

On the Approximation Ratio of the Path Matching Christofides Algorithm

Sacha Krug

Department of Computer Science, ETH Zurich, Switzerland

January 23rd, 2012

The Traveling Salesman Problem

Definition

The *traveling salesman problem (TSP)* is the following optimization problem. Given a complete graph G with edge costs, find a Hamiltonian cycle of minimum cost in G .

Definition

The *metric traveling salesman problem (Δ -TSP)* is the TSP restricted to instances satisfying the triangle inequality

$$c(\{v, w\}) \leq c(\{v, u\}) + c(\{u, w\}) \quad \forall u, v, w \in V(G).$$

The Traveling Salesman Problem

Definition

The *traveling salesman problem (TSP)* is the following optimization problem. Given a complete graph G with edge costs, find a Hamiltonian cycle of minimum cost in G .

Definition

The β -*metric traveling salesman problem* (Δ_β -TSP) is the TSP restricted to instances satisfying the β -triangle inequality

$$c(\{v, w\}) \leq \beta \cdot (c(\{v, u\}) + c(\{u, w\})) \quad \forall u, v, w \in V(G).$$

Christofides algorithm

Input: A complete, weighted, metric graph.

- 1: Find a **minimum spanning tree** T . Let U be the set of odd-degree vertices in T .
- 2: Find a **minimum perfect matching** M for U .
- 3: Construct an **Eulerian cycle** E in T and M .
- 4: Transform E into a **Hamiltonian cycle** H by skipping repeated occurrences of vertices.

Output: H . $\text{cost}(H) \leq \frac{3}{2} \cdot \text{cost}(H_{\text{Opt}})$

Christofides algorithm

Input: A complete, weighted, metric graph.

- 1: Find a **minimum spanning tree** T . Let U be the set of odd-degree vertices in T .
- 2: Find a **minimum perfect matching** M for U .
- 3: Construct an **Eulerian cycle** E in T and M .
- 4: Transform E into a **Hamiltonian cycle** H by skipping repeated occurrences of vertices.

Output: H . $\text{cost}(H) \leq \frac{3}{2} \cdot \text{cost}(H_{\text{Opt}})$

Christofides algorithm

Input: A complete, weighted, metric graph.

- 1: Find a **minimum spanning tree** T . Let U be the set of odd-degree vertices in T .
- 2: Find a **minimum perfect matching** M for U .
- 3: Construct an **Eulerian cycle** E in T and M .
- 4: Transform E into a **Hamiltonian cycle** H by skipping repeated occurrences of vertices.

Output: H . $\text{cost}(H) \leq \frac{3}{2} \cdot \text{cost}(H_{\text{Opt}})$

Christofides algorithm

Input: A complete, weighted, metric graph.

- 1: Find a **minimum spanning tree** T . Let U be the set of odd-degree vertices in T .
- 2: Find a **minimum perfect matching** M for U .
- 3: Construct an **Eulerian cycle** E in T and M .
- 4: Transform E into a **Hamiltonian cycle** H by skipping repeated occurrences of vertices.

Output: H . $\text{cost}(H) \leq \frac{3}{2} \cdot \text{cost}(H_{\text{Opt}})$

Christofides algorithm

Input: A complete, weighted, metric graph.

- 1: Find a **minimum spanning tree** T . Let U be the set of odd-degree vertices in T .
- 2: Find a **minimum perfect matching** M for U .
- 3: Construct an **Eulerian cycle** E in T and M .
- 4: Transform E into a **Hamiltonian cycle** H by skipping repeated occurrences of vertices.

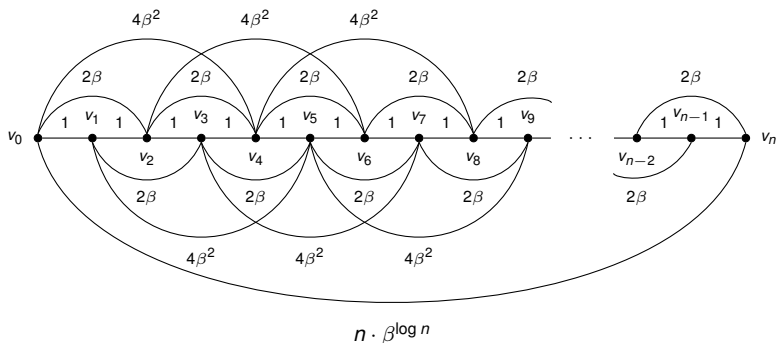
Output: H . $\text{cost}(H) \leq \frac{3}{2} \cdot \text{cost}(H_{\text{Opt}})$

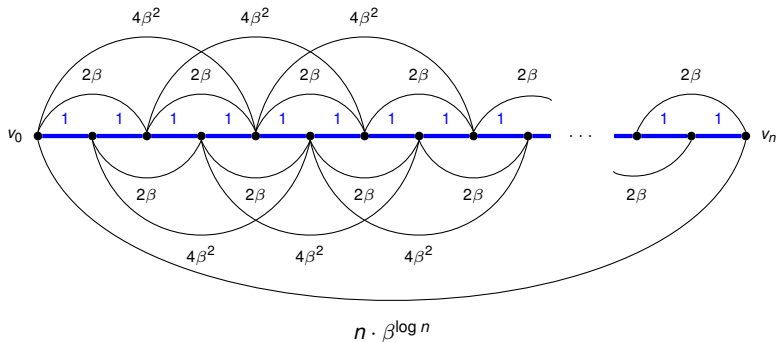
Christofides algorithm

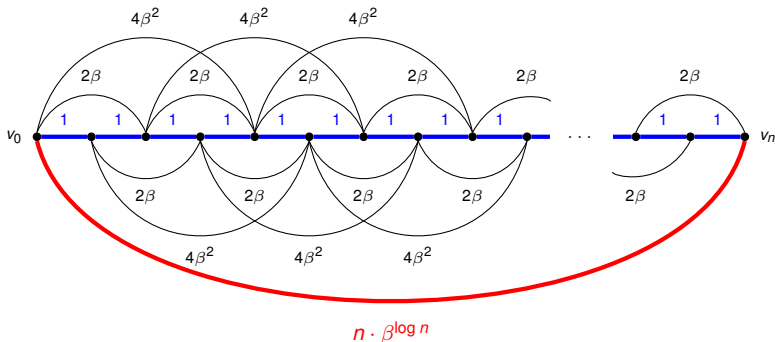
Input: A complete, weighted, metric graph.

- 1: Find a **minimum spanning tree** T . Let U be the set of odd-degree vertices in T .
- 2: Find a **minimum perfect matching** M for U .
- 3: Construct an **Eulerian cycle** E in T and M .
- 4: Transform E into a **Hamiltonian cycle** H by skipping repeated occurrences of vertices.

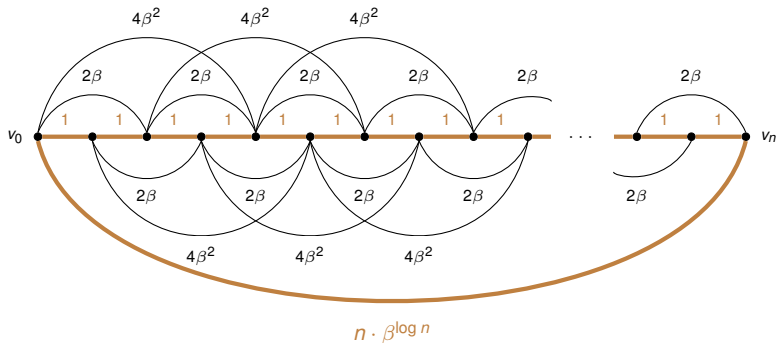
Output: H . $\text{cost}(H) \leq \frac{3}{2} \cdot \text{cost}(H_{\text{Opt}})$

Christofides algorithm not stable for $\beta > 1$ 

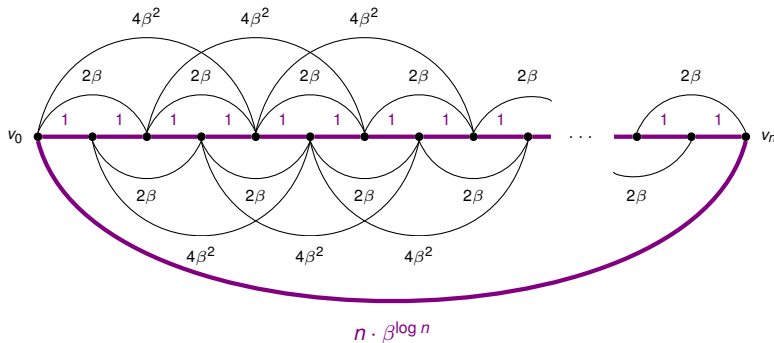
Christofides algorithm not stable for $\beta > 1$ 1. Minimum spanning tree T .

Christofides algorithm not stable for $\beta > 1$ 

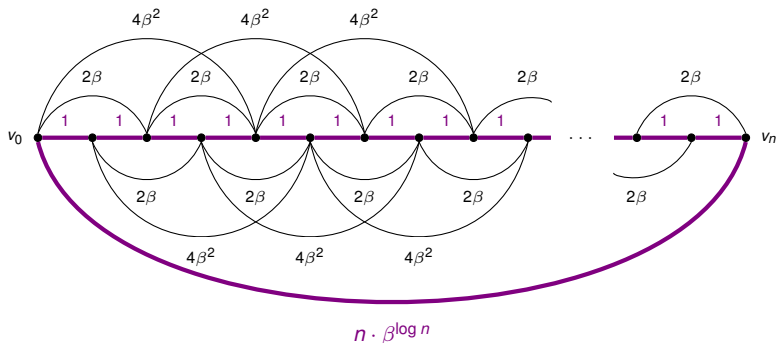
1. Minimum spanning tree T . 2. Minimum perfect matching M .

Christofides algorithm not stable for $\beta > 1$ 

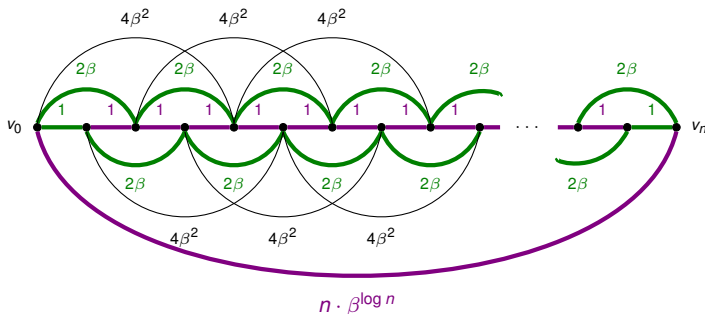
1. Minimum spanning tree T .
2. Minimum perfect matching M .
3. Eulerian cycle E .

Christofides algorithm not stable for $\beta > 1$ 

1. Minimum spanning tree T .
2. Minimum perfect matching M .
3. Eulerian cycle E .
4. Hamiltonian cycle H .

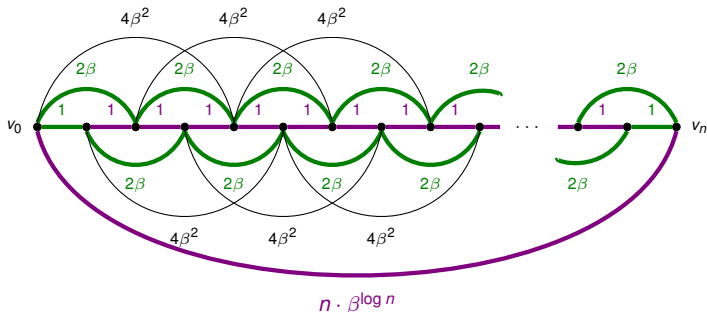
Christofides algorithm not stable for $\beta > 1$ 

Overall cost of H : $n + n \cdot \beta^{\log n} = n + n \cdot n^{\log \beta}$

Christofides algorithm not stable for $\beta > 1$ 

Overall cost of H : $n + n \cdot \beta^{\log n} = n + n \cdot n^{\log \beta}$

Overall cost of optimal solution: $(n-1) \cdot 2\beta + 2$

Christofides algorithm not stable for $\beta > 1$ 

Overall cost of H : $n + n \cdot \beta^{\log n} = n + n \cdot n^{\log \beta}$

Overall cost of optimal solution: $(n-1) \cdot 2\beta + 2$

Approximation ratio: $\frac{n + n \cdot n^{\log \beta}}{(n-1) \cdot 2\beta + 2} \geq n^{\log \beta} / 2\beta$

Motivation

- PMCA is a stable generalization of Christofides algorithm for $\beta \geq 1$ (approximation ratio of $\frac{3}{2}\beta^2$)
- PMCA as good as Christofides algorithm for Δ -TSP
- Worst-case example: lower bound, insight into how to improve the algorithm
- Improved PMCA might outperform Christofides algorithm in the metric case

Motivation

- PMCA is a stable generalization of Christofides algorithm for $\beta \geq 1$ (approximation ratio of $\frac{3}{2}\beta^2$)
- PMCA as good as Christofides algorithm for Δ -TSP
- Worst-case example: lower bound, insight into how to improve the algorithm
- Improved PMCA might outperform Christofides algorithm in the metric case

Motivation

- PMCA is a stable generalization of Christofides algorithm for $\beta \geq 1$ (approximation ratio of $\frac{3}{2}\beta^2$)
- PMCA as good as Christofides algorithm for Δ -TSP
- Worst-case example: lower bound, insight into how to improve the algorithm
- Improved PMCA might outperform Christofides algorithm in the metric case

Motivation

- PMCA is a stable generalization of Christofides algorithm for $\beta \geq 1$ (approximation ratio of $\frac{3}{2}\beta^2$)
- PMCA as good as Christofides algorithm for Δ -TSP
- Worst-case example: lower bound, insight into how to improve the algorithm
- Improved PMCA might outperform Christofides algorithm in the metric case

Basic Idea of the PMCA

Take Christofides algorithm, but compute a *path matching* instead of a normal matching in step 2.

Path matching: Two vertices are not necessarily matched by an edge, but by a path.

First Approach

So can we just do that?

Input: A complete, weighted, β -metric graph.

- 1: Find a minimum spanning tree T . Let U be the set of odd-degree vertices in T .
- 2: Find a minimum perfect **path matching** M for U .
- 3: Construct an Eulerian cycle E in T and M .
- 4: Transform E into a Hamiltonian cycle H by skipping repeated occurrences of vertices.

Output: H .

First Approach

So can we just do that?

Input: A complete, weighted, β -metric graph.

- 1: Find a minimum spanning tree T . Let U be the set of odd-degree vertices in T .
- 2: Find a minimum perfect **path matching** M for U .
- 3: Construct an Eulerian cycle E in T and M .
- 4: Transform E into a Hamiltonian cycle H by skipping repeated occurrences of vertices.

Output: H .

Yes, but ...

The Path Matching Christofides Algorithm

Input: A complete weighted β -metric graph G , for some $\beta \geq 1$.

- 1: Find a minimum spanning tree T in G . Let U be the set of odd-degree vertices in T .
- 2: Construct a minimum path matching M for U .
- 3: Resolve conflicts in M to get a vertex-disjoint path matching M' .
- 4: Construct an Eulerian cycle $E := (p_1, q_1, p_2, q_2, \dots)$ on T and M' such that p_1, p_2, \dots are paths in T and q_1, q_2, \dots are paths in M' .
- 5: Transform p_1, p_2, \dots into p'_1, p'_2, \dots such that the forest formed by p'_1, p'_2, \dots has maximum degree 3.
- 6: Resolve all remaining conflicts in $E' := (p'_1, q_1, p'_2, q_2, \dots)$ to obtain a Hamiltonian cycle H .

Output: H . $\text{cost}(H) \leq \frac{3}{2}\beta^2 \cdot \text{cost}(H_{\text{Opt}})$

The Path Matching Christofides Algorithm

- Input:** A complete weighted β -metric graph G , for some $\beta \geq 1$.
- 1: Find a minimum spanning tree T in G . Let U be the set of odd-degree vertices in T .
 - 2: Construct a minimum path matching M for U .
 - 3: Resolve conflicts in M to get a vertex-disjoint path matching M' .
 - 4: Construct an Eulerian cycle $E := (p_1, q_1, p_2, q_2, \dots)$ on T and M' such that p_1, p_2, \dots are paths in T and q_1, q_2, \dots are paths in M' .
 - 5: Transform p_1, p_2, \dots into p'_1, p'_2, \dots such that the forest formed by p'_1, p'_2, \dots has maximum degree 3.
 - 6: Resolve all remaining conflicts in $E' := (p'_1, q_1, p'_2, q_2, \dots)$ to obtain a Hamiltonian cycle H .

Output: H . $\text{cost}(H) \leq \frac{3}{2}\beta^2 \cdot \text{cost}(H_{\text{Opt}})$

The Path Matching Christofides Algorithm

- Input:** A complete weighted β -metric graph G , for some $\beta \geq 1$.
- 1: Find a minimum spanning tree T in G . Let U be the set of odd-degree vertices in T .
 - 2: Construct a minimum path matching M for U .
 - 3: Resolve conflicts in M to get a vertex-disjoint path matching M' .
 - 4: Construct an Eulerian cycle $E := (p_1, q_1, p_2, q_2, \dots)$ on T and M' such that p_1, p_2, \dots are paths in T and q_1, q_2, \dots are paths in M' .
 - 5: Transform p_1, p_2, \dots into p'_1, p'_2, \dots such that the forest formed by p'_1, p'_2, \dots has maximum degree 3.
 - 6: Resolve all remaining conflicts in $E' := (p'_1, q_1, p'_2, q_2, \dots)$ to obtain a Hamiltonian cycle H .

Output: H . $\text{cost}(H) \leq \frac{3}{2}\beta^2 \cdot \text{cost}(H_{\text{Opt}})$

The Path Matching Christofides Algorithm

- Input:** A complete weighted β -metric graph G , for some $\beta \geq 1$.
- 1: Find a minimum spanning tree T in G . Let U be the set of odd-degree vertices in T .
 - 2: Construct a minimum path matching M for U .
 - 3: Resolve conflicts in M to get a vertex-disjoint path matching M' .
 - 4: Construct an Eulerian cycle $E := (p_1, q_1, p_2, q_2, \dots)$ on T and M' such that p_1, p_2, \dots are paths in T and q_1, q_2, \dots are paths in M' .
 - 5: Transform p_1, p_2, \dots into p'_1, p'_2, \dots such that the forest formed by p'_1, p'_2, \dots has maximum degree 3.
 - 6: Resolve all remaining conflicts in $E' := (p'_1, q_1, p'_2, q_2, \dots)$ to obtain a Hamiltonian cycle H .

Output: H . $\text{cost}(H) \leq \frac{3}{2}\beta^2 \cdot \text{cost}(H_{\text{Opt}})$

The Path Matching Christofides Algorithm

- Input:** A complete weighted β -metric graph G , for some $\beta \geq 1$.
- 1: Find a minimum spanning tree T in G . Let U be the set of odd-degree vertices in T .
 - 2: Construct a minimum path matching M for U .
 - 3: Resolve conflicts in M to get a vertex-disjoint path matching M' .
 - 4: Construct an Eulerian cycle $E := (p_1, q_1, p_2, q_2, \dots)$ on T and M' such that p_1, p_2, \dots are paths in T and q_1, q_2, \dots are paths in M' .
 - 5: Transform p_1, p_2, \dots into p'_1, p'_2, \dots such that the forest formed by p'_1, p'_2, \dots has maximum degree 3.
 - 6: Resolve all remaining conflicts in $E' := (p'_1, q_1, p'_2, q_2, \dots)$ to obtain a Hamiltonian cycle H .

Output: H . $\text{cost}(H) \leq \frac{3}{2}\beta^2 \cdot \text{cost}(H_{\text{Opt}})$

The Path Matching Christofides Algorithm

- Input:** A complete weighted β -metric graph G , for some $\beta \geq 1$.
- 1: Find a minimum spanning tree T in G . Let U be the set of odd-degree vertices in T .
 - 2: Construct a minimum path matching M for U .
 - 3: Resolve conflicts in M to get a vertex-disjoint path matching M' .
 - 4: Construct an Eulerian cycle $E := (p_1, q_1, p_2, q_2, \dots)$ on T and M' such that p_1, p_2, \dots are paths in T and q_1, q_2, \dots are paths in M' .
 - 5: Transform p_1, p_2, \dots into p'_1, p'_2, \dots such that the forest formed by p'_1, p'_2, \dots has maximum degree 3.
 - 6: Resolve all remaining conflicts in $E' := (p'_1, q_1, p'_2, q_2, \dots)$ to obtain a Hamiltonian cycle H .

Output: H . $\text{cost}(H) \leq \frac{3}{2}\beta^2 \cdot \text{cost}(H_{\text{Opt}})$

The Path Matching Christofides Algorithm

- Input:** A complete weighted β -metric graph G , for some $\beta \geq 1$.
- 1: Find a minimum spanning tree T in G . Let U be the set of odd-degree vertices in T .
 - 2: Construct a minimum path matching M for U .
 - 3: Resolve conflicts in M to get a vertex-disjoint path matching M' .
 - 4: Construct an Eulerian cycle $E := (p_1, q_1, p_2, q_2, \dots)$ on T and M' such that p_1, p_2, \dots are paths in T and q_1, q_2, \dots are paths in M' .
 - 5: Transform p_1, p_2, \dots into p'_1, p'_2, \dots such that the forest formed by p'_1, p'_2, \dots has maximum degree 3.
 - 6: Resolve all remaining conflicts in $E' := (p'_1, q_1, p'_2, q_2, \dots)$ to obtain a Hamiltonian cycle H .

Output: H . $\text{cost}(H) \leq \frac{3}{2}\beta^2 \cdot \text{cost}(H_{\text{Opt}})$

The Path Matching Christofides Algorithm

- Input:** A complete weighted β -metric graph G , for some $\beta \geq 1$.
- 1: Find a minimum spanning tree T in G . Let U be the set of odd-degree vertices in T .
 - 2: Construct a minimum path matching M for U .
 - 3: Resolve conflicts in M to get a vertex-disjoint path matching M' .
 - 4: Construct an Eulerian cycle $E := (p_1, q_1, p_2, q_2, \dots)$ on T and M' such that p_1, p_2, \dots are paths in T and q_1, q_2, \dots are paths in M' .
 - 5: Transform p_1, p_2, \dots into p'_1, p'_2, \dots such that the forest formed by p'_1, p'_2, \dots has maximum degree 3.
 - 6: Resolve all remaining conflicts in $E' := (p'_1, q_1, p'_2, q_2, \dots)$ to obtain a Hamiltonian cycle H .

Output: H . $\text{cost}(H) \leq \frac{3}{2}\beta^2 \cdot \text{cost}(H_{\text{Opt}})$

Goal

Theorem

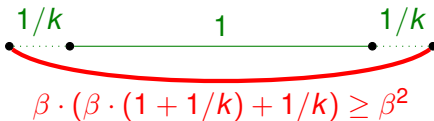
The upper bound of $\frac{3}{2}\beta^2$ on the approximation ratio of the PMCA is tight.

Two Tricks

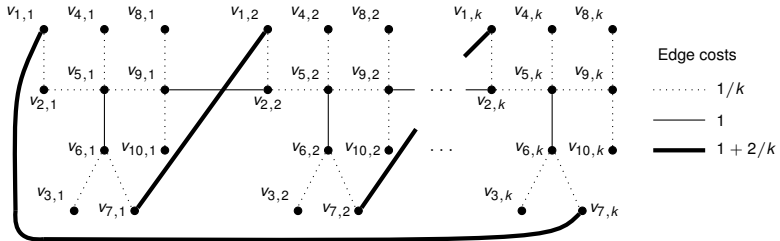
- $\frac{3}{2}$:



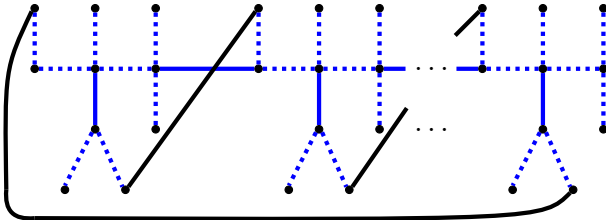
- β^2 :



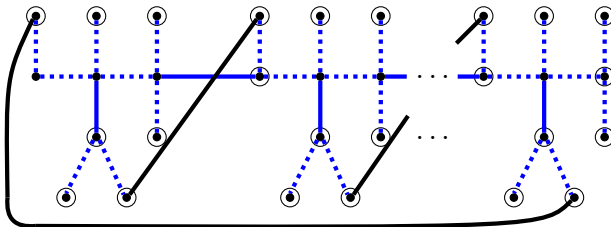
The Graph



Step 1: Minimum Spanning Tree T

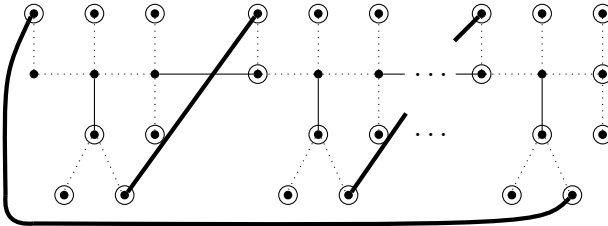


Step 1: Minimum Spanning Tree T

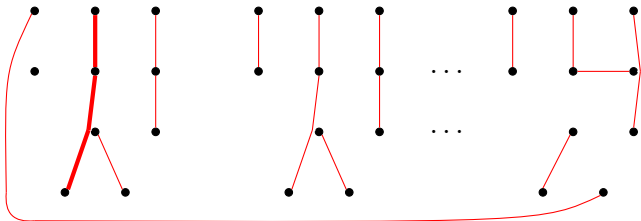


⊙ = U (set of odd-degree vertices)

Step 2: Minimum Path Matching M for U



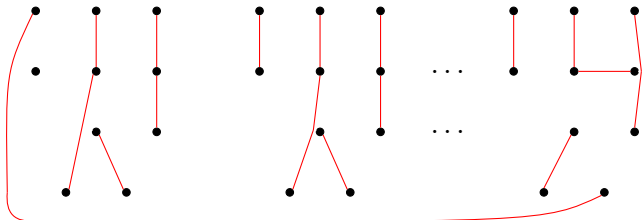
Step 3: Conflict Resolution in M



Part I: Finding a path with only one conflict

- 1 Start with an arbitrary path p with at least one conflict.
- 2 If p has more than one conflict, let v, w be the first and last.
- 3 Pick as new p one of the paths p_v, p_w that was not previously picked.

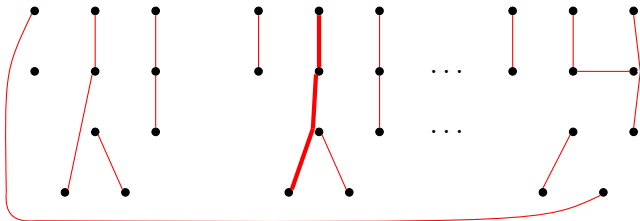
Step 3: Conflict Resolution in M



Part I: Finding a path with only one conflict

- 1 Start with an arbitrary path p with at least one conflict.
- 2 If p has more than one conflict, let v, w be the first and last.
- 3 Pick as new p one of the paths p_v, p_w that was not previously picked.

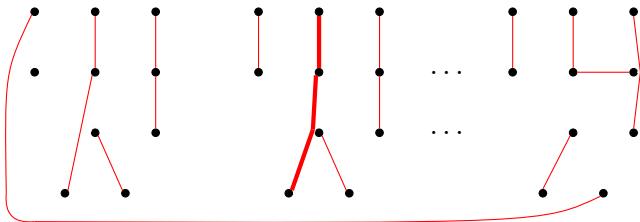
Step 3: Conflict Resolution in M



Part I: Finding a path with only one conflict

- 1 Start with an arbitrary path p with at least one conflict.
- 2 If p has more than one conflict, let v, w be the first and last.
- 3 Pick as new p one of the paths p_v, p_w that was not previously picked.

Step 3: Conflict Resolution in M

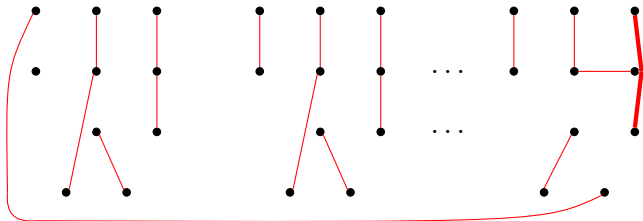


Part II: Resolving a conflict

Situation: We have a path p with only one conflict v

- If v is internal to p , bypass v in p .
- If v is an endpoint of p , do a certain transformation.

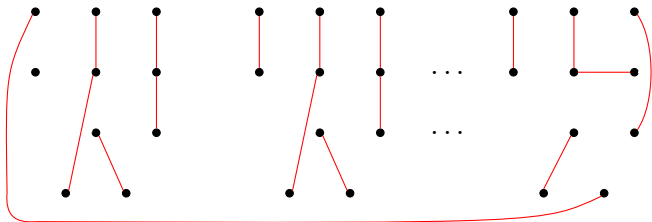
Step 3: Conflict Resolution in M



Part I: Finding a path with only one conflict

- 1 Start with an arbitrary path p with at least one conflict.
- 2 If p has more than one conflict, let v, w be the first and last.
- 3 Pick as new p one of the paths p_v, p_w that was not previously picked.

Step 3: Conflict Resolution in M

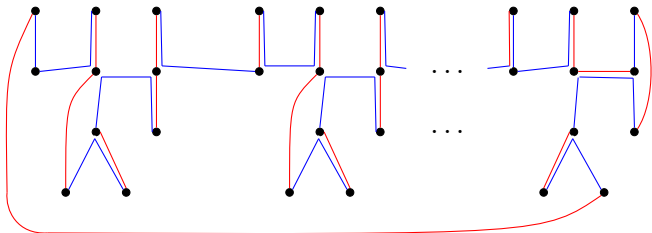


Part II: Resolving a conflict

Situation: We have a path p with only one conflict v

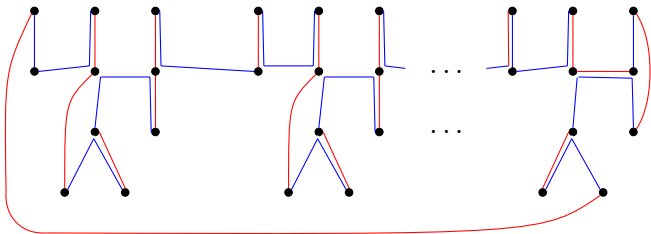
- If v is internal to p , bypass v in p .
- If v is an endpoint of p , do a certain transformation.

Step 4: Eulerian Cycle E



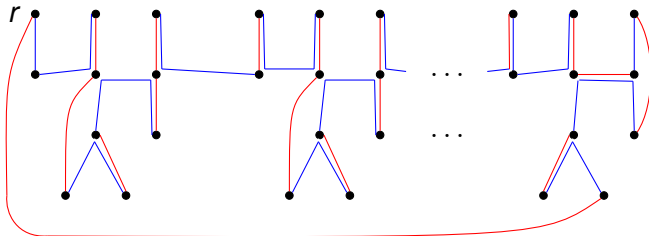
Combine T and M' to get E

Step 5: Conflict Resolution in the Paths of T



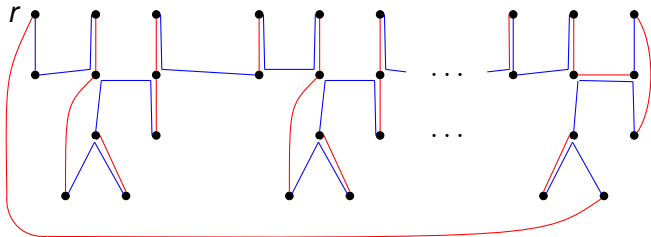
Goal: Every vertex has blue degree ≤ 3

Step 5: Conflict Resolution in the Paths of T



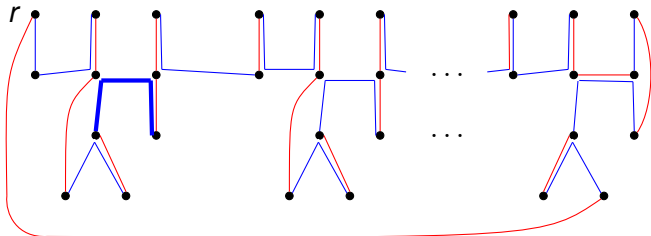
- 1 Pick a root vertex r (arbitrarily).

Step 5: Conflict Resolution in the Paths of T



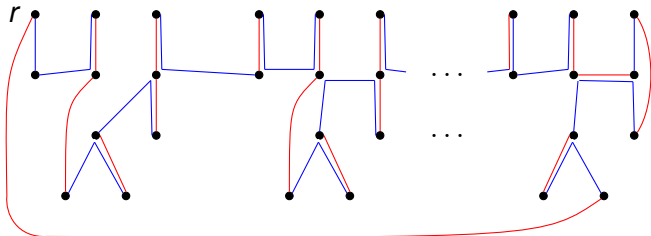
- 1 Pick a root vertex r (arbitrarily).
- 2 In every blue path p , bypass the vertex v closest to r in T if v is internal to p and if $\text{blue-deg}(v) \geq 4$.

Step 5: Conflict Resolution in the Paths of T



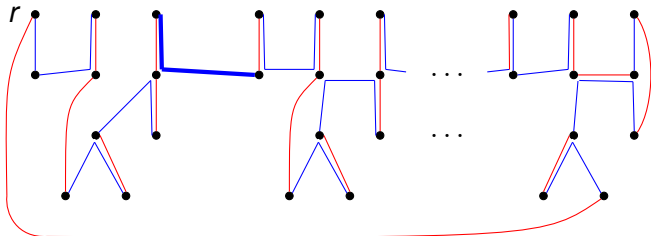
- 1 Pick a root vertex r (arbitrarily).
- 2 In every blue path p , bypass the vertex v closest to r in T if v is internal to p and if $\text{blue-deg}(v) \geq 4$.

Step 5: Conflict Resolution in the Paths of T



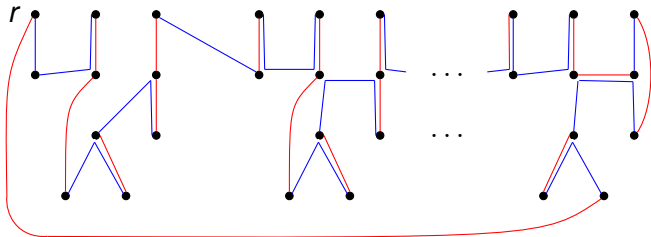
- 1 Pick a root vertex r (arbitrarily).
- 2 In every blue path p , bypass the vertex v closest to r in T if v is internal to p and if $\text{blue-deg}(v) \geq 4$.

Step 5: Conflict Resolution in the Paths of T



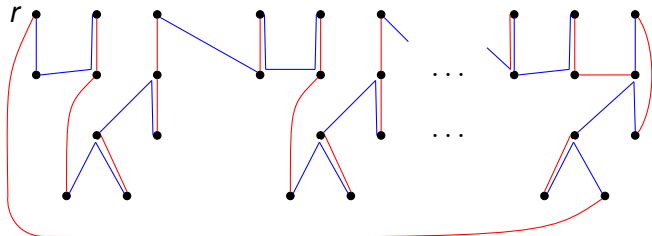
- 1 Pick a root vertex r (arbitrarily).
- 2 In every blue path p , bypass the vertex v closest to r in T if v is internal to p and if $\text{blue-deg}(v) \geq 4$.

Step 5: Conflict Resolution in the Paths of T



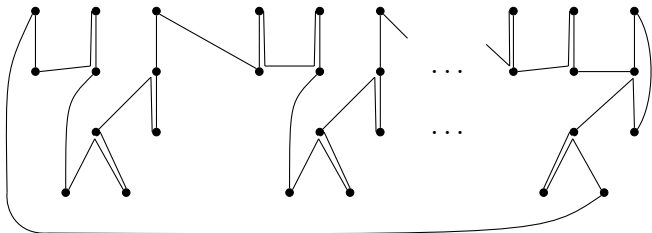
- 1 Pick a root vertex r (arbitrarily).
- 2 In every blue path p , bypass the vertex v closest to r in T if v is internal to p and if $\text{blue-deg}(v) \geq 4$.

Step 5: Conflict Resolution in the Paths of T



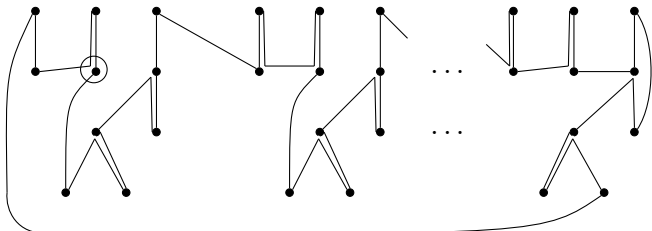
- 1 Pick a root vertex r (arbitrarily).
- 2 In every blue path p , bypass the vertex v closest to r in T if v is internal to p and if $\text{blue-deg}(v) \geq 4$.

Step 6: Resolution of Remaining Conflicts



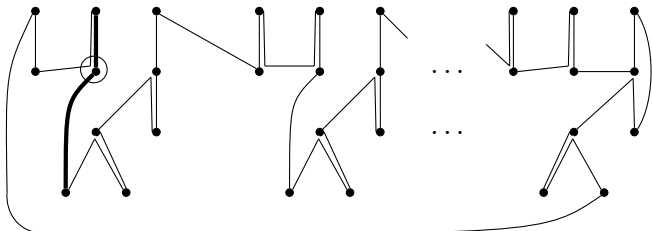
- 1 Bypass an arbitrary conflict v .

Step 6: Resolution of Remaining Conflicts



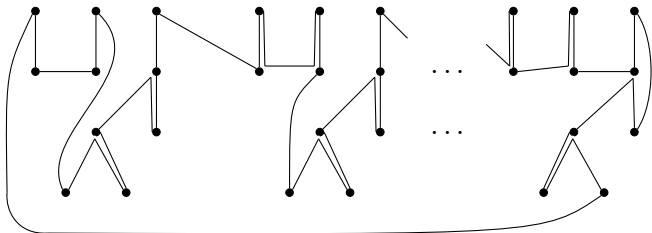
- 1 Bypass an arbitrary conflict v .

Step 6: Resolution of Remaining Conflicts



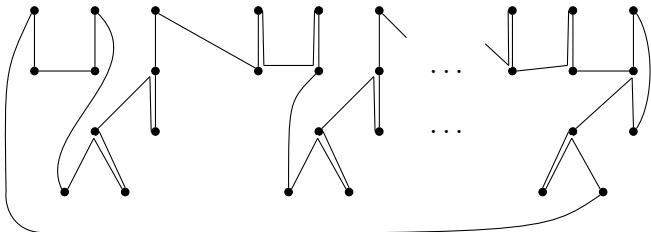
- 1 Bypass an arbitrary conflict v .

Step 6: Resolution of Remaining Conflicts



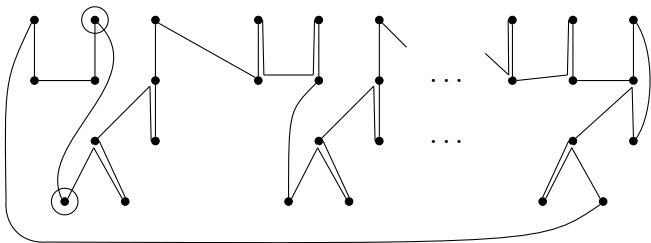
- 1 Bypass an arbitrary conflict v .

Step 6: Resolution of Remaining Conflicts



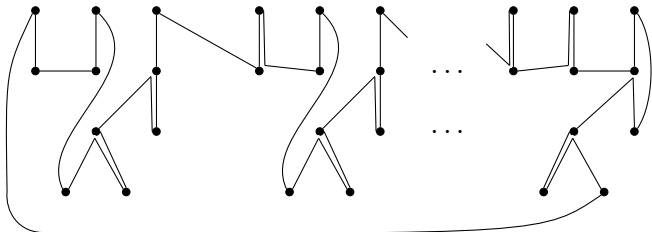
- 1 Bypass an arbitrary conflict v .
- 2 While there are conflicts: If v has conflicts as neighbors, let w be one of them. Let $v := w'$ and bypass v . Else let v be an arbitrary conflict and bypass v .

Step 6: Resolution of Remaining Conflicts



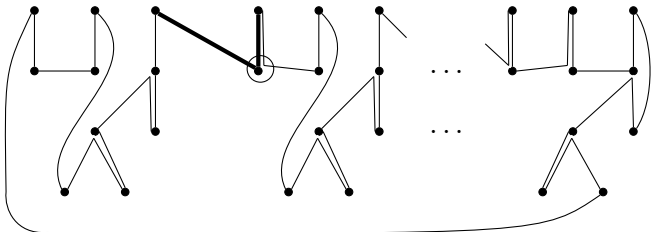
- 1 Bypass an arbitrary conflict v .
- 2 While there are conflicts: If v has conflicts as neighbors, let w be one of them. Let $v := w'$ and bypass v . Else let v be an arbitrary conflict and bypass v .

Step 6: Resolution of Remaining Conflicts



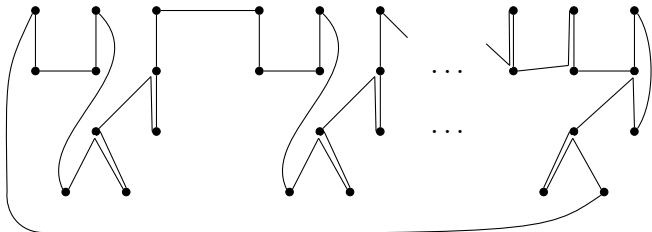
- 1 Bypass an arbitrary conflict v .
- 2 While there are conflicts: If v has conflicts as neighbors, let w be one of them. Let $v := w'$ and bypass v . Else let v be an arbitrary conflict and bypass v .

Step 6: Resolution of Remaining Conflicts



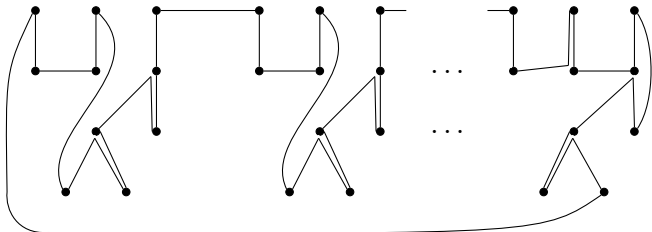
- 1 Bypass an arbitrary conflict v .
- 2 While there are conflicts: If v has conflicts as neighbors, let w be one of them. Let $v := w'$ and bypass v . Else let v be an arbitrary conflict and bypass v .

Step 6: Resolution of Remaining Conflicts



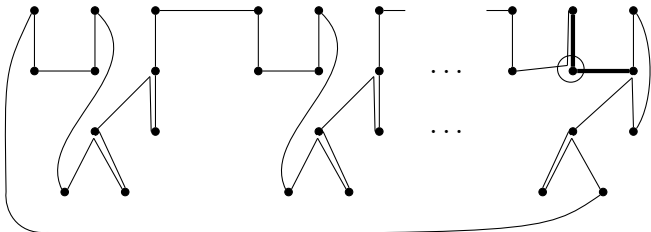
- 1 Bypass an arbitrary conflict v .
- 2 While there are conflicts: If v has conflicts as neighbors, let w be one of them. Let $v := w'$ and bypass v . Else let v be an arbitrary conflict and bypass v .

Step 6: Resolution of Remaining Conflicts



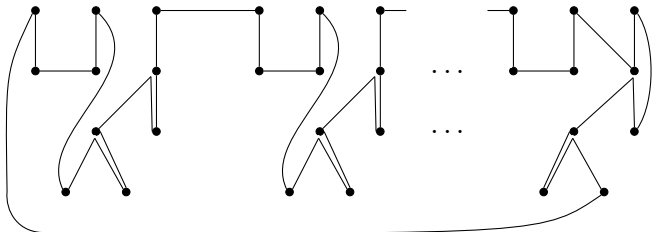
- 1 Bypass an arbitrary conflict v .
- 2 While there are conflicts: If v has conflicts as neighbors, let w be one of them. Let $v := w'$ and bypass v . Else let v be an arbitrary conflict and bypass v .

Step 6: Resolution of Remaining Conflicts



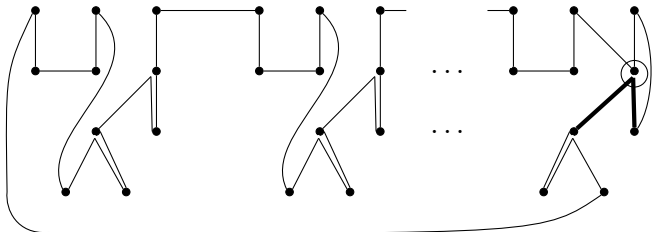
- 1 Bypass an arbitrary conflict v .
- 2 While there are conflicts: If v has conflicts as neighbors, let w be one of them. Let $v := w'$ and bypass v . Else let v be an arbitrary conflict and bypass v .

Step 6: Resolution of Remaining Conflicts



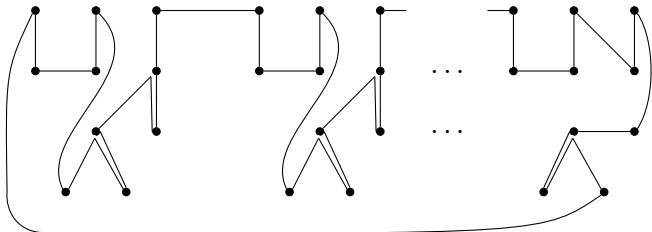
- 1 Bypass an arbitrary conflict v .
- 2 While there are conflicts: If v has conflicts as neighbors, let w be one of them. Let $v := w'$ and bypass v . Else let v be an arbitrary conflict and bypass v .

Step 6: Resolution of Remaining Conflicts



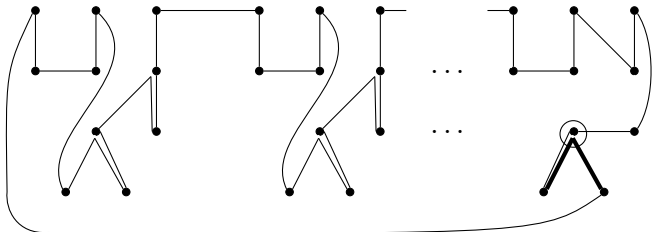
- 1 Bypass an arbitrary conflict v .
- 2 While there are conflicts: If v has conflicts as neighbors, let w be one of them. Let $v := w'$ and bypass v . Else let v be an arbitrary conflict and bypass v .

Step 6: Resolution of Remaining Conflicts



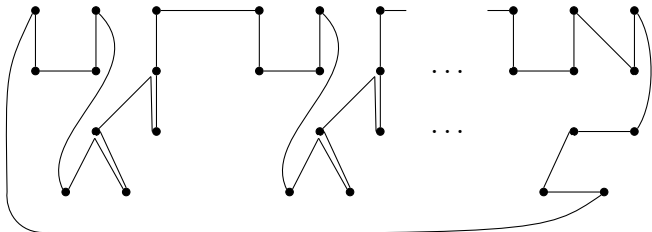
- 1 Bypass an arbitrary conflict v .
- 2 While there are conflicts: If v has conflicts as neighbors, let w be one of them. Let $v := w'$ and bypass v . Else let v be an arbitrary conflict and bypass v .

Step 6: Resolution of Remaining Conflicts



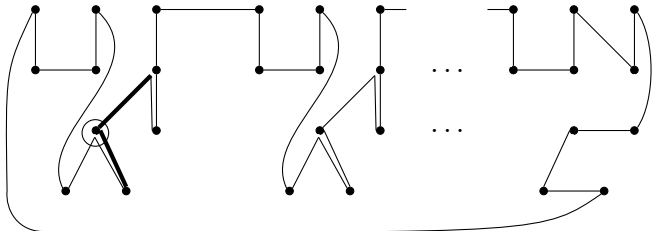
- 1 Bypass an arbitrary conflict v .
- 2 While there are conflicts: If v has conflicts as neighbors, let w be one of them. Let $v := w'$ and bypass v . Else let v be an arbitrary conflict and bypass v .

Step 6: Resolution of Remaining Conflicts



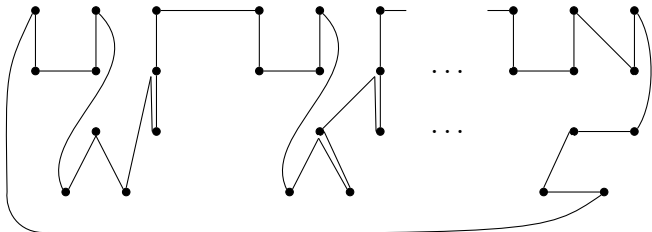
- 1 Bypass an arbitrary conflict v .
- 2 While there are conflicts: If v has conflicts as neighbors, let w be one of them. Let $v := w'$ and bypass v . Else let v be an arbitrary conflict and bypass v .

Step 6: Resolution of Remaining Conflicts



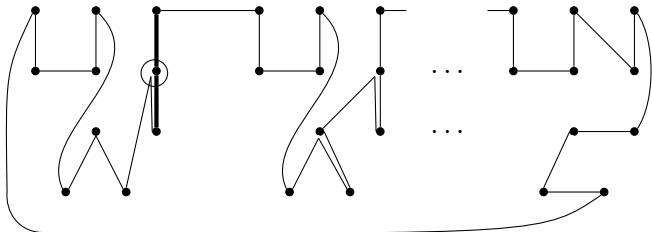
- 1 Bypass an arbitrary conflict v .
- 2 While there are conflicts: If v has conflicts as neighbors, let w be one of them. Let $v := w'$ and bypass v . Else let v be an arbitrary conflict and bypass v .

Step 6: Resolution of Remaining Conflicts



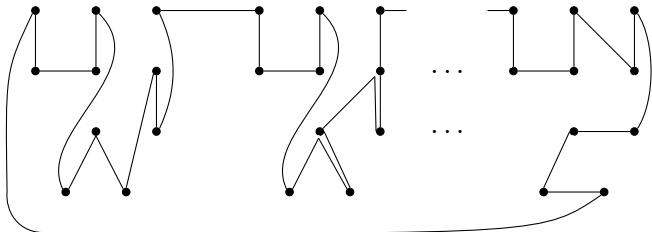
- 1 Bypass an arbitrary conflict v .
- 2 While there are conflicts: If v has conflicts as neighbors, let w be one of them. Let $v := w'$ and bypass v . Else let v be an arbitrary conflict and bypass v .

Step 6: Resolution of Remaining Conflicts



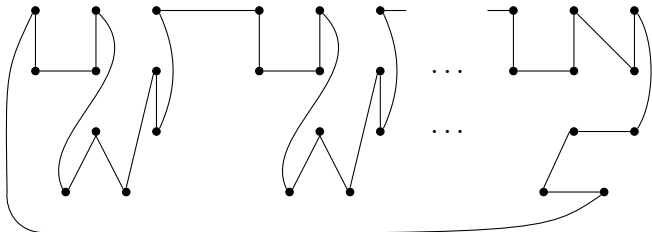
- 1 Bypass an arbitrary conflict v .
- 2 While there are conflicts: If v has conflicts as neighbors, let w be one of them. Let $v := w'$ and bypass v . Else let v be an arbitrary conflict and bypass v .

Step 6: Resolution of Remaining Conflicts



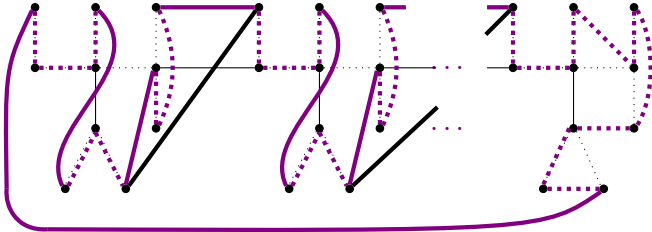
- 1 Bypass an arbitrary conflict v .
- 2 While there are conflicts: If v has conflicts as neighbors, let w be one of them. Let $v := w'$ and bypass v . Else let v be an arbitrary conflict and bypass v .

Step 6: Resolution of Remaining Conflicts



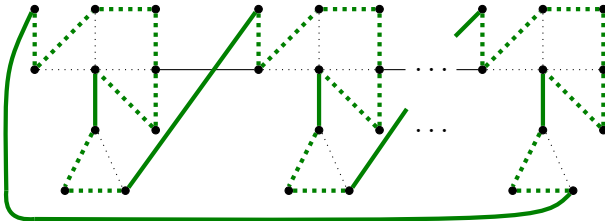
- 1 Bypass an arbitrary conflict v .
- 2 While there are conflicts: If v has conflicts as neighbors, let w be one of them. Let $v := w'$ and bypass v . Else let v be an arbitrary conflict and bypass v .

Cost of this Solution



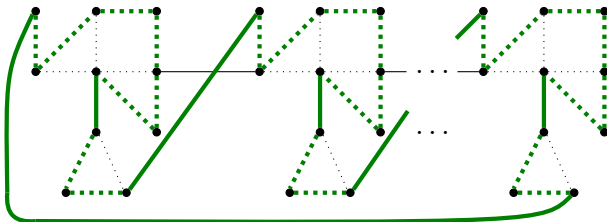
- Overall cost $\geq 3 \cdot (k - 1) \cdot \beta^2$

Cost of this Solution



- Overall cost $\geq 3 \cdot (k - 1) \cdot \beta^2$
- Cost of this solution: $2k + 2\beta^2 + 7\beta + 6$

Cost of this Solution



- Overall cost $\geq 3 \cdot (k - 1) \cdot \beta^2$
- Cost of this solution: $2k + 2\beta^2 + 7\beta + 6$
- $\forall \varepsilon > 0 : \frac{3 \cdot (k-1) \cdot \beta^2}{2k + 2\beta^2 + 7\beta + 6} \geq \frac{3}{2}\beta^2 - \varepsilon$, for sufficiently large k

The End

Thank you!