An approximation scheme for Planar Graph TSP

Thomas Grzanna

December 17, 2013

Traveling Salesman Problem

Definition (Traveling Salesman Problem (TSP))

Given an undirected, complete graph G with (symmetric) positive edge cost function c, find a minimum cost tour that visits all vertices exactly once and returns to its origin.

Traveling Salesman Problem

Definition (Traveling Salesman Problem (TSP))

Given an undirected, complete graph G with (symmetric) positive edge cost function c, find a minimum cost tour that visits all vertices exactly once and returns to its origin.

Special among NP-complete problems, often used to test new algorithmic ideas.
The decision problem versions of all TSP variants presented here are NP-hard (and we assume $P \neq NP$).

Traveling Salesman Problem

Definition (Traveling Salesman Problem (TSP))

Given an undirected, complete graph G with (symmetric) positive edge cost function c, find a minimum cost tour that visits all vertices exactly once and returns to its origin.

Special among NP-complete problems, often used to test new algorithmic ideas.
The decision problem versions of all TSP variants presented here are NP-hard (and we assume $P \neq NP$).

The general TSP is **not** approximable [BuK].

Approximation

Given a minimization problem and an optimal solution value OPT,

- the problem is α-**approximable**, if there is $\alpha > 1$ and a polynomial time algorithm that computes a solution with value at most $\alpha \cdot OPT$;
- the problem has a **polynomial-time approximation scheme (PTAS)**, if for any $\epsilon > 0$ it can be approximated in time $n^{\mathcal{O}(1/\epsilon)}$ with solution value at most $(1+\epsilon) \cdot OPT$ using that scheme.
  If $\epsilon = \frac{1}{k}$, the solution is at most k-**optimal**.

Metric TSP (M-TSP)

Definition (Metric TSP (M-TSP))

The Metric TSP is a TSP with additional conditions:
For adjacent vertices u, v, w:

- $c(u, u) = 0$ (no loops)
- $c(u, v) = c(v, u)$ (symmetry)
- $c(u, w) \leq c(u, v) + c(v, w)$ (triangle inequality)

- more practically relevant than general TSP
- $\frac{3}{2}$-approximable [Christofides '76]
- $\frac{220}{219}$ is lower bound for approximation factor α [PV '06]

(Planar) Graph TSP

Definition (Graph TSP)

The Graph TSP is a special case of the Metric TSP with cost 1 for all (non-loop) edges.

(Planar) Graph TSP

Definition (Graph TSP)

The Graph TSP is a special case of the Metric TSP with cost 1 for all (non-loop) edges.

For planar graphs, the definition is slightly different:

(Planar) Graph TSP

Definition (Graph TSP)

The Graph TSP is a special case of the Metric TSP with cost 1 for all (non-loop) edges.

For planar graphs, the definition is slightly different:

Definition (Planar Graph TSP)

Given an undirected, *planar* graph G with metric cost function c and cost 1 for all (non-loop) edges, find a minimum cost tour that visits all vertices *at least* once and returns to its origin.

Motivation: Euclidean TSP (E-TSP)

Definition (Euclidean TSP (E-TSP))

The Euclidean TSP is a special case of the metric TSP where c is given by the ordinary euclidean distance on a plane.

Motivation: Euclidean TSP (E-TSP)

Definition (Euclidean TSP (E-TSP))

The Euclidean TSP is a special case of the metric TSP where c is given by the ordinary euclidean distance on a plane.

- designers of planar graph PTAS considered it a step towards a PTAS for E-TSP

Motivation: Euclidean TSP (E-TSP)

Definition (Euclidean TSP (E-TSP))

The Euclidean TSP is a special case of the metric TSP where c is given by the ordinary euclidean distance on a plane.

- designers of planar graph PTAS considered it a step towards a PTAS for E-TSP
- major result: there is in fact a PTAS for E-TSP [Arora/Mitchell '98]

What is the goal?

We wish to obtain a polynomial-time approximation scheme for Planar Graph TSP:

Given a planar graph G with n vertices and parameter $\epsilon > 0$, the algorithm must

What is the goal?

We wish to obtain a polynomial-time approximation scheme for Planar Graph TSP:

Given a planar graph G with n vertices and parameter $\epsilon > 0$, the algorithm must

- run in $n^{\mathcal{O}(1/\epsilon)}$ time and

What is the goal?

We wish to obtain a polynomial-time approximation scheme for Planar Graph TSP:

Given a planar graph G with n vertices and parameter $\epsilon > 0$, the algorithm must

- run in $n^{\mathcal{O}(1/\epsilon)}$ time and
- compute a TSP tour of length at most $(1+\epsilon) \cdot OPT$.

What is the goal?

We wish to obtain a polynomial-time approximation scheme for Planar Graph TSP:

Given a planar graph G with n vertices and parameter $\epsilon > 0$, the algorithm must

- run in $n^{\mathcal{O}(1/\epsilon)}$ time and
- compute a TSP tour of length at most $(1+\epsilon) \cdot OPT$. Since $n \leq OPT$, we have $OPT + \epsilon n \leq (1+\epsilon)OPT$, so it suffices to stay within an **additive error** of ϵn.

The algorithm

The algorithm presented here is from the paper *An Approximation Scheme for Planar Graph TSP* by GRIGNI, KOUTSOUPIAS and PAPADIMITRIOU, published in '95.

Remark: Baker's framework (last week) cannot be applied to the TSP; however, there is a PTAS for Planar Graph TSP that modifies the framework and even runs in linear time [Klein, '05/'08].

Circle separators and face-edges

Circle separators, or **simple cycle separators**, partition the graph into

- an interior part A,
- an exterior part B
- and a circle C,

w.r.t. some size constraints.

Circle separators and face-edges

Circle separators, or **simple cycle separators**, partition the graph into

- an interior part A,
- an exterior part B
- and a circle C,

w.r.t. some size constraints.

A **face-edge** is a *virtual* edge through a face.
We will allow separators to use such edges (to some extend).

First approach: Miller's theorem

Theorem (Simple cycle separator, Miller)

Let H be a 2-connected planar graph with n vertices, edge weights and a maximum face size d.
Then H has a simple cycle separator C consisting of $\mathcal{O}(\sqrt{nd})$ edges, the interior and exterior of C both have at most $\frac{2}{3}n$ vertices and C can be found in polynomial time.

First approach: Miller's theorem

Theorem (Simple cycle separator, Miller)

Let H be a 2-connected planar graph with n vertices, edge weights and a maximum face size d.
Then H has a simple cycle separator C consisting of $\mathcal{O}(\sqrt{nd})$ edges, the interior and exterior of C both have at most $\frac{2}{3}n$ vertices and C can be found in polynomial time.

- attempts using this resulted in $n^{\mathcal{O}((log^2 n)/\epsilon^2)}$ complexity

First approach: Miller's theorem

Theorem (Simple cycle separator, Miller)

Let H be a 2-connected planar graph with n vertices, edge weights and a maximum face size d.
Then H has a simple cycle separator C consisting of $\mathcal{O}(\sqrt{nd})$ edges, the interior and exterior of C both have at most $\frac{2}{3}n$ vertices and C can be found in polynomial time.

- attempts using this resulted in $n^{\mathcal{O}((log^2 n)/\epsilon^2)}$ complexity
- a "more customizeable" theorem was needed

Final approach: A novel separator theorem

Theorem (Simple cycle separator with vertex weights)

Let H be a connected planar graph with n vertices, vertex weights and parameter f with $1 \leq f \leq \sqrt{n}$.
Then H has a simple cycle separator C through $\mathcal{O}(n/f)$ vertices, the interior and exterior of C both have at most $\frac{2}{3}$ of the total weight, C uses at most f face-edges and C can be found in polynomial (nearly linear) time.

Final approach: A novel separator theorem

Theorem (Simple cycle separator with vertex weights)

Let H be a connected planar graph with n vertices, vertex weights and parameter f with $1 \leq f \leq \sqrt{n}$.
Then H has a simple cycle separator C through $\mathcal{O}(n/f)$ vertices, the interior and exterior of C both have at most $\frac{2}{3}$ of the total weight, C uses at most f face-edges and C can be found in polynomial (nearly linear) time.

- f controls trade-off between size of C and amount of face-edges in C

Final approach: A novel separator theorem

Theorem (Simple cycle separator with vertex weights)

Let H be a connected planar graph with n vertices, vertex weights and parameter f with $1 \leq f \leq \sqrt{n}$.
Then H has a simple cycle separator C through $\mathcal{O}(n/f)$ vertices, the interior and exterior of C both have at most $\frac{2}{3}$ of the total weight, C uses at most f face-edges and C can be found in polynomial (nearly linear) time.

- f controls trade-off between size of C and amount of face-edges in C
- choice of f is crucial for efficiency of the algorithm

Where are the differences?

Theorem (Simple cycle separator, Miller)

Let H be a 2-connected planar graph with n vertices, edge weights and a maximum face size d.
Then H has a simple cycle separator C consisting of $\mathcal{O}(\sqrt{nd})$ edges, the interior and exterior of C both have at most $\frac{2}{3}n$ vertices and C can be found in polynomial time.

Where are the differences?

Theorem (Simple cycle separator with vertex weights)

Let H be a connected planar graph with n vertices, vertex weights and parameter f with $1 \leq f \leq \sqrt{n}$.
Then H has a simple cycle separator C through $\mathcal{O}(n/f)$ vertices, the interior and exterior of C both have at most $\frac{2}{3}$ of the total weight, C uses at most f face-edges and C can be found in polynomial (nearly linear) time.

Where are the differences?

Theorem (Simple cycle separator with vertex weights)

Let H be a connected planar graph with n vertices, vertex weights and parameter f with $1 \leq f \leq \sqrt{n}$.
Then H has a simple cycle separator C through $\mathcal{O}(n/f)$ vertices, the interior and exterior of C both have at most $\frac{2}{3}$ of the total weight, C uses at most f face-edges and C can be found in polynomial (nearly linear) time.

- more flexible separator parametrization

Where are the differences?

Theorem (Simple cycle separator with vertex weights)

*Let H be a connected planar graph with n vertices, vertex weights and parameter f with $1 \leq f \leq \sqrt{n}$.
Then H has a simple cycle separator C through $\mathcal{O}(n/f)$ vertices, the interior and exterior of C both have at most $\frac{2}{3}$ of the total weight, C uses at most f face-edges and C can be found in polynomial (nearly linear) time.*

- more flexible separator parametrization
- occurence of heavy-weighted vertices can be limited in exterior/interior parts

Decomposition - Principle

Input:

Connected graph G with planar embedding, vertex weights 1.

Decomposition - Principle

Input:
Connected graph G with planar embedding, vertex weights 1.
We choose $f = \Theta((logn)/\epsilon)$.

Decomposition - Principle

Input:
Connected graph G with planar embedding, vertex weights 1.
We choose $f = \Theta((logn)/\epsilon)$.

We decompose a given planar graph H into so-called *contracted subgraphs* H_1 and H_2. Start with $H = G$.

Decomposition - Principle

Input:
Connected graph G with planar embedding, vertex weights 1.
We choose $f = \Theta((logn)/\epsilon)$.

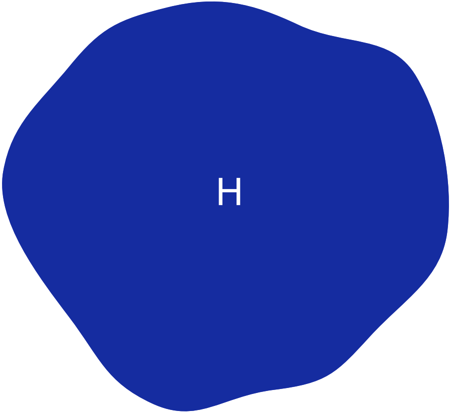
We decompose a given planar graph H into so-called *contracted subgraphs* H_1 and H_2. Start with $H = G$.

Steps:

1. Applying the separator
2. Contracting path segments
3. Removing face-edges
4. Weighting constraint points
5. Repeating decomposition recursively

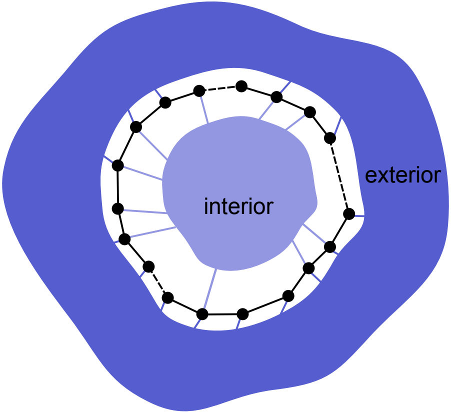
Decomposition

1. Applying the separator



Decomposition

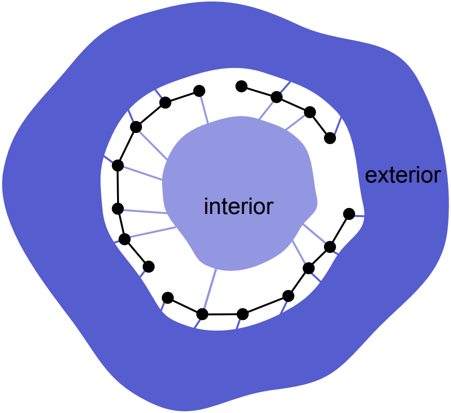
1. Applying the separator



Decomposition

2. Removing face-edges

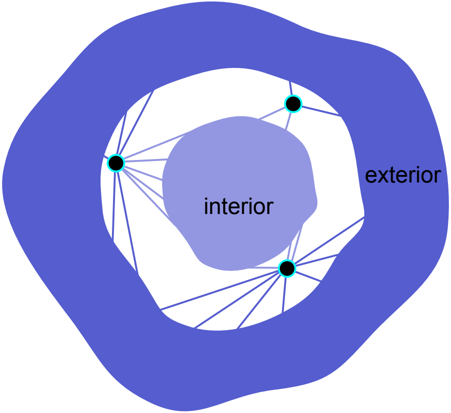
Removing at most f face-edges results in at most f path segments.



Decomposition

3. Contracting path segments

   The resulting nodes are called **constraint points (CPs)**.
   Result: H'

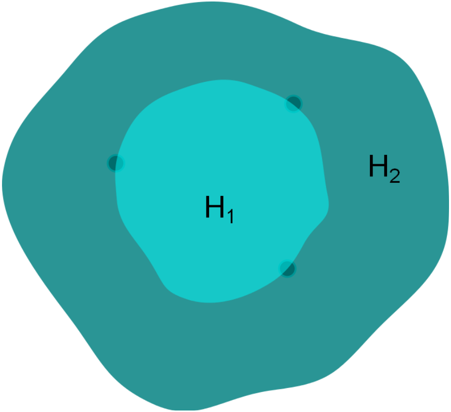


Decomposition

3. Contracting path segments

   The resulting nodes are called **constraint points (CPs)**.
   Result: H'

   Contracted subgraphs H_1 and H_2 share the new CPs created in this step.



Decomposition

4. Weighting constraint points

Let $W(H)$ be the total weight of H.

Decomposition

4. Weighting constraint points

Let $W(H)$ be the total weight of H.
In H_1 and H_2, we assign weight $\frac{W(H)}{6f}$ to each CP.

Decomposition

4. Weighting constraint points

   Let $W(H)$ be the total weight of H.
   In H_1 and H_2, we assign weight $\frac{W(H)}{6f}$ to each CP.

5. Repeating decomposition recursively

   Decompose H_1 and H_2 using the presented steps.

Decomposition

4. Weighting constraint points

   Let $W(H)$ be the total weight of H.
   In H_1 and H_2, we assign weight $\frac{W(H)}{6f}$ to each CP.

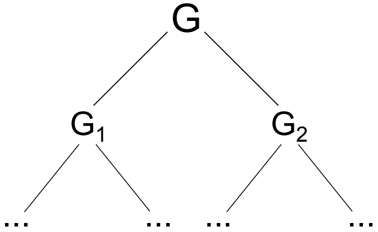
   Important: CPs from *previous* decomposition steps are also re-weighted!

5. Repeating decomposition recursively

   Decompose H_1 and H_2 using the presented steps.

Decomposition tree

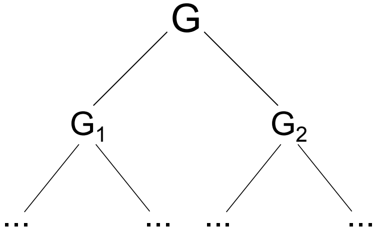
The binary **decomposition tree** $\mathcal{T}$ stores the decomposition results.



Decomposition tree

The binary **decomposition tree** $\mathcal{T}$ stores the decomposition results.

Edges represent recursive decomposition steps.

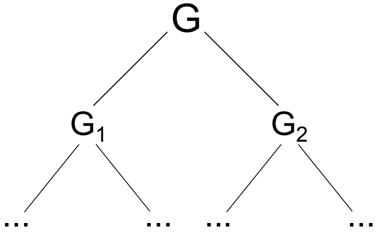


Decomposition tree

The binary **decomposition tree** $\mathcal{T}$ stores the decomposition results.

Edges represent recursive decomposition steps.

Stopping size:
$S = \Theta(f^2) = \Theta((log^2 n)/\epsilon^2)$
If $|H| \leq S$, stop recursion - H is a leaf.



Tree size and complexity

Some observations:

1. For all H in $\mathcal{T}$: $W(H_i) \leq \frac{5}{6}W(H)$.

Tree size and complexity

Some observations:

1. For all H in $\mathcal{T}$: $W(H_i) \leq \frac{5}{6}W(H)$.

Proof.

$W(H_i) \leq \frac{2}{3}W(H) + f \cdot \frac{W(H)}{6f} = \frac{5}{6}W(H)$. □

Tree size and complexity

Some observations:

1. For all H in $\mathcal{T}$: $W(H_i) \leq \frac{5}{6}W(H)$.

Proof.

$W(H_i) \leq \frac{2}{3}W(H) + f \cdot \frac{W(H)}{6f} = \frac{5}{6}W(H)$. □

2. For all H in $\mathcal{T}$: H contains at most $5f$ CPs.

Tree size and complexity

Some observations:

1. For all H in $\mathcal{T}$: $W(H_i) \leq \frac{5}{6}W(H)$.

Proof.

$W(H_i) \leq \frac{2}{3}W(H) + f \cdot \frac{W(H)}{6f} = \frac{5}{6}W(H)$. □

2. For all H in $\mathcal{T}$: H contains at most $5f$ CPs.

Proof.

Suppose $W(H_i) = x \cdot \frac{W(H)}{6f}$, with x being #CPs in H_i.
Then $x \cdot \frac{W(H)}{6f} \leq \frac{5}{6}W(H) \Leftrightarrow x \leq 5f$. □

Tree size and complexity

3. $\mathcal{T}$ has depth at most $D = log_{(\frac{6}{5})}n = \mathcal{O}(logn)$ (without proof).

Tree size and complexity

3. $\mathcal{T}$ has depth at most $D = log_{(\frac{6}{5})}n = \mathcal{O}(logn)$ (without proof).

Thus $\mathcal{T}$ has polynomial size, independent of ϵ.

Each decomposition step can be done in polynomial time, so the overall complexity of decomposition is $n^{\mathcal{O}(1)}$.

Approximation

Input:

Decomposition tree $\mathcal{T}$, parameter f, stopping size S.

Approximation

Input:
Decomposition tree $\mathcal{T}$, parameter f, stopping size S.

We compute a set of approximate solutions for every leaf of $\mathcal{T}$ and successively merge child graph solutions while going up the tree - but these will not be TSP solutions.

Approximation

Input:
Decomposition tree $\mathcal{T}$, parameter f, stopping size S.

We compute a set of approximate solutions for every leaf of $\mathcal{T}$ and successively merge child graph solutions while going up the tree - but these will not be TSP solutions.

Steps (for all inner nodes H):

1. Approximating leaf graph solutions
2. Building solutions in H'
3. Extending H'-solutions to H-solutions
4. Constructing the tour in root G

Path Covering

Definition (Path Cover)

Given a graph H and a set of chosen CPs in H, find the minimum length collection of paths that covers all vertices of H using each chosen CP as a path endpoint exactly once.

A path cover with no endpoints (0 is even) shall be a cycle.

Storing solutions

Every path has two endpoints, so we only consider even subsets of CPs in H.

Storing solutions

Every path has two endpoints, so we only consider even subsets of CPs in H.

Let $c(H) \leq 5f$ be #CPs in H. We identify a choice of CPs with a binary array $x \in \{0,1\}^{c(H)}$ and store the corresponding solution in $T(H)[x]$.

Storing solutions

Every path has two endpoints, so we only consider even subsets of CPs in H.

Let $c(H) \leq 5f$ be #CPs in H. We identify a choice of CPs with a binary array $x \in \{0,1\}^{c(H)}$ and store the corresponding solution in $T(H)[x]$.

$T(H)$ is a table of size $2^{c(H)-1} \leq 2^{5f-1} = 2^{5\Theta((logn)/\epsilon)-1} = n^{\mathcal{O}(1/\epsilon)}$.

Approximation in leaf graphs

Reminder: Leaf graphs L of $\mathcal{T}$ have size $|L| \leq S = \Theta((log^2 n)/\epsilon^2)$.

Approximation in leaf graphs

Reminder: Leaf graphs L of $\mathcal{T}$ have size $|L| \leq S = \Theta((log^2 n)/\epsilon^2)$.

Using a simpler approximation scheme based on the Lipton-Tarjan separator theorem and so-called *nonserial dynamic programming*, one can approximate path covers for the leaf graphs in time $2^{\mathcal{O}(\sqrt{|L|})} = n^{\mathcal{O}(1/\epsilon)}$.

Merging child graph solutions

The situation: For an inner graph H, we want to find a path cover approximation that is as small as possible.

Merging child graph solutions

The situation: For an inner graph H, we want to find a path cover approximation that is as small as possible.

1. For every choice x of endpoint CPs in H, consider contraction x' in H'

Merging child graph solutions

The situation: For an inner graph H, we want to find a path cover approximation that is as small as possible.

1. For every choice x of endpoint CPs in H, consider contraction x' in H'
2. Find solutions x_1 and x_2 for H_1 and H_2 such that their combination in H' is a minimal length solution matching x'

Merging child graph solutions

The situation: For an inner graph H, we want to find a path cover approximation that is as small as possible.

1. For every choice x of endpoint CPs in H, consider contraction x' in H'
2. Find solutions x_1 and x_2 for H_1 and H_2 such that their combination in H' is a minimal length solution matching x'
3. Extend the solution in H' to a solution in H

Merging child graph solutions

The situation: For an inner graph H, we want to find a path cover approximation that is as small as possible.

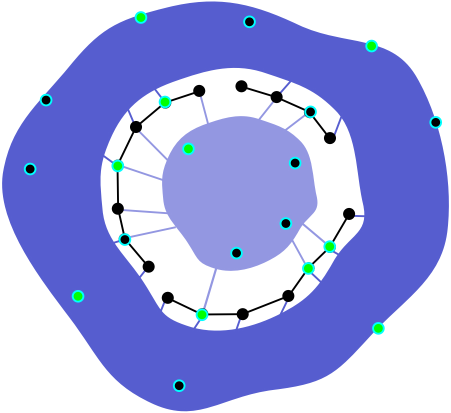
1. For every choice x of endpoint CPs in H, consider contraction x' in H'
2. Find solutions x_1 and x_2 for H_1 and H_2 such that their combination in H' is a minimal length solution matching x'
3. Extend the solution in H' to a solution in H

We will look at this process for a given x and H.

Merging in H'

1. For a choice x of endpoint CPs in H, consider contraction x' in H'

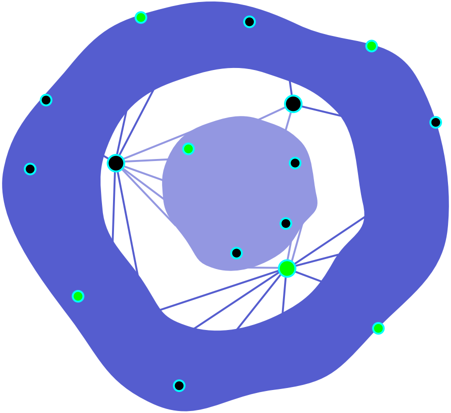
   New CP shall be endpoint in x' iff. path segment contained odd number of endpoints in x.



Merging in H'

1. For a choice x of endpoint CPs in H, consider contraction x' in H'

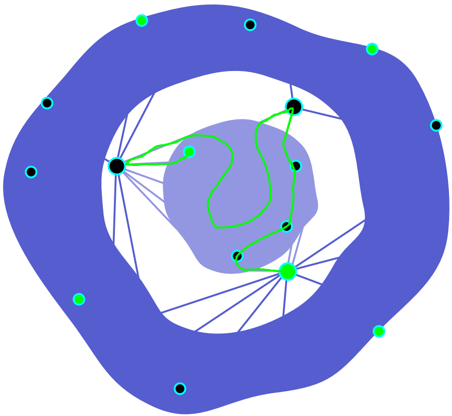
   New CP shall be endpoint in x' iff. path segment contained odd number of endpoints in x.



Merging in H'

2. Find solutions x_1 and x_2 for H_1 and H_2 such that their combination in H' is a minimal length solution matching x'

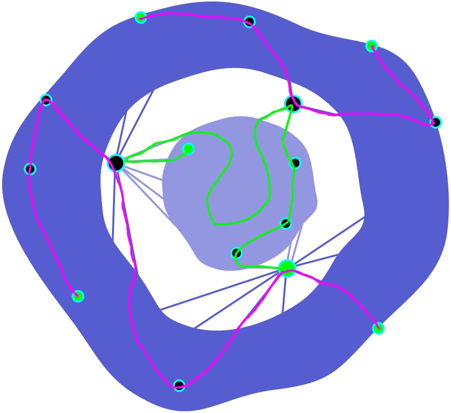
   Approach: Choose x_1 that matches x'..



Merging in H'

2. Find solutions x_1 and x_2 for H_1 and H_2 such that their combination in H' is a minimal length solution matching x'

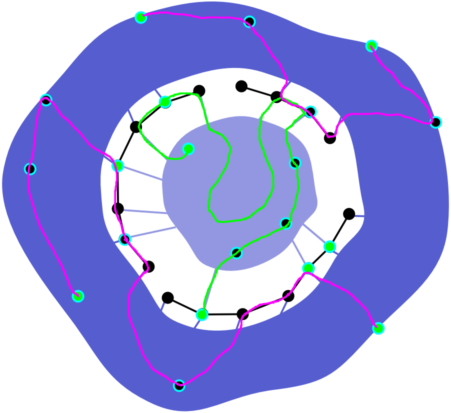
   Approach: Choose x_1 that matches x'.. .. then pick the shortest matching x_2.



Extending to H

3. Extend the solution in H' to a solution in H

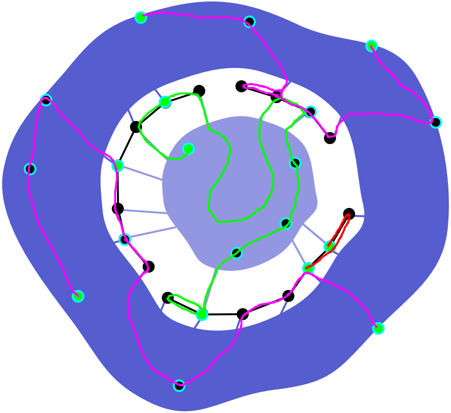
   Usually requires some additional operations, the paper did not provide any details here.



Extending to H

3. Extend the solution in H' to a solution in H

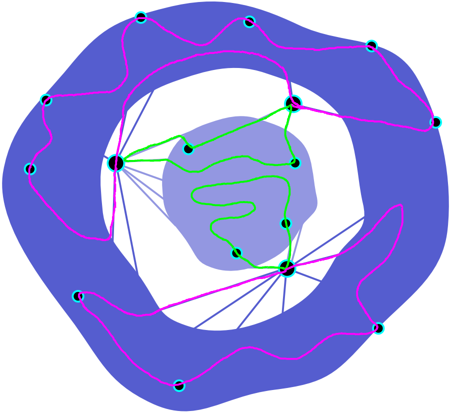
   Usually requires some additional operations, the paper did not provide any details here.



Special case: Input graph G

G' does not contain "old" CPs - build solution for empty x: a tour.

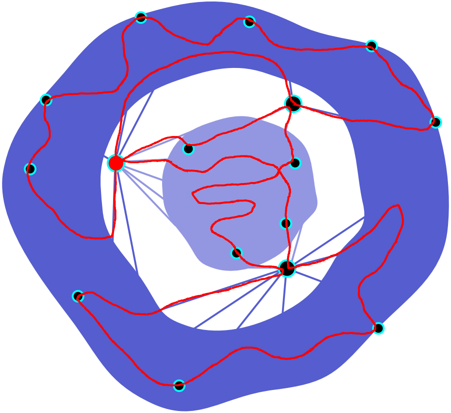
Since G is connected and all vertices are covered, it's possible to connect multiple tours to get a single tour.



Special case: Input graph G

G' does not contain "old" CPs - build solution for empty x: a tour.

Since G is connected and all vertices are covered, it's possible to connect multiple tours to get a single tour.



Runtime summary

Decomposition:

- Single decomposition step: $n^{\mathcal{O}(1)}$
- Amount of decompositions done: $n^{\mathcal{O}(1)}$
- Decomposition complexity in total: $n^{\mathcal{O}(1)}$

Runtime summary

Decomposition:

- Single decomposition step: $n^{\mathcal{O}(1)}$
- Amount of decompositions done: $n^{\mathcal{O}(1)}$
- Decomposition complexity in total: $n^{\mathcal{O}(1)}$

Approximation:

- Single leaf approximation step: $n^{\mathcal{O}(1/\epsilon)} \cdot n^{\mathcal{O}(1/\epsilon)}$
- Single inner node approximation step: $n^{\mathcal{O}(1/\epsilon)} \cdot n^{\mathcal{O}(1/\epsilon)}$
- Approximation complexity in total: $n^{\mathcal{O}(1)} \cdot n^{\mathcal{O}(1/\epsilon)} = n^{\mathcal{O}(1/\epsilon)}$

Runtime summary

Decomposition:

- Single decomposition step: $n^{\mathcal{O}(1)}$
- Amount of decompositions done: $n^{\mathcal{O}(1)}$
- Decomposition complexity in total: $n^{\mathcal{O}(1)}$

Approximation:

- Single leaf approximation step: $n^{\mathcal{O}(1/\epsilon)} \cdot n^{\mathcal{O}(1/\epsilon)}$
- Single inner node approximation step: $n^{\mathcal{O}(1/\epsilon)} \cdot n^{\mathcal{O}(1/\epsilon)}$
- Approximation complexity in total: $n^{\mathcal{O}(1)} \cdot n^{\mathcal{O}(1/\epsilon)} = n^{\mathcal{O}(1/\epsilon)}$

Total runtime: $n^{\mathcal{O}(1/\epsilon)}$

Approximation error

The additive error can be shown to be at most ϵn, but this requires more detailed analysis than we can do here.

All good things come to an end

Thank you for your attention! Questions..?