Reconfiguration and graph structure

Paul Bonsma Marcin Kamiński Marcin Wrochna

University of Twente University of Warsaw

May 2014

Outline

- Reconfiguration – what & why
- Bounded pathwidth: PSPACE-hard
- Claw-free: P

Outline

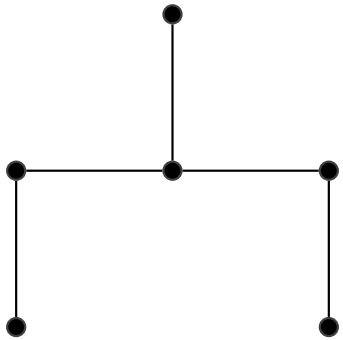
- Reconfiguration – what & why
- Bounded pathwidth: PSPACE-hard
- Claw-free: P

Outline

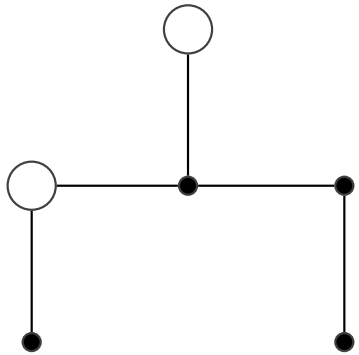
- Reconfiguration – what & why
- Bounded pathwidth: PSPACE-hard
- Claw-free: P

Reconfiguration – an example

Token sliding

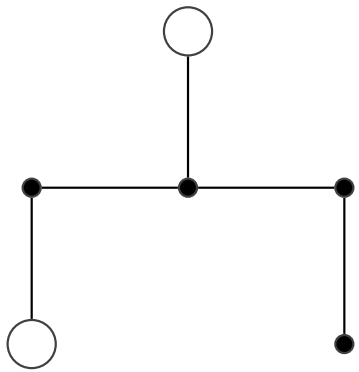


Reconfiguration – an example
Token sliding

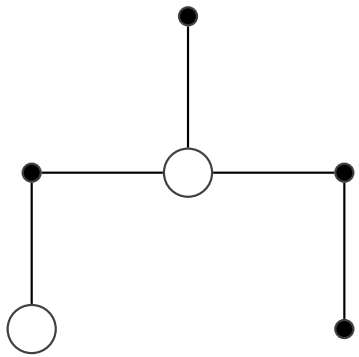


Reconfiguration – an example

Token sliding

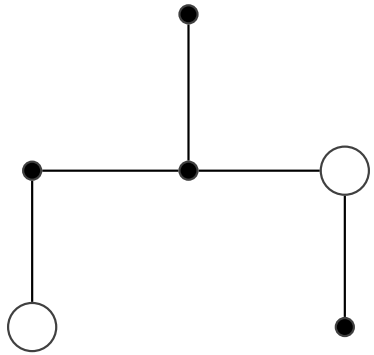


Reconfiguration – an example
Token sliding

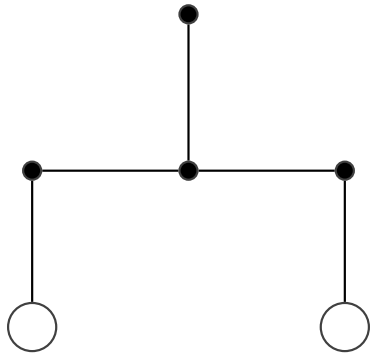


Reconfiguration – an example

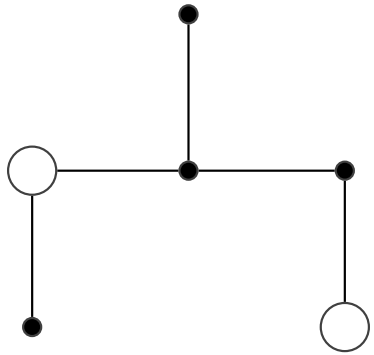
Token sliding



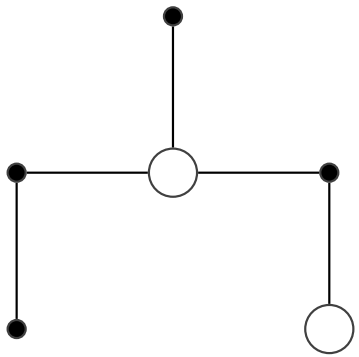
Reconfiguration – an example
Token sliding



Reconfiguration – an example
Token sliding

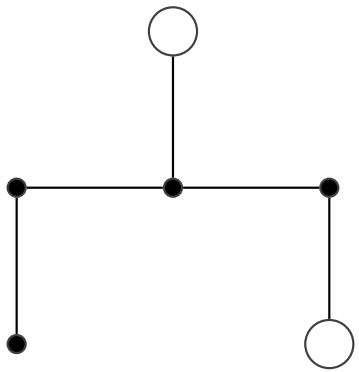


Reconfiguration – an example
Token sliding



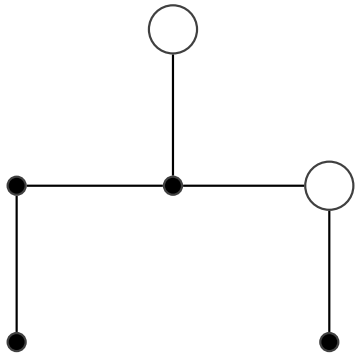
Reconfiguration – an example

Token sliding

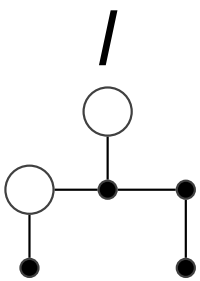


Reconfiguration – an example

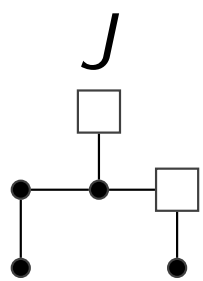
Token sliding



Token sliding

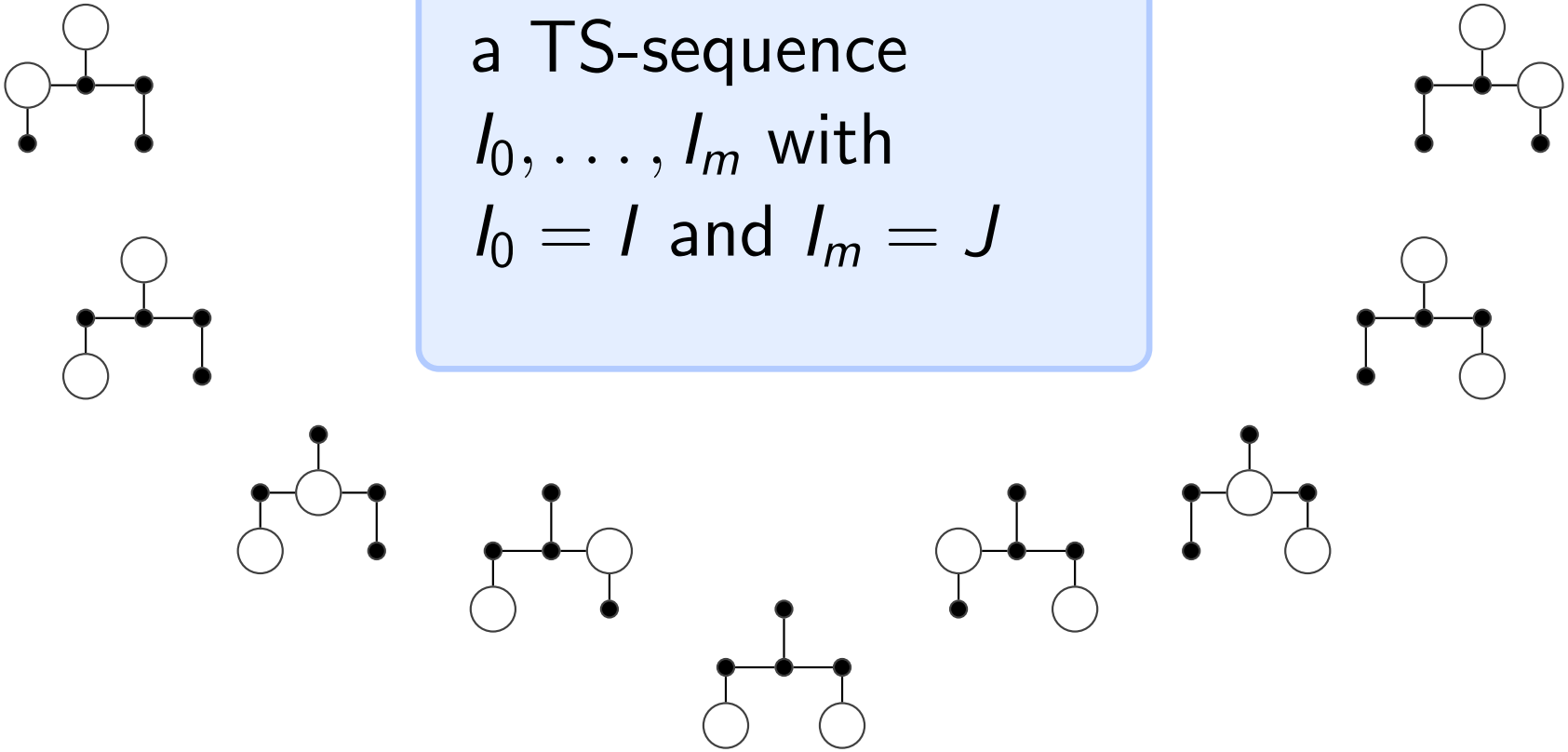


Token Sliding Reach.
Input: G, I, J
Question: $I \leftrightarrow_{\mathsf{TS}} J$?



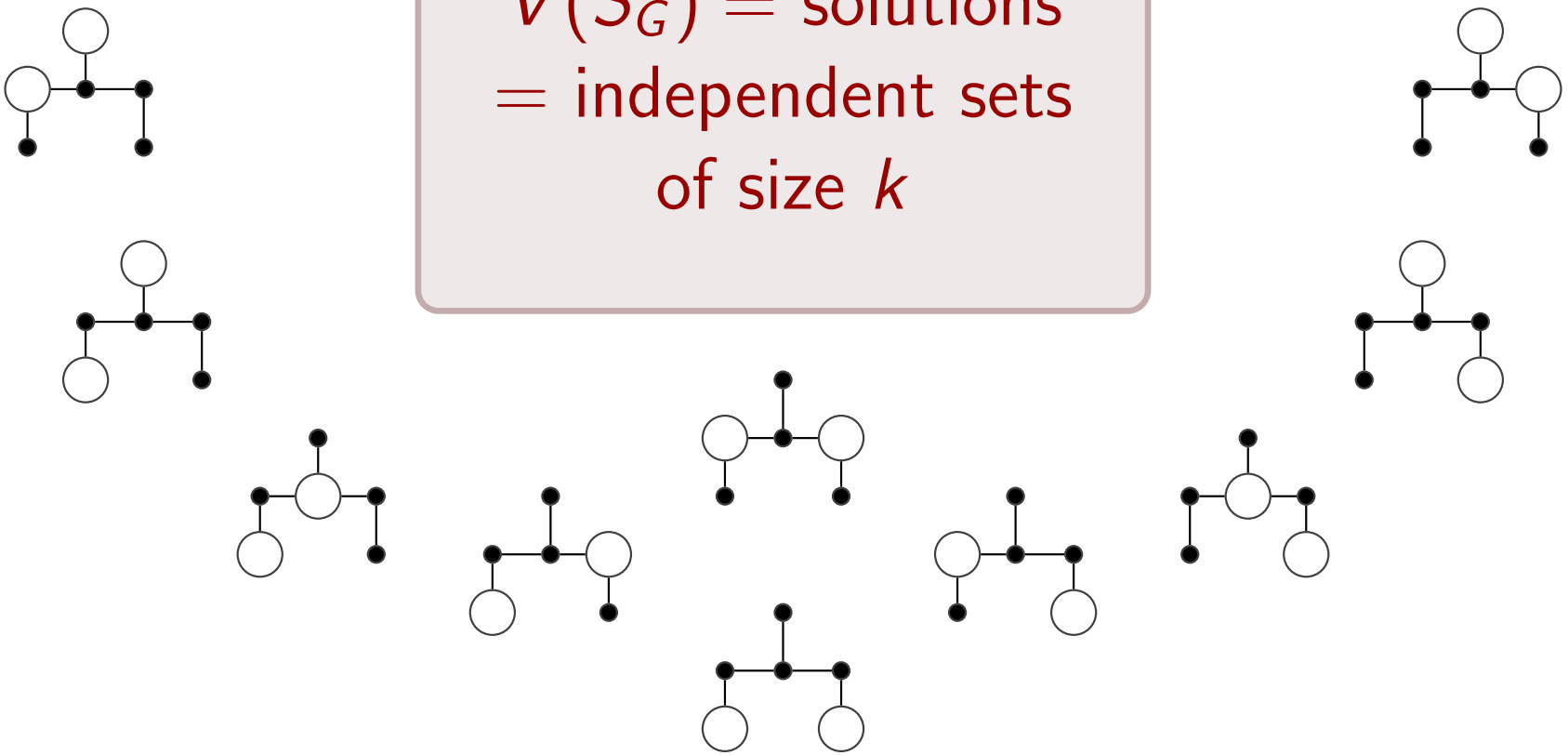
Token sliding

a TS-sequence $I_0, \ldots, I_m$ with $I_0 = I$ and $I_m = J$



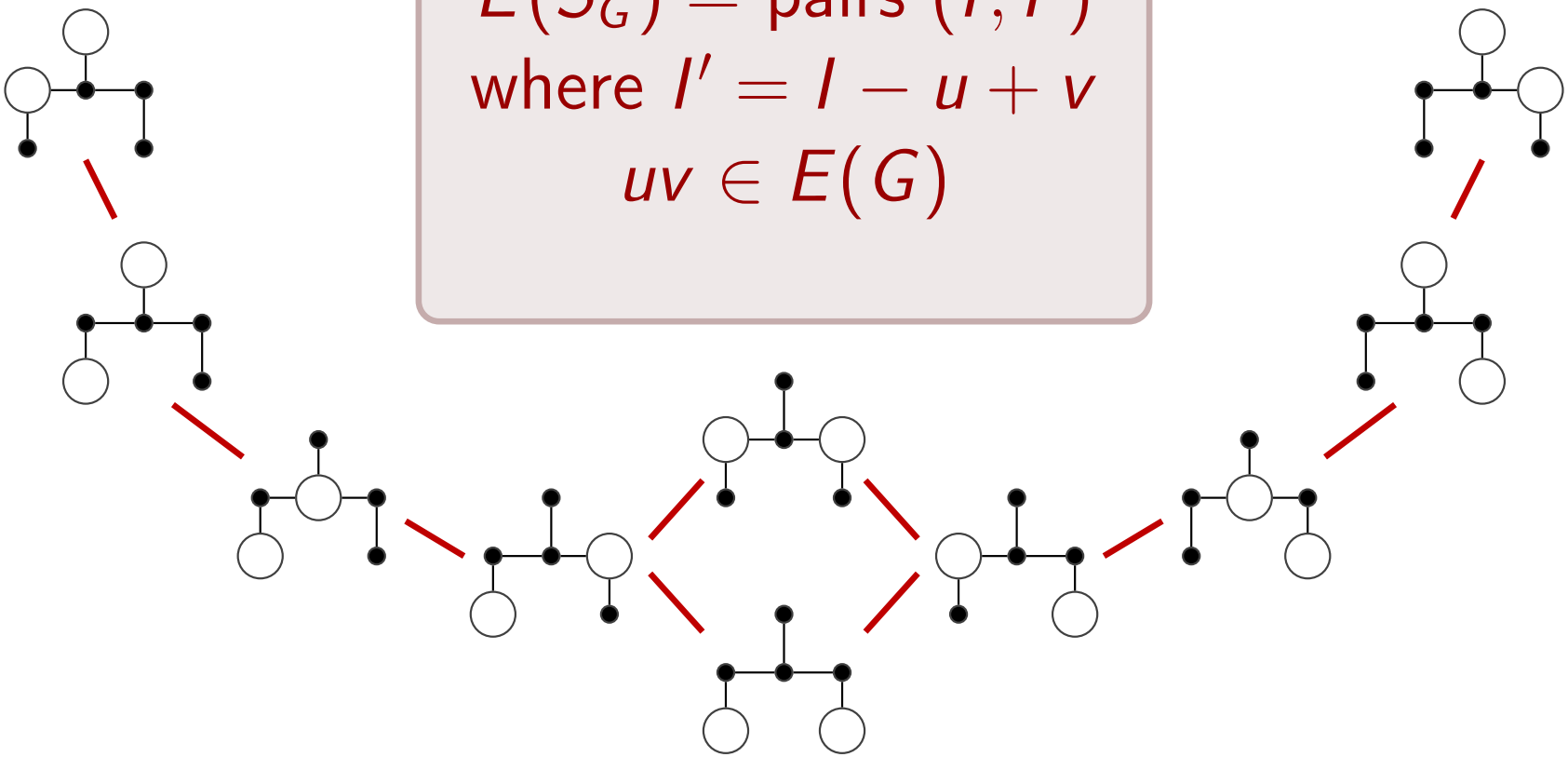
Solution graph

$V(S_G)$ = solutions
= independent sets
of size k



Solution graph

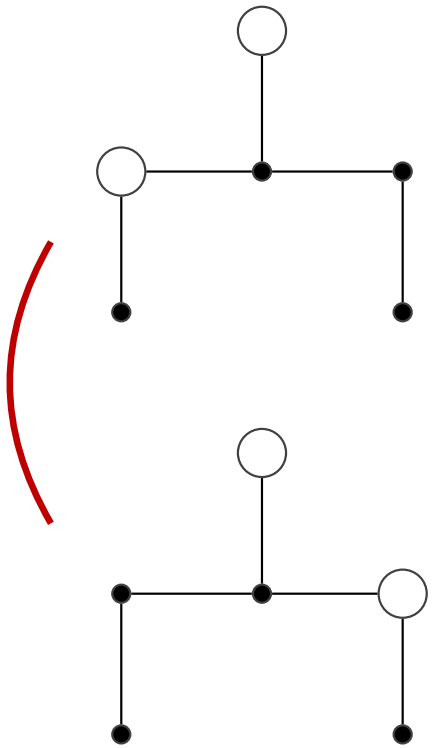
$$E(S_G) = \text{pairs } (I, I')$$
$$\text{where } I' = I - u + v$$
$$uv \in E(G)$$



Independent Set – TS, TJ and TAR

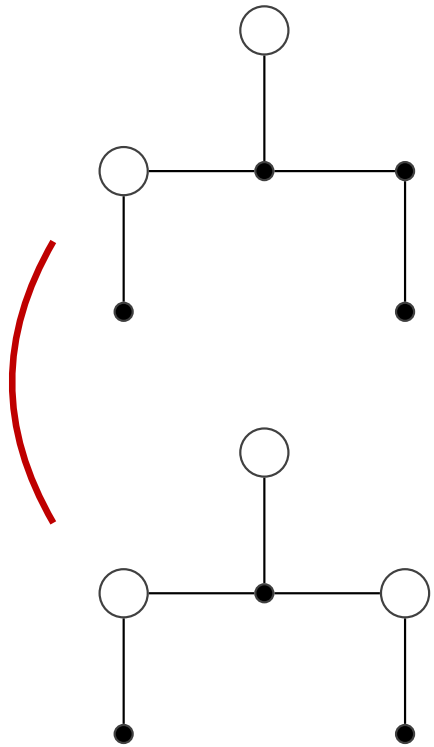
Token Jumping

$I' = I - u + v$ for any u, v



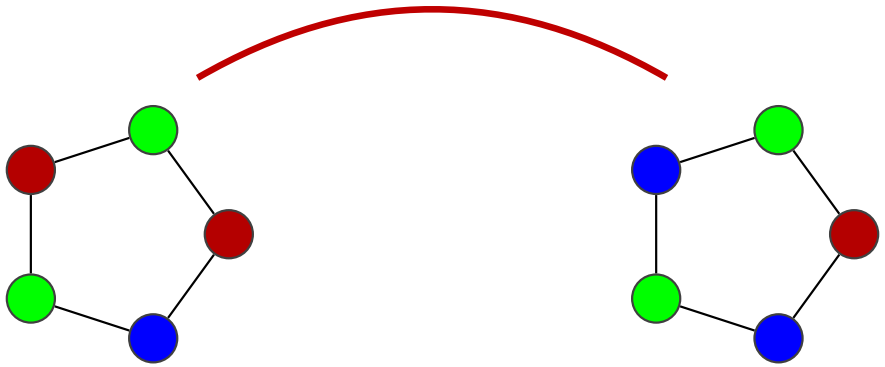
Token Addition & Removal

$I' = I \pm v$ and $|I| \geq k$



k-Recoloring
$k = 3$

$V = k$-colorings, $\quad E =$ switching one color



Reconfiguration – why

- Explore the solution space
- Practical problems(?) and puzzles
- Unexpected, nontrivial results

Reconfiguration – why

- Explore the solution space
- Practical problems(?) and puzzles
- Unexpected, nontrivial results

Reconfiguration – why

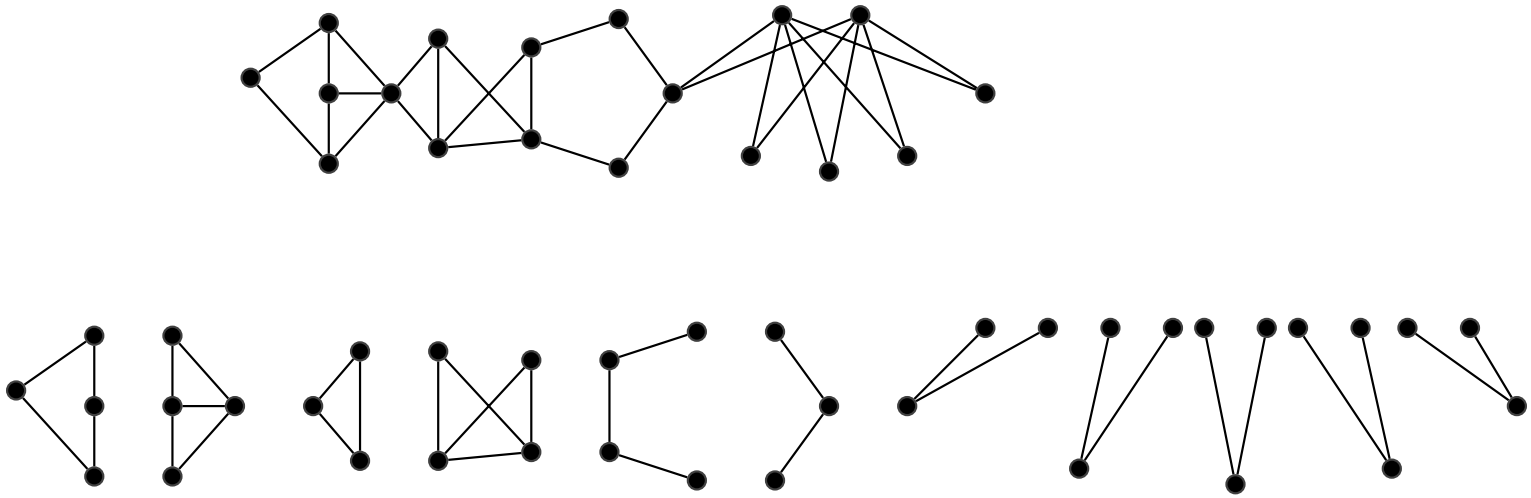
- Explore the solution space
- Practical problems(?) and puzzles
- Unexpected, nontrivial results

Theorem [Cereceda, van den Heuvel, Johnson 2011]

3-recoloring is P.

Pathwidth

$p = 3$



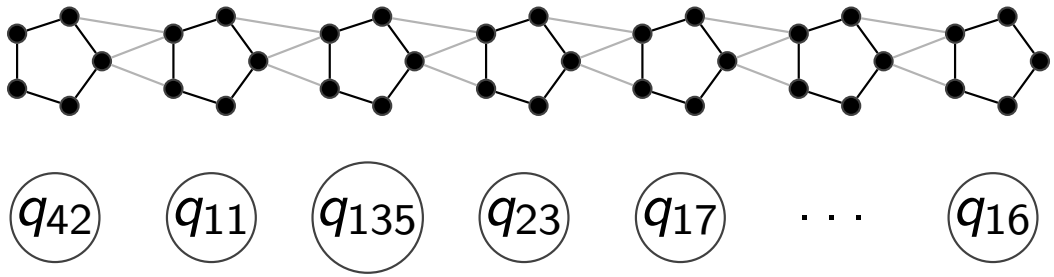
Take a sequence of graphs on $\leq p + 1$ vertices.
Glue any vertices from two consecutive graphs.

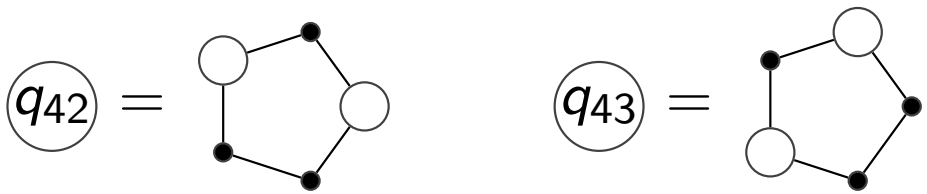
Pathwidth

- pathwidth $\leq p \implies$ treewidth $\leq p$
- pathwidth is treewidth without branching/merging steps.
- Max Independent Set, k-Coloring, ...
  are P in bounded pathwidth.
- so is counting solutions, extensions to solutions, ...

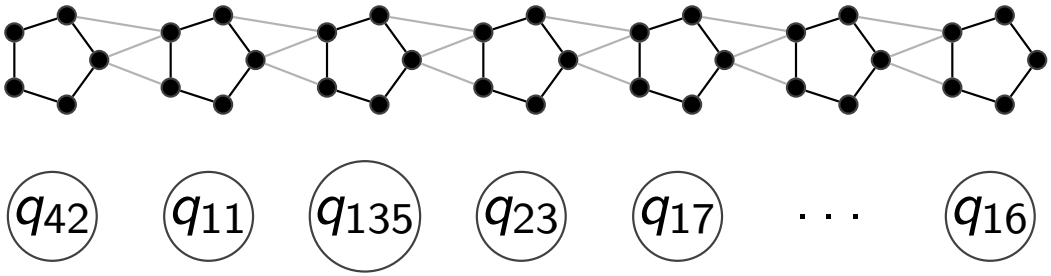
- Max Independent Set, k-Coloring, ...
  are P in bounded pathwidth.
- so is counting solutions, extensions to solutions, ...
- Token Jumping, k-Recoloring is P in trees.
- Token Jumping has a dynamic algorithm in cographs.

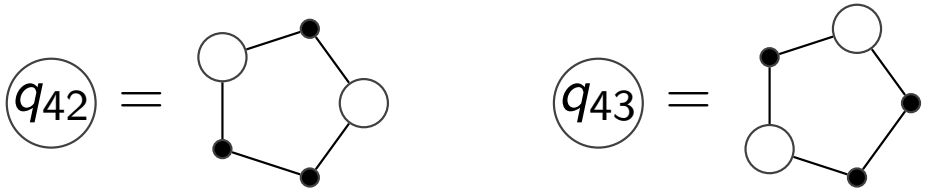
Why is reconfiguration hard?



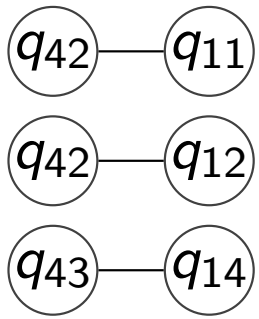


Why is reconfiguration hard?

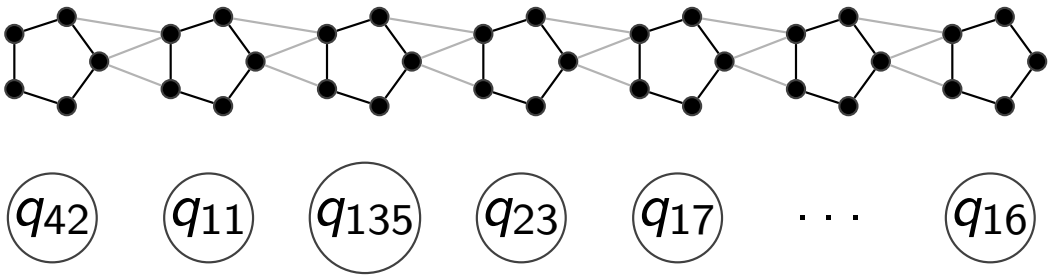




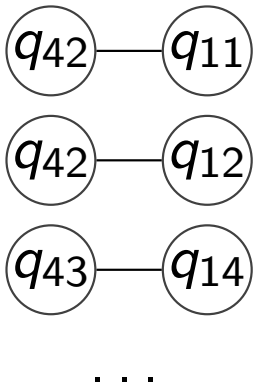
Rules:

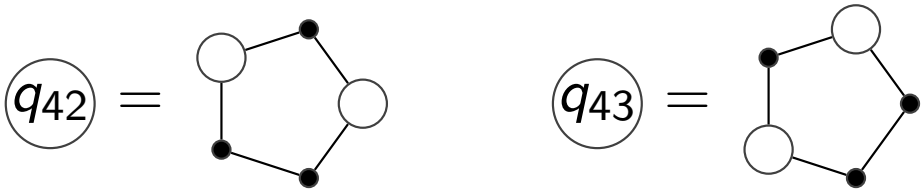


Why is reconfiguration hard?



Rules:





If we can construct enough states and any set of rules, we can simulate a Turing Machine's tape.

String Rewriting Systems

A *string rewriting system* is a pair (Σ, R)

Σ – alphabet

$R \subseteq \Sigma^* \times \Sigma^*$ – set of rules

Apply a rule (α, β) to a word u by replacing a subword α by β.

$u \rightsquigarrow_R v :=$ u can be transformed into v by applying rules.

String Rewriting Systems

A *string rewriting system* is a pair (Σ, R)

Σ – alphabet

$R \subseteq \Sigma^* \times \Sigma^*$ – set of rules

Apply a rule (α, β) to a word u by replacing a subword α by β.

$u \rightsquigarrow_R v :=$ u can be transformed into v by applying rules.

String Rewriting Systems

A *string rewriting system* is a pair (Σ, R)

Σ – alphabet

$R \subseteq \Sigma^* \times \Sigma^*$ – set of rules

Apply a rule (α, β) to a word u by replacing a subword α by β.

$u \rightsquigarrow_R v :=$ u can be transformed into v by applying rules.

Word problem

Apply a rule (α, β) to a word u by replacing a subword α by β.
$u \rightsquigarrow_R v$: $=$ u can be transformed into v by applying rules.

Word Problem for (Σ, R)

Input: $u, v \in \Sigma^*$
Question: $u \rightsquigarrow_R v$?

Theorem [Post 1947]

There is a SRS whose word problem is undecidable.

Also known as semi-Thue systems, finitely presented monoids, unrestricted grammars.

Word problem

Apply a rule (α, β) to a word u by replacing a subword α by β.
$u \rightsquigarrow_R v$: $=$ u can be transformed into v by applying rules.

Word Problem for (Σ, R)

Input: $u, v \in \Sigma^*$
Question: $u \rightsquigarrow_R v$?

Theorem [Post 1947]

There is a SRS whose word problem is undecidable.

Also known as semi-Thue systems, finitely presented monoids, unrestricted grammars.

Word problem

Apply a rule (α, β) to a word u by replacing a subword α by β.
$u \rightsquigarrow_R v$: $=$ u can be transformed into v by applying rules.

Word Problem for (Σ, R)

Input: $u, v \in \Sigma^*$
Question: $u \rightsquigarrow_R v$?

Theorem [Post 1947]

There is a SRS whose word problem is undecidable.

Also known as semi-Thue systems, finitely presented monoids, unrestricted grammars.

balanced SRS

$$\begin{aligned}
\text{symmetric SRS} &:= (\alpha,\beta) \in R \iff (\beta,\alpha) \in R \\
\text{balanced SRS} &:= |\alpha| = |\beta| && \text{for all } (\alpha,\beta) \in R \\
\text{2-balanced SRS} &:= |\alpha| = |\beta| = 2 && \text{for all } (\alpha,\beta) \in R
\end{aligned}$$

balanced $\implies$ word problem can be solved in polynomial space.

a variant of Post's theorem

There is a 2-balanced symmetric SRS whose word problem is PSPACE-complete.

balanced SRS

$$\begin{aligned}
\text{symmetric SRS} &:= (\alpha,\beta) \in R \iff (\beta,\alpha) \in R \\
\text{balanced SRS} &:= |\alpha| = |\beta| && \text{for all } (\alpha,\beta) \in R \\
\text{2-balanced SRS} &:= |\alpha| = |\beta| = 2 && \text{for all } (\alpha,\beta) \in R
\end{aligned}$$

balanced $\implies$ word problem can be solved in polynomial space.

a variant of Post's theorem

There is a 2-balanced symmetric SRS whose word problem is PSPACE-complete.

balanced SRS

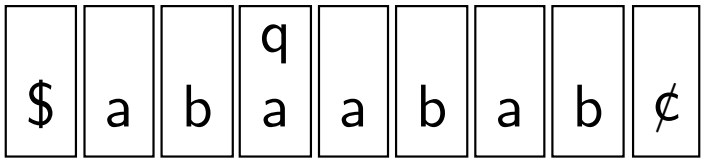
$$\begin{aligned}
\text{symmetric SRS} &:= (\alpha,\beta) \in R \iff (\beta,\alpha) \in R \\
\text{balanced SRS} &:= |\alpha| = |\beta| && \text{for all } (\alpha,\beta) \in R \\
\text{2-balanced SRS} &:= |\alpha| = |\beta| = 2 && \text{for all } (\alpha,\beta) \in R
\end{aligned}$$

balanced $\implies$ word problem can be solved in polynomial space.

a variant of Post's theorem

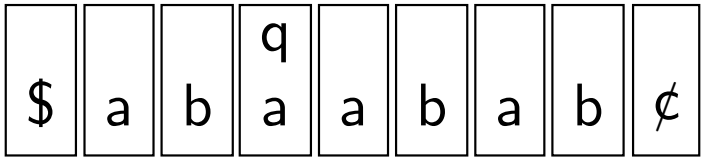
There is a 2-balanced symmetric SRS whose word problem is PSPACE-complete.

a Turing Machine is an SRS



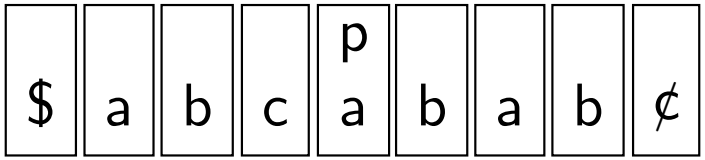
a Turing Machine is an SRS

$$\delta(q, a) = (p, c, \rightarrow)$$



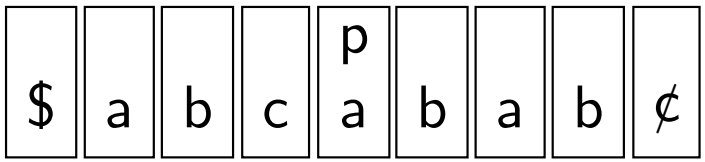
a Turing Machine is an SRS

$$\delta(q, a) = (p, c, \rightarrow)$$



a Turing Machine is an SRS

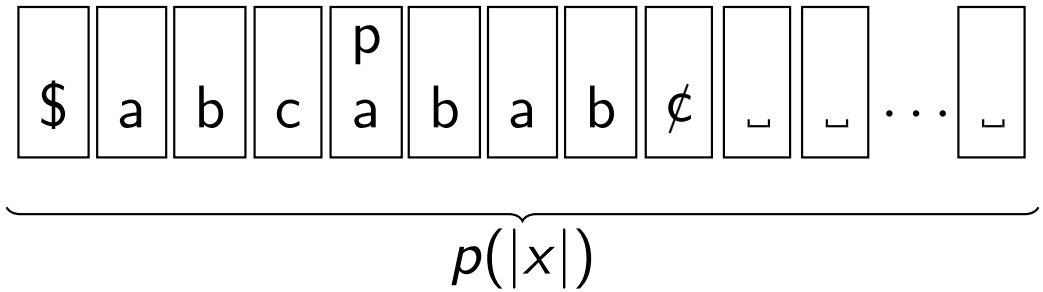
$$\delta(q, a) = (p, c, \rightarrow)$$



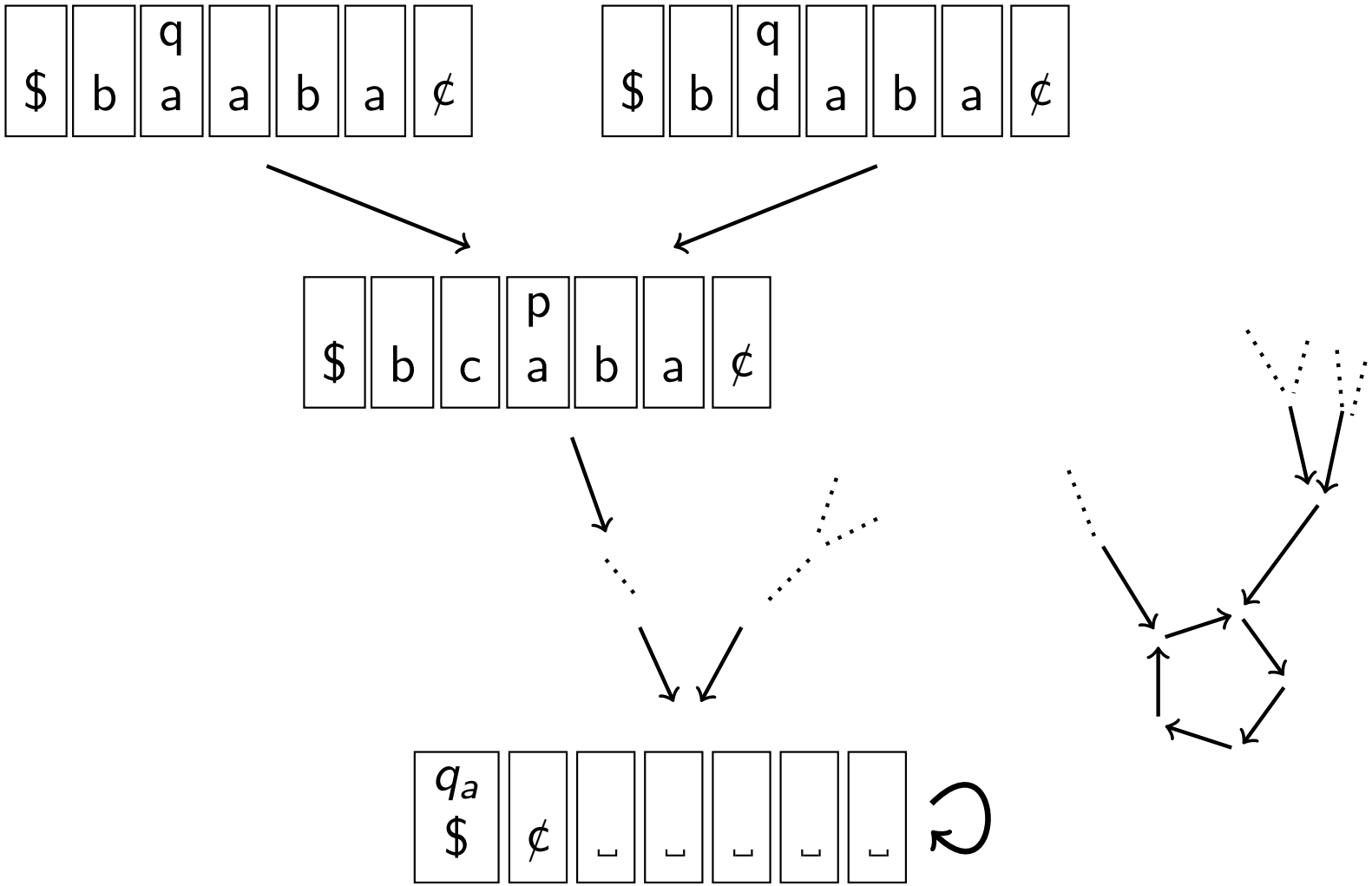
$$\left(\begin{array}{|c|}\hline q \\ \hline a \\ \hline \end{array} \begin{array}{|c|}\hline \\ \hline a \\ \hline \end{array} , \begin{array}{|c|}\hline \\ \hline c \\ \hline \end{array} \begin{array}{|c|}\hline p \\ \hline a \\ \hline \end{array} \right) \in R$$

a PSPACE Turing Machine is a balanced SRS

$$\delta(q, a) = (p, c, \rightarrow)$$



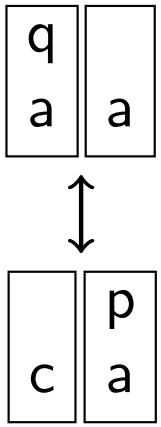
a deterministic Turing Machine is a symmetric SRS



Simplifying the system further

Lemma

There is a 2-balanced symmetric SRS (Σ, R) whose word problem is PSPACE-complete.



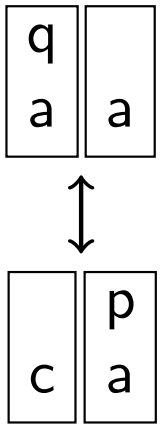
Simplifying the system further

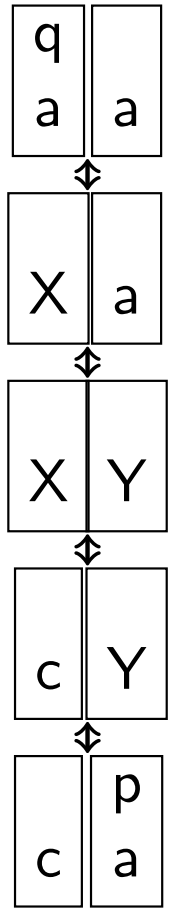
Lemma

There is a 2-balanced symmetric SRS (Σ, R) such that for all $\{a_1a_2, b_1b_2\} \in R$

$$\text{either } a_1 = b_1 \text{ or } a_2 = b_2$$

whose word problem is PSPACE-complete.

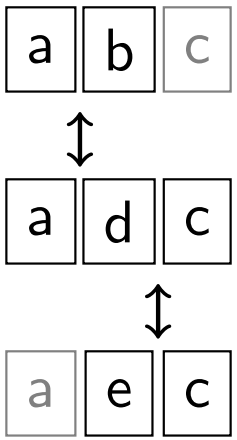




Simplifying the system even further

Lemma

There is a 2-balanced symmetric SRS (Σ, R)
such that for all $\{a_1a_2, b_1b_2\} \in R$ either $a_1 = b_1$ or $a_2 = b_2$
whose word problem is PSPACE-complete.



H-words

Let $H = (\Sigma, R)$ where $R \subseteq \Sigma \times \Sigma$ is a relation between symbols.
H-word := consecutive symbols are related = directed walk in H

H-WORD RECONFIGURATION

Input: $u, v \in \Sigma^n$ – two H-words of equal length
Question: Is there a sequence of H-words between u and v such that consecutive H-words differ in only one symbol?

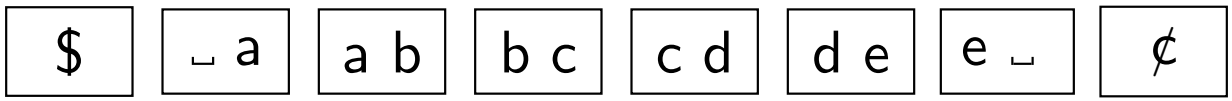
H-WORD RECONFIGURATION is hard

Let $\Big(\{a, b, c, \dots\}\ ,\ \big\{\{a_1 b_1, a_1, c_1\}, \dots\big\}\Big)$ be our SRS.

Let $\Sigma =$

pair symbols: $ab, ac, \dots, \sqcup a, \sqcup b, \dots$

special symbols: $\$, \not\subset, x_1, \dots$



Allow:

a b	b c	for any a, b, c
$	␣ a	for any a
a ␣	⊄	for any a

$\cdot\ a_i$	x_i	
x_i	$b_i\ \cdot$	for any $\cdot$
x_i	$c_i\ \cdot$	for every rule

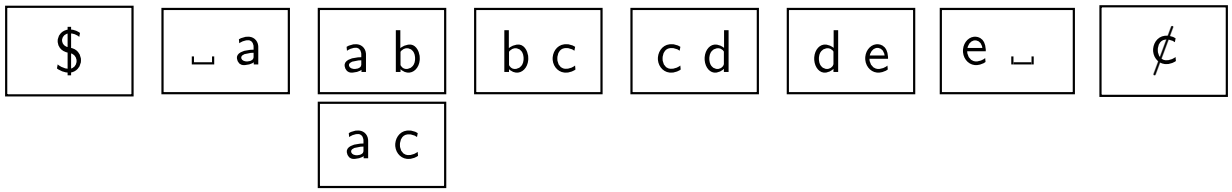
H-WORD RECONFIGURATION is hard

Let $\Big(\{a, b, c, \dots\}\ ,\ \big\{\{a_1 b_1, a_1, c_1\}, \dots\big\}\Big)$ be our SRS.

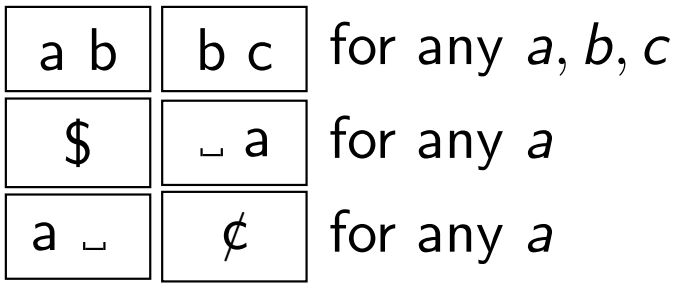
Let $\Sigma =$

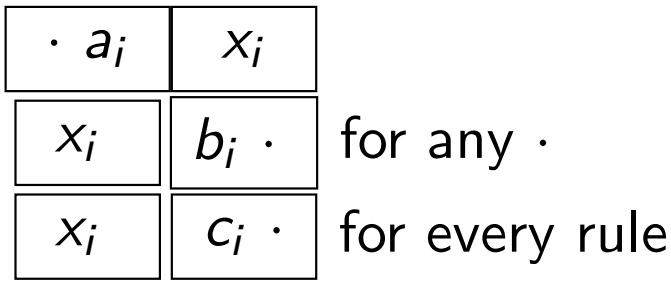
pair symbols: $ab, ac, \dots, \sqcup a, \sqcup b, \dots$

special symbols: $\$, \not\subset, x_1, \dots$



Allow:





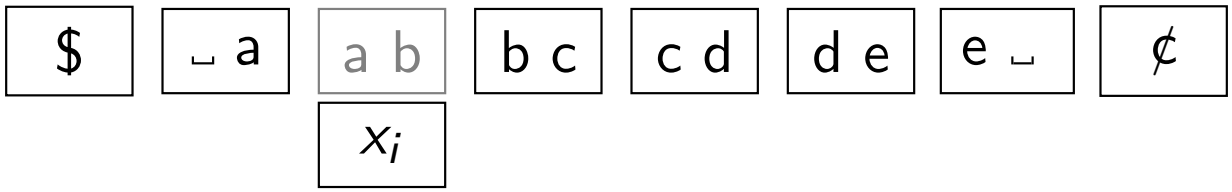
H-WORD RECONFIGURATION is hard

Let $\Big(\{a, b, c, \dots\}\ ,\ \{\{a_1 b_1, a_1, c_1\}, \dots\}\Big)$ be our SRS.

Let $\Sigma =$

pair symbols: $ab, ac, \dots, \sqcup a, \sqcup b, \dots$

special symbols: $\$, \not\subset, x_1, \dots$



Allow:

a b	b c	for any a, b, c
$	␣ a	for any a
a ␣	⊄	for any a

$\cdot\ a_i$	x_i	
x_i	$b_i\ \cdot$	for any $\cdot$
x_i	$c_i\ \cdot$	for every rule

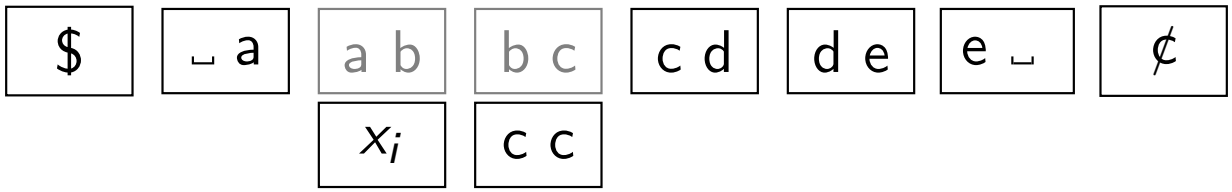
H-WORD RECONFIGURATION is hard

Let $\Big(\{a, b, c, \dots\}\ ,\ \big\{\{a_1 b_1, a_1, c_1\}, \dots\big\}\Big)$ be our SRS.

Let $\Sigma =$

pair symbols: $ab, ac, \dots, \sqcup a, \sqcup b, \dots$

special symbols: $\$, \not\subset, x_1, \dots$



Allow:

a b	b c	for any a, b, c
$	␣ a	for any a
a ␣	¢	for any a

$\cdot\ a_i$	x_i	
x_i	$b_i\ \cdot$	for any $\cdot$
x_i	$c_i\ \cdot$	for every rule

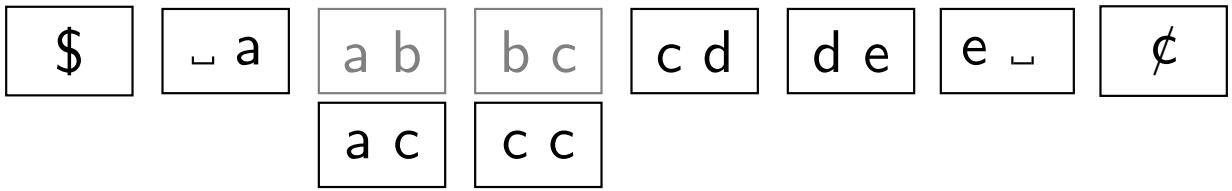
H-WORD RECONFIGURATION is hard

Let $\Big(\{a, b, c, \dots\}\ ,\ \big\{\{a_1 b_1, a_1, c_1\}, \dots\big\}\Big)$ be our SRS.

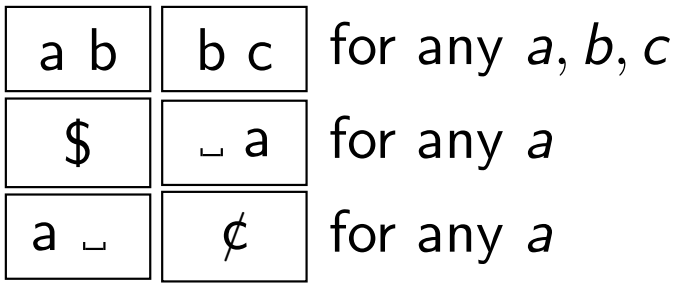
Let $\Sigma =$

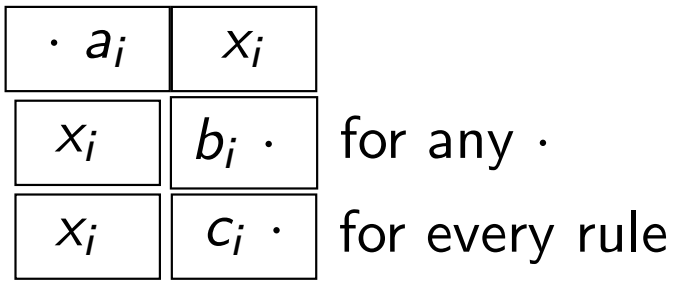
pair symbols: $ab, ac, \dots, \sqcup a, \sqcup b, \dots$

special symbols: $\$, \not\subset, x_1, \dots$

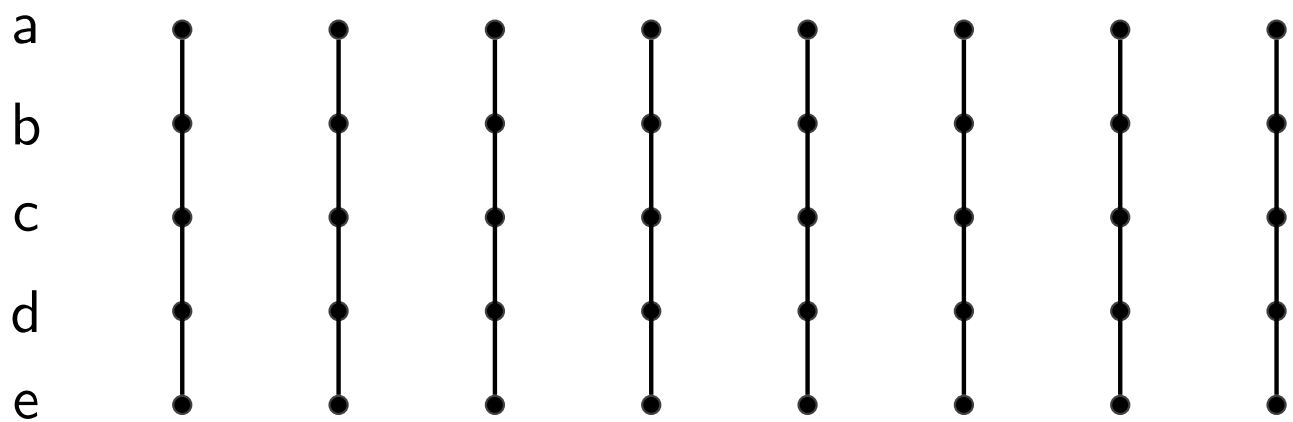


Allow:

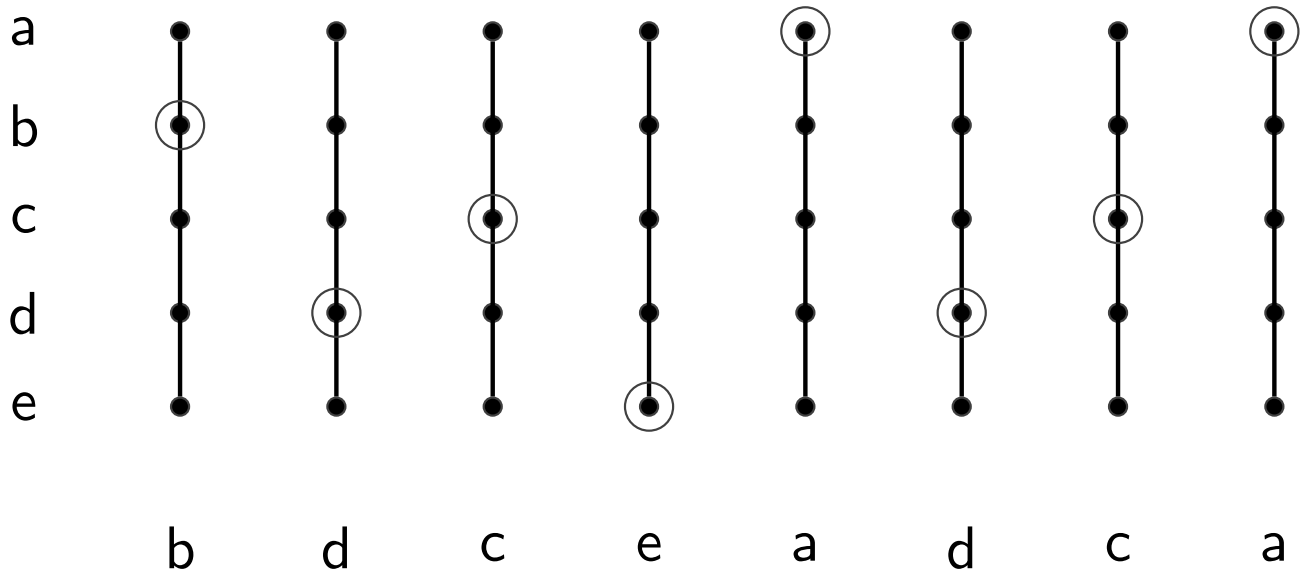




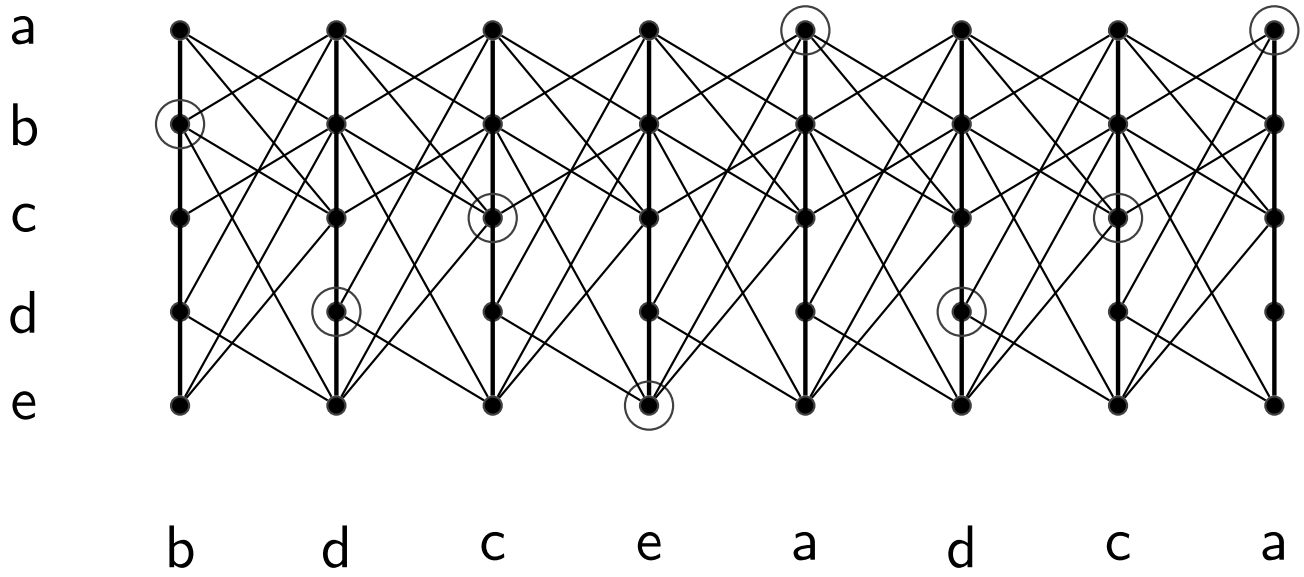
Token Sliding and Jumping



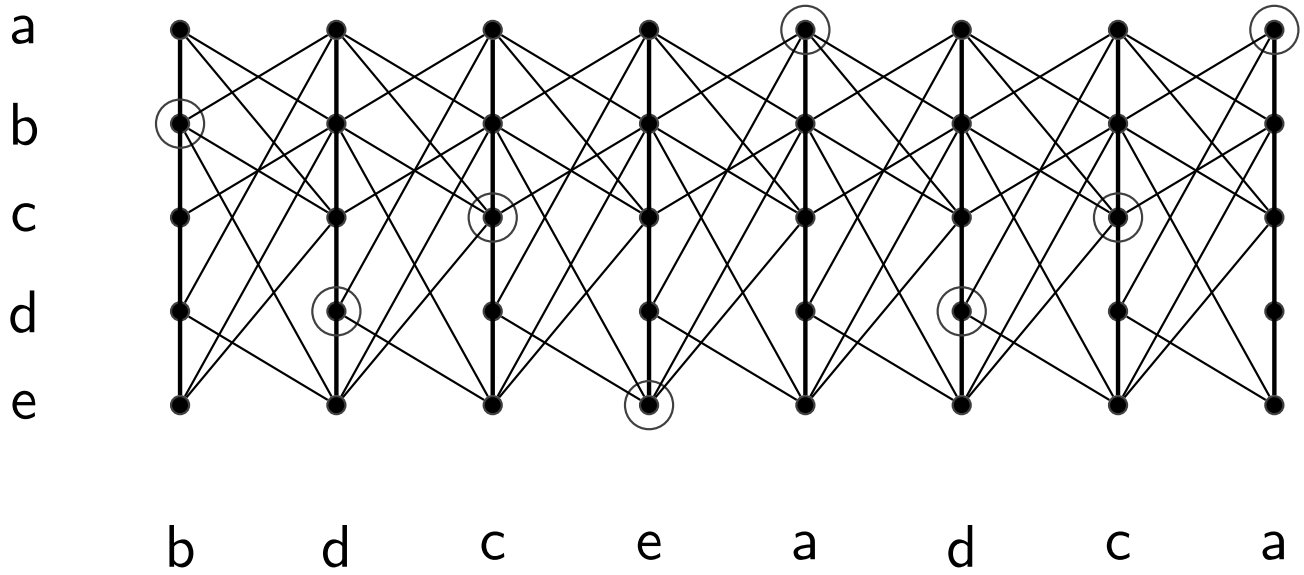
Token Sliding and Jumping



Token Sliding and Jumping

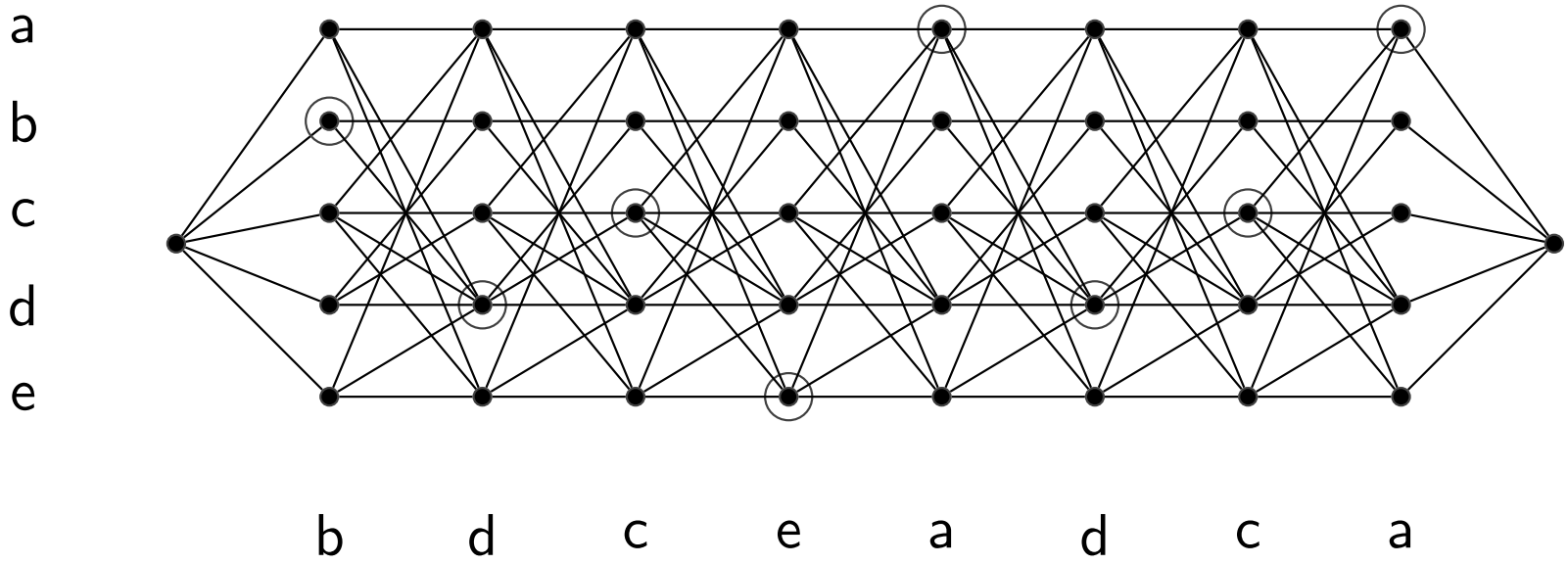


Token Sliding and Jumping



There is a p such that Token Sliding and Token Jumping are PSPACE-complete on graphs of pathwidth $\leq p$.

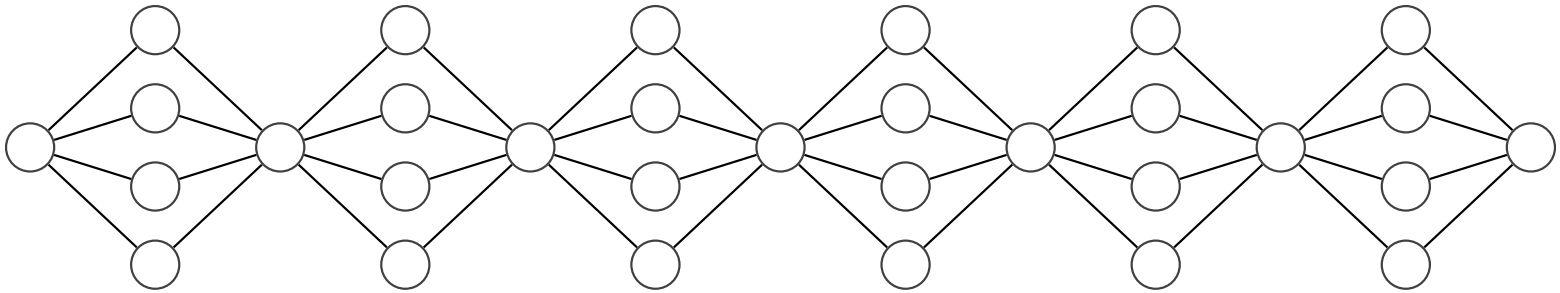
Shortest Path



There is a p such that Shortest Path Reconfiguration is PSPACE-complete on graphs of pathwidth $\leq p$.

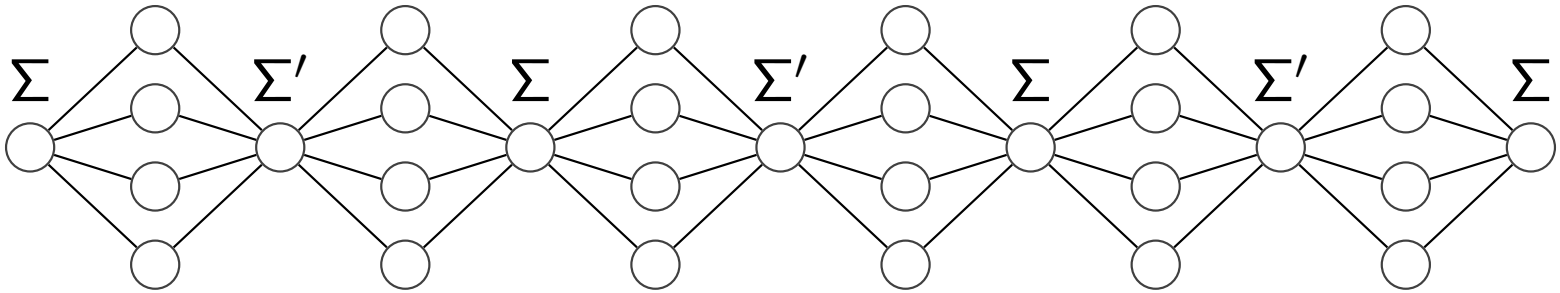
k-list-recoloring

Let $H = (\Sigma, R)$, let Σ' be a copy of Σ.



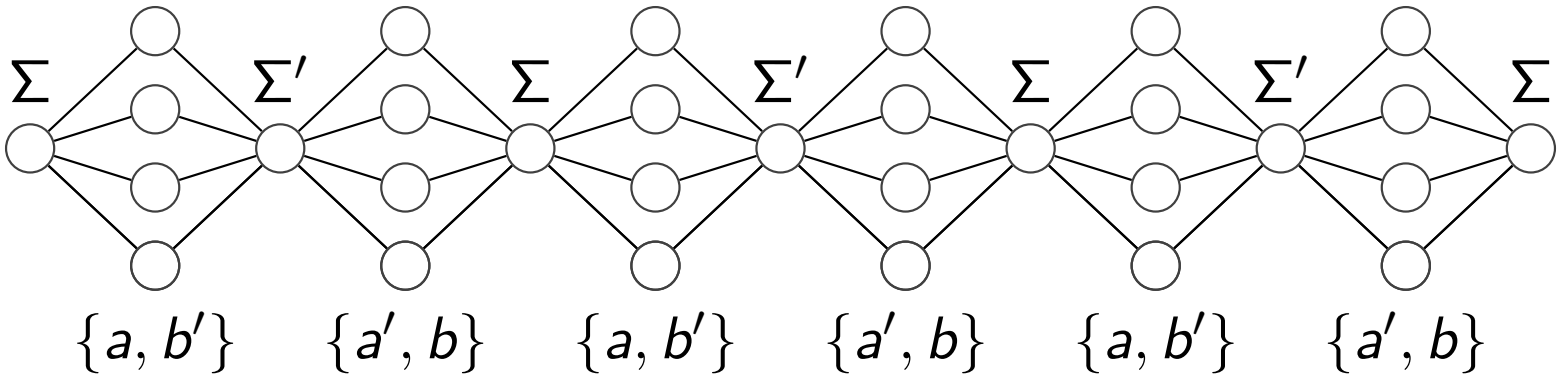
k-list-recoloring

Let $H = (\Sigma, R)$, let Σ' be a copy of Σ.



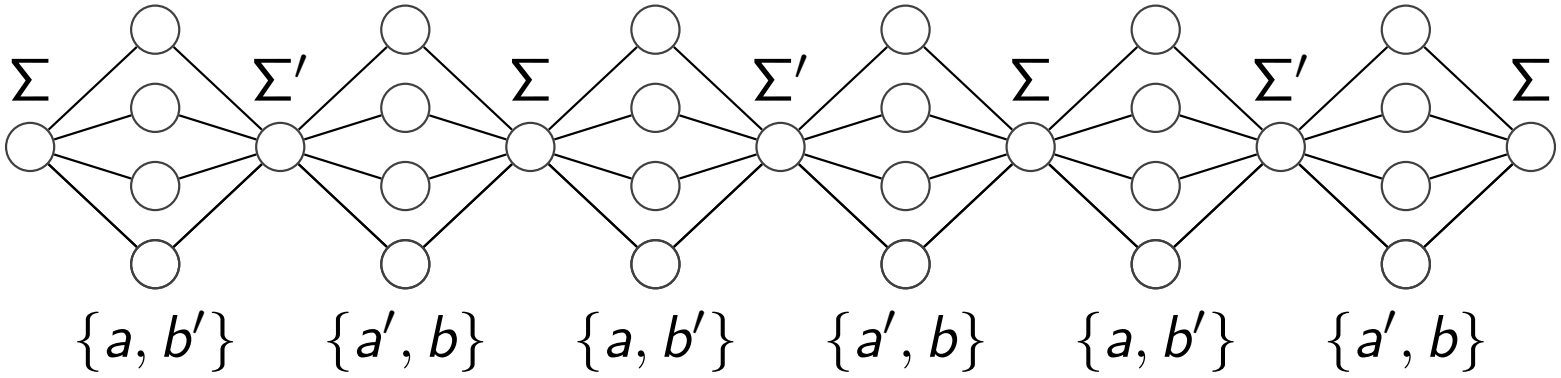
k-list-recoloring

Let $H = (\Sigma, R)$, let Σ' be a copy of Σ.



k-list-recoloring

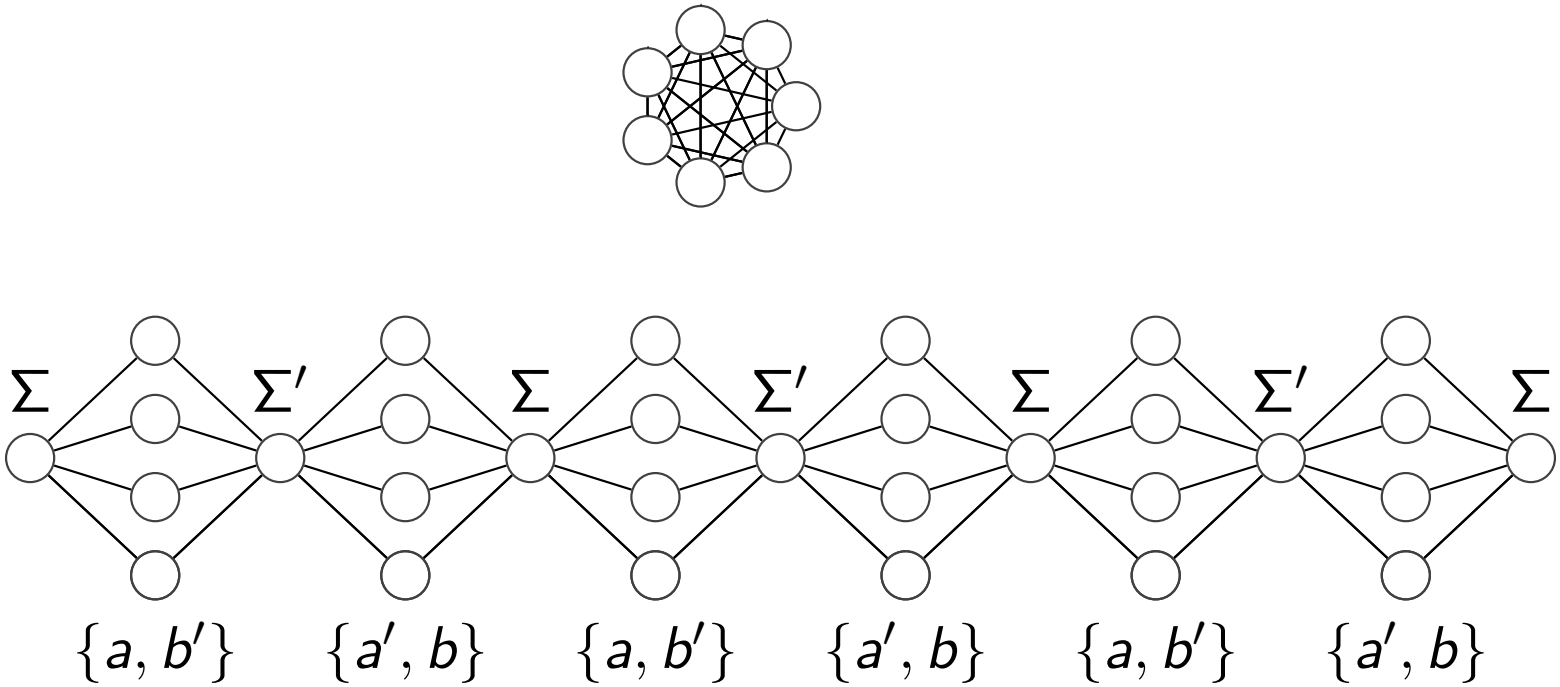
Let $H = (\Sigma, R)$, let Σ' be a copy of Σ.



There are integers k, p such that k-LIST-RECOLORING is PSPACE-complete on chains of p-onions.

k-recoloring

Just add a k-clique, give it a fixed coloring, replace lists with edges.



There are integers k, p such that k-RECOLORING is PSPACE-complete on graphs of pathwidth $\leq p$.

Pathwidth - conclusions

- Many reconfiguration problems are PSPACE-complete.
- But not all. Clique reconfiguration is trivial:
  all cliques can be listed in $\mathcal{O}(2^p n)$ time in treewidth $\leq p$.
- Is connectivity hard?
- Parameterized hardness – what parameter?

Pathwidth - conclusions

- Many reconfiguration problems are PSPACE-complete.
- But not all. Clique reconfiguration is trivial:
  all cliques can be listed in $\mathcal{O}(2^p n)$ time in treewidth $\leq p$.
- Is connectivity hard?
- Parameterized hardness – what parameter?

Pathwidth - conclusions

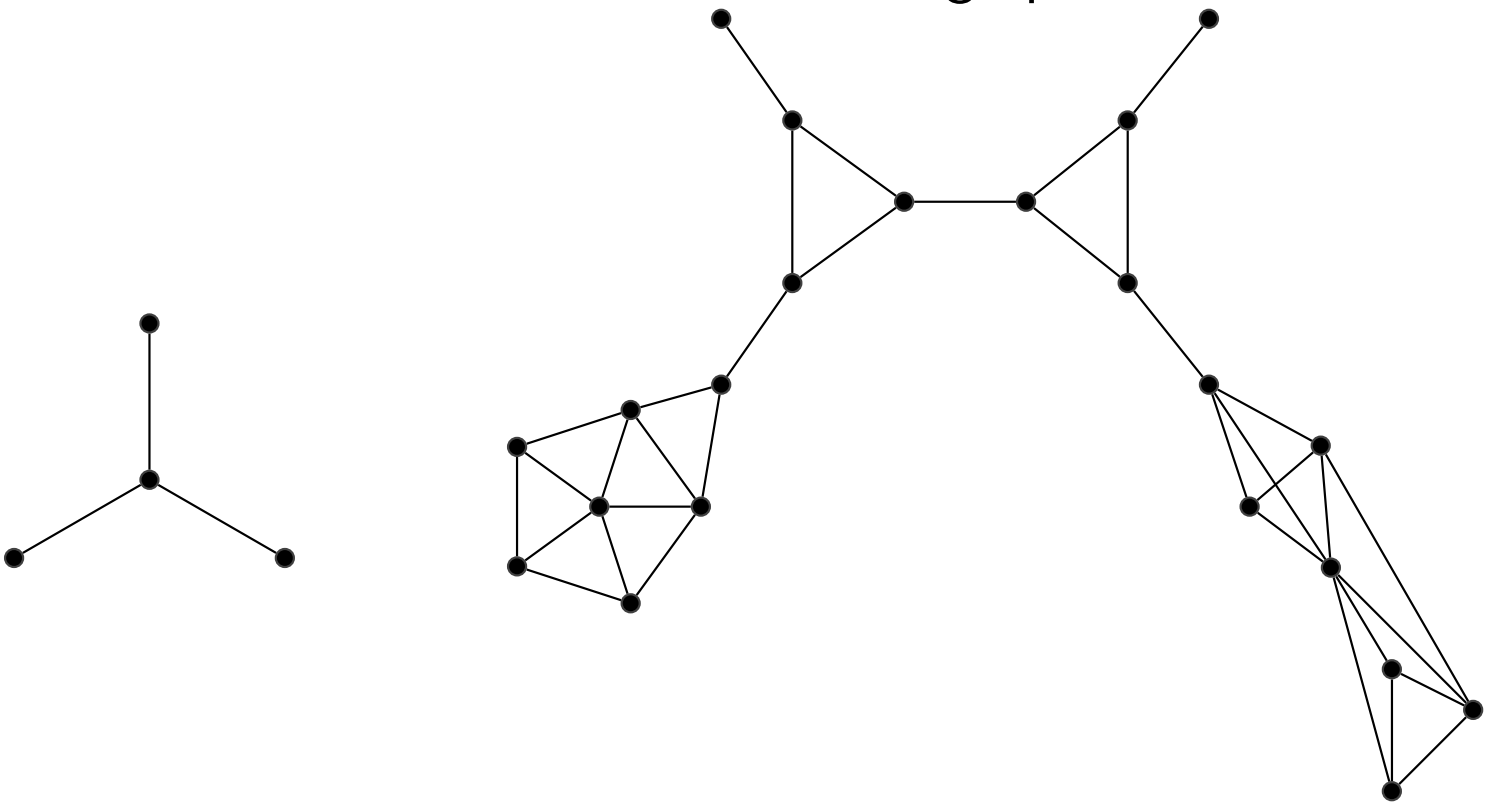
- Many reconfiguration problems are PSPACE-complete.
- But not all. Clique reconfiguration is trivial:
  all cliques can be listed in $\mathcal{O}(2^p n)$ time in treewidth $\leq p$.
- Is connectivity hard?
- Parameterized hardness – what parameter?

Pathwidth - conclusions

- Many reconfiguration problems are PSPACE-complete.
- But not all. Clique reconfiguration is trivial:
  all cliques can be listed in $\mathcal{O}(2^p n)$ time in treewidth $\leq p$.
- Is connectivity hard?
- Parameterized hardness – what parameter?

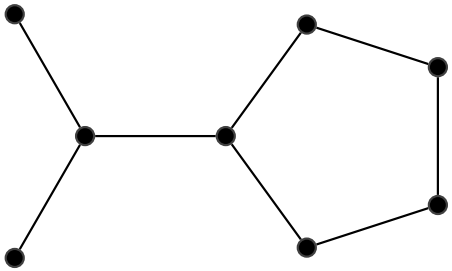
Claw-free graphs

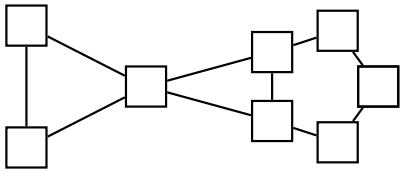
Claw-free := no claw as an induced subgraph.



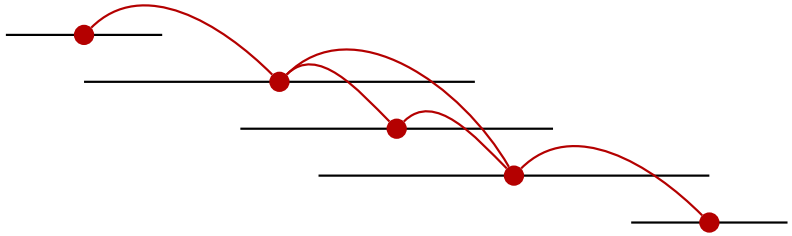
Claw-free graphs

Line graphs



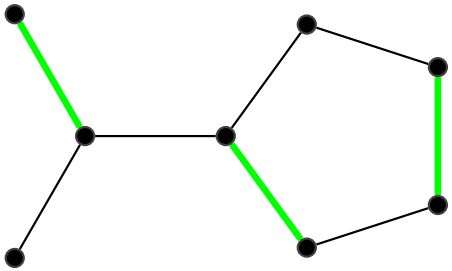


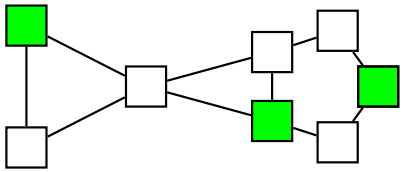
Proper interval graphs



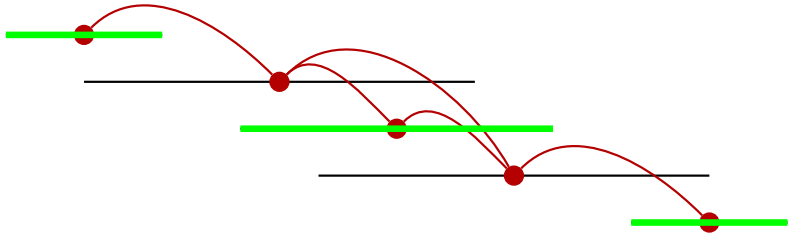
Claw-free graphs

Line graphs





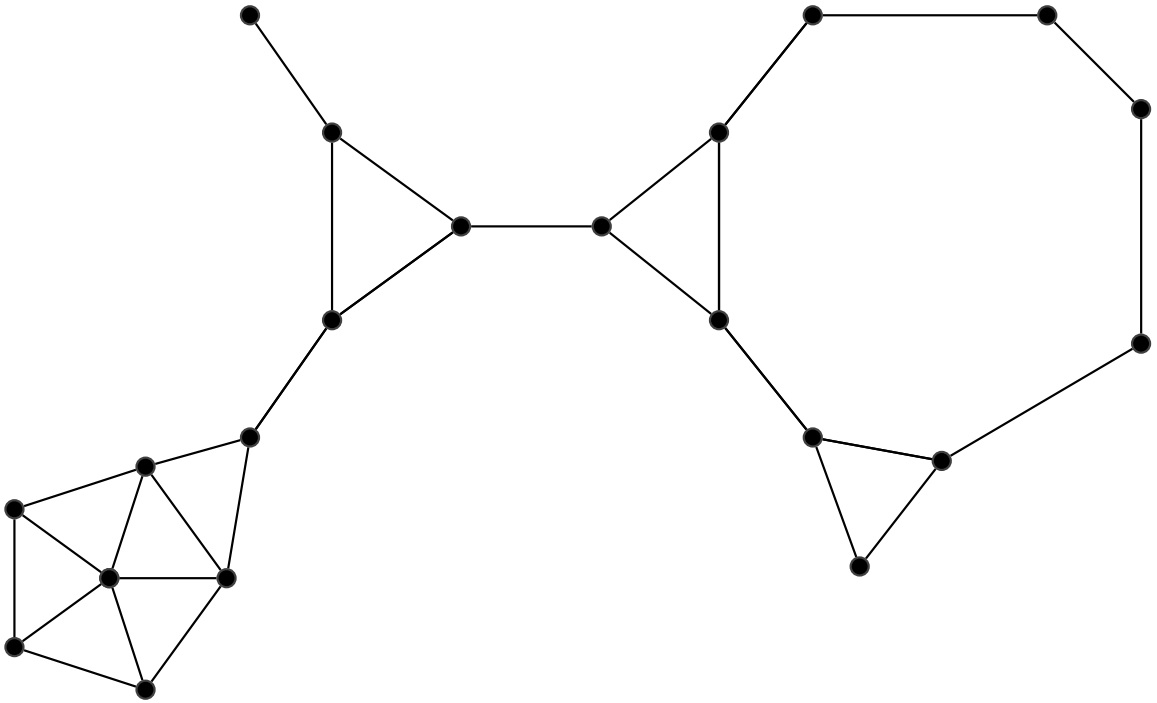
Proper interval graphs



No-cycle case

No-cycle lemma

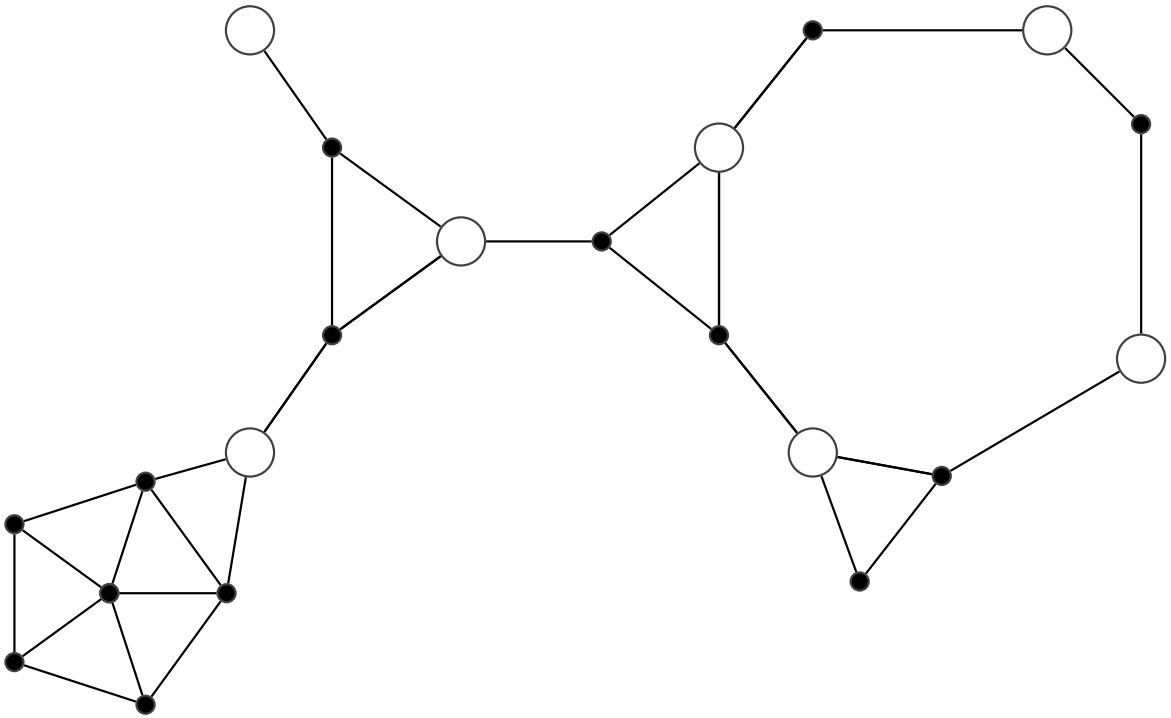
Let I, J be independent sets in a connected claw-free G.
If $G[I \Delta J]$ contains no cycles, then $I \leftrightarrow_{\mathsf{TS}} J$.



No-cycle case

No-cycle lemma

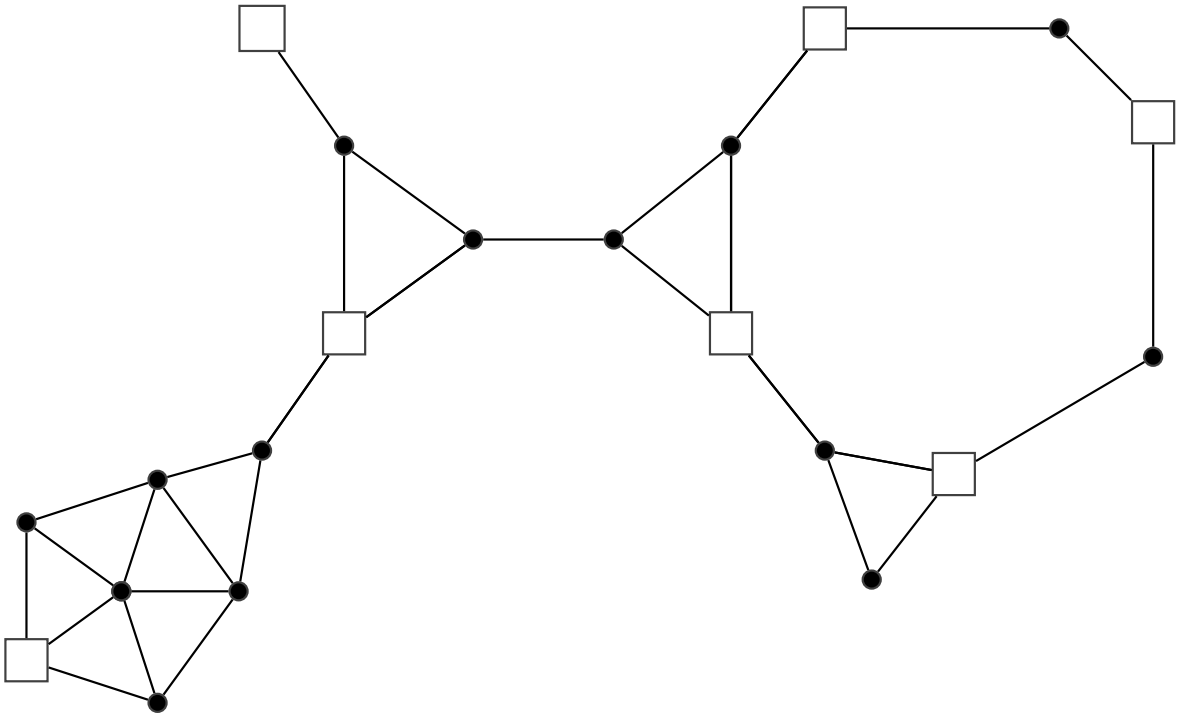
Let I, J be independent sets in a connected claw-free G.
If $G[I \Delta J]$ contains no cycles, then $I \leftrightarrow_{\mathsf{TS}} J$.



No-cycle case

No-cycle lemma

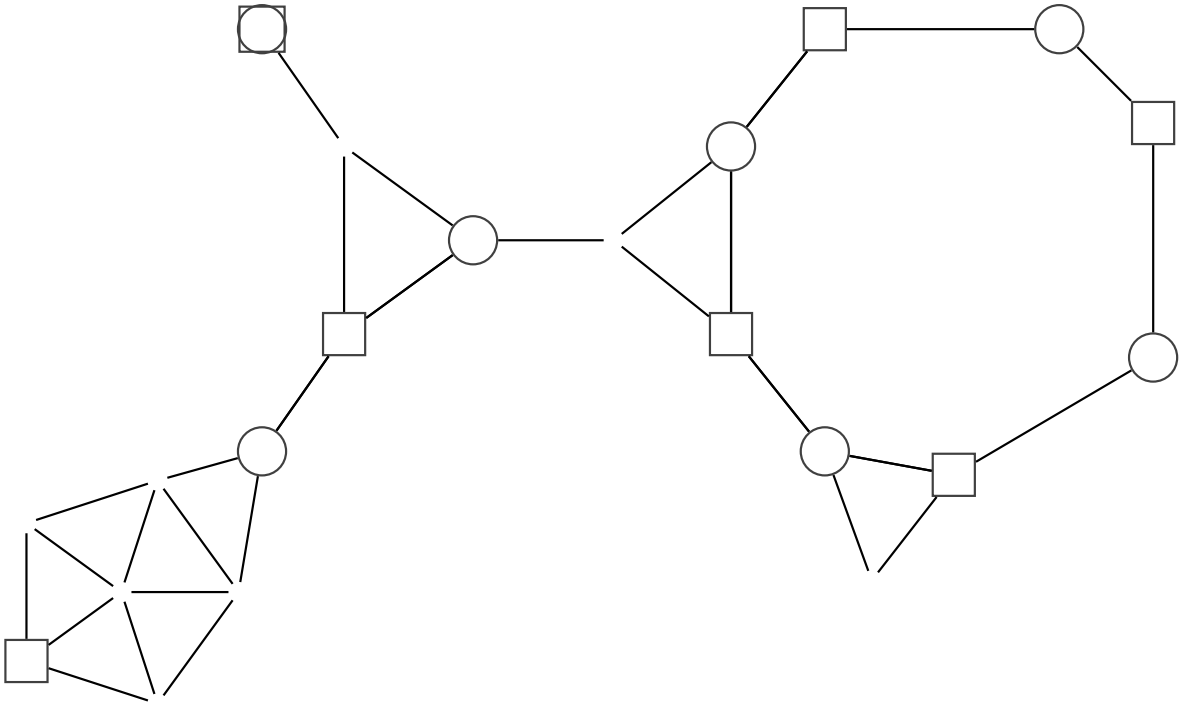
Let I, J be independent sets in a connected claw-free G.
If $G[I \Delta J]$ contains no cycles, then $I \leftrightarrow_{\mathsf{TS}} J$.



No-cycle case

No-cycle lemma

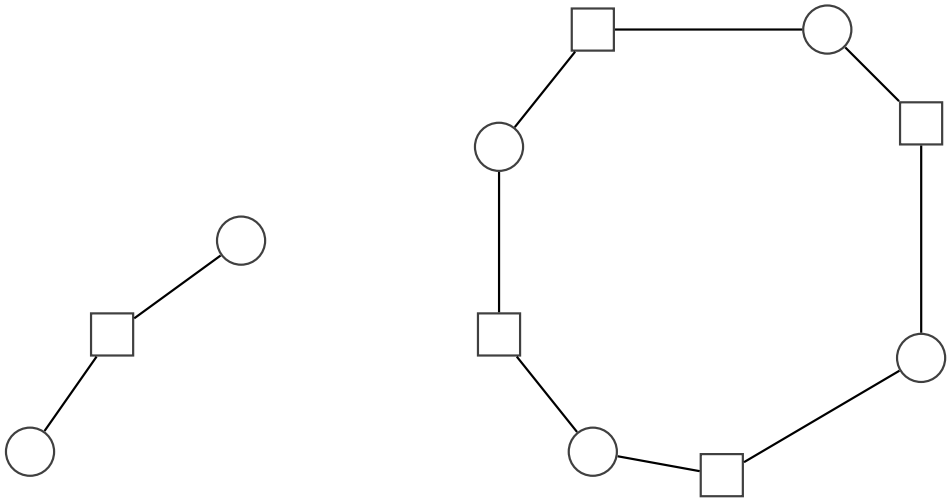
Let I, J be independent sets in a connected claw-free G.
If $G[I \Delta J]$ contains no cycles, then $I \leftrightarrow_{\mathsf{TS}} J$.



No-cycle case

No-cycle lemma

Let I, J be independent sets in a connected claw-free G.
If $G[I \Delta J]$ contains no cycles, then $I \leftrightarrow_{\mathsf{TS}} J$.



No-cycle case

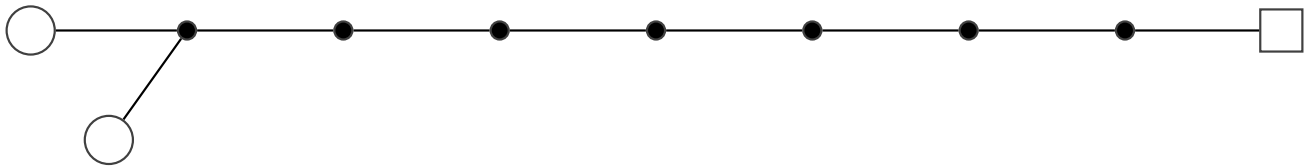
No-cycle lemma

Let I, J be independent sets in a connected claw-free G.
If $G[I \Delta J]$ contains no cycles, then $I \leftrightarrow_{\mathsf{TS}} J$.

No-cycle case

No-cycle lemma

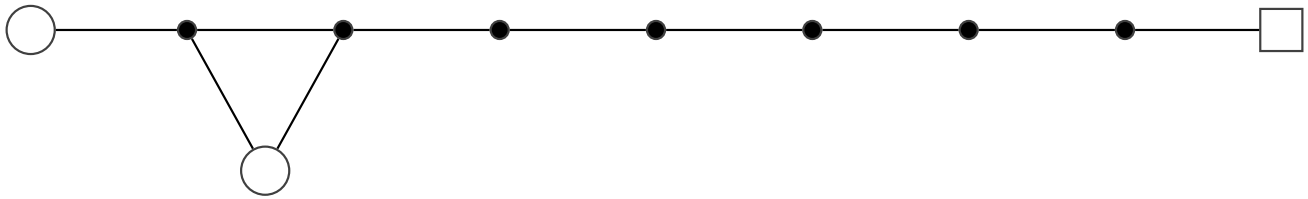
Let I, J be independent sets in a connected claw-free G.
If $G[I \Delta J]$ contains no cycles, then $I \leftrightarrow_{\mathsf{TS}} J$.



No-cycle case

No-cycle lemma

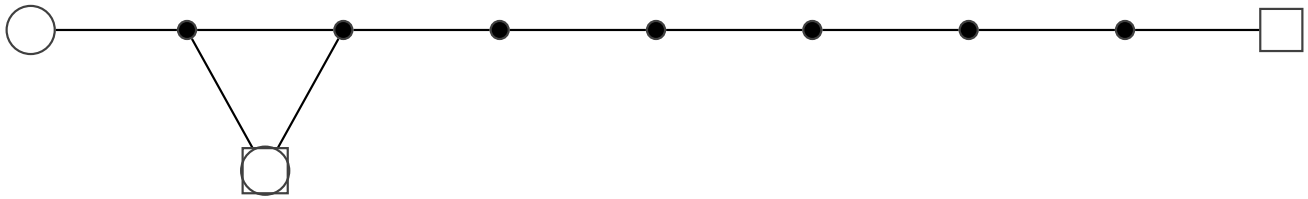
Let I, J be independent sets in a connected claw-free G.
If $G[I \Delta J]$ contains no cycles, then $I \leftrightarrow_{\mathsf{TS}} J$.



No-cycle case

No-cycle lemma

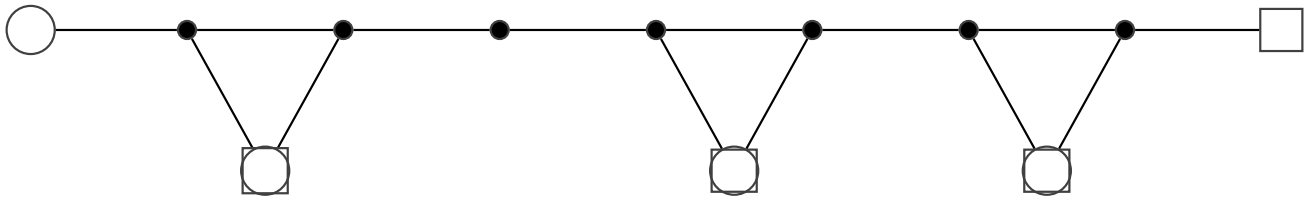
Let I, J be independent sets in a connected claw-free G.
If $G[I \Delta J]$ contains no cycles, then $I \leftrightarrow_{\mathsf{TS}} J$.



No-cycle case

No-cycle lemma

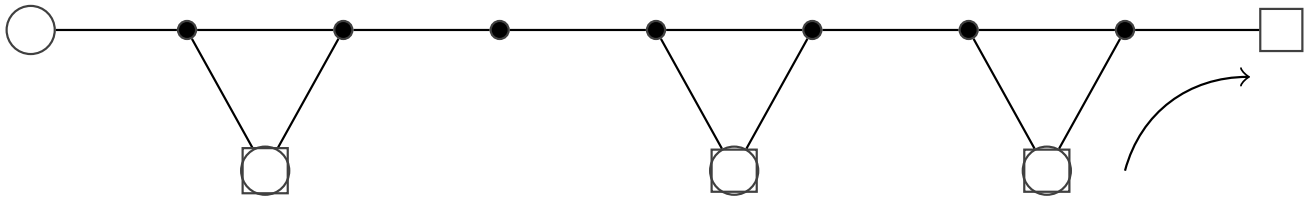
Let I, J be independent sets in a connected claw-free G.
If $G[I \Delta J]$ contains no cycles, then $I \leftrightarrow_{\mathsf{TS}} J$.



No-cycle case

No-cycle lemma

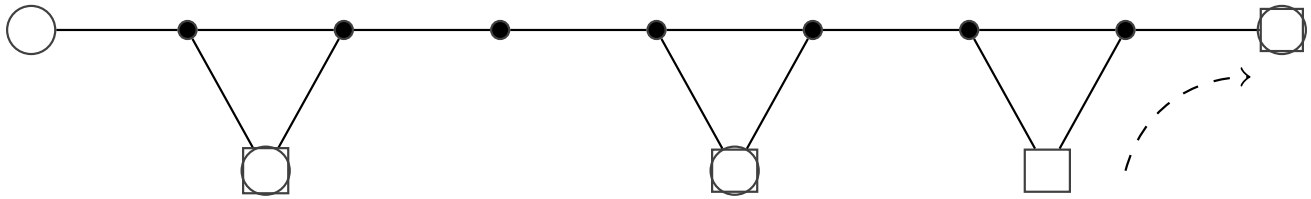
Let I, J be independent sets in a connected claw-free G.
If $G[I \Delta J]$ contains no cycles, then $I \leftrightarrow_{\mathsf{TS}} J$.



No-cycle case

No-cycle lemma

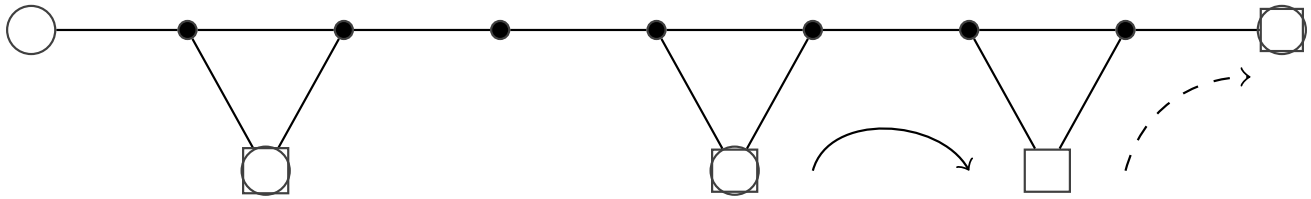
Let I, J be independent sets in a connected claw-free G.
If $G[I \Delta J]$ contains no cycles, then $I \leftrightarrow_{\mathsf{TS}} J$.



No-cycle case

No-cycle lemma

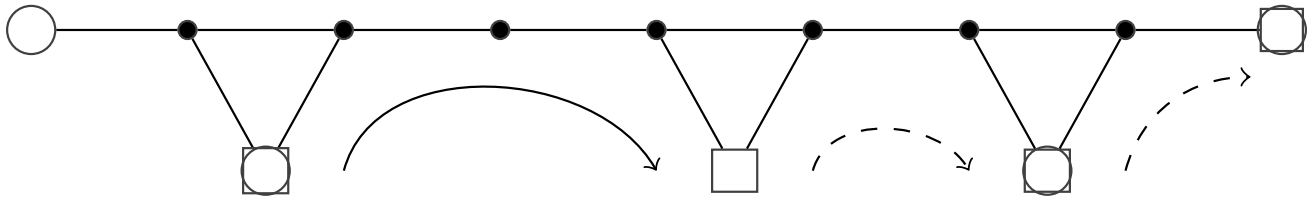
Let I, J be independent sets in a connected claw-free G.
If $G[I \Delta J]$ contains no cycles, then $I \leftrightarrow_{\mathsf{TS}} J$.



No-cycle case

No-cycle lemma

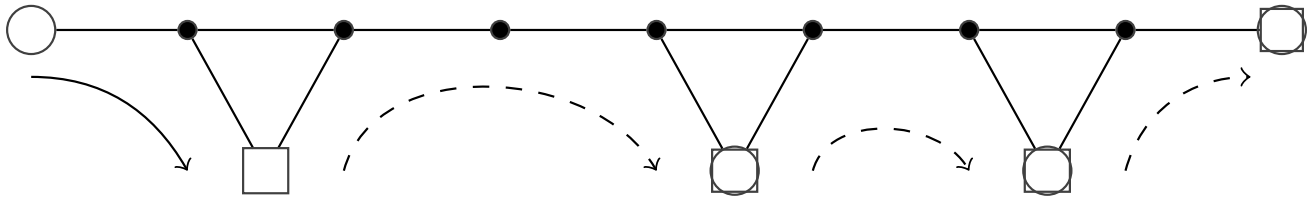
Let I, J be independent sets in a connected claw-free G.
If $G[I \Delta J]$ contains no cycles, then $I \leftrightarrow_{\mathsf{TS}} J$.



No-cycle case

No-cycle lemma

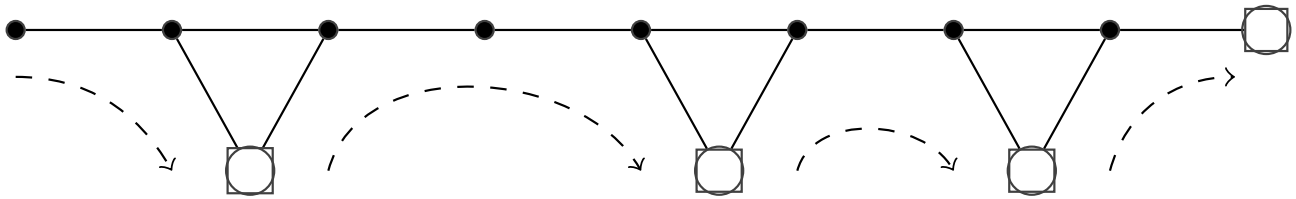
Let I, J be independent sets in a connected claw-free G.
If $G[I \Delta J]$ contains no cycles, then $I \leftrightarrow_{\mathsf{TS}} J$.



No-cycle case

No-cycle lemma

Let I, J be independent sets in a connected claw-free G.
If $G[I \Delta J]$ contains no cycles, then $I \leftrightarrow_{\mathsf{TS}} J$.



TS is equivalent to TJ

No-cycle lemma

Let I, J be independent sets in a connected claw-free G.
If $G[I \Delta J]$ contains no cycles, then $I \leftrightarrow_{\mathsf{TS}} J$.

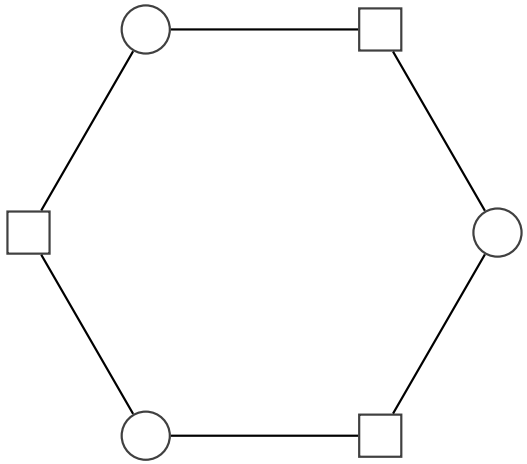
The symmetric difference between I and $I - u + v$ has no cycles!

Corollary

In a connected graph G, $I \leftrightarrow_{\mathsf{TS}} J$ if and only if $I \leftrightarrow_{\mathsf{TJ}} J$.

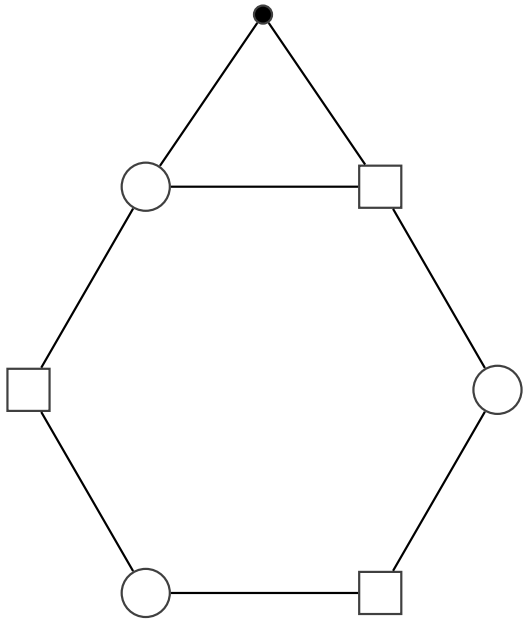
Breaking cycles

NO instance – can't move any token.



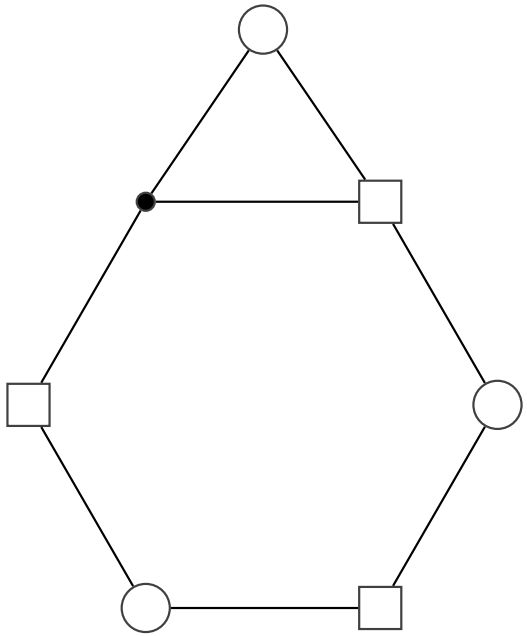
Breaking cycles

YES instance – we can break the cycle.



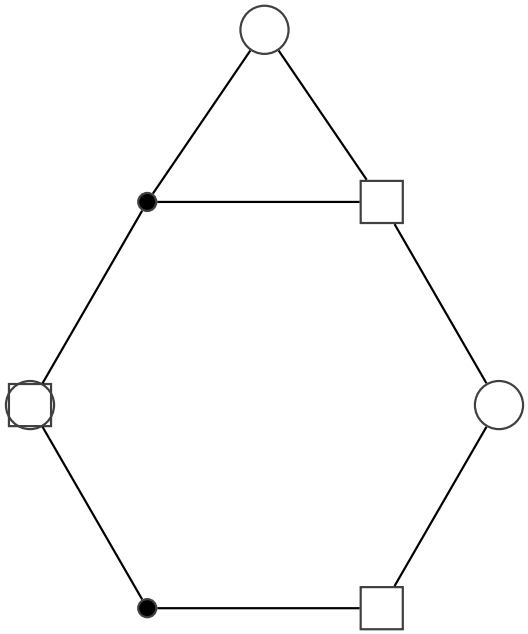
Breaking cycles

YES instance – we can break the cycle.



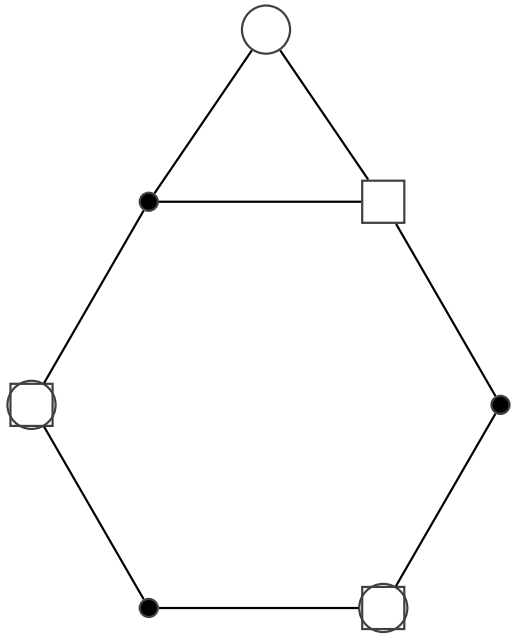
Breaking cycles

YES instance – we can break the cycle.



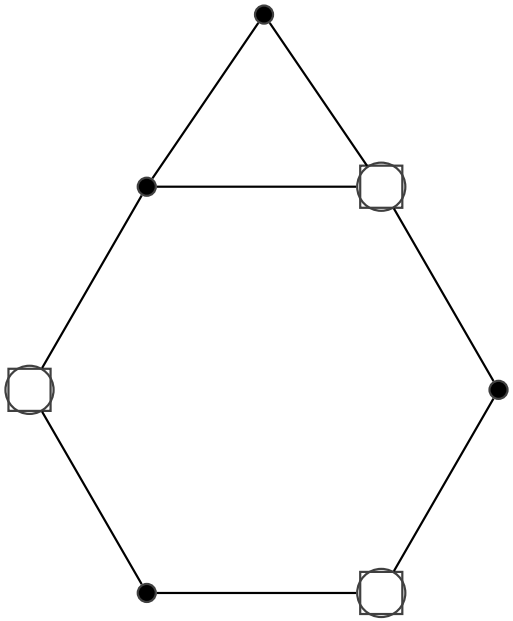
Breaking cycles

YES instance – we can break the cycle.



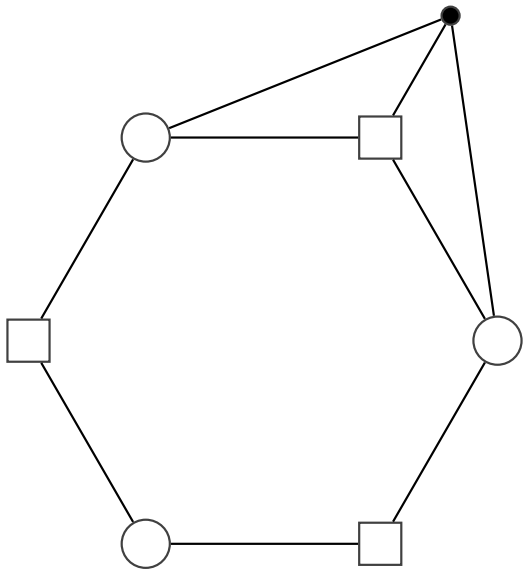
Breaking cycles

YES instance – we can break the cycle.



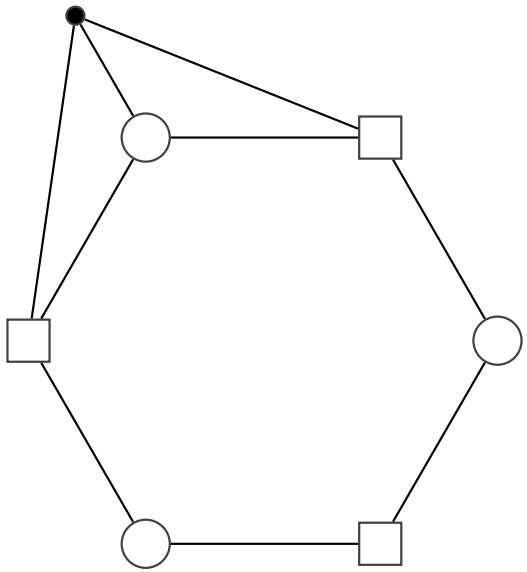
Breaking cycles

NO instance – can't move any token.



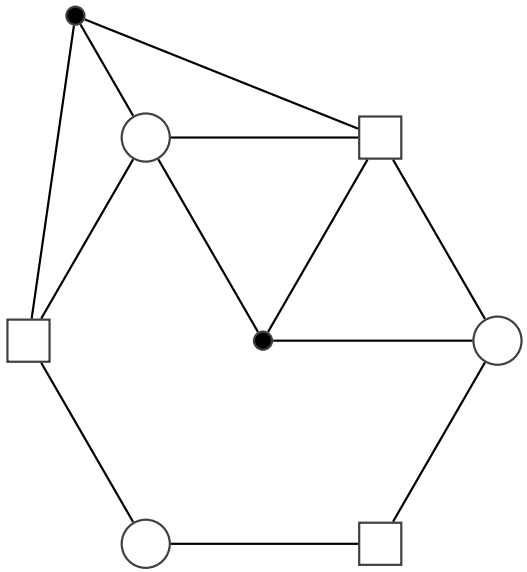
Breaking cycles

NO instance – can't break the cycle.



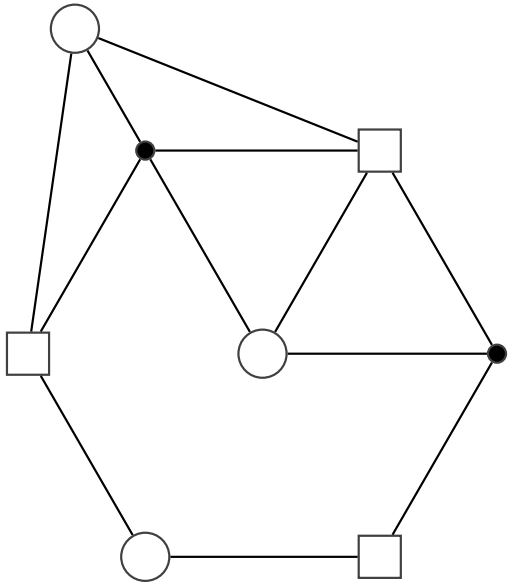
Breaking cycles

YES instance – we can break the cycle.



Breaking cycles

YES instance – we can break the cycle.



Internal and external

$\square \cap C$ – target vertices on the cycle.
Internal vertex := has exactly 2 neighbors in $\square \cap C$.
Breaking vertex := has exactly 1 neighbor in $\square \cap C$.
External vertex := has exactly 0 neighbors in $\square \cap C$.
V = internal $\uplus$ breaking $\uplus$ external $\uplus$ $(\square \cap C)$

Observation

In any token sliding sequence, a move is the first to break the cycle iff it is the first to move a token to a breaking vertex.

There are no edges between external and internal vertices!

Internal and external

$\square \cap C$ – target vertices on the cycle.
Internal vertex := has exactly 2 neighbors in $\square \cap C$.
Breaking vertex := has exactly 1 neighbor in $\square \cap C$.
External vertex := has exactly 0 neighbors in $\square \cap C$.
V = internal $\uplus$ breaking $\uplus$ external $\uplus$ $(\square \cap C)$

Observation

In any token sliding sequence, a move is the first to break the cycle iff it is the first to move a token to a breaking vertex.

There are no edges between external and internal vertices!

Internal and external

$\square \cap C$ – target vertices on the cycle.
Internal vertex := has exactly 2 neighbors in $\square \cap C$.
Breaking vertex := has exactly 1 neighbor in $\square \cap C$.
External vertex := has exactly 0 neighbors in $\square \cap C$.
V = internal $\uplus$ breaking $\uplus$ external $\uplus$ $(\square \cap C)$

Observation

In any token sliding sequence, a move is the first to break the cycle iff it is the first to move a token to a breaking vertex.

There are no edges between external and internal vertices!

Internal and external

There are no edges between external and internal vertices.

Let S be a shortest sequence breaking the cycle.
S ends with an internal-to-breaking move.
Any move before that is internal-to-internal or external-to-external.

External moves are completely independent from internal moves.
Only one move is needed to make the last vertex free.
So we can forget all moves of the other type.

S has only external moves or only internal moves.

Internal and external

There are no edges between external and internal vertices.

Let S be a shortest sequence breaking the cycle.
S ends with an internal-to-breaking move.
Any move before that is internal-to-internal or external-to-external.

External moves are completely independent from internal moves.
Only one move is needed to make the last vertex free.
So we can forget all moves of the other type.

S has only external moves or only internal moves.

Internal and external

There are no edges between external and internal vertices.

Let S be a shortest sequence breaking the cycle.
S ends with an internal-to-breaking move.
Any move before that is internal-to-internal or external-to-external.

External moves are completely independent from internal moves.
Only one move is needed to make the last vertex free.
So we can forget all moves of the other type.

S has only external moves or only internal moves.

Internal and external

There are no edges between external and internal vertices.

Let S be a shortest sequence breaking the cycle.
S ends with an internal-to-breaking move.
Any move before that is internal-to-internal or external-to-external.

External moves are completely independent from internal moves.
Only one move is needed to make the last vertex free.
So we can forget all moves of the other type.

S has only external moves or only internal moves.

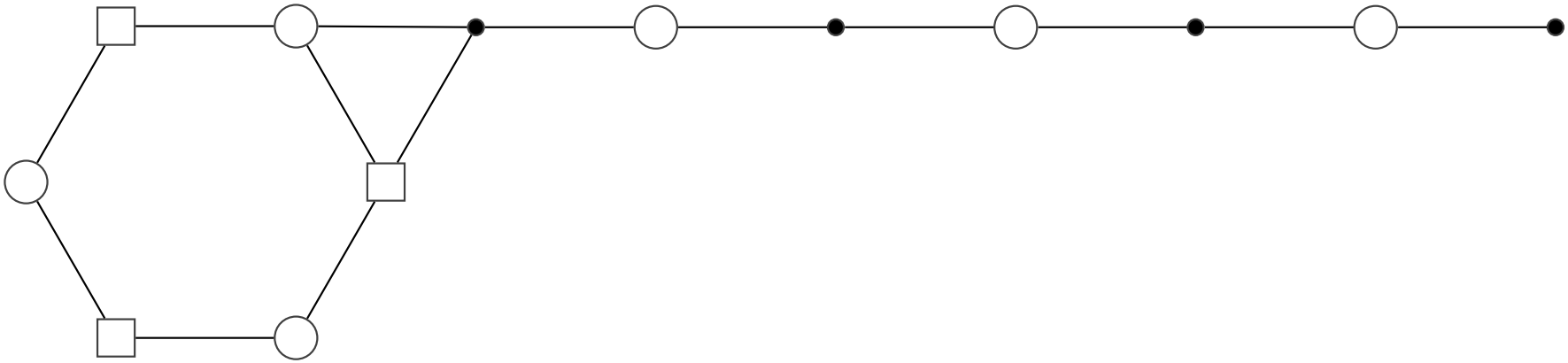
Resolving all cycles

Let S be a shortest sequence breaking the cycle.
S has only external moves or only internal moves.

Then I can be reconfigured to $I \Delta C$.
So if each is breakable, then we can 'turn' them one by one.

Thus it's enough to ask for each cycle independently: is it externally or internally breakable?

External case



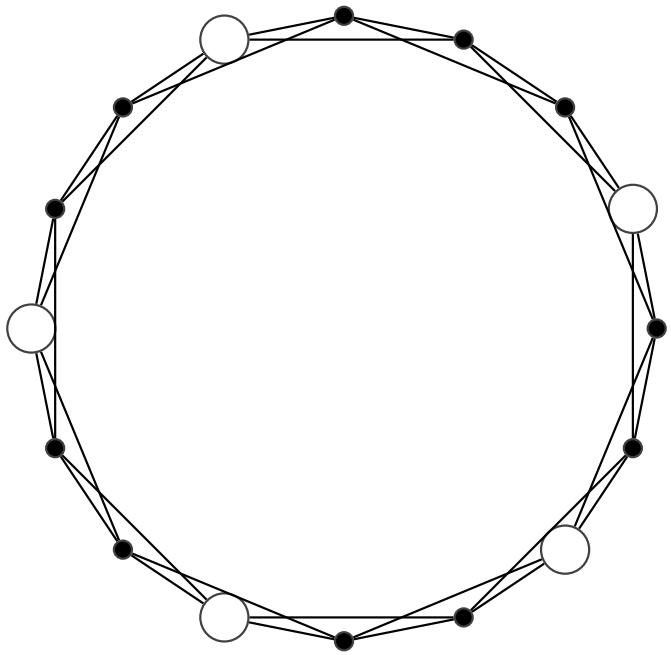
External case

A cycle is externally resolvable iff there is an alternating path as above.

Warning: finding such a path is not easy! [Minty 1980]

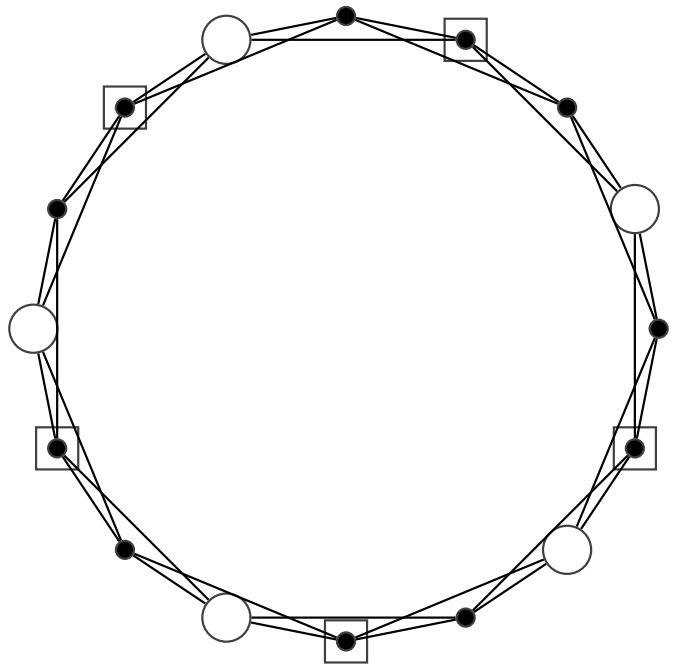
Internal case
Example

$V = \{1, \ldots, nk + 1\}$
edges go $\pm 1, \ \ldots, \ \pm(k-1)$
$n = 5, k = 3$



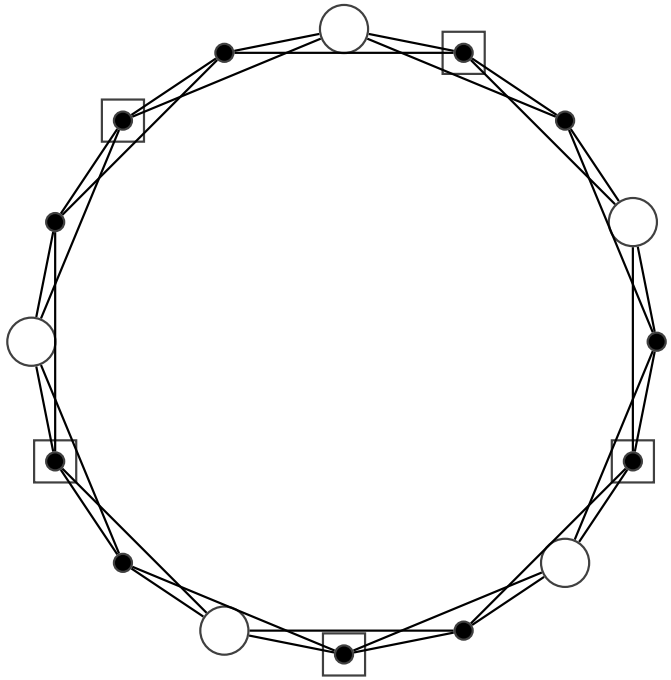
Internal case
Example

$V = \{1, \ldots, nk + 1\}$
edges go $\pm 1, \ \ldots, \ \pm(k-1)$
$n = 5, k = 3$



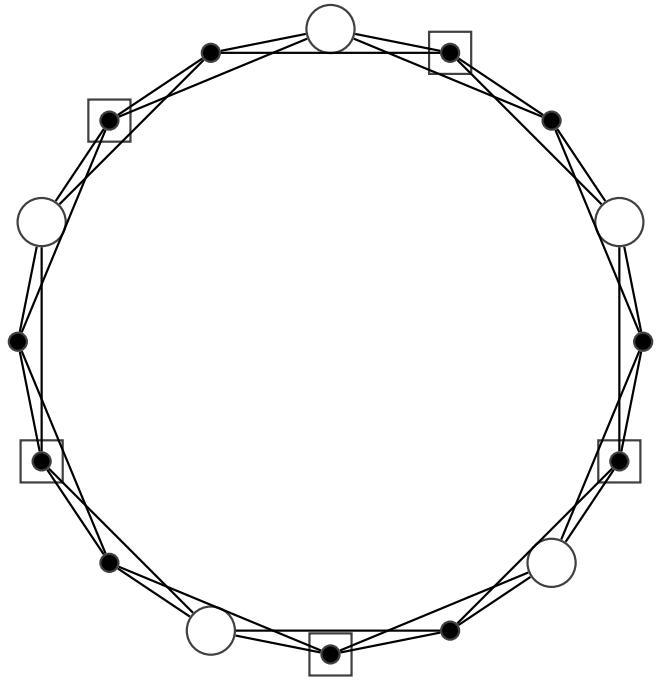
Internal case
Example

$V = \{1, \ldots, nk + 1\}$
edges go $\pm 1, \ \ldots, \ \pm(k-1)$
$n = 5, k = 3$



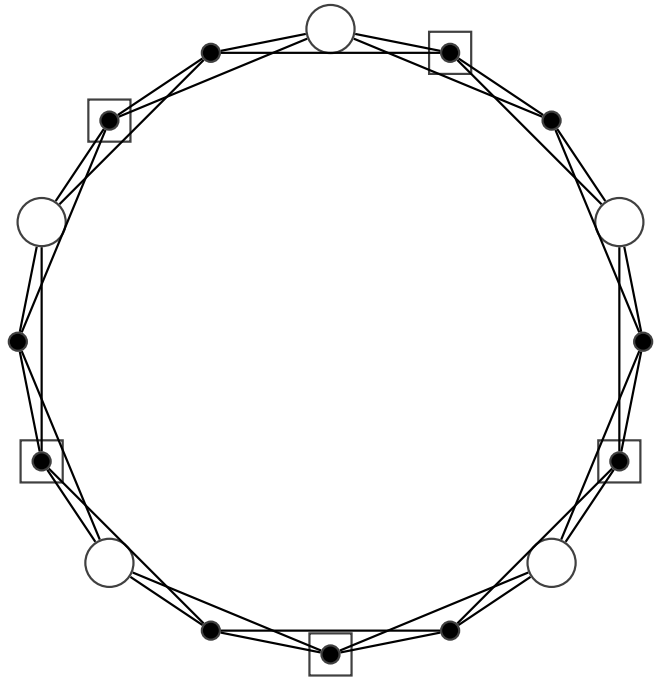
Internal case
Example

$V = \{1, \ldots, nk + 1\}$
edges go $\pm 1, \ \ldots, \ \pm(k-1)$
$n = 5, k = 3$



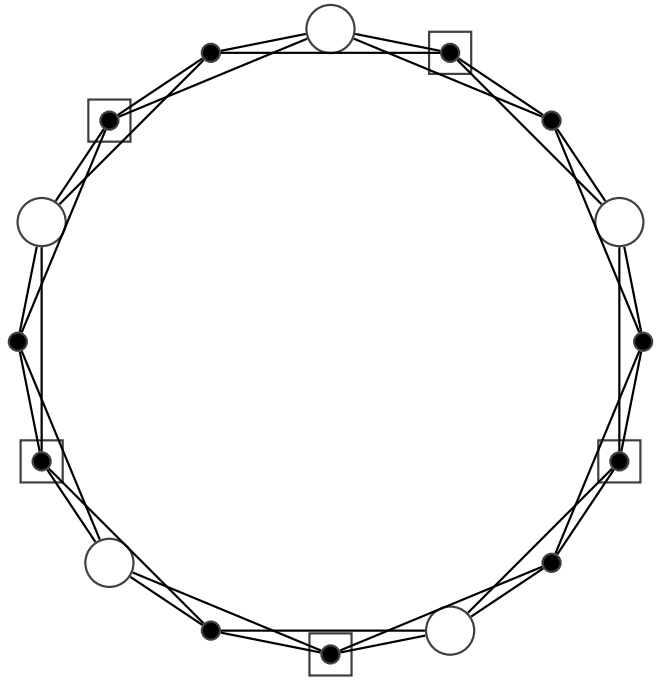
Internal case
Example

$V = \{1, \ldots, nk + 1\}$
edges go $\pm 1, \ \ldots, \ \pm(k-1)$
$n = 5, k = 3$



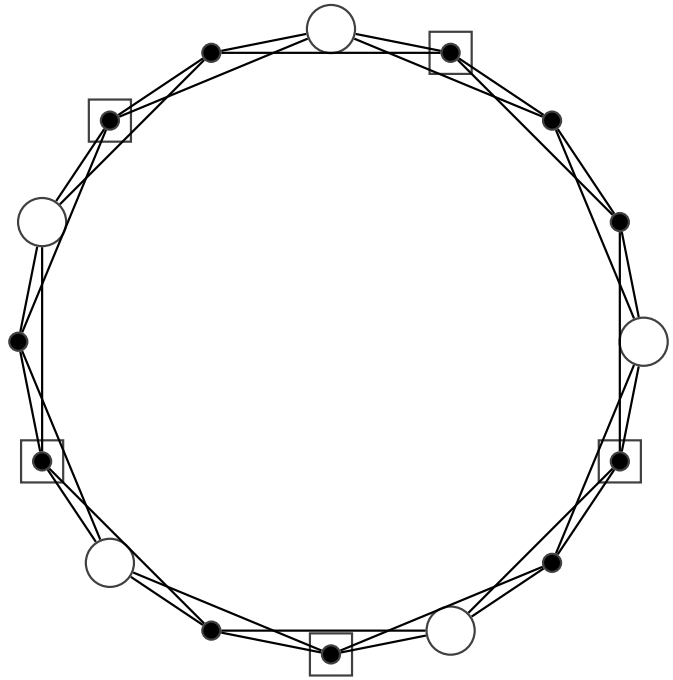
Internal case
Example

$V = \{1, \ldots, nk + 1\}$
edges go $\pm 1, \ \ldots, \ \pm(k-1)$
$n = 5, k = 3$



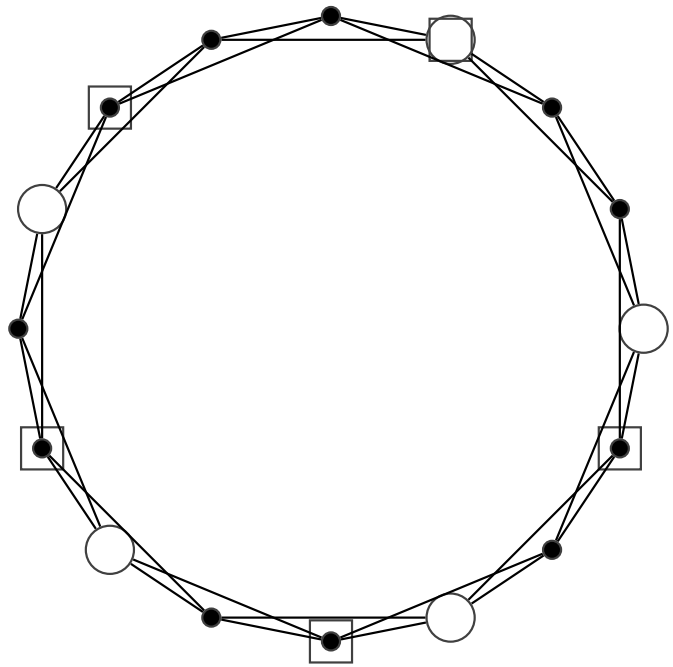
Internal case
Example

$V = \{1, \ldots, nk + 1\}$
edges go $\pm 1, \ \ldots, \ \pm(k-1)$
$n = 5, k = 3$



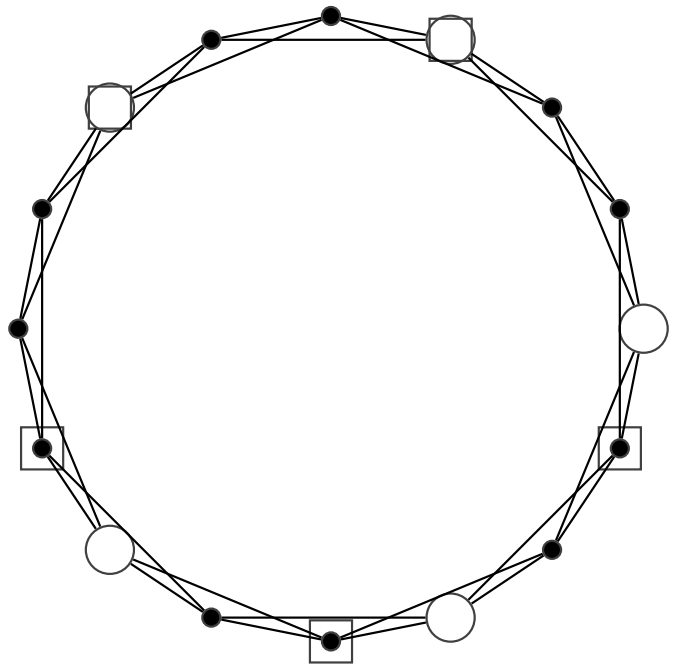
Internal case
Example

$V = \{1, \ldots, nk + 1\}$
edges go $\pm 1, \ \ldots, \ \pm(k-1)$
$n = 5, k = 3$



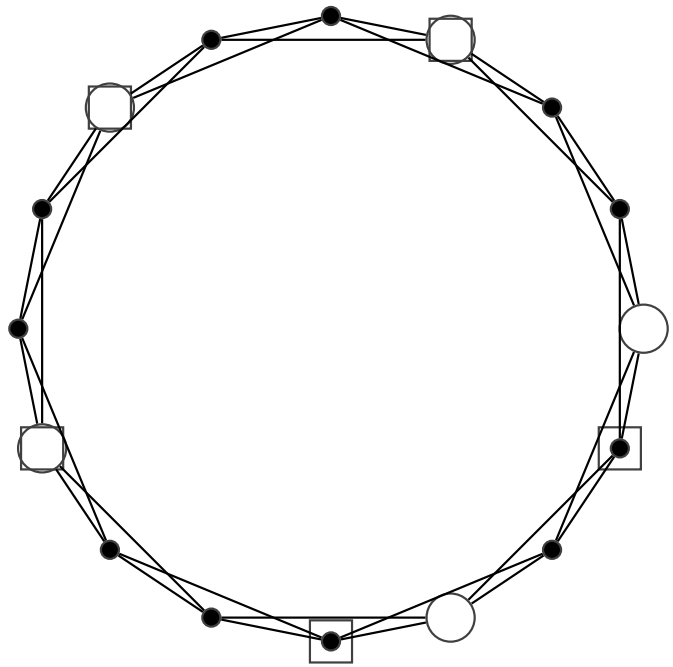
Internal case
Example

$V = \{1, \ldots, nk + 1\}$
edges go $\pm 1, \ \ldots, \ \pm(k-1)$
$n = 5, k = 3$



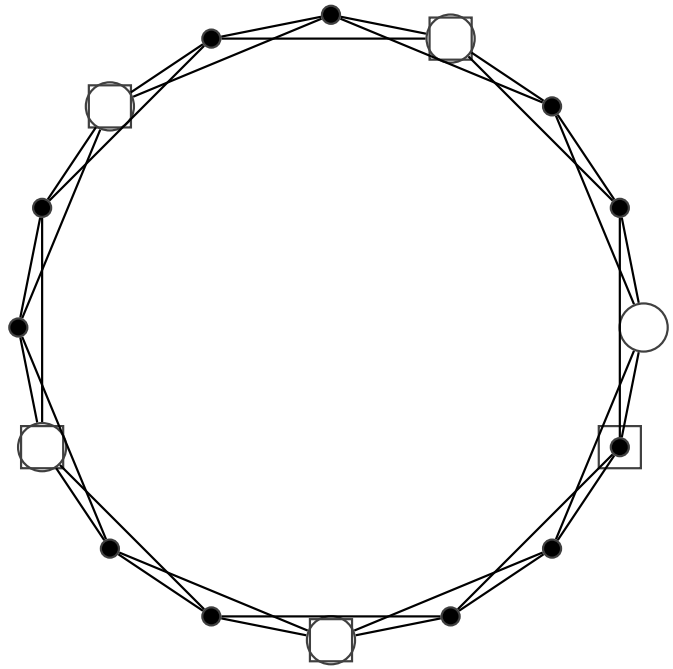
Internal case
Example

$V = \{1, \ldots, nk + 1\}$
edges go $\pm 1, \ \ldots, \ \pm(k-1)$
$n = 5, k = 3$



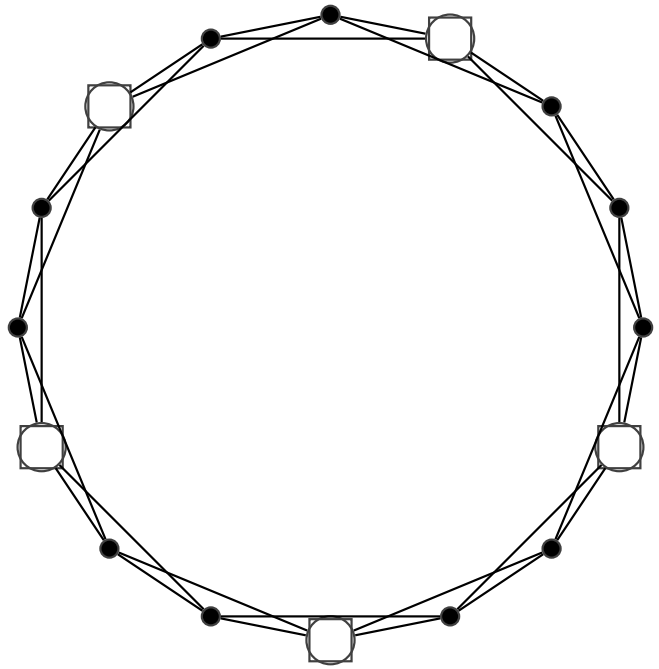
Internal case
Example

$V = \{1, \ldots, nk + 1\}$
edges go $\pm 1, \ \ldots, \ \pm(k-1)$
$n = 5, k = 3$

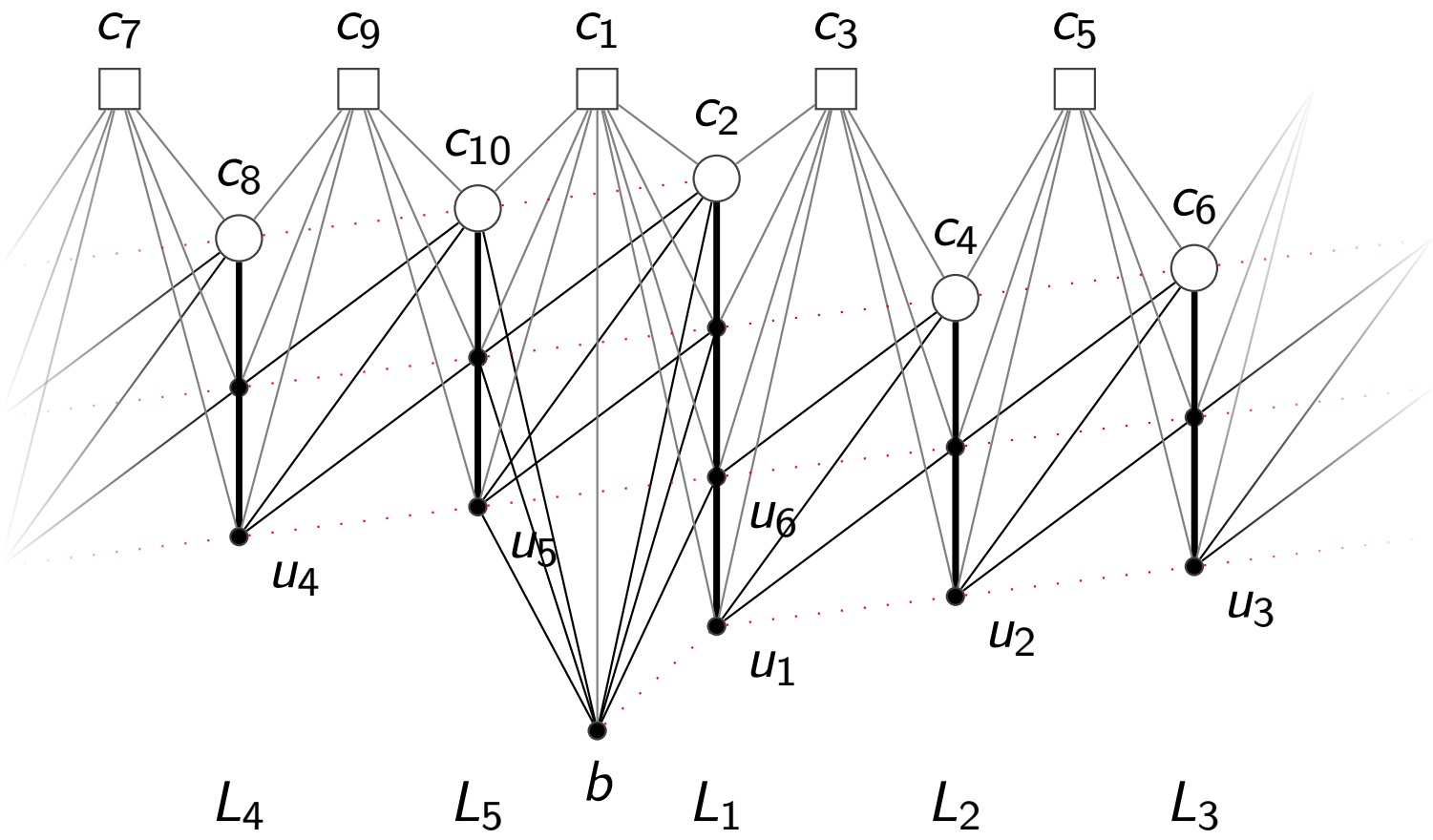


Internal case
Example

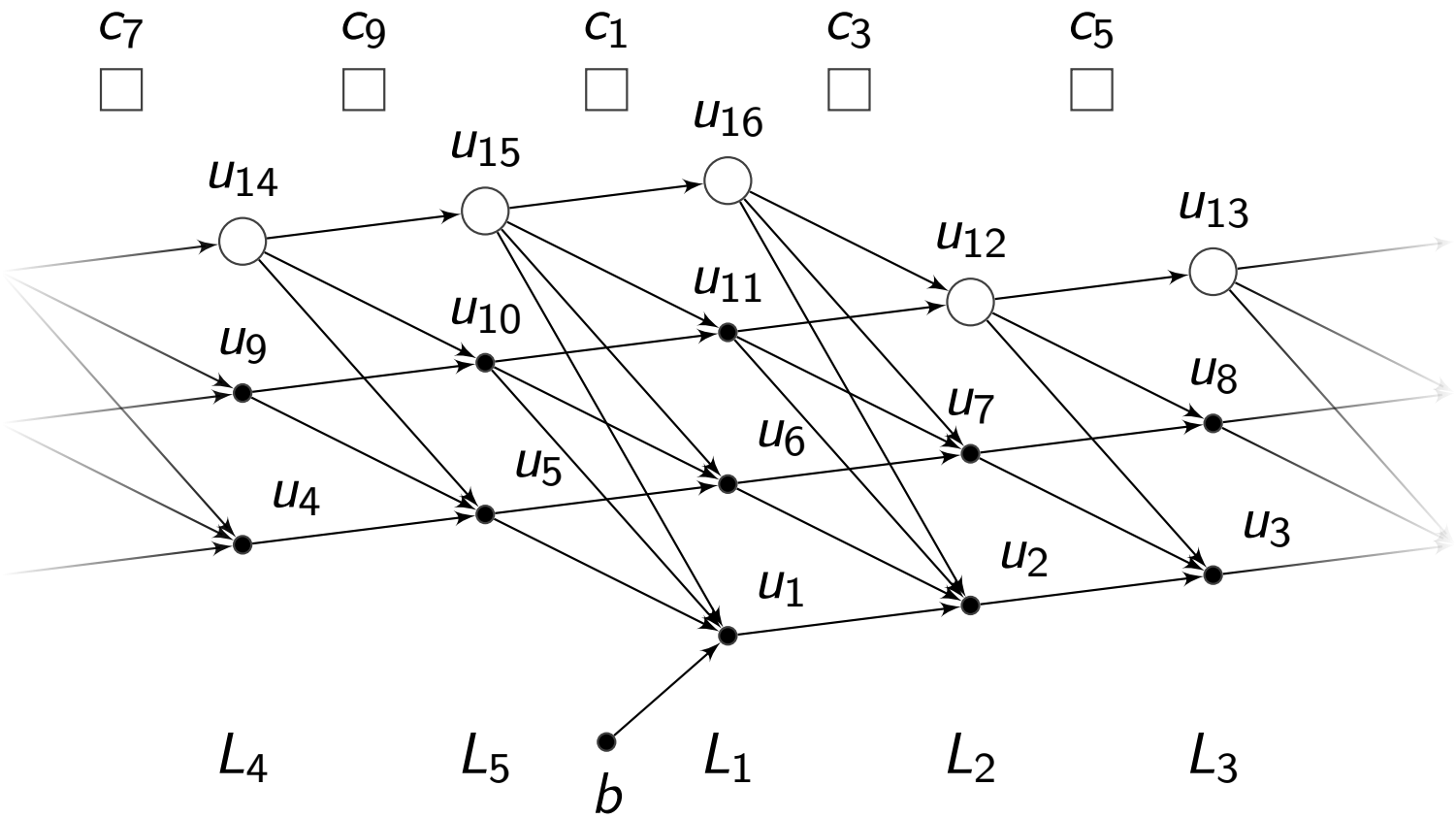
$V = \{1, \ldots, nk + 1\}$
edges go $\pm 1, \ \ldots, \ \pm(k-1)$
$n = 5, k = 3$



Internal case



Internal case



Thank you!
Questions?