Dynamic Vehicle Routing for Robotic Networks
Lecture #2: Preliminary Results in Combinatorics

Francesco Bullo[1] Emilio Frazzoli[2] Marco Pavone[2]
Ketan Savla[2] Stephen L. Smith[2]

[1]CCDC
University of California, Santa Barbara
bullo@engineering.ucsb.edu

[2]LIDS and CSAIL
Massachusetts Institute of Technology
{frazzoli,pavone,ksavla,slsmith}@mit.edu

Workshop at the 2010 American Control Conference
Baltimore, Maryland, USA, June 29, 2010, 8:30am to 5:00pm

Lecture outline

Key references for this lecture

Graph Theory Basics:
R. Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer, 2 edition, 2000

Combinatorial Optimization:
B. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms*, volume 21 of *Algorithmics and Combinatorics*. Springer, 4 edition, 2007

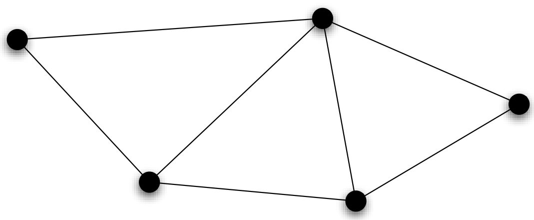
Stochastic TSP:
J. M. Steele. *Probability Theory and Combinatorial Optimization*. SIAM, 1987

Basic Queueing Theory:
L. Kleinrock. *Queueing Systems. Volume I: Theory*. Wiley, 1975

Outline

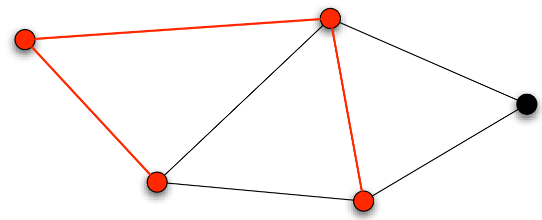
Graph Theory Review

- An undirected graph $G = (V, E)$.
- a **path** in G is a sequence $v_1, e_1, v_2, \ldots, v_k, e_k, v_{k+1}$, with
  - $e_i \neq e_j$ for $i \neq j$.
  - $v_i \neq v_j$ for all $i \neq j$.
- A **circuit** or **cycle** has $v_1 = v_{k+1}$.
- A **Hamiltonian path** is a path that contains all vertices.
- Similarly define a **Hamiltonian cycle** or **tour**.



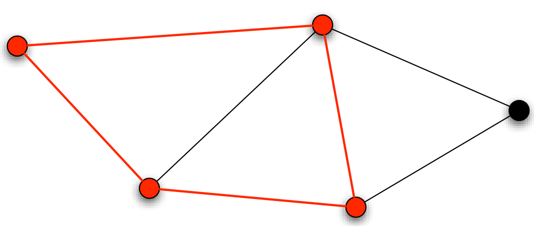
Graph Theory Review

- An undirected graph $G = (V, E)$.
- a **path** in G is a sequence $v_1, e_1, v_2, \ldots, v_k, e_k, v_{k+1}$, with
  - $e_i \neq e_j$ for $i \neq j$.
  - $v_i \neq v_j$ for all $i \neq j$.
- A **circuit** or **cycle** has $v_1 = v_{k+1}$.
- A **Hamiltonian path** is a path that contains all vertices.
- Similarly define a **Hamiltonian cycle** or **tour**.



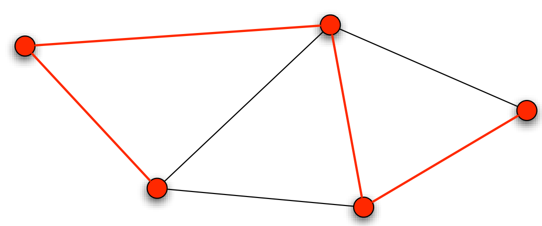
Graph Theory Review

- An undirected graph $G = (V, E)$.
- a **path** in G is a sequence $v_1, e_1, v_2, \ldots, v_k, e_k, v_{k+1}$, with
  - $e_i \neq e_j$ for $i \neq j$.
  - $v_i \neq v_j$ for all $i \neq j$.
- A **circuit** or **cycle** has $v_1 = v_{k+1}$.
- A **Hamiltonian path** is a path that contains all vertices.
- Similarly define a **Hamiltonian cycle** or **tour**.



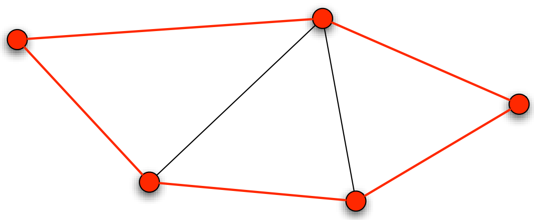
Graph Theory Review

- An undirected graph $G = (V, E)$.
- a **path** in G is a sequence $v_1, e_1, v_2, \ldots, v_k, e_k, v_{k+1}$, with
  - $e_i \neq e_j$ for $i \neq j$.
  - $v_i \neq v_j$ for all $i \neq j$.
- A **circuit** or **cycle** has $v_1 = v_{k+1}$.
- A **Hamiltonian path** is a path that contains all vertices.
- Similarly define a **Hamiltonian cycle** or **tour**.



Graph Theory Review

- An undirected graph $G = (V, E)$.
- a **path** in G is a sequence $v_1, e_1, v_2, \dots, v_k, e_k, v_{k+1}$, with
  - $e_i \neq e_j$ for $i \neq j$.
  - $v_i \neq v_j$ for all $i \neq j$.
- A **circuit** or **cycle** has $v_1 = v_{k+1}$.
- A **Hamiltonian path** is a path that contains all vertices.
- Similarly define a **Hamiltonian cycle** or **tour**.



Weighted Graphs

- A **weighted graph** $G = (V, E, c)$ has edge weights $c : E \to \mathbb{R}_{>0}$.
- In a **complete graph**, $E = V \times V$.

Special classes of **complete weighted graphs**:

- **Metric** if

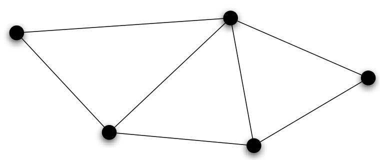
$$c(\{v_1, v_2\}) + c(\{v_2, v_3\}) \geq c(\{v_1, v_3\}) \text{ for all } v_1, v_2, v_3 \in V.$$

- **Euclidean** if

$$V \subset \mathbb{R}^d \quad \text{and} \quad c(\{v_i, v_j\}) = \|v_i - v_j\|_2.$$

Minimum Spanning Tree

- A **tree** is a graph with no cycles
- A **spanning tree** of G is a subgraph that
  1. is a tree
  2. connects all vertices together



Minimum Spanning Tree Problem

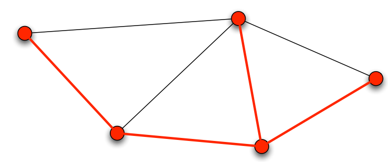
Given: a weighted graph $G = (V, E, c)$
Task: find a spanning tree $T = (E_T, V_T)$ such that $\sum_{e \in E_T} c(e)$ is minimum.

Can be solved in **greedy fashion** using **Kruskal's algorithm**:

- Recursively adds shortest edge that does not create a cycle
- Runs in $O(n^2)$ time (where $|V| = n$)

Minimum Spanning Tree

- A **tree** is a graph with no cycles
- A **spanning tree** of G is a subgraph that
  1. is a tree
  2. connects all vertices together



Minimum Spanning Tree Problem

Given: a weighted graph $G = (V, E, c)$
Task: find a spanning tree $T = (E_T, V_T)$ such that $\sum_{e \in E_T} c(e)$ is minimum.

Can be solved in **greedy fashion** using **Kruskal's algorithm**:

- Recursively adds shortest edge that does not create a cycle
- Runs in $O(n^2)$ time (where $|V| = n$)

Hamiltonian Cycle Decision Problem

Hamiltonian Cycle

Given: An undirected graph G.

Question: Does G contain a Hamiltonian cycle?

Hamiltonian Cycle is **NP-complete**

(One of Karp's 21 NP-complete problems)

Recall, a problem is NP-complete if

- Every solution can be **verified** in polynomial time (NP).
- Every problem in NP can be **reduced** to it.

Outline

Traveling Salesman Problem

Traveling Salesman Problem (TSP)

Given: A complete graph $G_n = (V_n, E_n)$ and weights $c : E_n \to \mathbb{R}_{>0}$.

Task: Find a Hamiltonian cycle with minimum weight.

- TSP is **NP-hard**
- To show NP-hard: Reduce Hamiltonian Cycle to TSP.

Given an undirected graph $G = (V, E)$ with $|V| = n$:

1. Construct complete graph G_n with weight 1 for each edge in E and weight 2 for all other edges.
2. Then G is Hamiltonian $\Leftrightarrow$ optimum TSP tour has length n.

Approximation Algorithms for the TSP

Theorem (Sahni and Gonzalez, 1976)

Unless $P = NP$, there is no k-factor approx alg for the TSP for any $k \geq 1$.

Proof Idea: k-factor approx would imply poly time algorithm for Hamiltonian Cycle.

In practice for metric and non-metric problems:

- Heuristic: Lin-Kernighan based solvers (Lin and Kernighan, 1973)
  - Empirically $\sim 5\%$ of optimal in $O(n^{2.2})$ time.
- Exact: Concorde TSP Solver (Applegate, Bixby, Chvatal, Cook, 2007)
  - Exact solution of Euclidean TSP on $85,900$ points!

Metric TSP

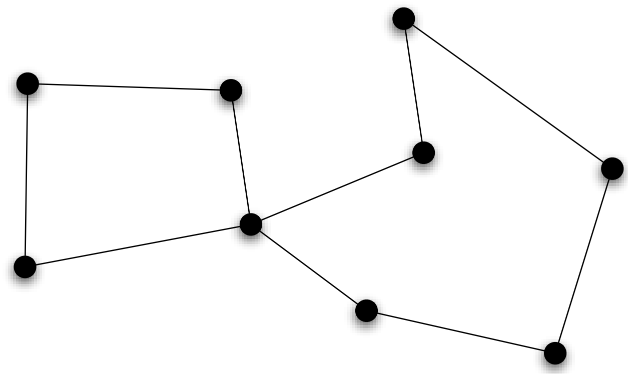
Metric TSP

Given: A complete metric graph $G_n = (V_n, E_n)$
Task: Find a Hamiltonian cycle with minimum weight.

- The Metric TSP is **NP-hard**.
- There exist approximation algorithms!

Eulerian Graphs

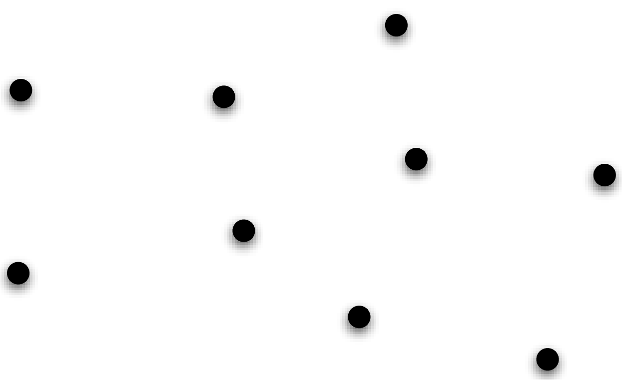
- **Eulerian graph**: degree of each vertex is even
- **Eulerian walk**: Closed walk containing every edge.
- Graph has Eulerian walk $\Leftrightarrow$ Eulerian.
- Eulerian walk can be computed in $O(|V| + |E|)$ time.



Double-Tree Algorithm

Double-Tree Algorithm

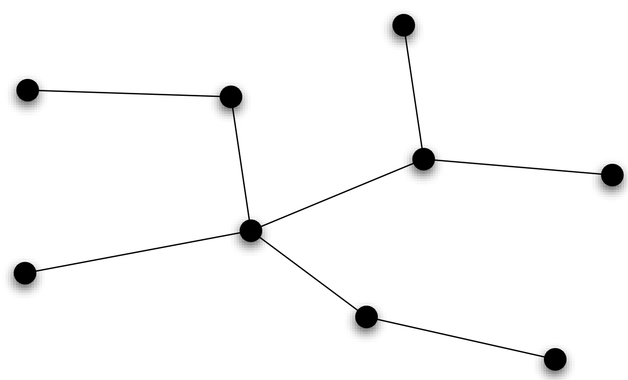
```
1: Find a minimum spanning tree T of graph G_n.
2: G‾ := graph containing two copies of each edge in T.
3: Compute Eulerian walk in Eulerian graph G‾.
4: Walk gives ordering, ignore all but first occurrence
   of vertex.
```



Double-Tree Algorithm

Double-Tree Algorithm

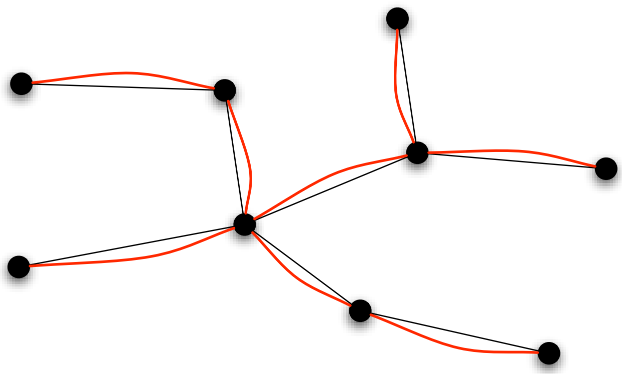
```
1: Find a minimum spanning tree T of graph G_n.
2: G‾ := graph containing two copies of each edge in T.
3: Compute Eulerian walk in Eulerian graph G‾.
4: Walk gives ordering, ignore all but first occurrence
   of vertex.
```

Double-Tree Algorithm

Double-Tree Algorithm

1: Find a minimum spanning tree T of graph G_n.
2: $\overline{G} :=$ graph containing two copies of each edge in T.
3: Compute Eulerian walk in Eulerian graph $\overline{G}$.
4: Walk gives ordering, ignore all but first occurrence of vertex.



Double-Tree Algorithm

Theorem

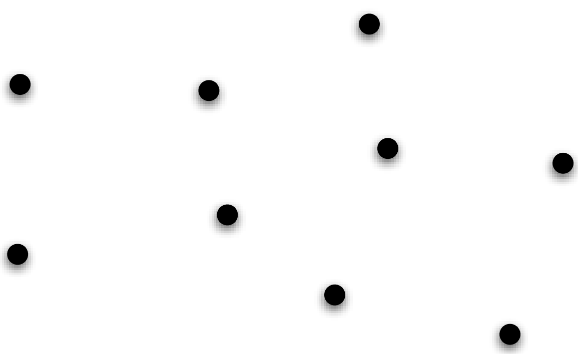
Double-Tree Algorithm is a 2-approx algorithm for the Metric TSP. Its running time is $O(n^2)$.

- Deleting one edge from a tour gives a spanning tree.
- Thus minimum spanning tree is shorter than optimal tour.
- Each edge is doubled.
- Spanning tree can be computed in $O(n^2)$ time.
- Eulerian walk computed in $O(n)$ time.

Christofides' Algorithm

Christofides' Algorithm

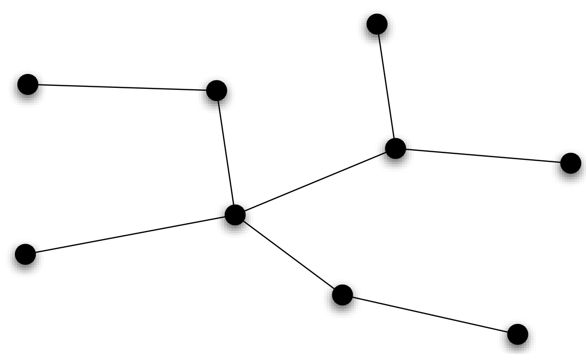
1: Find a minimum spanning tree T of G.
2: Let W be the set of vertices with odd degree in T.
3: Find the minimum weight perfect matching M in subgraph generated by W.
4: Find an Eulerian path in $G := (V_n, E(T) \cup M)$, (skip vertices already seen).



Christofides' Algorithm

Christofides' Algorithm

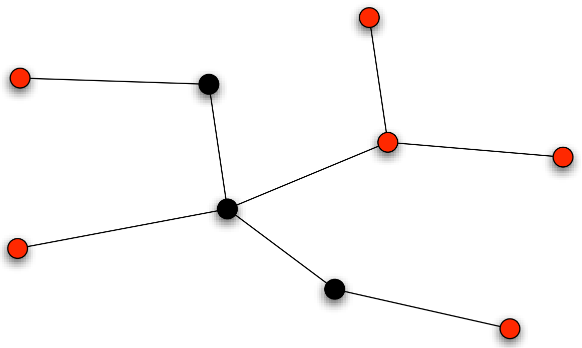
1: Find a minimum spanning tree T of G.
2: Let W be the set of vertices with odd degree in T.
3: Find the minimum weight perfect matching M in subgraph generated by W.
4: Find an Eulerian path in $G := (V_n, E(T) \cup M)$, (skip vertices already seen).



Christofides' Algorithm

Christofides' Algorithm

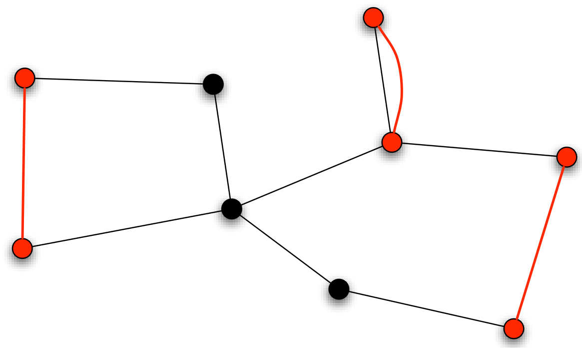
1: Find a minimum spanning tree T of G.
2: Let W be the set of vertices with odd degree in T.
3: Find the minimum weight perfect matching M in subgraph generated by W.
4: Find an Eulerian path in $G := (V_n, E(T) \cup M)$, (skip vertices already seen).



Christofides' Algorithm

Christofides' Algorithm

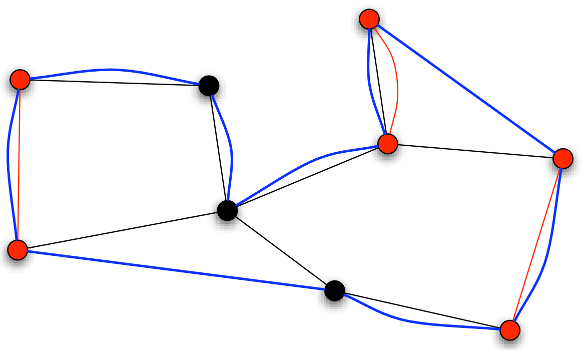
1: Find a minimum spanning tree T of G.
2: Let W be the set of vertices with odd degree in T.
3: Find the minimum weight perfect matching M in subgraph generated by W.
4: Find an Eulerian path in $G := (V_n, E(T) \cup M)$, (skip vertices already seen).



Christofides' Algorithm

Christofides' Algorithm

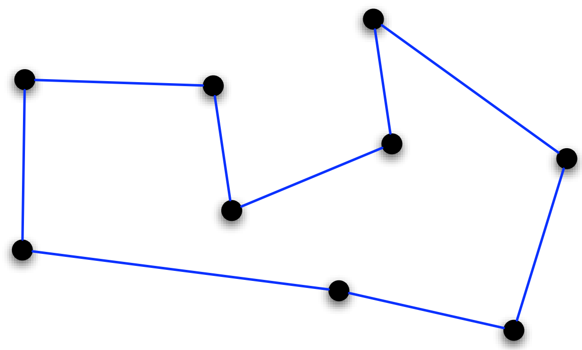
1: Find a minimum spanning tree T of G.
2: Let W be the set of vertices with odd degree in T.
3: Find the minimum weight perfect matching M in subgraph generated by W.
4: Find an Eulerian path in $G := (V_n, E(T) \cup M)$, (skip vertices already seen).



Christofides' Algorithm

Christofides' Algorithm

1: Find a minimum spanning tree T of G.
2: Let W be the set of vertices with odd degree in T.
3: Find the minimum weight perfect matching M in subgraph generated by W.
4: Find an Eulerian path in $G := (V_n, E(T) \cup M)$, (skip vertices already seen).



Christofides' Algorithm

Theorem

Christofides' Algorithm gives a 3/2-approx algorithm for the Metric TSP. Its running time is $O(n^3)$.

- $L(\text{Christofides}) = L(\text{MST}) + L(M)$.
- But, $L(\text{MST}) < L(\text{TSP})$, and
- $L(M) \leq L(M') \leq L(\text{TSP})/2$.
  Where M' is the minimum perfect matching of W using edges that are part of TSP.

Best known approx algorithm for Metric TSP

Euclidean TSP

Theorem (Arora, 1998; Mitchell, 1999)

For each fixed $\epsilon > 0$, a $(1+\epsilon)$-approximate solution can be found in $O\big(n^3(\log n)^c\big)$ time.

Practical value limited to due c's dependence on ϵ.

Length Bounds for Euclidean TSP

How long is the TSP tour through n points in unit square?

Theorem (Few, 1955)

For every set Q_n of n points in the unit square

$$\text{ETSP}(Q_n) \leq \sqrt{2n} + 7/4.$$

Worst-case lower bound matches:

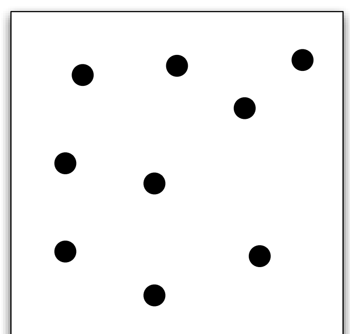
- Equally space n points on a grid
- Then $\text{ETSP}(Q_n) = C\sqrt{n}$.
- So, worst-case length $\geq C\sqrt{n}$.

Worst-case TSP Length Upper Bound (Intuition)

- Consider $Q_n := \{x_1, \dots, x_n\}$ of n points in unit square.
- There exists $c > 0$ such that

$$\min\left\{\|x_i - x_j\| : x_i, x_j \in Q_n\right\} \leq \frac{c}{\sqrt{n}}.$$

- Let ℓ_n denote worst-case TSP length through n pts.
- Then $\ell_n \leq \ell_{n-1} + 2c/\sqrt{n}$.
- Summing we get $\ell(n) \leq C\sqrt{n}$.

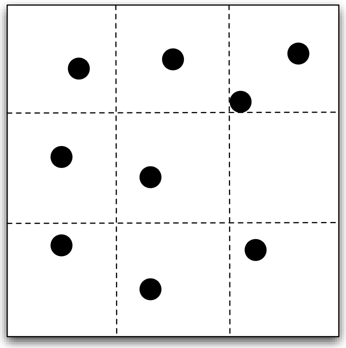


Worst-case TSP Length Upper Bound (Intuition)

- Consider $Q_n := \{x_1, \dots, x_n\}$ of n points in unit square.
- There exists $c > 0$ such that

$$\min \left\{ \|x_i - x_j\| : x_i, x_j \in Q_n \right\} \leq \frac{c}{\sqrt{n}}.$$

- Let ℓ_n denote worst-case TSP length through n pts.
- Then $\ell_n \leq \ell_{n-1} + 2c/\sqrt{n}$.
- Summing we get $\ell(n) \leq C\sqrt{n}$.

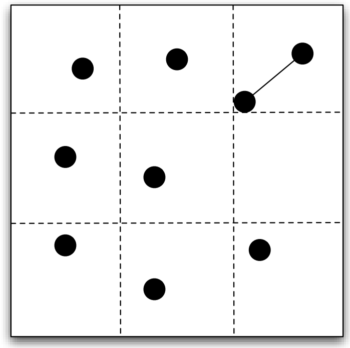


Worst-case TSP Length Upper Bound (Intuition)

- Consider $Q_n := \{x_1, \dots, x_n\}$ of n points in unit square.
- There exists $c > 0$ such that

$$\min \left\{ \|x_i - x_j\| : x_i, x_j \in Q_n \right\} \leq \frac{c}{\sqrt{n}}.$$

- Let ℓ_n denote worst-case TSP length through n pts.
- Then $\ell_n \leq \ell_{n-1} + 2c/\sqrt{n}$.
- Summing we get $\ell(n) \leq C\sqrt{n}$.

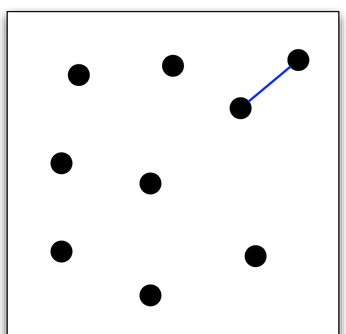


Worst-case TSP Length Upper Bound (Intuition)

- Consider $Q_n := \{x_1, \dots, x_n\}$ of n points in unit square.
- There exists $c > 0$ such that

$$\min \left\{ \|x_i - x_j\| : x_i, x_j \in Q_n \right\} \leq \frac{c}{\sqrt{n}}.$$

- Let ℓ_n denote worst-case TSP length through n pts.
- Then $\ell_n \leq \ell_{n-1} + 2c/\sqrt{n}$.
- Summing we get $\ell(n) \leq C\sqrt{n}$.

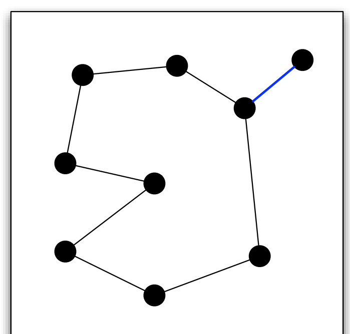


Worst-case TSP Length Upper Bound (Intuition)

- Consider $Q_n := \{x_1, \dots, x_n\}$ of n points in unit square.
- There exists $c > 0$ such that

$$\min \left\{ \|x_i - x_j\| : x_i, x_j \in Q_n \right\} \leq \frac{c}{\sqrt{n}}.$$

- Let ℓ_n denote worst-case TSP length through n pts.
- Then $\ell_n \leq \ell_{n-1} + 2c/\sqrt{n}$.
- Summing we get $\ell(n) \leq C\sqrt{n}$.



TSP Length for Random Points

Theorem (Beardwood, Halton, and Hammersley, 1959)

Let Q_n be a set of n i.i.d. random variables with compact support in $\mathbb{R}^d$ and distribution $\varphi(x)$. Then, with prob. 1

$$\lim_{n\to+\infty} \frac{\mathrm{ETSP}(Q_n)}{n^{(d-1)/d}} = \beta_{\mathrm{TSP},d} \int_{\mathbb{R}^d} \bar{\varphi}(x)^{(d-1)/d} dx,$$

where $\beta_{\mathrm{TSP},d}$ is a constant independent of φ, and $\bar{\varphi}$ is absolutely continuous part of φ.

For uniform distribution in square of area A

$$\frac{\mathrm{ETSP}(Q_n)}{\sqrt{n}} \to \beta_{\mathrm{TSP},2}\sqrt{A} \quad \text{as } n \to +\infty.$$

Best estimate of $\beta_{\mathrm{TSP},2}$ is Percus and Martin, 1996

$$\beta_{\mathrm{TSP},2} \simeq 0.7120.$$

Summary of Traveling Salesman Problem

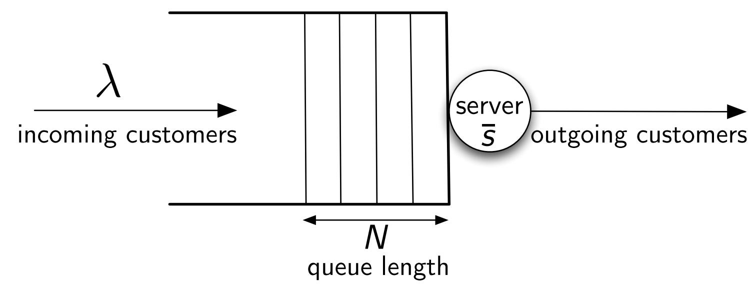
- Solving TSP is *NP*-**hard**, and no approx algorithms exist.
- For **metric TSP**, still *NP*-hard but good **approx algs exist**.
- For Euclidean TSP, very good heuristics exist.
- Length of tour through n points in unit square:
  - Worst-case is $\Theta(\sqrt{n})$.
  - Uniform random is $\Theta(\sqrt{n})$.
  - For all density functions $O(\sqrt{n})$.

Outline

1. Graph Theory
2. The Traveling Salesman Problem
3. Queueing Theory
   - Kendall's Notation
   - Little's Law and Load Factor

Basic Queueing Model

- Customers arrive, wait in a queue, and are then processed
- Queue length builds up when arrival rate is larger than service rate



- **Arrivals** modeled as stochastic process with rate λ
- **Service time** of each customer is a r.v. with finite mean $\bar{s}$ and second moment $\bar{s^2}$.
- **Service rate** is $1/\bar{s}$.

Queueing Notation

Kendall's Queueing notation $A/B/C$:

- $A =$ the arrival process
- $B =$ the service time distribution
- $C =$ the number of servers

Main codes:

- $D =$ Deterministic
- $M =$ Markovian
  - for arrivals: Poisson process
  - for service times: Exponential distribution
- G (or GI) $=$ General distribution (independent among customers)

Example $M/G/m$ queue:

- Poisson arrivals with rate λ
- General service times with mean $\bar{s}$
- m servers

Little's Law and Load Factor

Define:

- average wait-time in queue as $\overline{W}$
- average system as $\overline{T} := \overline{W} + \bar{s}$.

Little's Law/Theorem

For a stable queue $\overline{N} = \lambda \overline{W}$

- For m servers, define **load factor** as

$$\varrho := \frac{\lambda \bar{s}}{m}$$

- **Necessary condition** for stable queue is $\varrho < 1$.

Wait-time examples

For $M/D/1$ queue:

$$\overline{W} = \frac{\varrho \bar{s}}{2(1-\varrho)}$$

For $M/G/1$ queue:

$$\overline{W} = \frac{\lambda \bar{s^2}}{2(1-\varrho)}$$

For $G/G/1$ queue (Kingman, 1962):

$$\overline{W} \leq \frac{\lambda(\sigma_a^2 + \sigma_s^2)}{2(1-\varrho)}$$

and the upper bound becomes exact as $\varrho \to 1^-$.

Lecture outline

Workshop Structure and Schedule

8:00-8:30am	*Coffee Break*	
8:30-9:00am	Lecture #1:	Intro to dynamic vehicle routing
9:05-9:50am	Lecture #2:	Prelims: graphs, TSPs and queues
9:55-10:40am	Lecture #3:	The single-vehicle DVR problem
10:40-11:00am	*Break*	
11:00-11:45pm	Lecture #4:	The multi-vehicle DVR problem
11:45-1:10pm	*Lunch Break*	
1:10-2:10pm	Lecture #5:	Extensions to vehicle networks
2:15-3:00pm	Lecture #6:	Extensions to different demand models
3:00-3:20pm	*Coffee Break*	
3:20-4:20pm	Lecture #7:	Extensions to different vehicle models
4:25-4:40pm	Lecture #8:	Extensions to different task models
4:45-5:00pm		Final open-floor discussion