An algorithmic study of switch graphs

Bastian Katz[1] Ignaz Rutter[1] Gerhard J. Woeginger[2]

[1] Algorithmics Group
Faculty of Informatics
Universität Karlsruhe (TH), KIT

[2] Department of Mathematics and
Computer Science
TU Eindhoven

June 26, WG 2009

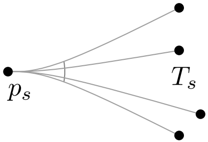
What is a switch?

Definition (Switch)

A *switch* on a vertex set V is a pair (p_s, T_s).

- $p_s \in V$ is the *pivot* vertex
- $\emptyset \neq T_s \subseteq V$ is the set of *target* vertices

We may enable exactly one of the edges $\{\{p_s, t_s\} \mid t_s \in T_s\}$



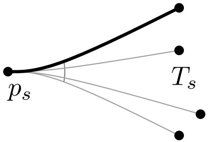
What is a switch?

Definition (Switch)

A *switch* on a vertex set V is a pair (p_s, T_s).

- $p_s \in V$ is the *pivot* vertex
- $\emptyset \neq T_s \subseteq V$ is the set of *target* vertices

We may enable exactly one of the edges $\{\{p_s, t_s\} \mid t_s \in T_s\}$



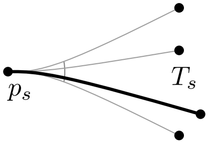
What is a switch?

Definition (Switch)

A *switch* on a vertex set V is a pair (p_s, T_s).

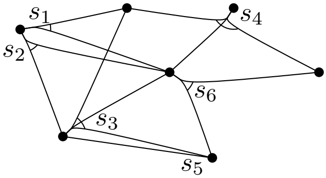
- $p_s \in V$ is the *pivot* vertex
- $\emptyset \neq T_s \subseteq V$ is the set of *target* vertices

We may enable exactly one of the edges $\{\{p_s, t_s\} \mid t_s \in T_s\}$



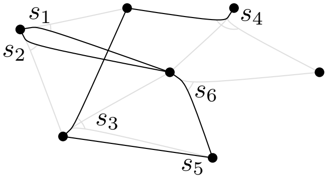
What is a switch graph?

Switch graph $G = (V, S)$: S set of switches on vertex set V.



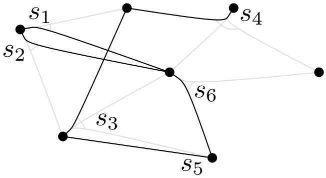
What is a switch graph?

Switch graph $G = (V, S)$: S set of switches on vertex set V.



What is a switch graph?

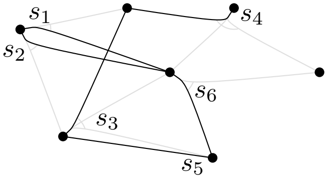
Switch graph $G = (V, S)$: S set of switches on vertex set V.



A *configuration* is a mapping $c : S \to V$ s.t. $c(s) \in T_s \quad \forall s \in S$.

What is a switch graph?

Switch graph $G = (V, S)$: S set of switches on vertex set V.

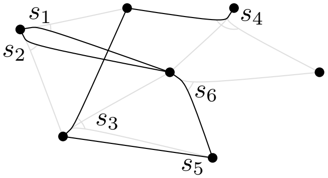


A *configuration* is a mapping $c : S \to V$ s.t. $c(s) \in T_s \quad \forall s \in S$.

- c picks exactly one edge $e_c(s) := \{p_s, c(s)\}$ for every switch s.
- Yields multigraph $G_c = (V, E_c)$ with $E_c = \{e_c(s) : s \in S\}$.

What is a switch graph?

Switch graph $G = (V, S)$: S set of switches on vertex set V.



A *configuration* is a mapping $c : S \to V$ s.t. $c(s) \in T_s \quad \forall s \in S$.

- c picks exactly one edge $e_c(s) := \{p_s, c(s)\}$ for every switch s.
- Yields multigraph $G_c = (V, E_c)$ with $E_c = \{e_c(s) : s \in S\}$.

A switch graph represents a population of graphs, a configuration describes a concrete member of that family.

Problems on switch graphs

Every graph property $\mathcal{P}$ yields problem on switch graphs:

Problem Switch-$\mathcal{P}$

Input: Switch graph $G = (V, S)$ with $n := |V|$, $m := |S|$ and fan-out $k := \max_{s \in S} |T_s|$.
Question: Does G have a configuration c such that G_c has property $\mathcal{P}$?

- Bipartiteness
- Global Connectivity
- Local Connectivity (given vertices a, b are connected)
- Biconnectivity
- Planarity
- Eulerianness
- Acyclicity

Related work

Cook introduced switch graphs to study certain stable configurations of Rule 101: [Cook '03]

- $O(n^2)$ algorithm detecting a configuration containing a cycle.
- *binary* switches, only vertices with degree 3 may have a switch

Groote and Ploeger study complexity of certain graph properties on switch graphs with binary switches: [Groote, Ploeger '08]

- mostly *forward directed* switch graphs (pivot $\rightarrow$ target)
- $O(n)$ algorithm for finding configuration with a directed a–b path, for given a, b.
- Several open questions, our work is heavily inspired by these.

Reinhardt gives $O(n^4)$ algorithm for finding a configuration with an a–b path in a model similar to Cook's. [Reinhardt '09]

Our results.

The following problems are NP-hard for switch graphs:

- SwitchBipartite
- SwitchTriangleFree
- SwitchPlanar
- SwitchBiconnected
- SwitchEulerian (directed and undirected)
- SwitchStronglyConnected
- SwitchMinCycles

We give efficient algorithms for the following problems:

- SwitchDirectedAcyclic: $O(n + km)$ time
- Global connectivity: $O(km + kn^2)$ time
- Local connectivity: $O(km + n\alpha(n))$ time

Our results.

The following problems are NP-hard for switch graphs:

- SwitchBipartite
- SwitchTriangleFree
- SwitchPlanar
- SwitchBiconnected
- SwitchEulerian (directed and undirected)
- SwitchStronglyConnected
- SwitchMinCycles

We give efficient algorithms for the following problems:

- SwitchDirectedAcyclic: $O(n + km)$ time
- Global connectivity: $O(km + kn^2)$ time
- Local connectivity: $O(km + n\alpha(n))$ time

SwitchPlanar

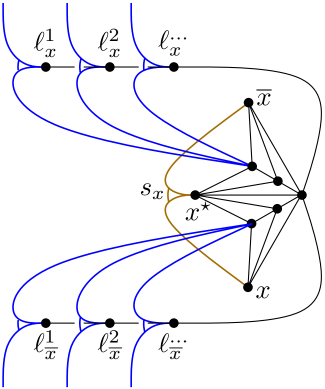
Theorem

Given a switch graph $G = (V, S)$ it is NP-hard to decide whether there exists a configuration c such that G_c is planar.

Proof: Gadget proof, reduction from a variant of Planar 3Sat.

Hardness of SWITCHPLANAR

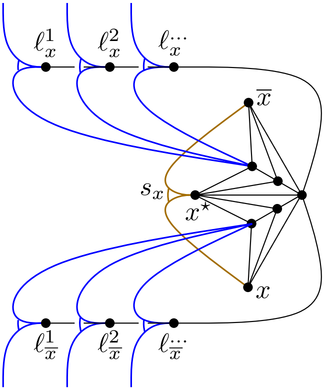
Gadget proof:



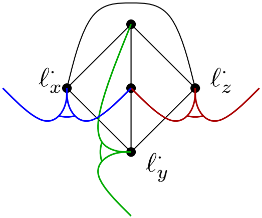
Variable

Hardness of SwitchPlanar

Gadget proof:



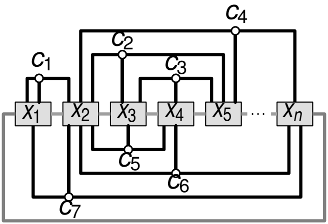
Variable



Clause

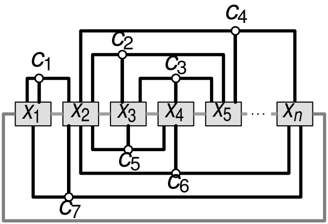
Monotone Planar 3SAT

Planar 3SAT:



Monotone Planar 3SAT

Planar 3SAT:

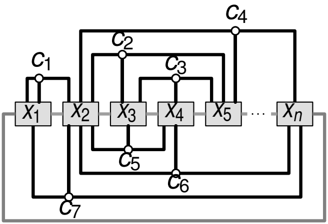


Monotone:

- literals of each clause are either all positive or all negative
- clauses with positive literals drawn above x-axis, negative clauses below

Monotone Planar 3Sat

Planar 3Sat:



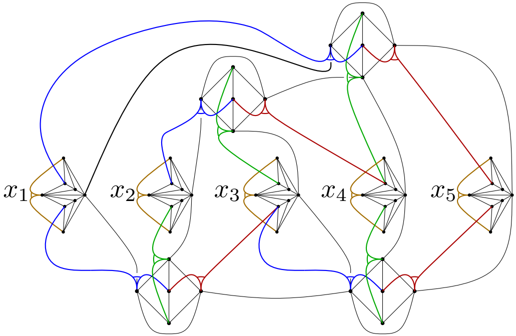
Monotone:

- literals of each clause are either all positive or all negative
- clauses with positive literals drawn above x-axis, negative clauses below

Theorem (de Berg, Koshravi, submitted)

Monotone Planar 3Sat *is NP-hard.*

SwitchPlanar *(example)*



$$(x_1 \vee x_4 \vee x_5) \wedge (x_2 \vee x_3 \vee x_4) \wedge$$
$$(\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3) \wedge (\overline{x}_3 \vee \overline{x}_4 \vee \overline{x}_5)$$

Global Connectivity

Definition (SwitchConnect)

Given a switch graph $G = (V, S)$. Is there a configuration c of the switches such that G_c is connected?

A polynomial-time algorithm

Switch graph $G = (V, S)$,
$E :=$ all multi-edges on V possibly resulting from S.

A polynomial-time algorithm

Switch graph $G = (V, S)$,
$E :=$ all multi-edges on V possibly resulting from S.

Connectivity via matroid intersection

Define two matroids $(E, \mathcal{I}_1)$, $(E, \mathcal{I}_2)$:

- $E' \in \mathcal{I}_1 \Leftrightarrow E'$ is acyclic
- $E' \in \mathcal{I}_2 \Leftrightarrow E'$ contains ≤ 1 edge per switch

A polynomial-time algorithm

Switch graph $G = (V, S)$,
$E :=$ all multi-edges on V possibly resulting from S.

Connectivity via matroid intersection

Define two matroids $(E, \mathcal{I}_1)$, $(E, \mathcal{I}_2)$:

- $E' \in \mathcal{I}_1 \Leftrightarrow E'$ is acyclic
- $E' \in \mathcal{I}_2 \Leftrightarrow E'$ contains ≤ 1 edge per switch

G has connected configuration $\Leftrightarrow$ ex. $E' \in \mathcal{I}_1 \cap \mathcal{I}_2$ with $|E'| = n - 1$.

A polynomial-time algorithm

Switch graph $G = (V, S)$,
$E :=$ all multi-edges on V possibly resulting from S.

Connectivity via matroid intersection

Define two matroids $(E, \mathcal{I}_1)$, $(E, \mathcal{I}_2)$:

- $E' \in \mathcal{I}_1 \Leftrightarrow E'$ is acyclic
- $E' \in \mathcal{I}_2 \Leftrightarrow E'$ contains ≤ 1 edge per switch

G has connected configuration $\Leftrightarrow$ ex. $E' \in \mathcal{I}_1 \cap \mathcal{I}_2$ with $|E'| = n - 1$.

$\rightsquigarrow$ polynomial algorithm $\approx O(n^4)$

A new matroid

A matroid of switches

Define new matroid $(S, \mathcal{I}_3)$:
$S' \in \mathcal{I}_3 \Leftrightarrow$ ex. configuration c with $G_c(S')$ acyclic.

A new matroid

A matroid of switches

Define new matroid $(S, \mathcal{I}_3)$:
$S' \in \mathcal{I}_3 \Leftrightarrow$ ex. configuration c with $G_c(S')$ acyclic.

G Yes-Instance of SwitchConnect $\Leftrightarrow$ $(S, \mathcal{I}_3)$ has independent set of size $n-1$

A new matroid

A matroid of switches

Define new matroid $(S, \mathcal{I}_3)$:
$S' \in \mathcal{I}_3 \Leftrightarrow$ ex. configuration c with $G_c(S')$ acyclic.

G YES-Instance of SWITCHCONNECT $\Leftrightarrow$ $(S, \mathcal{I}_3)$ has independent set of size $n-1$

To solve SWITCHCONNECT we need to steps:

1. Show that $(S, \mathcal{I}_3)$ is a matroid.
2. Give a fast independence test for $(S, \mathcal{I}_3)$.

A new matroid

Theorem

$(S, \mathcal{I}_3)$ is a matroid.

Proof:

- $\emptyset \in \mathcal{I}_3$.
- $A \in \mathcal{I}_3$, $B \subseteq A \Rightarrow B \in \mathcal{I}_3$.

Remains to show: $A, B \in \mathcal{I}_3$, $|A| < |B| \Rightarrow \exists x \in B \setminus A : A \cup \{x\} \in \mathcal{I}_3$.

A new matroid

Theorem

$(S, \mathcal{I}_3)$ is a matroid.

Proof:

- $\emptyset \in \mathcal{I}_3$.
- $A \in \mathcal{I}_3$, $B \subseteq A \Rightarrow B \in \mathcal{I}_3$.

Remains to show: $A, B \in \mathcal{I}_3$, $|A| < |B| \Rightarrow \exists x \in B \setminus A : A \cup \{x\} \in \mathcal{I}_3$.

- Let c_A, c_B acyclic configurations of A, B with max. similarity, i.e., $|\{s \in A \cap B \mid c_A(s) = c_B(s)\}|$ maximum.
  Let E_A, E_B be the corresponding acyclic edge sets.
- $|E_A| < |E_B| \Rightarrow$ ex. $e_b \in E_B \setminus E_A$ with $E_A \cup \{e_b\}$ acyclic. Let $x \in B$ the corresponding switch.
- Now $A \cup \{x\}$ is independent and $x \notin A$, otherwise could make c_A, c_B more similar.

Charaterization and Test of Independence

Lemma

Let $S' \subseteq S$. *Then* $S' \in \mathcal{I}_3 \quad \Rightarrow \quad \forall S'' \subseteq S' \colon |S''| < |V(S'')|$.

Charaterization and Test of Independence

Lemma

Let $S' \subseteq S$. *Then* $S' \in \mathcal{I}_3 \quad \Rightarrow \quad \forall S'' \subseteq S' \colon |S''| < |V(S'')|$.

Charaterization and Test of Independence

Lemma

Let $S' \subseteq S$. *Then* $S' \in \mathcal{I}_3 \iff \forall S'' \subseteq S' \colon |S''| < |V(S'')|$.

Charaterization and Test of Independence

Lemma

Let $S' \subseteq S$. *Then* $S' \in \mathcal{I}_3 \iff \forall S'' \subseteq S' \colon |S''| < |V(S'')|$.

Proof: "$\Rightarrow$": $\surd$

Charaterization and Test of Independence

Lemma

Let $S' \subseteq S$. Then $S' \in \mathcal{I}_3 \iff \forall S'' \subseteq S'\colon |S''| < |V(S'')|$.

Proof: "$\Rightarrow$": $\surd$

"$\Leftarrow$": Let S' be minimal dependent set of switches.

$\Rightarrow \exists$ configuration of S with exactly one cycle

Charaterization and Test of Independence

Lemma

Let $S' \subseteq S$. Then $S' \in \mathcal{I}_3 \iff \forall S'' \subseteq S'\colon |S''| < |V(S'')|$.

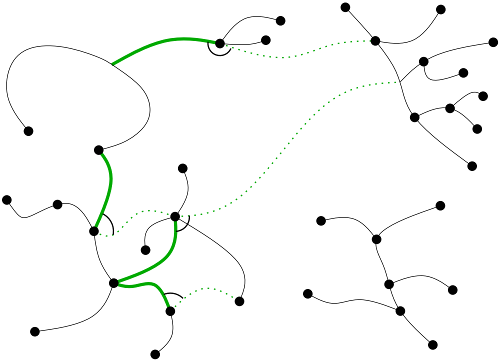
Proof: "$\Rightarrow$": $\surd$

"$\Leftarrow$": Let S' be minimal dependent set of switches.

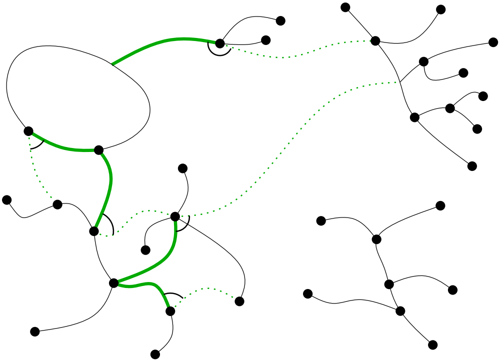
$\Rightarrow \exists$ configuration of S with exactly one cycle

We show that S' contains a subset with few vertices.

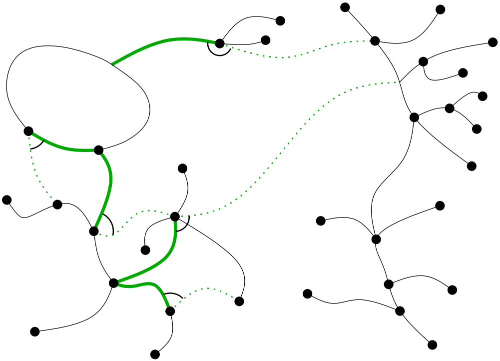
Connectivity



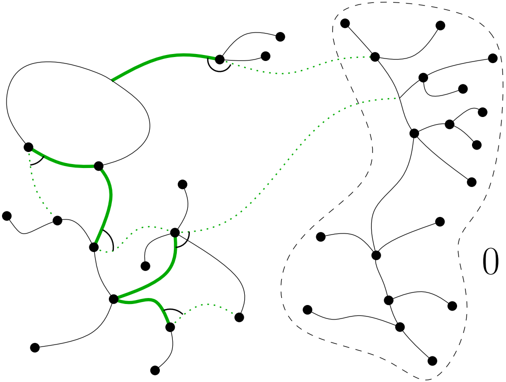
Connectivity



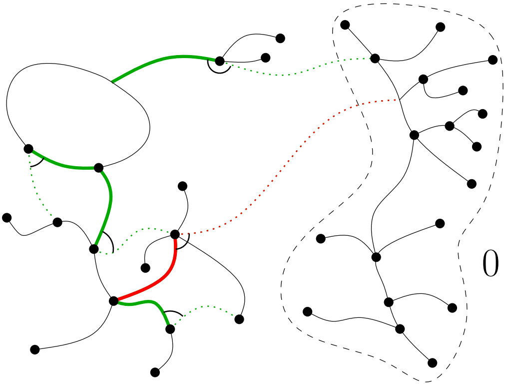
Connectivity



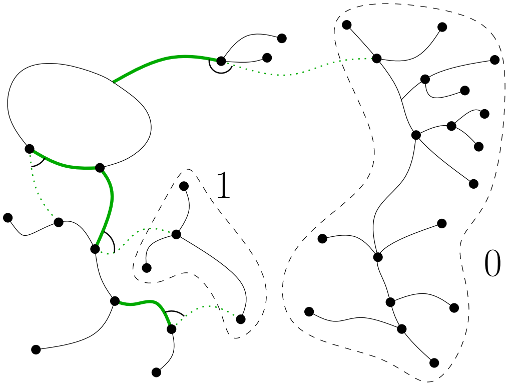
Connectivity



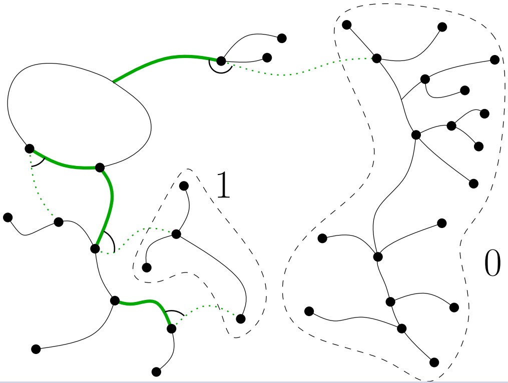
Connectivity



Connectivity



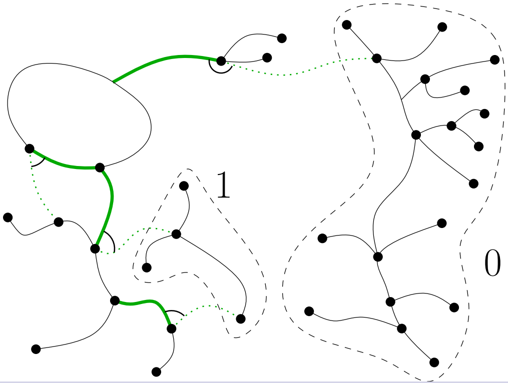
Connectivity



Continue picking switches until

1. !∃ selectable switch

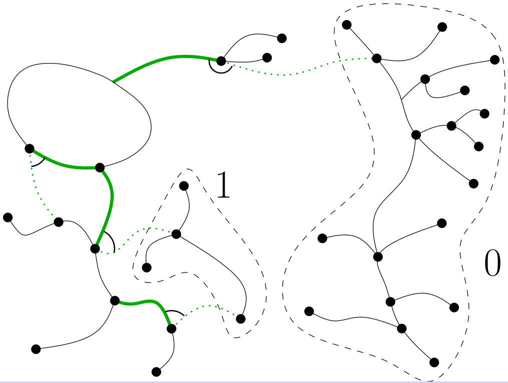
Connectivity



Continue picking switches until

1. !∃ selectable switch ⇒ set of vertices with too many switches

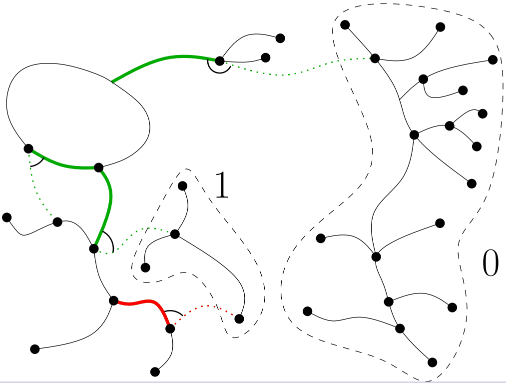
Connectivity



Continue picking switches until

1. $!\exists$ selectable switch $\Rightarrow$ set of vertices with too many switches
2. After removal of $s_1, \ldots, s_k$ remaining graph is acyclic

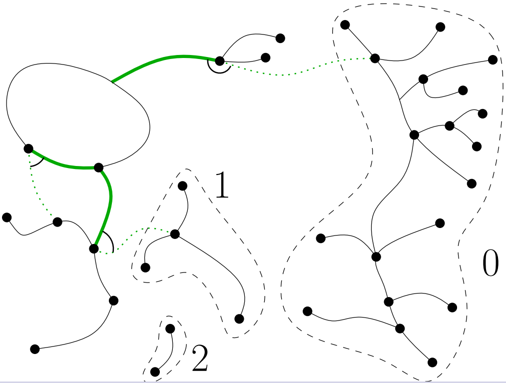
Connectivity



Continue picking switches until

1. !∃ selectable switch ⇒ set of vertices with too many switches
2. After removal of $s_1, \ldots, s_k$ remaining graph is acyclic

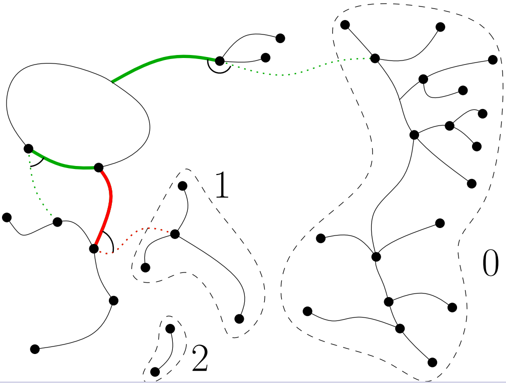
Connectivity



Continue picking switches until

1. !∃ selectable switch ⇒ set of vertices with too many switches
2. After removal of $s_1, \ldots, s_k$ remaining graph is acyclic

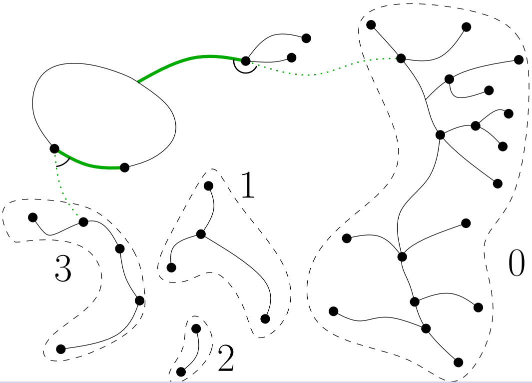
Connectivity



Continue picking switches until

1. !∃ selectable switch ⇒ set of vertices with too many switches
2. After removal of $s_1, \ldots, s_k$ remaining graph is acyclic

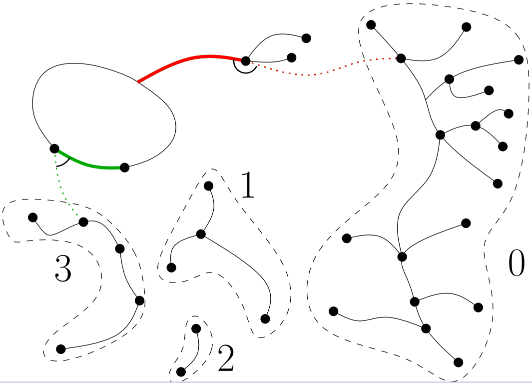
Connectivity



Continue picking switches until

1. $!\exists$ selectable switch $\Rightarrow$ set of vertices with too many switches
2. After removal of $s_1, \ldots, s_k$ remaining graph is acyclic

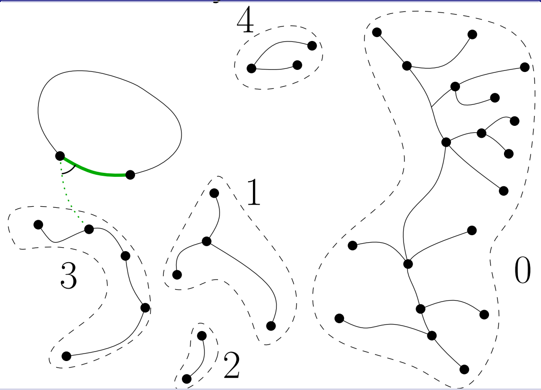
Connectivity



Continue picking switches until

1. $!\exists$ selectable switch $\Rightarrow$ set of vertices with too many switches
2. After removal of $s_1, \ldots, s_k$ remaining graph is acyclic

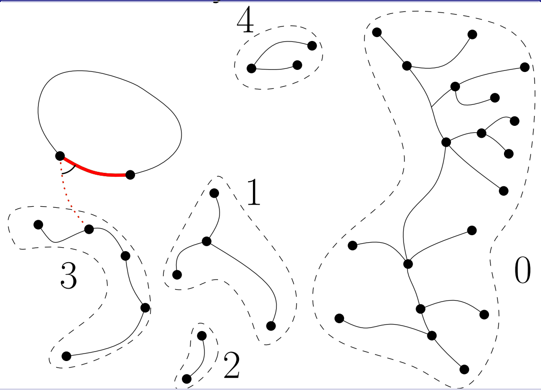
Connectivity



Continue picking switches until

1. $!\exists$ selectable switch $\Rightarrow$ set of vertices with too many switches
2. After removal of $s_1, \ldots, s_k$ remaining graph is acyclic

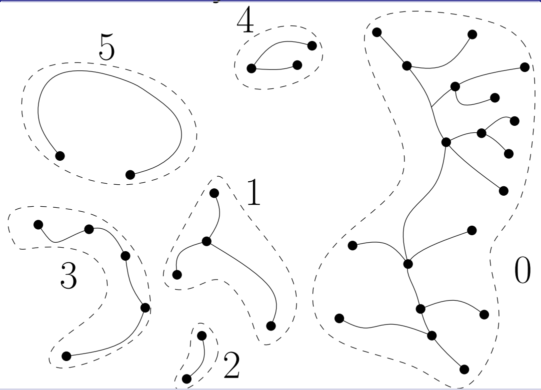
Connectivity



Continue picking switches until

1. $!\exists$ selectable switch $\Rightarrow$ set of vertices with too many switches
2. After removal of $s_1, \ldots, s_k$ remaining graph is acyclic

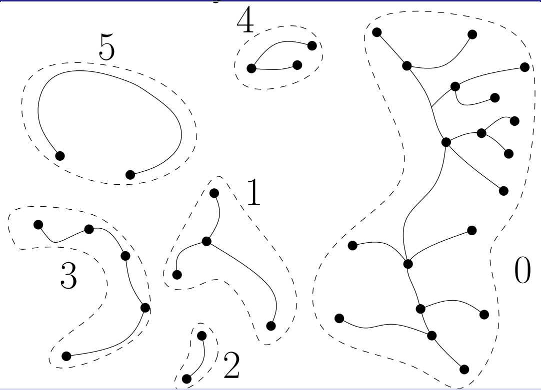
Connectivity



Continue picking switches until

1. !∃ selectable switch ⇒ set of vertices with too many switches
2. After removal of $s_1, \ldots, s_k$ remaining graph is acyclic

Connectivity

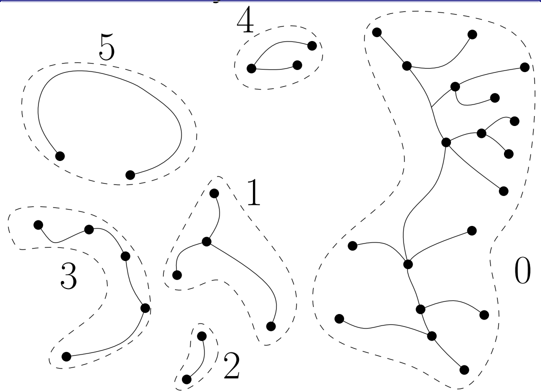


V_i := nodes with label i

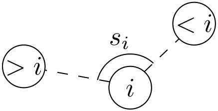
Continue picking switches until

1. !∃ selectable switch ⇒ set of vertices with too many switches
2. After removal of $s_1, \ldots, s_k$ remaining graph is acyclic

Connectivity



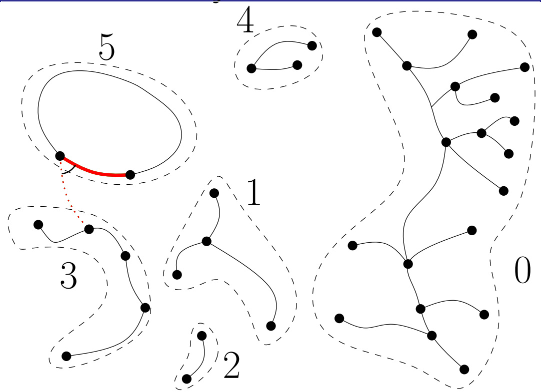
V_i := nodes with label i



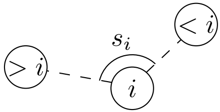
Switch s_i, $i = 1, \ldots, k-1$

- s_i connects V_i to V_j with $j > i$
- unused vertex of s_i is in V_ℓ with $\ell < i$

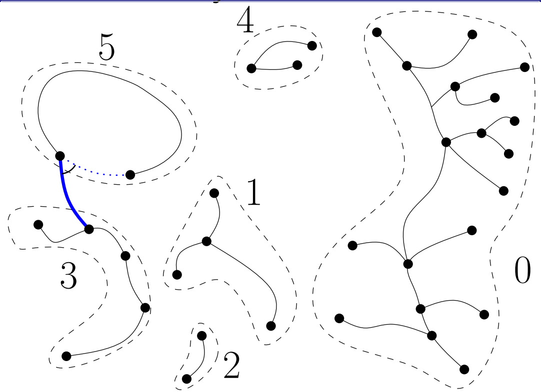
Connectivity



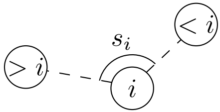
V_i := nodes with label i



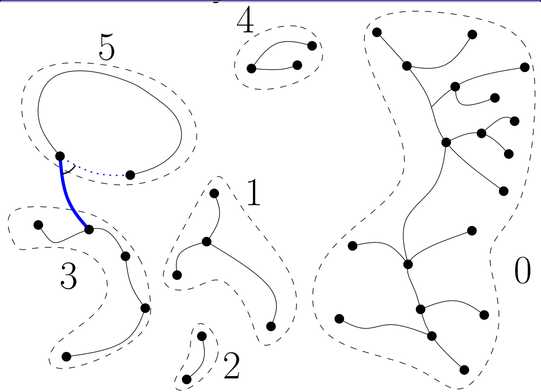
Connectivity



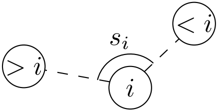
V_i := nodes with label i



Connectivity



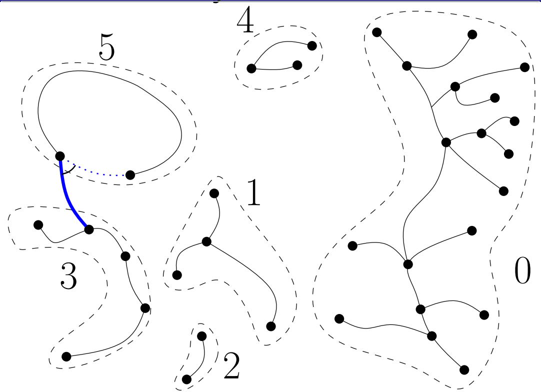
V_i := nodes with label i



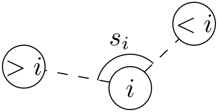
After switch s_i is inserted:

- ≫ $V_k, \ldots, V_i$ are acyclically connected
- ≫ $\exists_1\, j < i$ s.t. V_j is no component

Connectivity

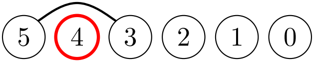


V_i := nodes with label i

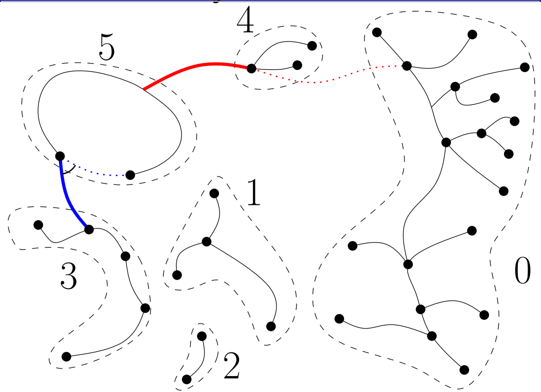


After switch s_i is inserted:

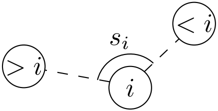
- $V_k, \ldots, V_i$ are acyclically connected
- $\exists_1\, j < i$ s.t. V_j is no component



Connectivity

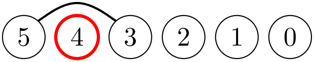


V_i := nodes with label i

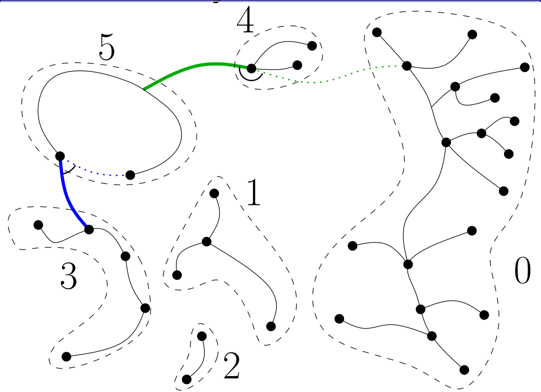


After switch s_i is inserted:

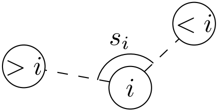
- $V_k, \dots, V_i$ are acyclically connected
- $\exists_1\, j < i$ s.t. V_j is no component



Connectivity

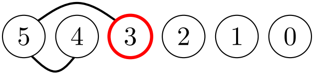


V_i := nodes with label i

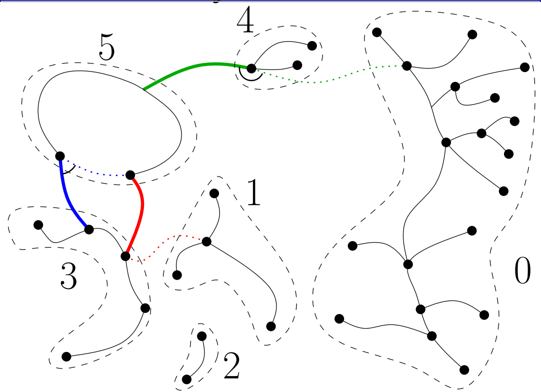


After switch s_i is inserted:

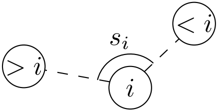
- $V_k, \ldots, V_i$ are acyclically connected
- $\exists_1\, j < i$ s.t. V_j is no component



Connectivity

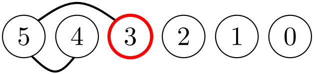


V_i := nodes with label i

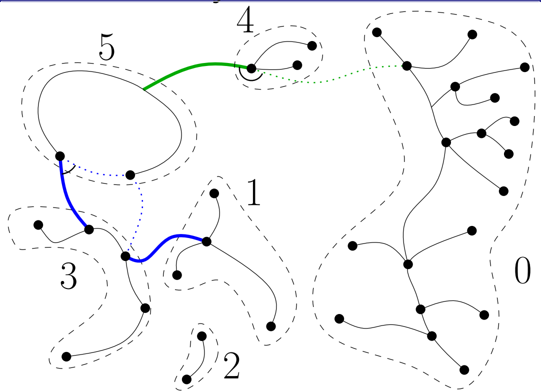


After switch s_i is inserted:

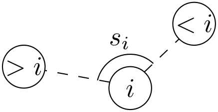
- $\gg$ $V_k, \ldots, V_i$ are acyclically connected
- $\gg$ $\exists_1\, j < i$ s.t. V_j is no component



Connectivity

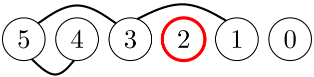


V_i := nodes with label i

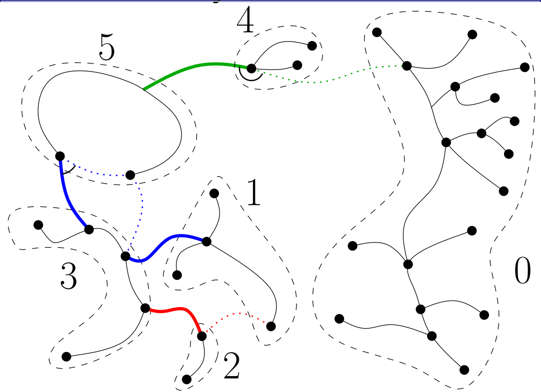


After switch s_i is inserted:

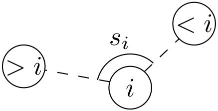
- $V_k, \ldots, V_i$ are acyclically connected
- $\exists_1\, j < i$ s.t. V_j is no component



Connectivity

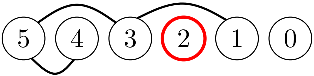


V_i := nodes with label i

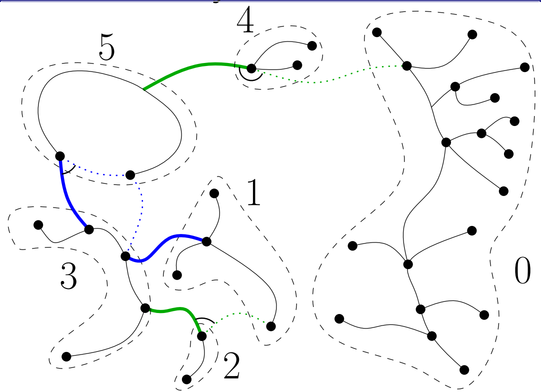


After switch s_i is inserted:

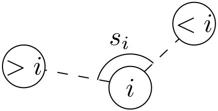
- $V_k, \ldots, V_i$ are acyclically connected
- $\exists_1\, j < i$ s.t. V_j is no component



Connectivity

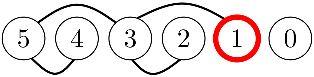


V_i := nodes with label i

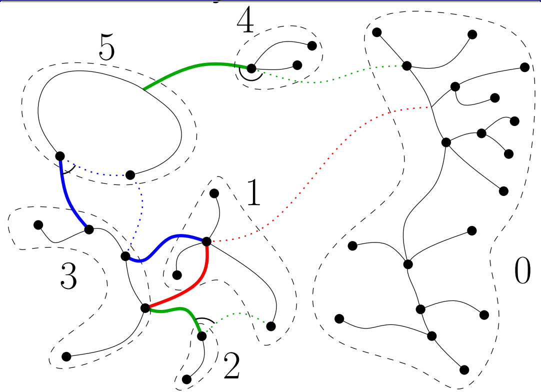


After switch s_i is inserted:

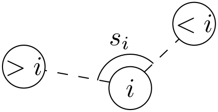
- $V_k, \ldots, V_i$ are acyclically connected
- $\exists_1\, j < i$ s.t. V_j is no component



Connectivity

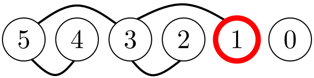


V_i := nodes with label i

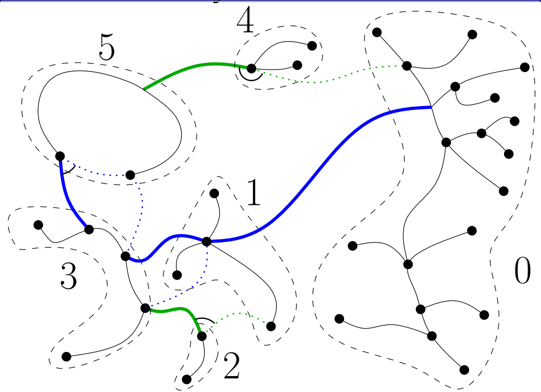


After switch s_i is inserted:

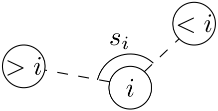
- $V_k, \ldots, V_i$ are acyclically connected
- $\exists_1\, j < i$ s.t. V_j is no component



Connectivity

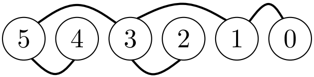


V_i := nodes with label i



After switch s_i is inserted:

- $V_k, \ldots, V_i$ are acyclically connected
- $\exists_1\ j < i$ s.t. V_j is no component



Testing independence

Lemma (Independence Test)

Given independent set of switches S' with acyclic configuration and an additional switch s we can compute in $O(kn)$ time either

- $S'' \subseteq S' \cup \{s\}$ *with* $|S''| >= |V(S'')|$ *or*
- *acyclic configuration of* $S' \cup \{s\}$.

Testing independence

Lemma (Independence Test)

Given independent set of switches S' with acyclic configuration and an additional switch s we can compute in $O(kn)$ time either

- $S'' \subseteq S' \cup \{s\}$ *with* $|S''| >= |V(S'')|$ *or*
- *acyclic configuration of* $S' \cup \{s\}$.

Theorem

Given switch graph $G = (V, S)$ can compute in $O(m \cdot kn)$ time a maximal independent set of switches with an acyclic configuration.

Testing independence

Lemma (Independence Test)

Given independent set of switches S' with acyclic configuration and an additional switch s we can compute in $O(kn)$ time either

- $S'' \subseteq S' \cup \{s\}$ *with* $|S''| >= |V(S'')|$ *or*
- *acyclic configuration of* $S' \cup \{s\}$.

Theorem

Given switch graph $G = (V, S)$ can compute in $O(m \cdot kn)$ time a maximal independent set of switches with an acyclic configuration.

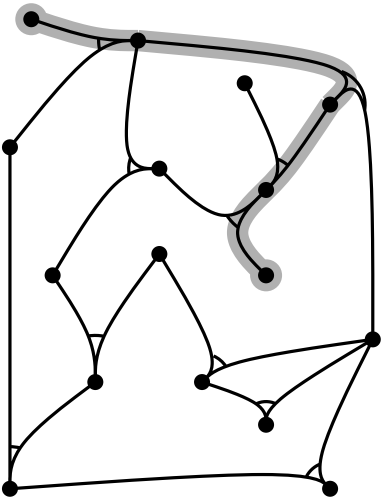
Running time can be improved to $O(km + kn^2)$.

Local Connectivity

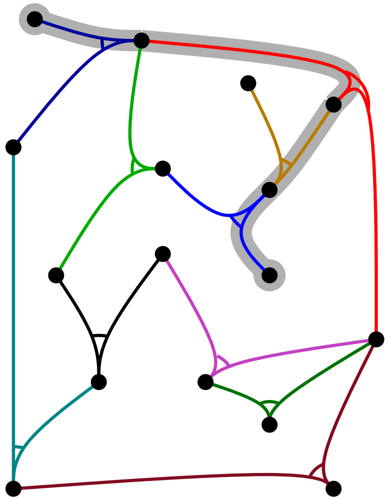
Definition (a–b-Connectivity)

Given a switch graph $G = (V, S)$ and $a, b \in V$.
Is there a configuration c such that a, b are connected in $G_c(S)$?

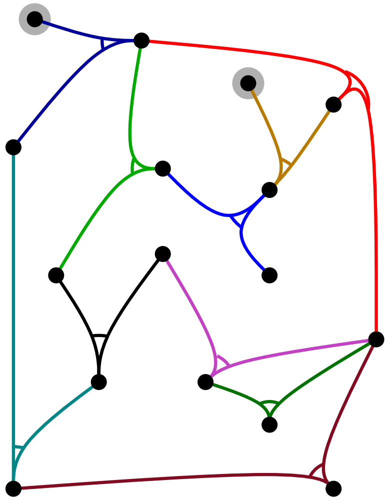
a-b-Connectivity (examples)



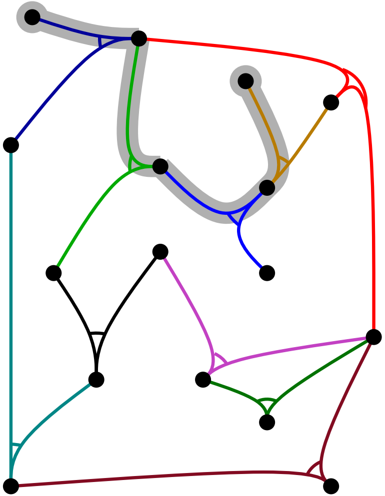
a-b-Connectivity (examples)



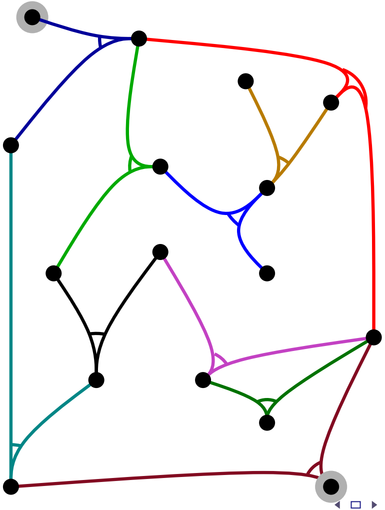
a-b-Connectivity (examples)



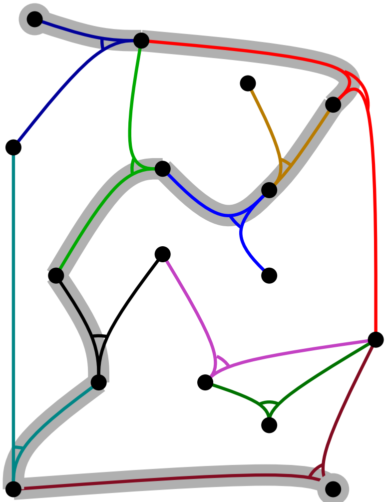
a-b-Connectivity (examples)



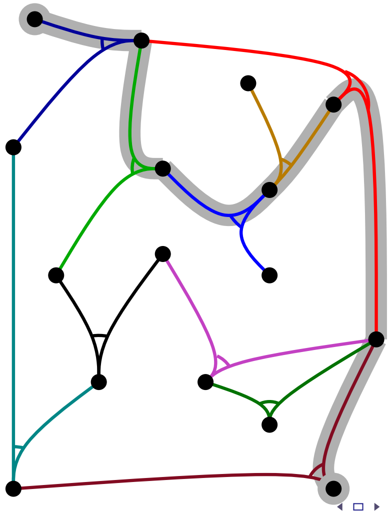
a-b-Connectivity (examples)



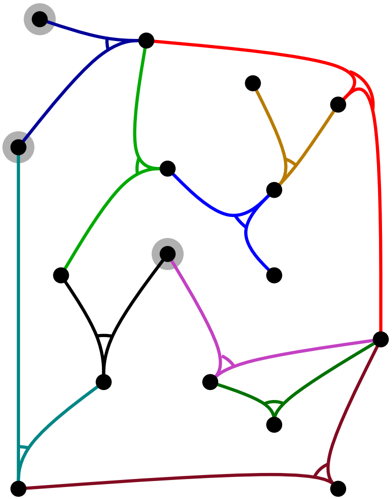
a-b-Connectivity (examples)



a-b-Connectivity (examples)

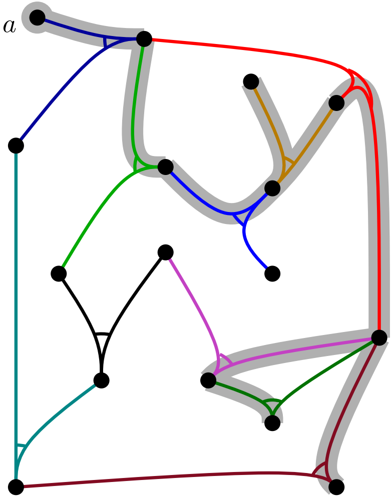


a-b-Connectivity (examples)



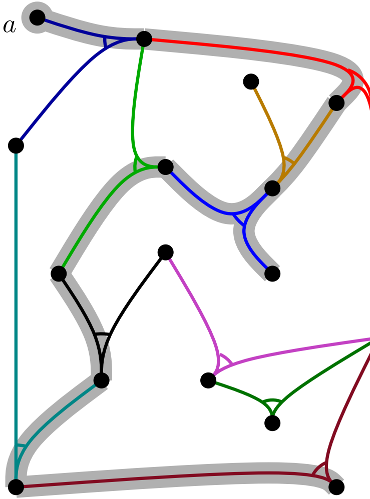
Searching along a tree

- Build search tree with root *a*
  - Edges used from target to pivot block the switch.
  - Edges from pivot to target may all be explored.



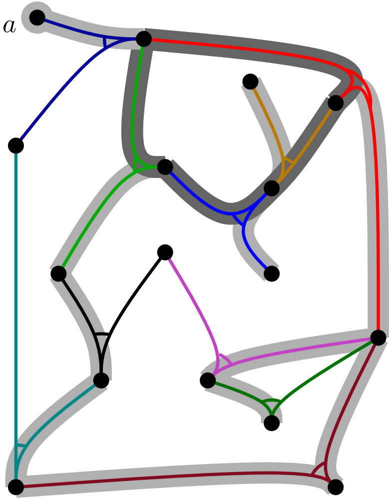
Searching along a tree

- Build search tree with root *a*
  - Edges used from target to pivot block the switch.
  - Edges from pivot to target may all be explored.



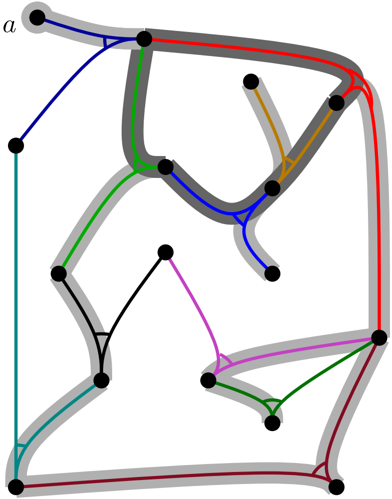
Searching along a tree

- Build search tree with root *a*
  - Edges used from target to pivot block the switch.
  - Edges from pivot to target may all be explored.
- Does not find all paths!
  - Direction of traversal is important for cycles of switches.
  - Idea: contract cycles.



Searching along a tree

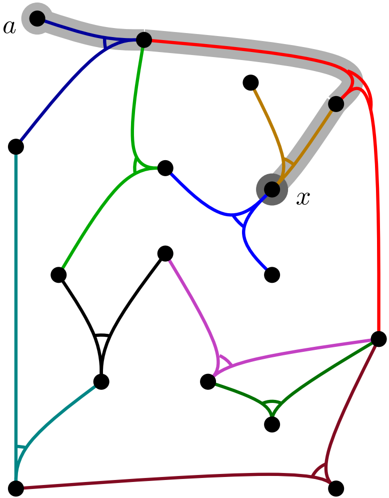
- Build search tree with root *a*
  - Edges used from target to pivot block the switch.
  - Edges from pivot to target may all be explored.
- Does not find all paths!
  - Direction of traversal is important for cycles of switches.
  - Idea: contract cycles.
- Correctness? Running time?



A contraction lemma

Lemma

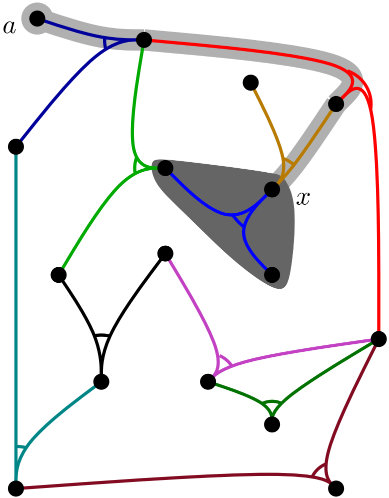
Let $x \in V$ with an x–a path P s.t. all switches of P are used “forward”. Contracting switches with pivot x that are not on P does not change the vertices reachable from a.



A contraction lemma

Lemma

Let $x \in V$ with an x–a path P s.t. all switches of P are used "forward". Contracting switches with pivot x that are not on P does not change the vertices reachable from a.

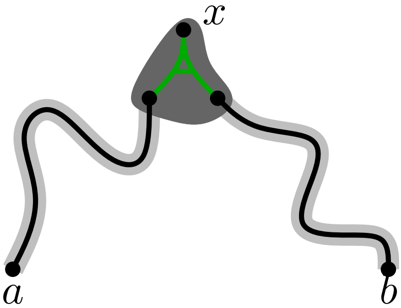


A contraction lemma

Lemma

Let $x \in V$ with an x–a path P s.t. all switches of P are used "forward". Contracting switches with pivot x that are not on P does not change the vertices reachable from a.

≫ preserves reachability from a

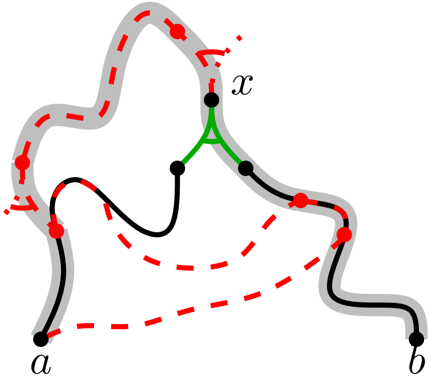


A contraction lemma

Lemma

Let $x \in V$ with an x–a path P s.t. all switches of P are used "forward". Contracting switches with pivot x that are not on P does not change the vertices reachable from a.

≫ preserves reachability from a

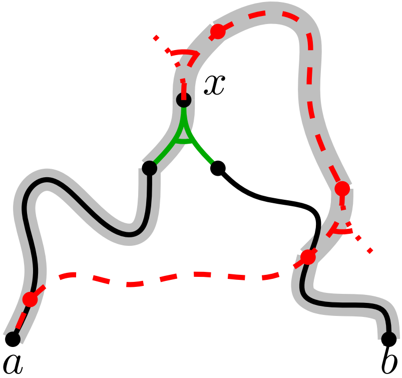


A contraction lemma

Lemma

Let $x \in V$ with an x–a path P s.t. all switches of P are used "forward". Contracting switches with pivot x that are not on P does not change the vertices reachable from a.

≫ preserves reachability from a

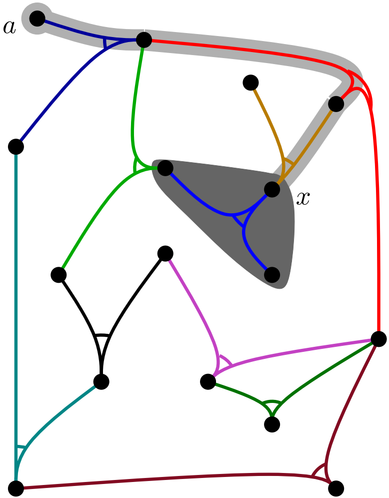


A contraction lemma

Lemma

Let $x \in V$ with an x–a path P s.t. all switches of P are used "forward". Contracting switches with pivot x that are not on P does not change the vertices reachable from a.

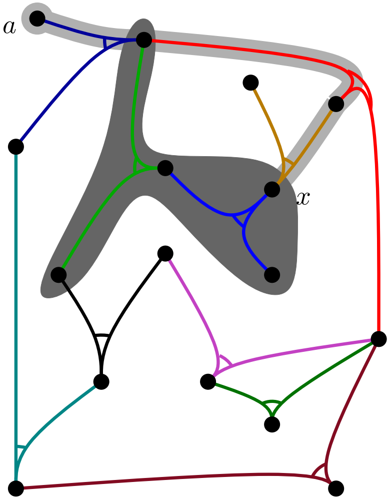
≫ preserves reachability from a



A contraction lemma

Lemma

Let $x \in V$ with an x–a path P s.t. all switches of P are used "forward". Contracting switches with pivot x that are not on P does not change the vertices reachable from a.



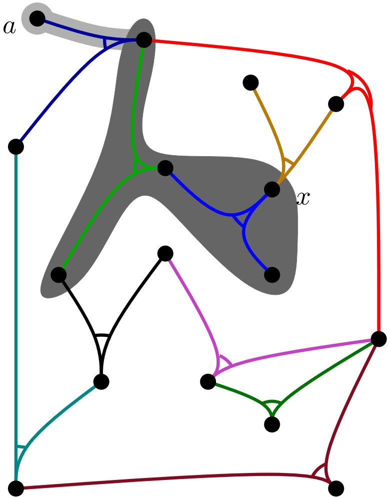
- preserves reachability from a
- contracts cycles of switches

A contraction lemma

Lemma

Let $x \in V$ with an x–a path P s.t. all switches of P are used "forward". Contracting switches with pivot x that are not on P does not change the vertices reachable from a.

- preserves reachability from a
- contracts cycles of switches

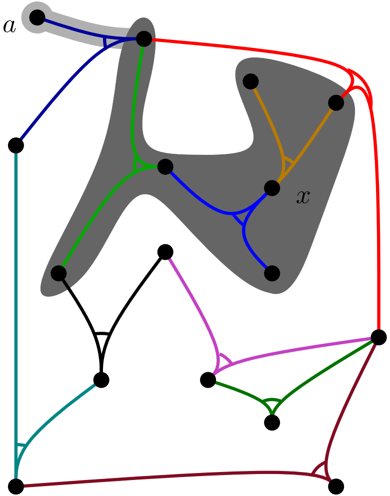


A contraction lemma

Lemma

Let $x \in V$ with an x–a path P s.t. all switches of P are used "forward". Contracting switches with pivot x that are not on P does not change the vertices reachable from a.

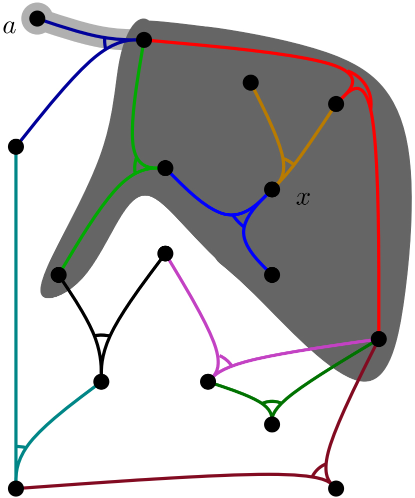
- preserves reachability from a
- contracts cycles of switches



A contraction lemma

Lemma

Let $x \in V$ with an x–a path P s.t. all switches of P are used "forward". Contracting switches with pivot x that are not on P does not change the vertices reachable from a.



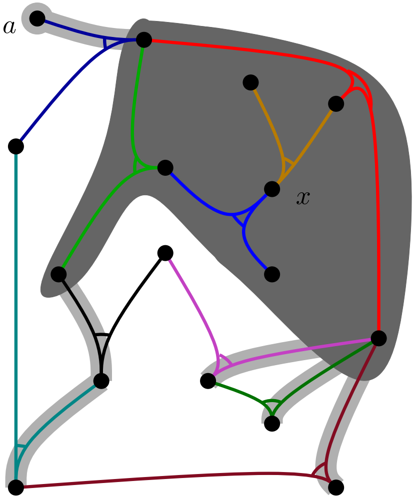
- preserves reachability from a
- contracts cycles of switches

A contraction lemma

Lemma

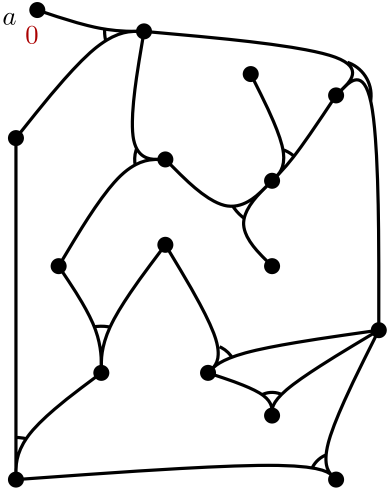
Let $x \in V$ with an x–a path P s.t. all switches of P are used "forward". Contracting switches with pivot x that are not on P does not change the vertices reachable from a.

- preserves reachability from a
- contracts cycles of switches
- b reachable from a iff b has forward path from b to a
- $O(n^2 + nmk)$ algorithm!



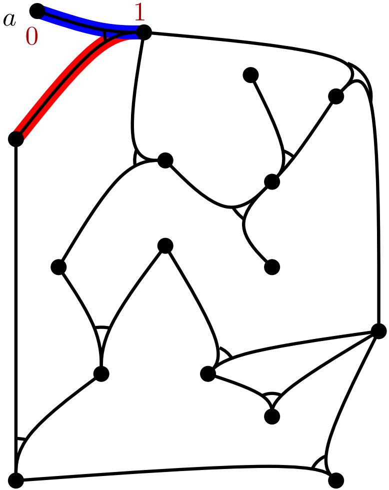
Near linear time: search tree

- Pivot receives label:
  - Mark tree edge blue
  - Mark blocked edges red



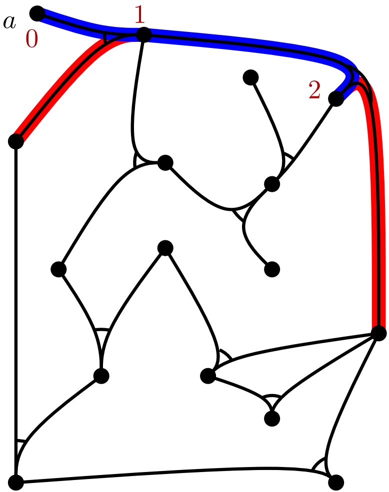
Near linear time: search tree

- Pivot receives label:
  - Mark tree edge blue
  - Mark blocked edges red



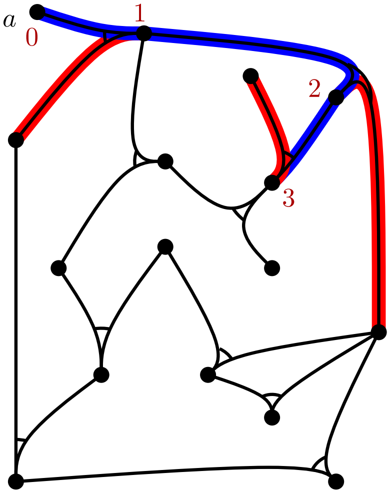
Near linear time: search tree

- Pivot receives label:
  - Mark tree edge blue
  - Mark blocked edges red



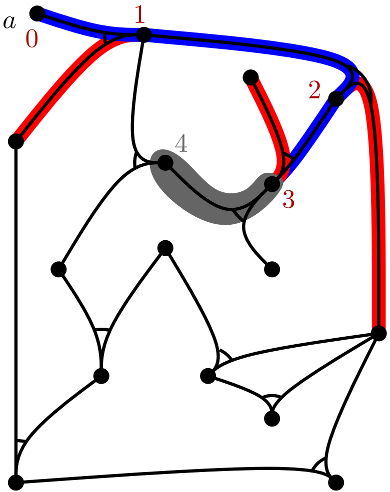
Near linear time: search tree

- Pivot receives label:
  - Mark tree edge blue
  - Mark blocked edges red
- Target receives label:
  - Contract edge



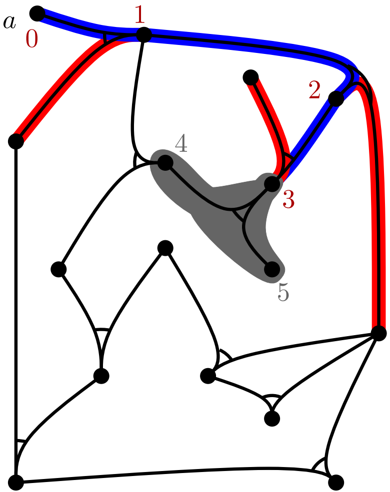
Near linear time: search tree

- Pivot receives label:
  - Mark tree edge blue
  - Mark blocked edges red
- Target receives label:
  - Contract edge



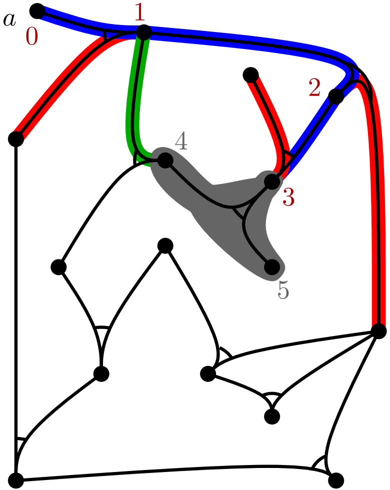
Near linear time: search tree

- Pivot receives label:
  - Mark tree edge blue
  - Mark blocked edges red
- Target receives label:
  - Contract edge
- Both already labeled:
  - mark edge green



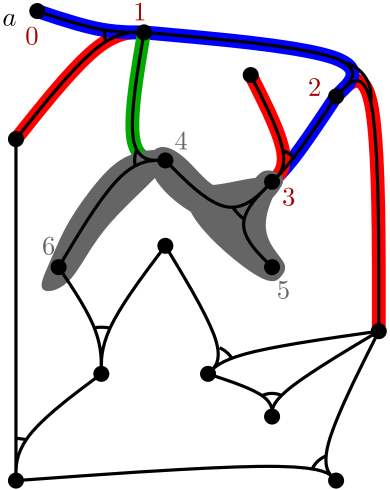
Near linear time: search tree

- Pivot receives label:
  - Mark tree edge blue
  - Mark blocked edges red
- Target receives label:
  - Contract edge
- Both already labeled:
  - mark edge green



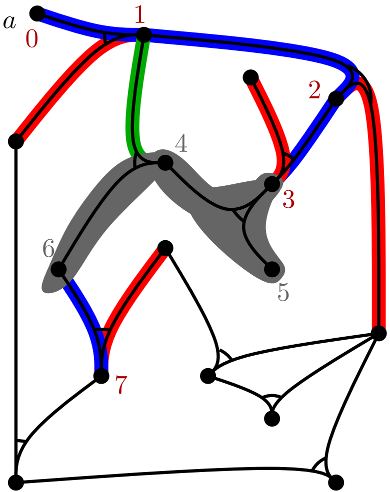
Near linear time: search tree

- Pivot receives label:
  - Mark tree edge blue
  - Mark blocked edges red
- Target receives label:
  - Contract edge
- Both already labeled:
  - mark edge green



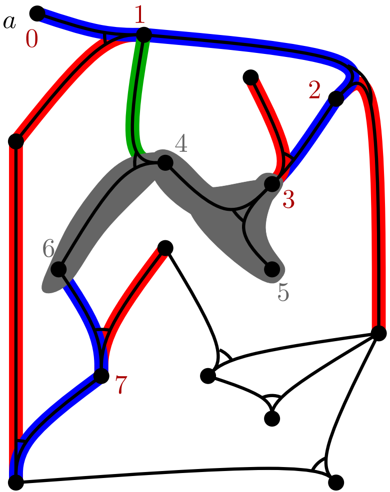
Near linear time: search tree

- Pivot receives label:
  - Mark tree edge blue
  - Mark blocked edges red
- Target receives label:
  - Contract edge
- Both already labeled:
  - mark edge green



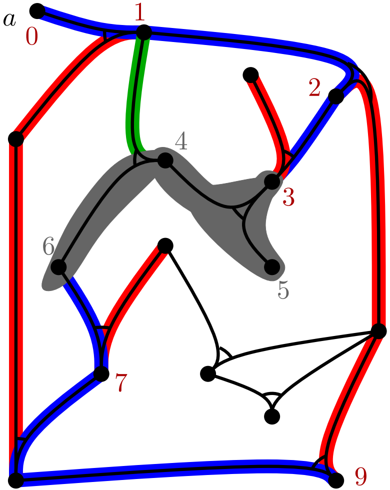
Near linear time: search tree

- Pivot receives label:
  - Mark tree edge blue
  - Mark blocked edges red
- Target receives label:
  - Contract edge
- Both already labeled:
  - mark edge green



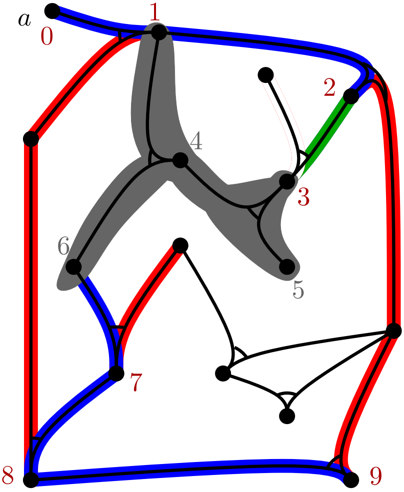
Near linear time: search tree

- Pivot receives label:
  - Mark tree edge blue
  - Mark blocked edges red
- Target receives label:
  - Contract edge
- Both already labeled:
  - mark edge green
- Green edges may be contracted: node with bigger label changes marking of its incident edges:
  - blue → green, red → black



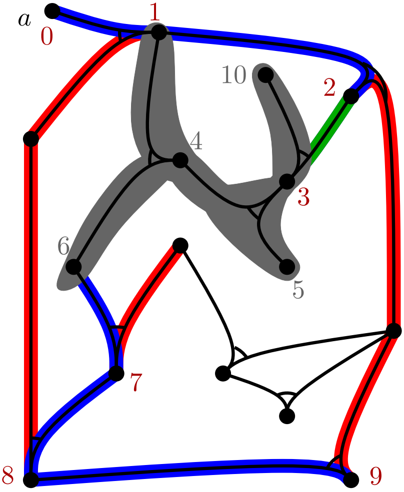
Near linear time: search tree

- Pivot receives label:
  - Mark tree edge blue
  - Mark blocked edges red
- Target receives label:
  - Contract edge
- Both already labeled:
  - mark edge green
- Green edges may be contracted: node with bigger label changes marking of its incident edges:
  - blue → green, red → black



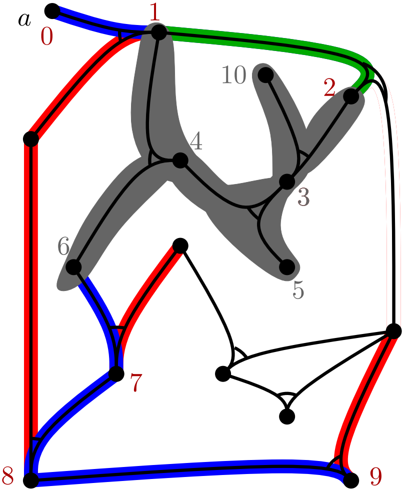
Near linear time: search tree

- Pivot receives label:
  - Mark tree edge blue
  - Mark blocked edges red
- Target receives label:
  - Contract edge
- Both already labeled:
  - mark edge green
- Green edges may be contracted: node with bigger label changes marking of its incident edges:
  - blue → green, red → black



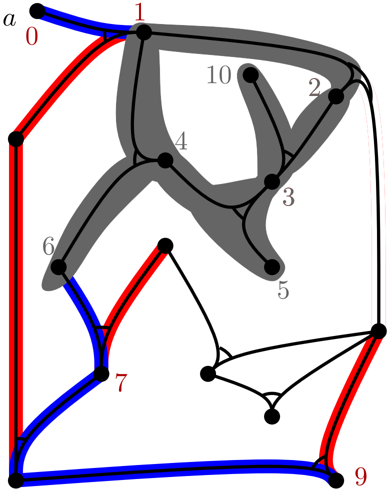
Near linear time: search tree

- Pivot receives label:
  - Mark tree edge blue
  - Mark blocked edges red
- Target receives label:
  - Contract edge
- Both already labeled:
  - mark edge green
- Green edges may be contracted: node with bigger label changes marking of its incident edges:
  - blue → green, red → black



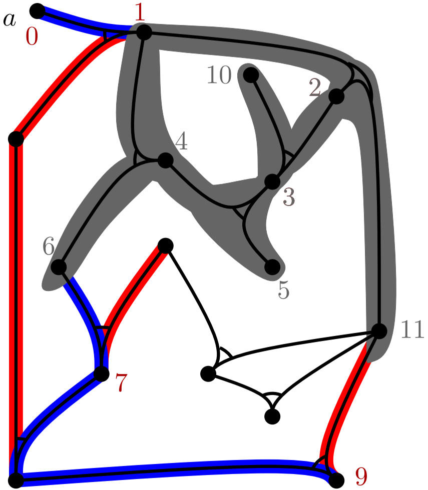
Near linear time: search tree

- Pivot receives label:
  - Mark tree edge blue
  - Mark blocked edges red
- Target receives label:
  - Contract edge
- Both already labeled:
  - mark edge green
- Green edges may be contracted: node with bigger label changes marking of its incident edges:
  - blue → green, red → black



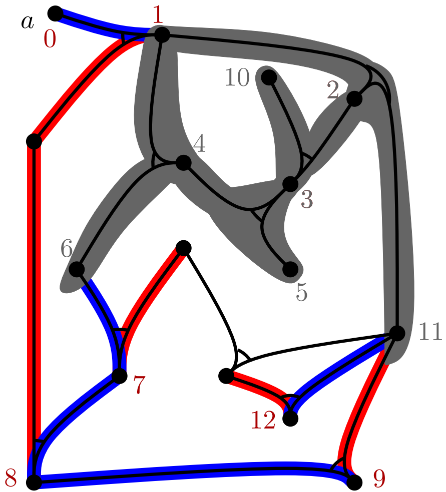
Near linear time: search tree

- Pivot receives label:
  - Mark tree edge blue
  - Mark blocked edges red
- Target receives label:
  - Contract edge
- Both already labeled:
  - mark edge green
- Green edges may be contracted: node with bigger label changes marking of its incident edges:
  - blue → green, red → black



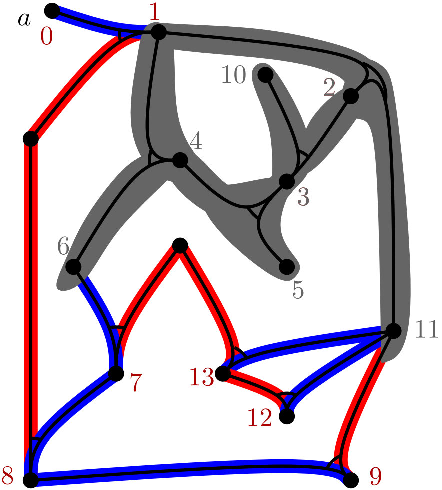
Near linear time: search tree

- Pivot receives label:
  - Mark tree edge blue
  - Mark blocked edges red
- Target receives label:
  - Contract edge
- Both already labeled:
  - mark edge green
- Green edges may be contracted: node with bigger label changes marking of its incident edges:
  - blue → green, red → black



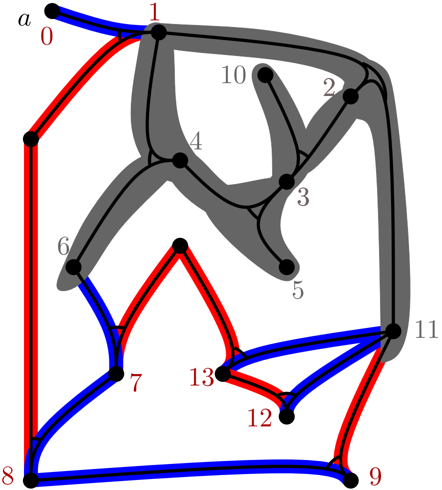
Near linear time: search tree

- Pivot receives label:
  - Mark tree edge blue
  - Mark blocked edges red
- Target receives label:
  - Contract edge
- Both already labeled:
  - mark edge green
- Green edges may be contracted: node with bigger label changes marking of its incident edges:
  - blue → green, red → black



Near linear time: search tree

- Pivot receives label:
  - Mark tree edge blue
  - Mark blocked edges red
- Target receives label:
  - Contract edge
- Both already labeled:
  - mark edge green
- Green edges may be contracted: node with bigger label changes marking of its incident edges:
  - blue → green, red → black
- Running time: $O(km + n\alpha(n))$



Conclusion and Open Problems

Known: Finding configurations with the following properties is NP-hard: Bipartite, triangle-free, Eulerian, biconnected, strongly connected, planar, minimum number of directed cycles

Problem (SwitchConnect-T)

Input: Given a switch graph $G = (V, S)$ and a set $T \subseteq V$.
Question: Is there a configuration c such that in G_c all vertices in T are in the same connected component.

Results:

- Efficient algorithms for $|T| = 2$ and $T = V$.
- NP-hard for arbitrary T.

Open: What about $|T| = 3$? FPT with parameter $|T|$?