large ratio: $\frac{mfvs_{pn}(D)}{mfvs(D)}$.



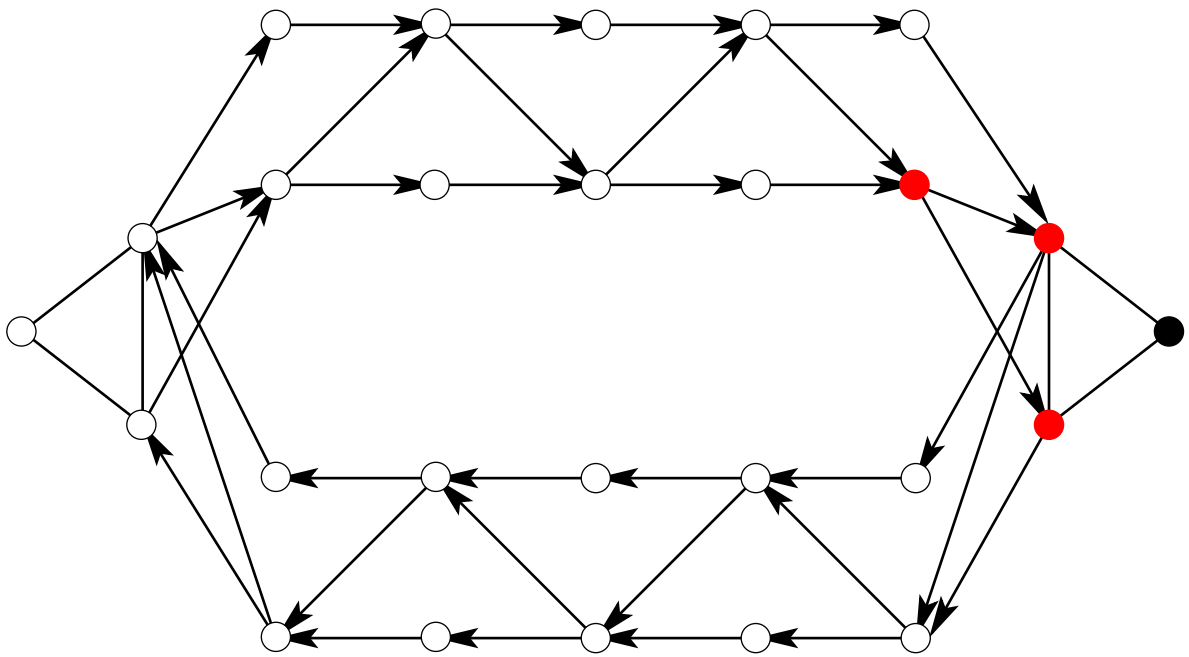
$mfvs(D) = 4$

$pn(D) = 3$

$mfvs_{pn}(D) = n + 4$

occupied vertices by the minimum # agents

$\exists$ digraphs with arbitrary large ratio: $\frac{mfvs_{pn}(D)}{mfvs(D)}$.



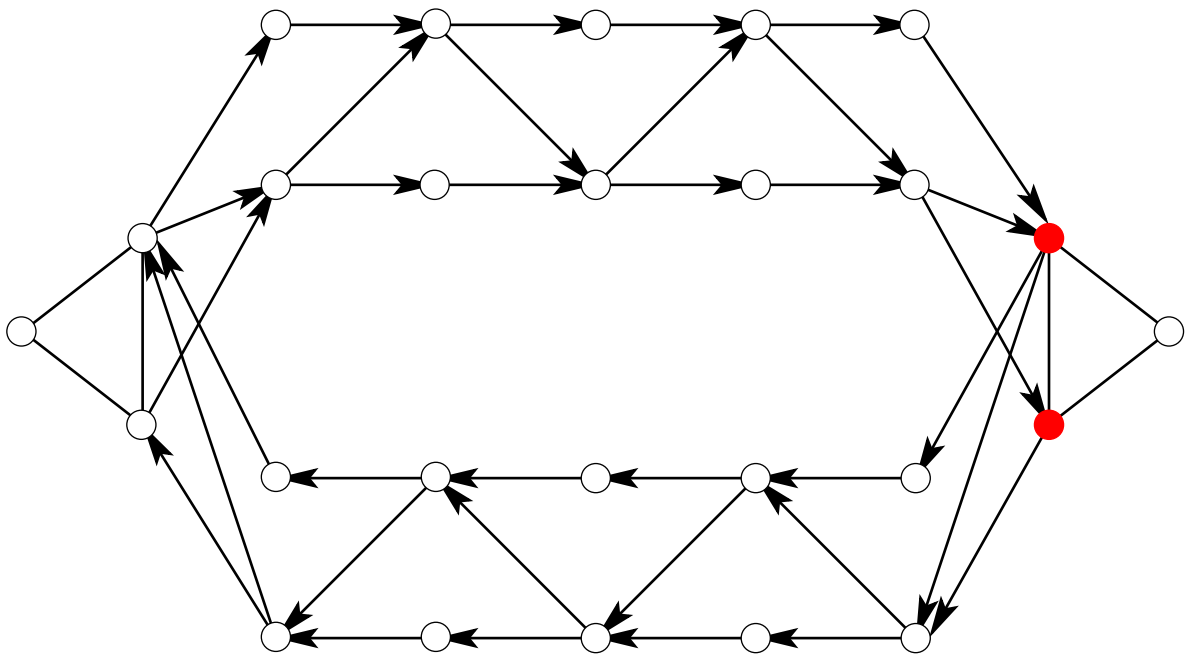
$mfvs(D) = 4$

$pn(D) = 3$

$mfvs_{pn}(D) = n + 4$

occupied vertices by the minimum # agents

$\exists$ digraphs with arbitrary large ratio: $\frac{mfvs_{pn}(D)}{mfvs(D)}$.



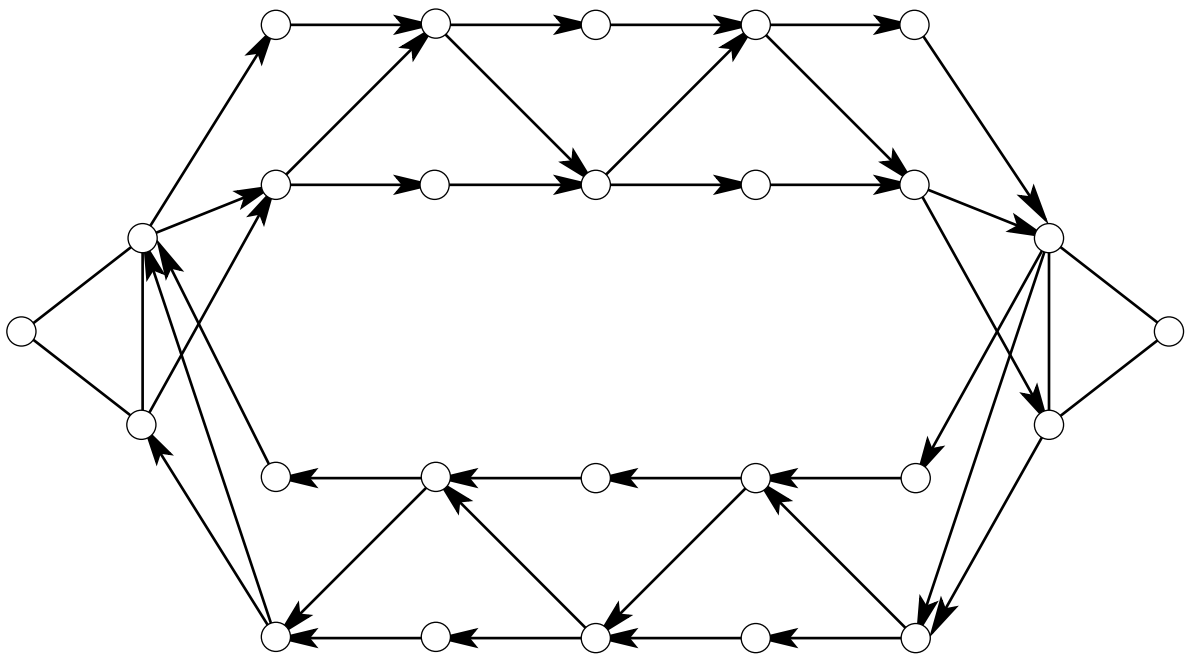
$mfvs(D) = 4$

$pn(D) = 3$

$mfvs_{pn}(D) = n + 4$

occupied vertices by the minimum # agents

$\exists$ digraphs with arbitrary large ratio: $\frac{mfvs_{pn}(D)}{mfvs(D)}$.



$mfvs(D) = 4$

$pn(D) = 3$

$mfvs_{pn}(D) = n + 4$

Open Question

Symmetric Digraphs

Maximum value of $\frac{mfvs_{pn}(D)}{mfvs(D)}$ for symmetric digraphs?

Lemma

There exists a symmetric digraph D with $\lambda > 3 - \varepsilon$ and $\forall D$ symmetric digraph, $\frac{mfvs_{pn}(D)}{mfvs(D)} \leq pn(D)$.

Conjecture: 3 is tight.

Perspectives

Among others...

- Determine routing + process strategy
- Mixte strategies using available resources
  - Tradeoff number of interruptions / number of switchings
- Inclusion of physical constraints (with Alcatel-Lucent)
  - Time to retune wavelengths after addition/removal
- Distributed strategies
- Sequence of maintenance operations
  - Mutualize route changes to reduce interruptions

Merch'i !!!! A l'année prochaine !!!!!