max planck institut
informatik

Certifying 3-Edge-Connectivity

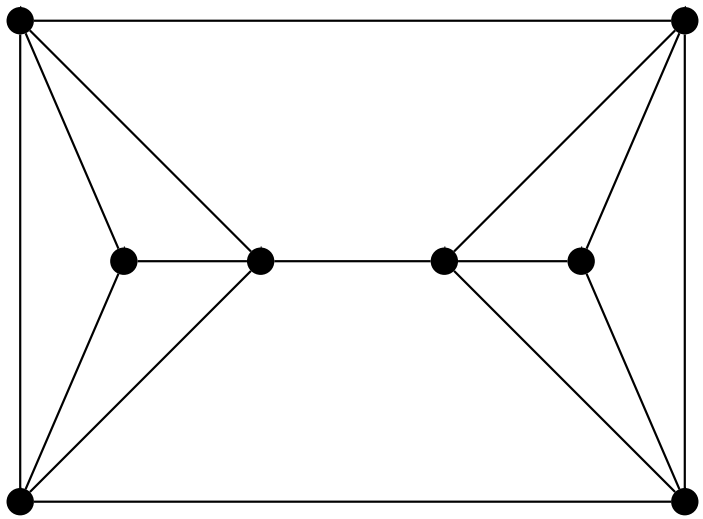
Kurt Mehlhorn Adrian Neumann Jens M. Schmidt

Max Planck Institute for Informatics

June 20, 2013

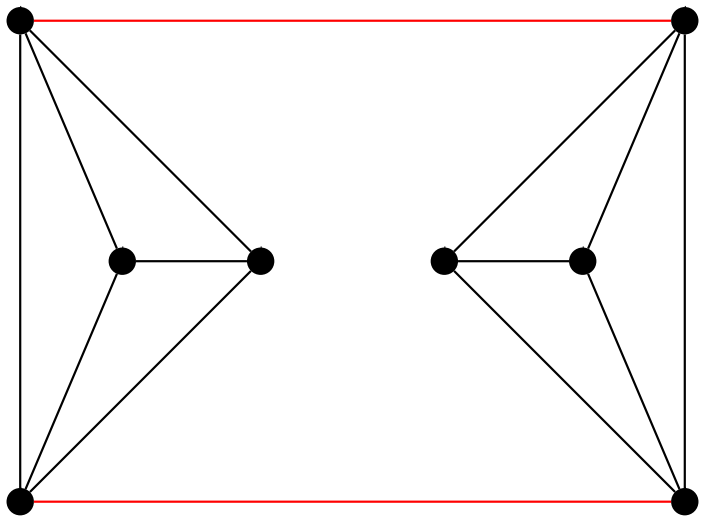
Problem

Given a (multi-)graph G, check whether there is a pair of edges e_1, e_2 such that $G - \{e_1, e_2\}$ is disconnected.



Problem

Given a (multi-)graph G, check whether there is a pair of edges e_1, e_2 such that $G - \{e_1, e_2\}$ is disconnected.



Related Work

3-edge-connectivity is a well studied problem. Many linear time solutions known, e.g.:

- 1992: Nagamochi and Ibaraki
- 1992: Taoka, Watanabe, and Onaga
- 2007, 2009: Tsin

Related Work

3-edge-connectivity is a well studied problem. Many linear time solutions known, e.g.:

- 1992: Nagamochi and Ibaraki
- 1992: Taoka, Watanabe, and Onaga
- 2007, 2009: Tsin

Why yet another algorithm?

Certifying Algorithms

Instead of answering *yes* or *no*, a **certifying** algorithm proves its answer.

Certifying Algorithms

Instead of answering *yes* or *no*, a **certifying** algorithm proves its answer.

- Provide a certificate of correctness for every answer
- A simple checking routine verifies the certificate

Certifying Algorithms

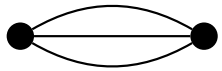
Instead of answering *yes* or *no*, a **certifying** algorithm proves its answer.

- Provide a certificate of correctness for every answer
- A simple checking routine verifies the certificate

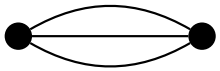
Goal

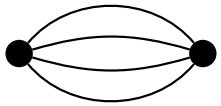
Make the checker implementation simple enough to allow formal verification. Then every answer is correct *by formal proof*.

Construction Sequence

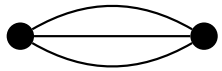
For 3-edge-connectivity we use a *construction sequence* by Mader (1978). Start with a $K_2^3 =$ 

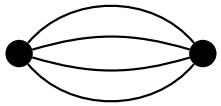
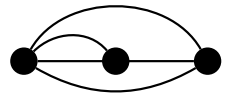
Construction Sequence

For 3-edge-connectivity we use a *construction sequence* by Mader (1978). Start with a $K_2^3 =$ 

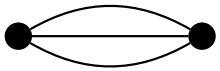
- Add an edge 

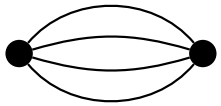
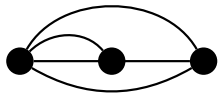
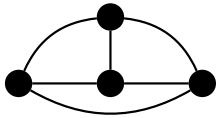
Construction Sequence

For 3-edge-connectivity we use a *construction sequence* by Mader (1978). Start with a $K_2^3 =$ 

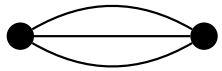
- Add an edge 
- Split an edge, connect the new node with an old node 

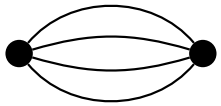
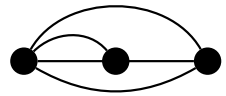
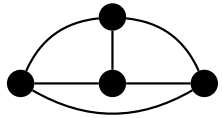
Construction Sequence

For 3-edge-connectivity we use a *construction sequence* by Mader (1978). Start with a $K_2^3 =$ 

- Add an edge 
- Split an edge, connect the new node with an old node 
- Split two edges and connect the new nodes 

Construction Sequence

For 3-edge-connectivity we use a *construction sequence* by Mader (1978). Start with a $K_2^3 =$ 

- Add an edge 
- Split an edge, connect the new node with an old node 
- Split two edges and connect the new nodes 

Mader

Using these operations one can construct exactly the 3-edge-connected graphs.

Content

In This Talk

How to find a construction sequence for a given 3-connected graph.

Content

In This Talk

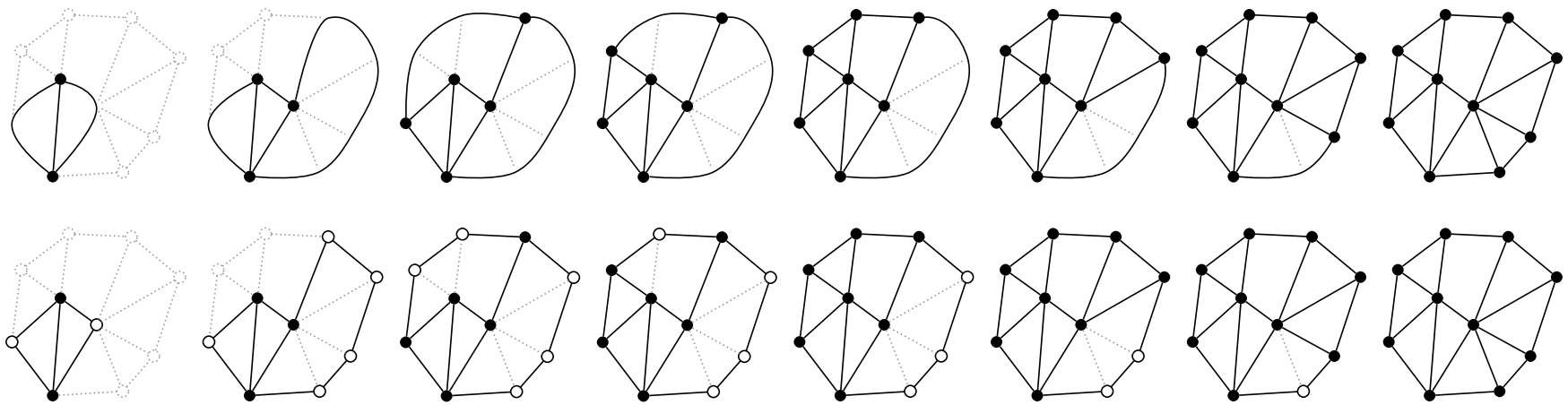
How to find a construction sequence for a given 3-connected graph.

In the paper:

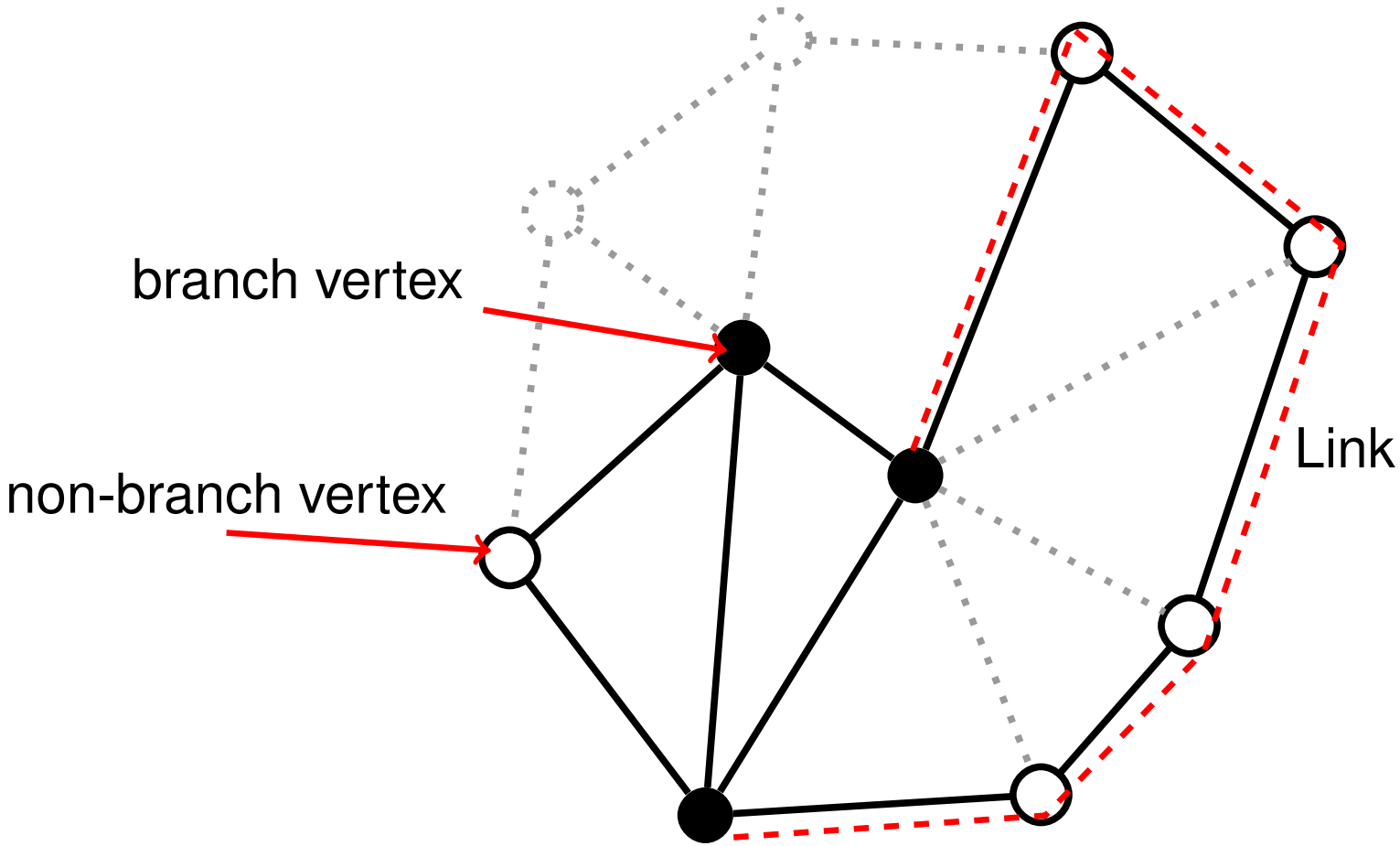
- Theorems for correctness.
- Details for the linear time algorithm.
- How to verify the certificate.
- How to find the 3-edge-connected components (upcoming journal version).

Subdivisions

Subdivisions



Subdivisions

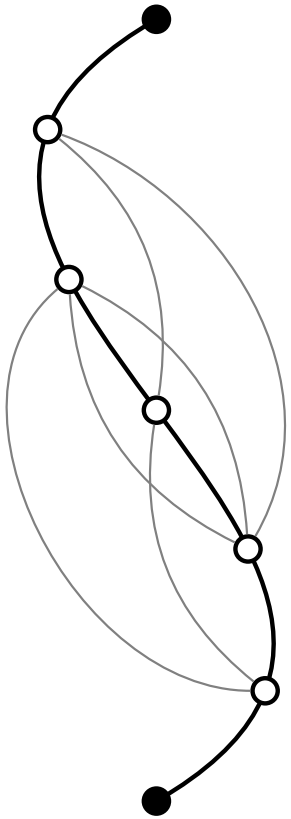


A First Algorithm

1. Find a K_2^3 subdivision. Initialize $G_c = K_2^3$
2. Find a path P in $G - G_c$ from a node u to a node v, such that
   a) at least one of $\{u, v\}$ has degree at least three, or
   b) v lies on a different link from u
3. Add P to the current subgraph
4. If the current subgraph is not G, goto 2.

A First Algorithm

1. Find a K_2^3 subdivision. Initialize $G_c = K_2^3$
2. Find a path P in $G - G_c$ from a node u to a node v, such that
   a) at least one of $\{u, v\}$ has degree at least three, or
   b) v lies on a different link from u
3. Add P to the current subgraph
4. If the current subgraph is not G, goto 2.



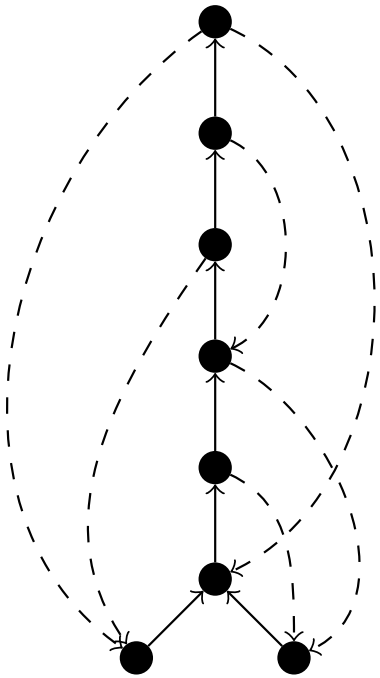
Proof by picture.

Chain Decomposition

A structure to help find a K_2^3 and subsequent paths. Very similar to an ear-decomposition.

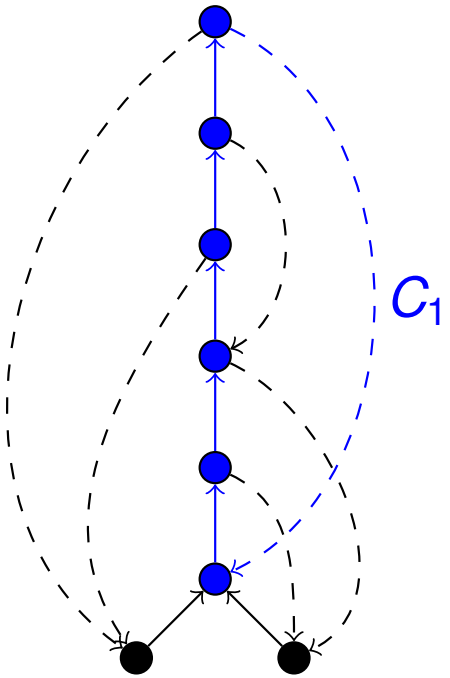
Chain Decomposition

A structure to help find a K_2^3 and subsequent paths. Very similar to an ear-decomposition.



Chain Decomposition

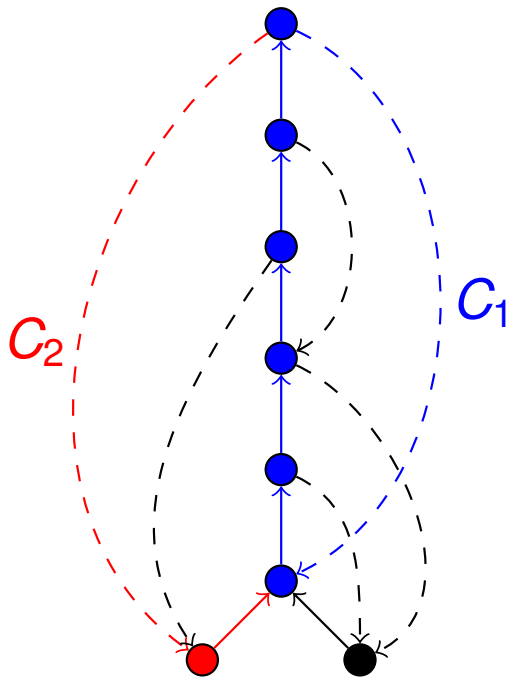
A structure to help find a K_2^3 and subsequent paths. Very similar to an ear-decomposition.

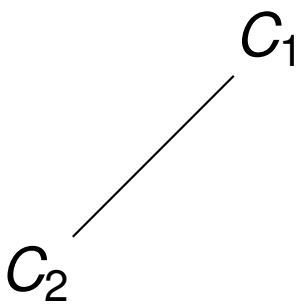


c_1

Chain Decomposition

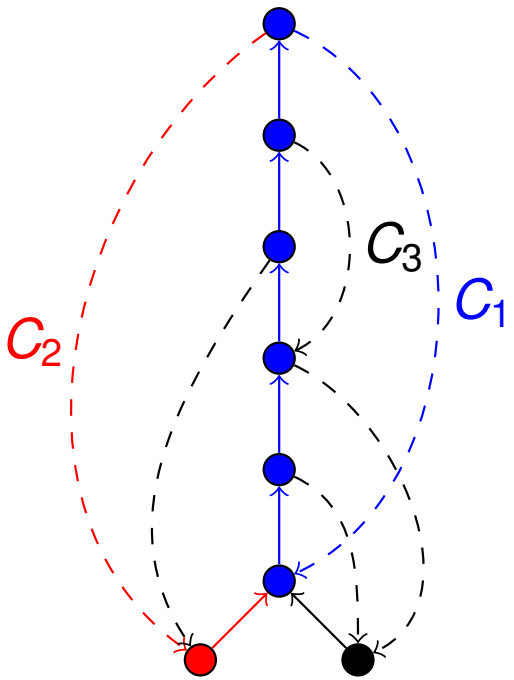
A structure to help find a K_2^3 and subsequent paths. Very similar to an ear-decomposition.

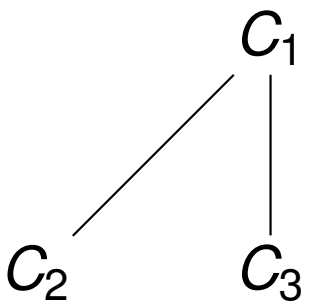




Chain Decomposition

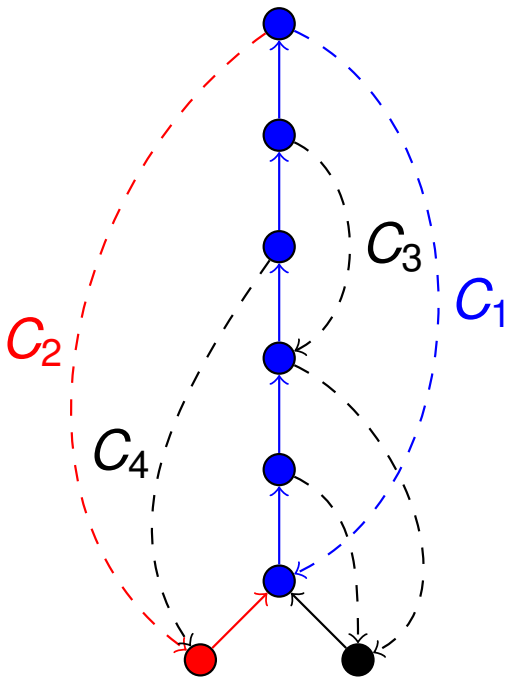
A structure to help find a K_2^3 and subsequent paths. Very similar to an ear-decomposition.

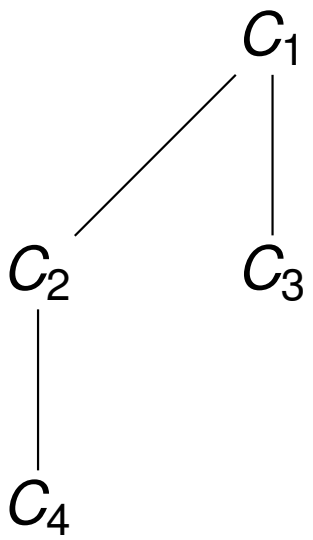




Chain Decomposition

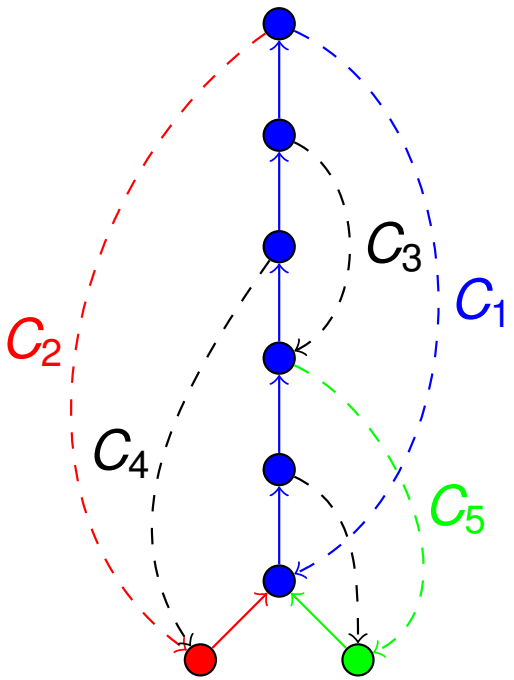
A structure to help find a K_2^3 and subsequent paths. Very similar to an ear-decomposition.

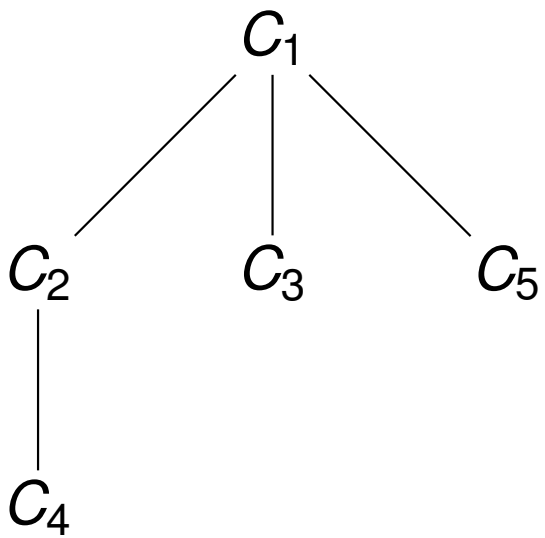




Chain Decomposition

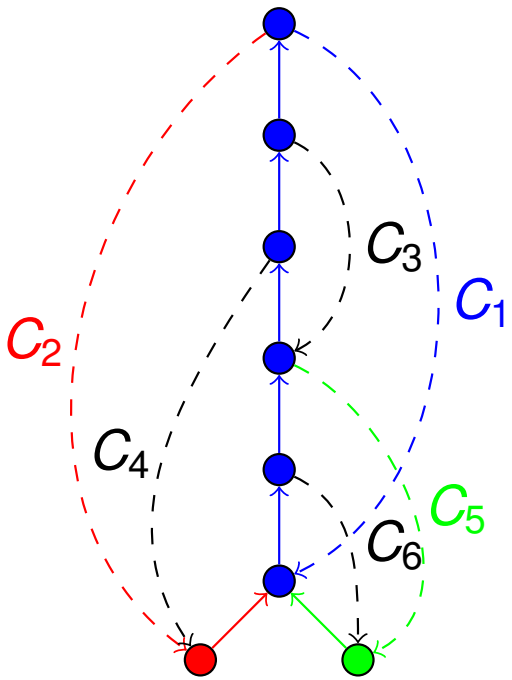
A structure to help find a K_2^3 and subsequent paths. Very similar to an ear-decomposition.

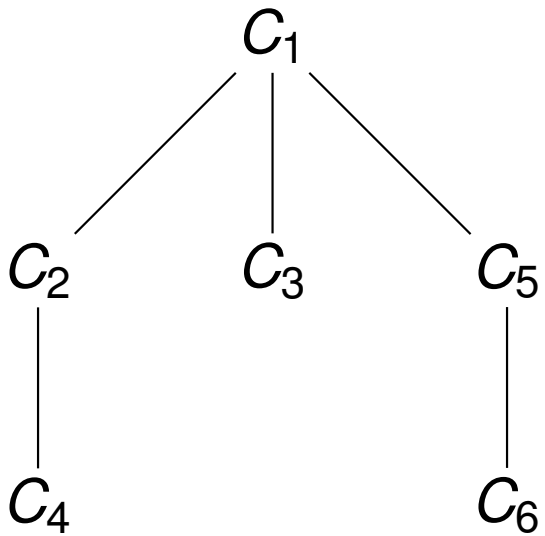




Chain Decomposition

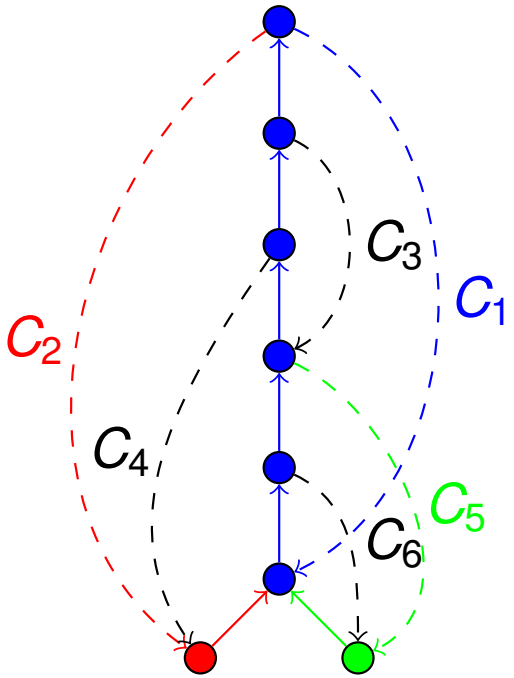
A structure to help find a K_2^3 and subsequent paths. Very similar to an ear-decomposition.

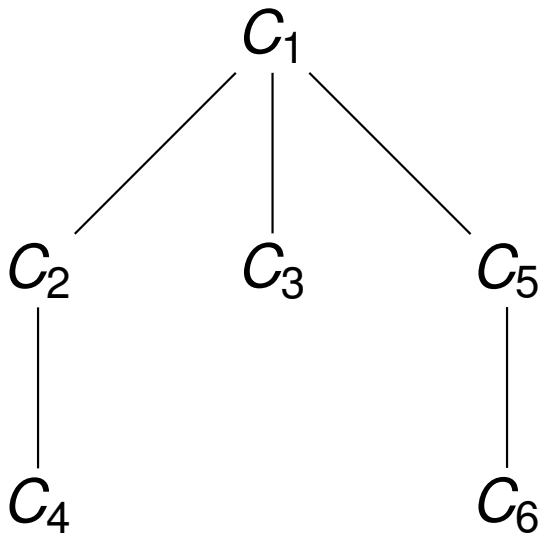




Chain Decomposition

A structure to help find a K_2^3 and subsequent paths. Very similar to an ear-decomposition.





Also useful for certifying 2-vertex-connectivity, 2-edge-connectivity, and 3-vertex-connectivity!

An Improved Algorithm

Observations

- C_1 and C_2 can be chosen s.t. $C_1 \cup C_2 = K_2^3$
- A chain can be added to a graph if its endpoints lie on different links or one is a branch vertex.
- Adding a chain makes its endpoints branch vertices.

An Improved Algorithm

1. Compute a chain decomposition such that $C_1 \cup C_2 \sim K_2^3$
2. Initialize subgraph to $C_1 \cup C_2$.
3. Iterate over children C of C_1 and C_2. If C's endpoints are branching or on different links, add C to a list of addable chains.

An Improved Algorithm

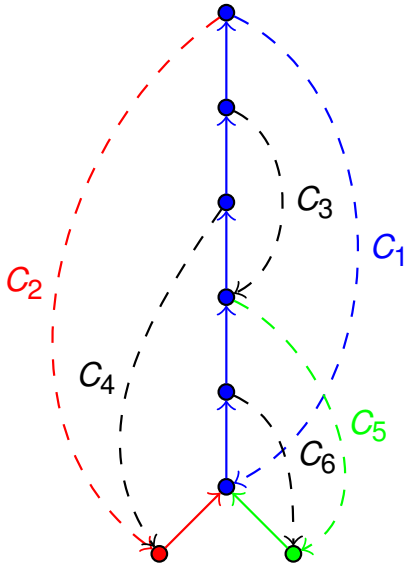
1. Compute a chain decomposition such that $C_1 \cup C_2 \sim K_2^3$
2. Initialize subgraph to $C_1 \cup C_2$.
3. Iterate over children C of C_1 and C_2. If C's endpoints are branching or on different links, add C to a list of addable chains.
4. Take a chain C from the list of addable chains.
   a) Add C. This makes the endpoints branch vertices, splits the links.
   b) Check whether splitting the links made chains addable.
   c) Check whether the children of C are addable.
5. If there are addable chains left, goto 4.

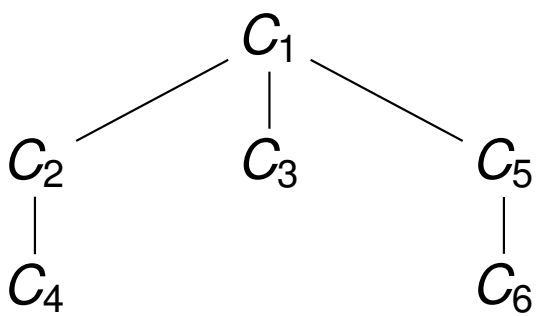
An Improved Algorithm

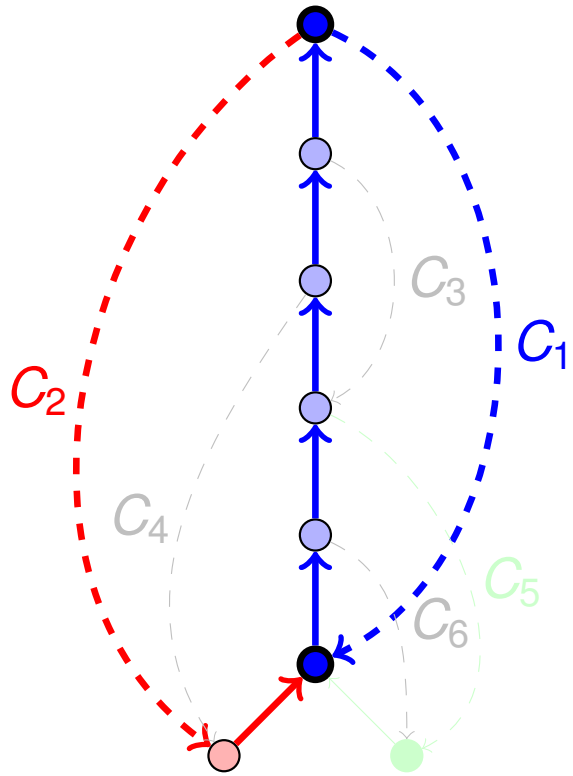
1. Compute a chain decomposition such that $C_1 \cup C_2 \sim K_2^3$
2. Initialize subgraph to $C_1 \cup C_2$.
3. Iterate over children C of C_1 and C_2. If C's endpoints are branching or on different links, add C to a list of addable chains.
4. Take a chain C from the list of addable chains.
   a) Add C. This makes the endpoints branch vertices, splits the links.
   b) Check whether splitting the links made chains addable.
   c) Check whether the children of C are addable.
5. If there are addable chains left, goto 4.

Can be implemented such that the runtime is in

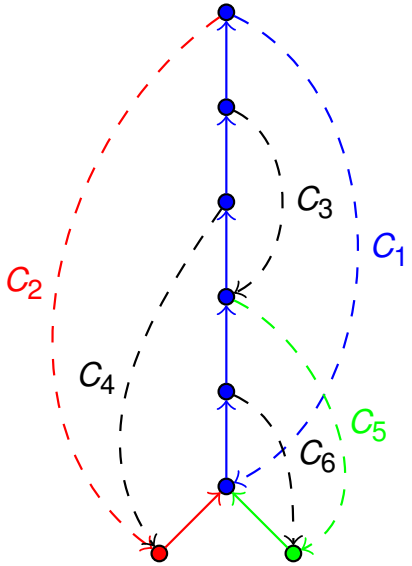
$$O((n+m)\log(n+m)).$$

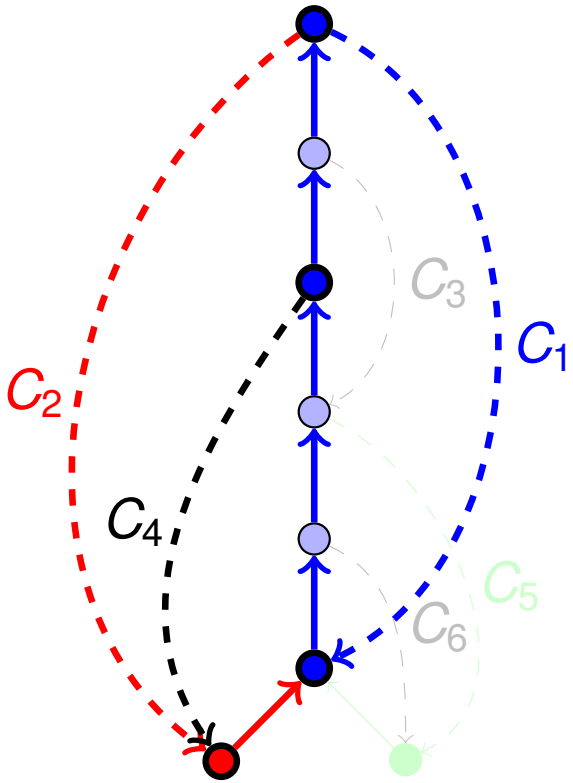




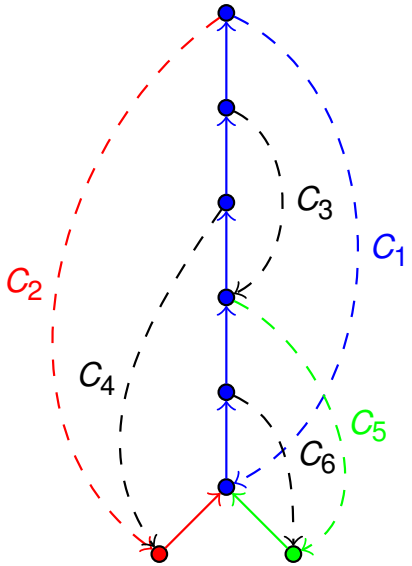


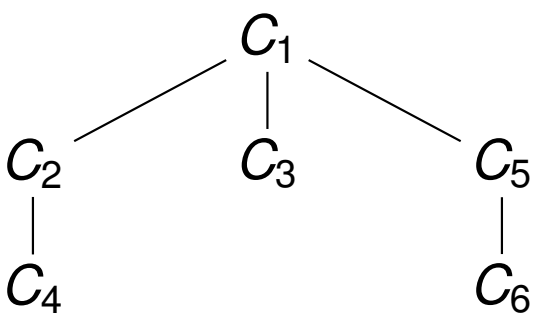
Addable: C_4, C_5

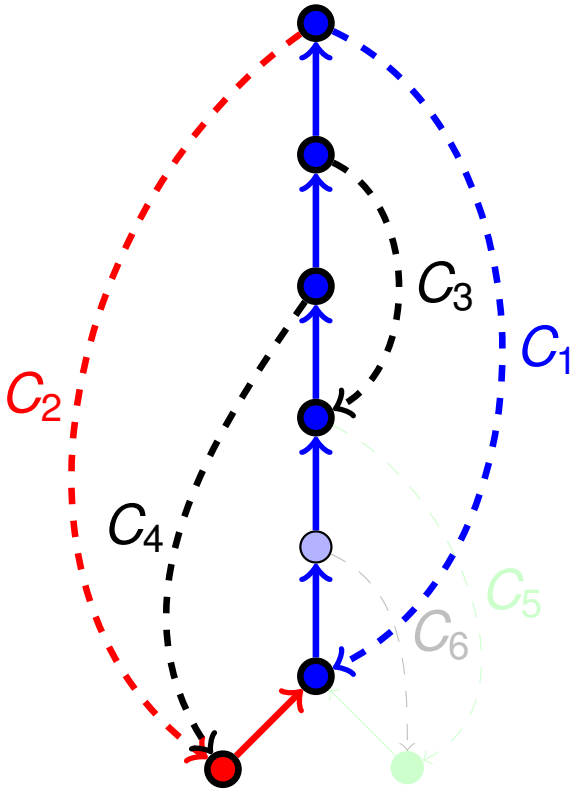




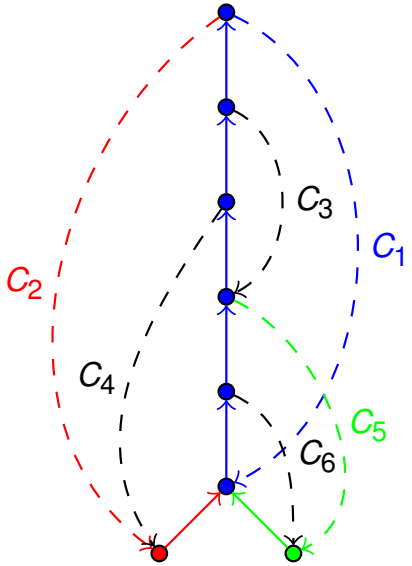
Addable: C_3, C_5

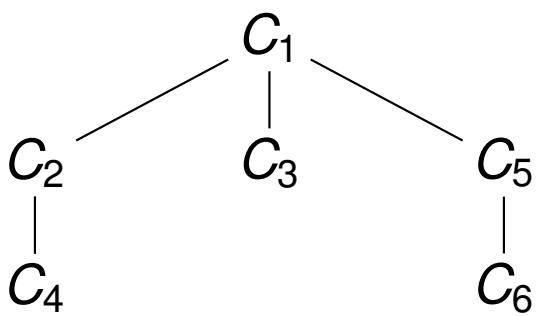


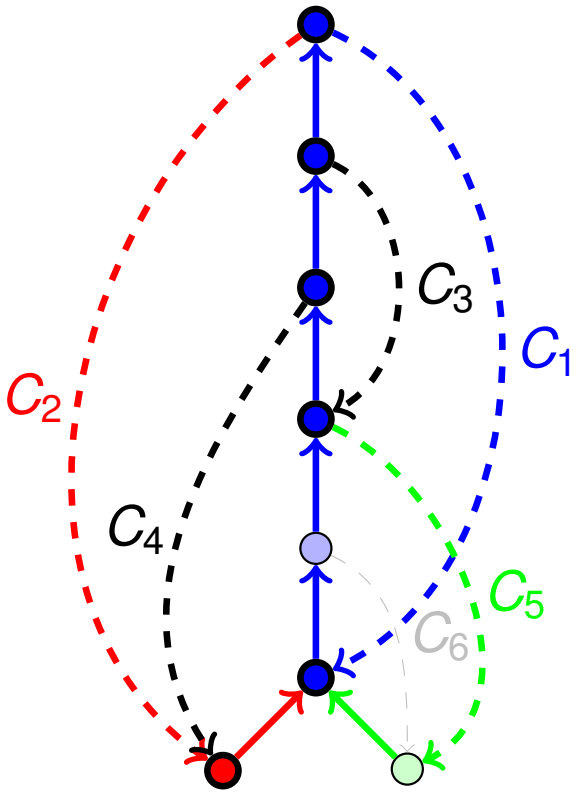




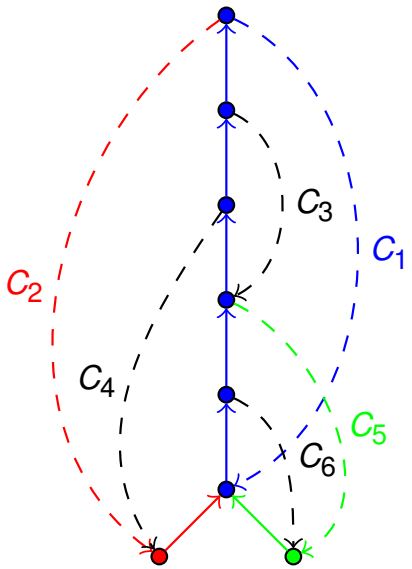
Addable: C_5

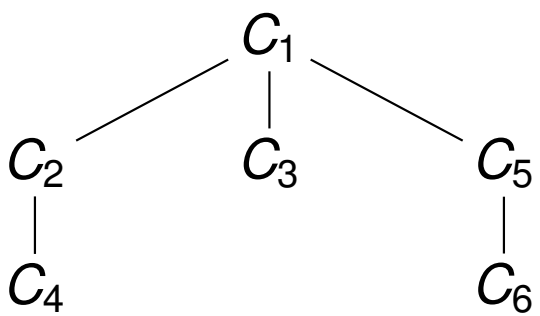


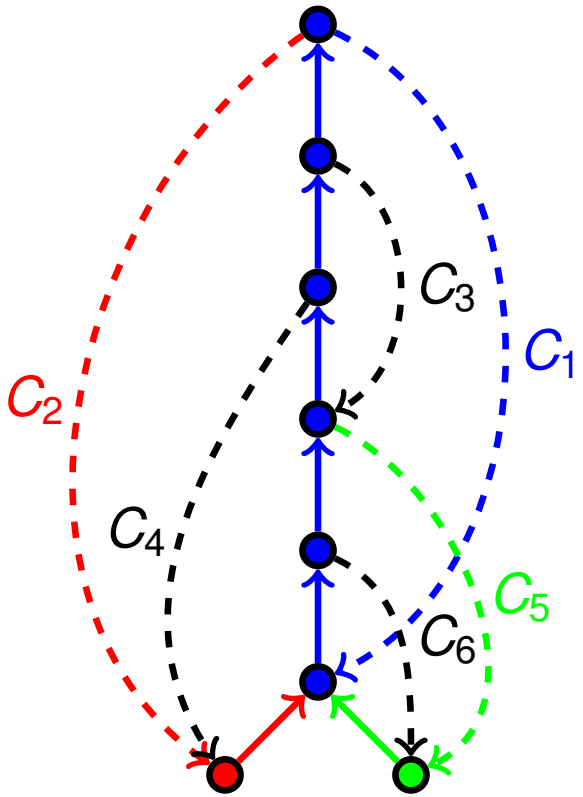




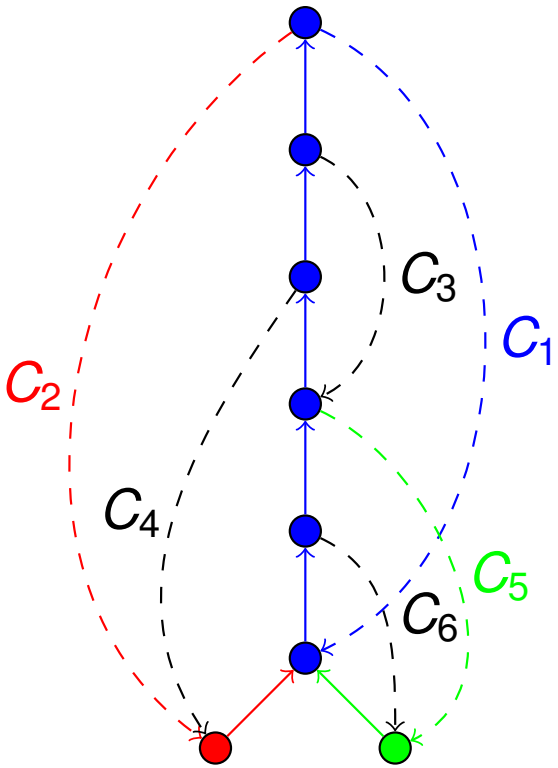
Addable: C_6

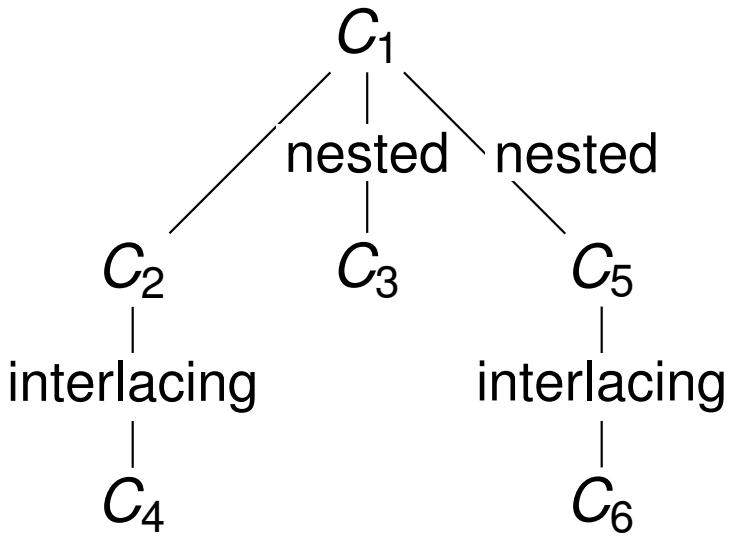

C_1
C_2
C_3
C_4
C_5
C_6


C_1
C_2
C_3
C_5
C_4
C_6


C_1
C_2
C_3
C_4
C_5
C_6

Classifying Chains





Observations

- Nested children can be added after their parent if there is a branch vertex on the tree path between their endpoints
- Interlacing children can be added immediately after their parent.

Observations

- Nested children can be added after their parent if there is a branch vertex on the tree path between their endpoints
- Interlacing children can be added immediately after their parent.

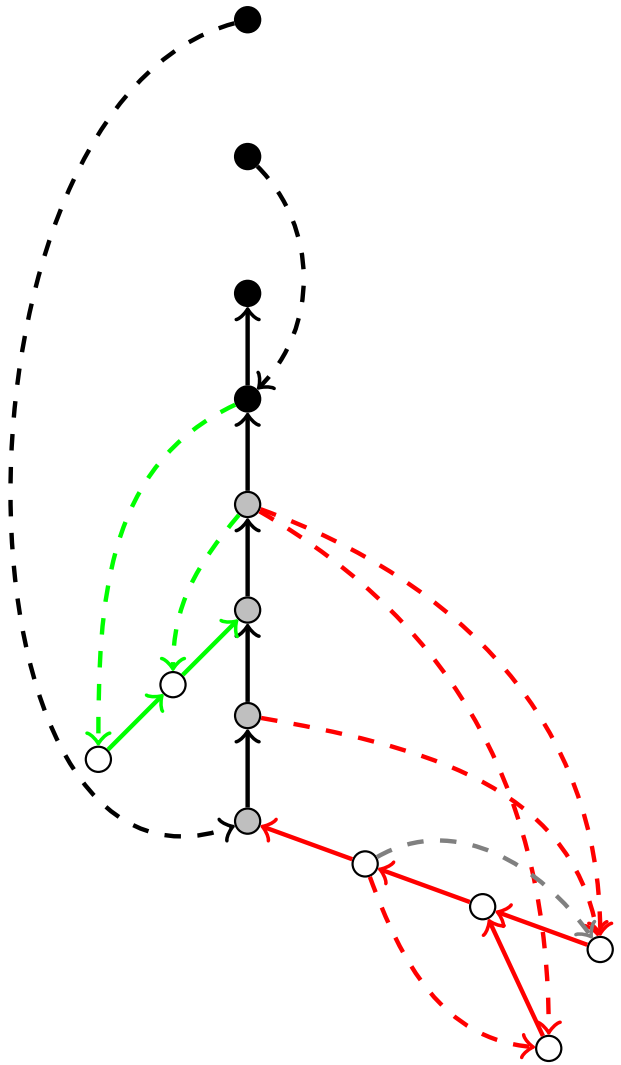
Idea

Add interlacing chains (and their interlacing offspring) as early as possible.
$\Rightarrow$ After adding a nested chain, add all interlacing offspring.

Segments

Segment: A nested child and all its interlacing offspring.

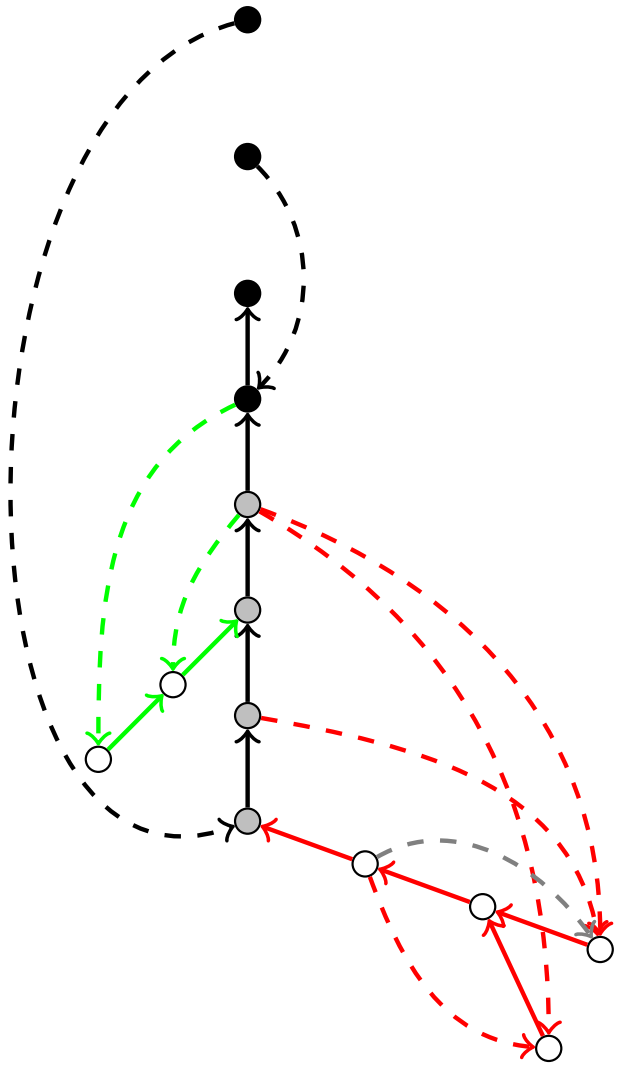
Can be added if there is a branch vertex between the endpoints of the nested chain.



Segments

Problem: Find an order on the segments.

Solution: Traverse the overlap graph.



A Linear Time Algorithm

Compute a chain decomposition $\{C_1, C_2, \ldots, C_{m-n+1}\}$
Initialize G_c to $C_1 \cup C_2$;
for i from 1 to $m - n + 1$ **do**
    Compute the segments w.r.t. C_i
    Find an insertion order $S_1, \ldots, S_k$ on the segments
    for j from 1 to k **do**
        Add the chains contained in S_j parent-first
    end for
end for

Wrap Up

- Certifying Algorithms: An important tool for reliable implementations.
- Construction Sequences: Certificates for connectivity properties.
- Chain decompositions: Structure for computing construction sequences.

Wrap Up

- Certifying Algorithms: An important tool for reliable implementations.
- Construction Sequences: Certificates for connectivity properties.
- Chain decompositions: Structure for computing construction sequences.

Open Problem

Can we find the construction sequence for k-(edge)-connected graphs efficiently?