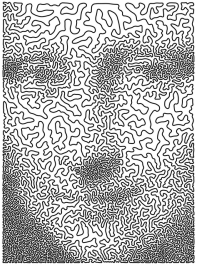
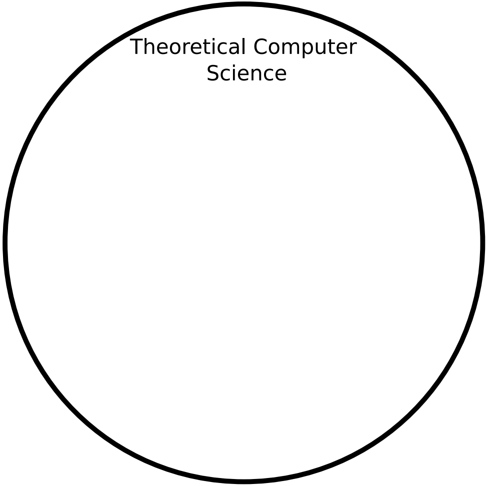
The Art of Incremental Improvements

Karol Węgrzycki

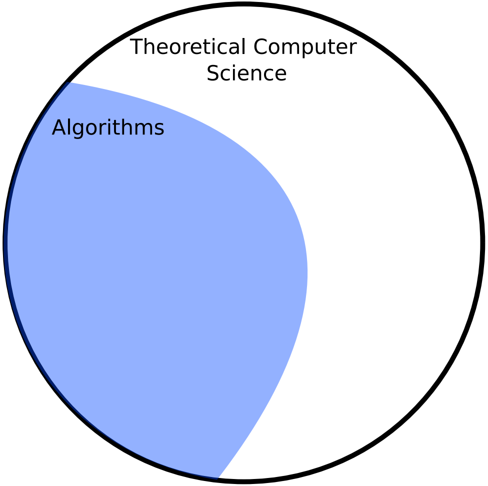


Context

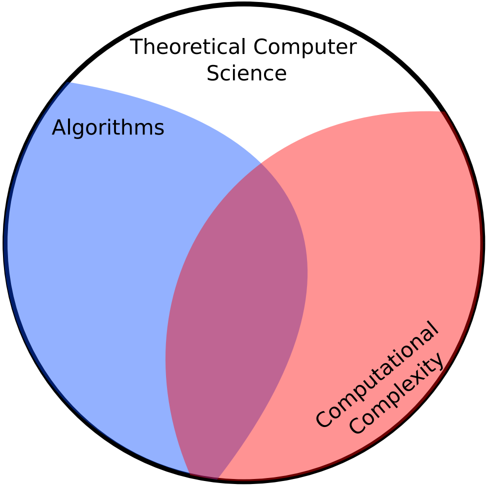
Context



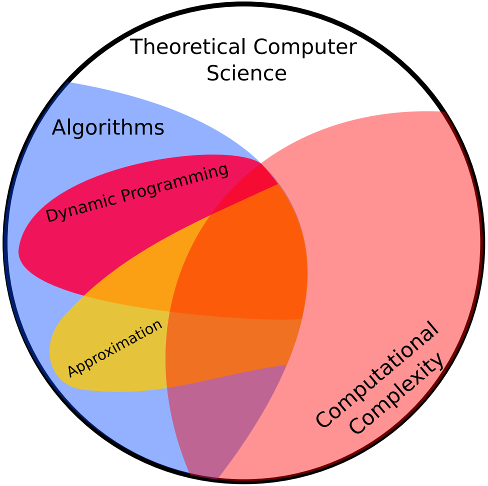
Context



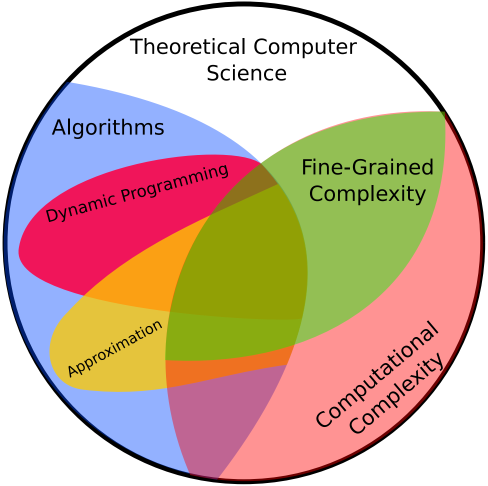
Context


Theoretical Computer Science
Algorithms
Computational Complexity

Context

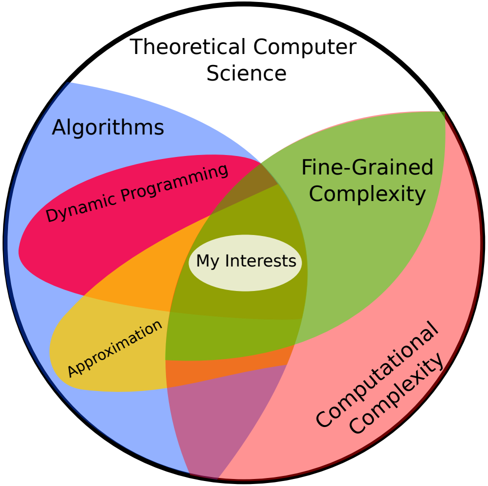

Theoretical Computer Science
Algorithms
Dynamic Programming
Approximation
Computational Complexity

Context

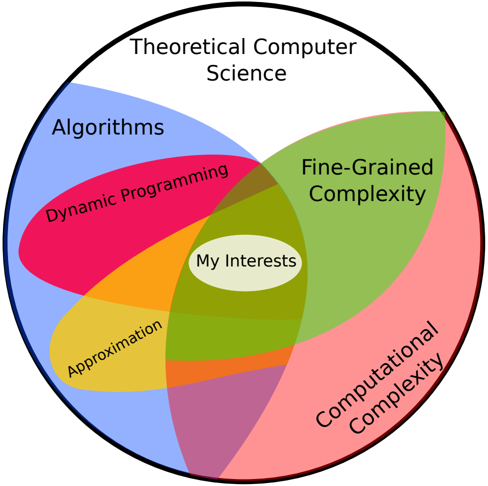


Theoretical Computer Science
Algorithms
Dynamic Programming
Fine-Grained Complexity
Approximation
Computational Complexity

Context


Theoretical Computer Science
Algorithms
Dynamic Programming
Fine-Grained Complexity
My Interests
Approximation
Computational Complexity

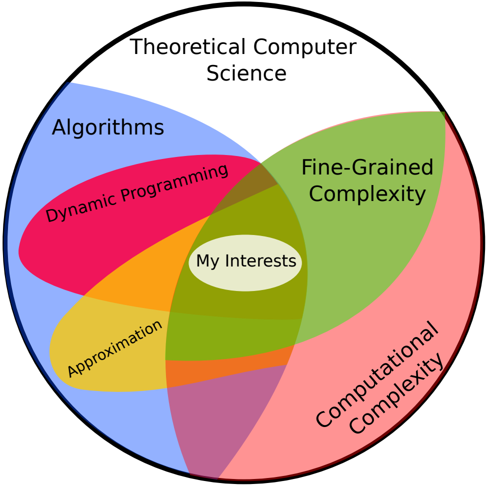
Context



Goal: For *important* problem give:

- fast algorithm,
- argument that it cannot solved faster.

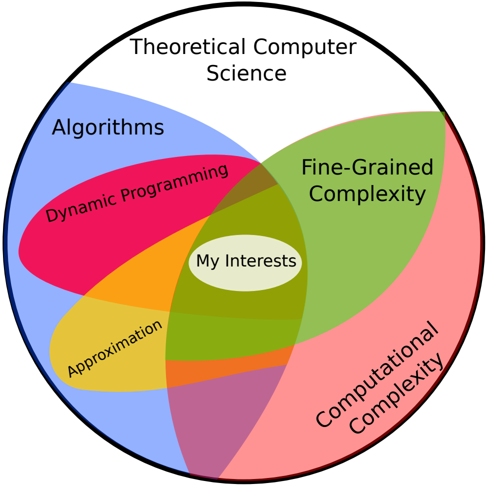
Context



Goal: For *important* problem give:

- fast algorithm,
- argument that it cannot solved faster.

Context



Goal: For *important* problem give:

- fast algorithm,
- argument that it cannot solved faster.

Thank you: Antonios Antoniadis, Karl Bringmann, Jana Cslovjecsek, Marek Cygan, François Dross, Krzysztof Fleszar, Sándor Kisfaludi-Bak, Jędrzej Olkowski, Marcin Mucha, Jesper Nederlof, Marvin Künnemann, Andrzej Pacut, Jakub Pawlewicz, Michał Pilipczuk, Adam Polak, Lars Rohwedder, Mateusz Rychlewicz, Piotr Sankowski, Céline Swennenhuis, Adam Witkowski, Michał Włodarczyk, Anna Zych-Pawlewicz

Most of my talks look like this:

1972:
Problem X
is NP-hard

Most of my talks look like this:

1972:
Problem X
is NP-hard

1973:
Exponential
Time Algorithm

Most of my talks look like this:

1972:
Problem X
is NP-hard

1973:
Exponential
Time Algorithm

1974:
Faster Algorithm

Most of my talks look like this:

1972:
Problem X
is NP-hard

1973:
Exponential
Time Algorithm

1974:
Faster Algorithm

•
•
•

2021:
Currently Best
Algorithm

Most of my talks look like this:

1972:
Problem X
is NP-hard

1973:
Exponential
Time Algorithm

1980:
Approximation
Algorithm

1974:
Faster Algorithm

⋮

2021:
Currently Best
Algorithm

Most of my talks look like this:

1972:
Problem X
is NP-hard

1973:
Exponential
Time Algorithm

1974:
Faster Algorithm

⋮

2021:
Currently Best
Algorithm

1980:
Approximation
Algorithm

1981:
Better Approximation

⋮

2021:
0.001-Approximation

Most of my talks look like this:

1972:
Problem X
is NP-hard

1973:
Exponential
Time Algorithm

1974:
Faster Algorithm

⋮

2021:
Currently Best
Algorithm

1980:
Approximation
Algorithm

1981:
Better Approximation

⋮

2021:
0.001-Approximation

1990:
FPT Algorithm

. . .

1991:
Faster
FPT Algorithm

⋮

2021
Currently fastest
FPT algorithm

Most of my talks look like this:

1972:
Problem X
is NP-hard

Incremental
Improvements?

1973:
Exponential
Time Algorithm

1974:
Faster Algorithm

1980:
Approximation
Algorithm

1981:
Better Approximation

1990:
FPT Algorithm

• • •

1991:
Faster
FPT Algorithm

⋮ ⋮ ⋮

2021:
Currently Best
Algorithm

2021:
0.001-Approximation

2021
Currently fastest
FPT algorithm

Most of my talks look like this:

1972:
Problem X
is NP-hard

Incremental
Improvements?

1973:
Exponential
Time Algorithm

1980:
Approximation
Algorithm

1990:
FPT Algorithm

...

1974:
Faster Algorithm

1981:
Better Approximation

1991:
Faster
FPT Algorithm

⋮

2021:
Currently Best
Algorithm

2021:
0.001-Approximation

2021
Currently fastest
FPT algorithm

Most of my talks look like this:

1972:
Problem X
is NP-hard

Incremental
Improvements?

1973:
Exponential
Time Algorithm

1980:
Approximation
Algorithm

1990:
FPT Algorithm

...

1974:
Faster Algorithm

⋮

2021:
Currently Best
Algorithm

1981:
Better Approximation

⋮

2021:
0.001-Approximation

1991:
Faster
FPT Algorithm

⋮

2021
Currently fastest
FPT algorithm

Most of my talks look like this:

1972:
Problem X
is NP-hard

Incremental
Improvements?

1973:
Exponential
Time Algorithm

1974:
Faster Algorithm

⋮

2021:
Currently Best
Algorithm

1980:
Approximation
Algorithm

1981:
Better Approximation

⋮

2021:
0.001-Approximation

1990:
FPT Algorithm

…

1991:
Faster
FPT Algorithm

⋮

2021
Currently fastest
FPT algorithm

Most of my talks look like this:

1972:
Problem X
is NP-hard

Incremental
Improvements?

1973:
Exponential
Time Algorithm

1974:
Faster Algorithm

⋮

2021:
Currently Best
Algorithm

1980:
Approximation
Algorithm

1981:
Better Approximation

⋮

2021:
0.001-Approximation

1990:
FPT Algorithm

…

1991:
Faster
FPT Algorithm

⋮

2021
Currently fastest
FPT algorithm

Most of my talks look like this:

1972:
Problem X
is NP-hard

Most of my results
look like this

1973:
Exponential
Time Algorithm

1974:
Faster Algorithm

⋮

2021:
Currently Best
Algorithm

1980:
Approximation
Algorithm

1981:
Better Approximation

⋮

2021:
0.001-Approximation

1990:
FPT Algorithm

…

1991:
Faster
FPT Algorithm

⋮

2021
Currently fastest
FPT algorithm

Most of my talks look like this:

1972:
Problem X
is NP-hard

I'll focus on
most recent result

1973:
Exponential
Time Algorithm

1974:
Faster Algorithm

⋮

2021:
Currently Best
Algorithm

1980:
Approximation
Algorithm

1981:
Better Approximation

⋮

2021:
0.001-Approximation

1990
FPT Algorithm

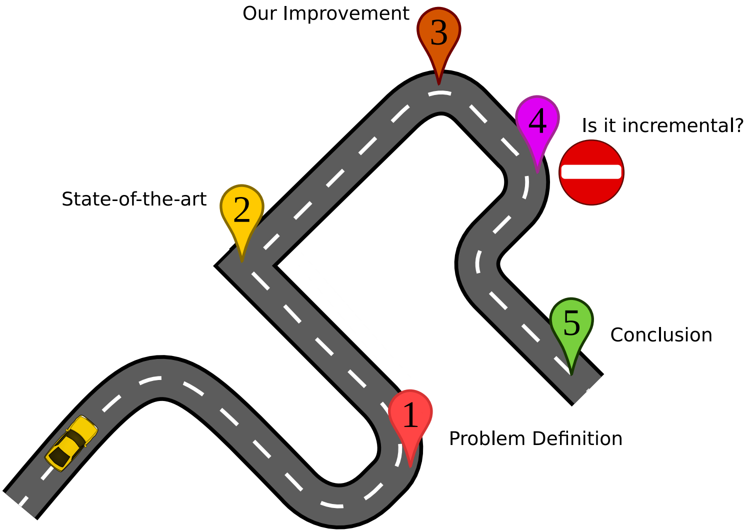
• • •

1991:
Faster
FPT Algorithm

⋮

2021
Currently fastest
FPT algorithm

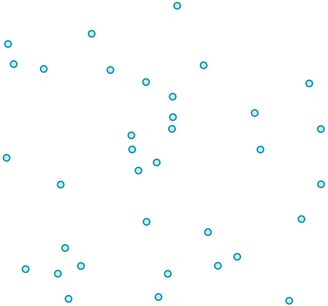
Plan for the talk



Part 1: Problem Definition

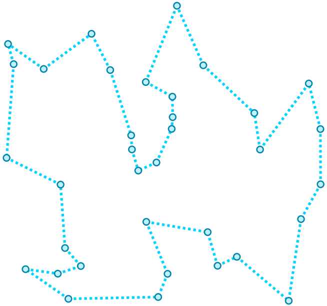
Part 1: Problem Definition

Euclidean Traveling Salesman Problem (Euclidean TSP): Given n points in $\mathbb{R}^2$, find the shortest roundtrip tour that contains all the points.



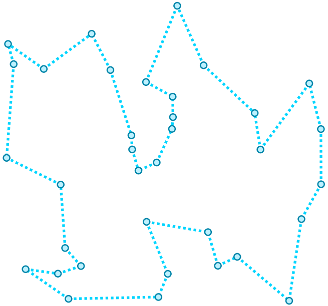
Part 1: Problem Definition

Euclidean Traveling Salesman Problem (Euclidean TSP): Given n points in $\mathbb{R}^2$, find the shortest roundtrip tour that contains all the points.



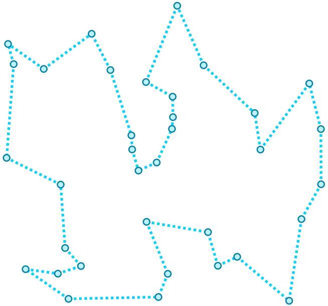
Part 1: Problem Definition

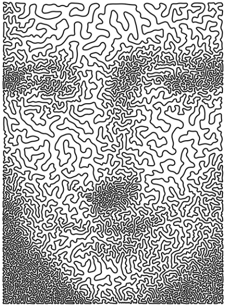
Euclidean Traveling Salesman Problem (Euclidean TSP): Given n points in $\mathbb{R}^2$, find the shortest roundtrip tour that contains all the points.



Part 1: Problem Definition

Euclidean Traveling Salesman Problem (Euclidean TSP): Given n points in $\mathbb{R}^2$, find the shortest roundtrip tour that contains all the points.

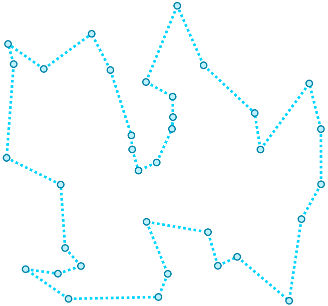


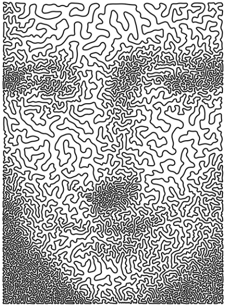


Source: Robert Bosch

Part 1: Problem Definition

Euclidean Traveling Salesman Problem (Euclidean TSP): Given n points in $\mathbb{R}^2$, find the shortest roundtrip tour that contains all the points.





Source: Robert Bosch

Euclidean TSP is NP-hard [Papadimitriou 1977]. We need to settle for approximation

Part 2: State-of-the-art

Part 2: State-of-the-art

PTAS for Euclidean TSP: for any fixed $\varepsilon > 0$ find a tour that is at most $(1 + \varepsilon)$ times longer than optimal in $n^{\mathcal{O}(1)}$ time.

Part 2: State-of-the-art

PTAS for Euclidean TSP: for any fixed $\varepsilon > 0$ find a tour that is at most $(1+\varepsilon)$ times longer than optimal in $n^{\mathcal{O}(1)}$ time.

History of PTASes for Euclidean TSP:

Arora (J.ACM 1998) $n(\log(n)/\varepsilon)^{\mathcal{O}(1/\varepsilon)}$ time

Mitchell (SICOMP 1999) $n^{\mathcal{O}(1/\varepsilon)}$ time

2010 Gödel prize winners!

Part 2: State-of-the-art

PTAS for Euclidean TSP: for any fixed $\varepsilon > 0$ find a tour that is at most $(1+\varepsilon)$ times longer than optimal in $n^{\mathcal{O}(1)}$ time.

History of PTASes for Euclidean TSP:

Arora (J.ACM 1998) $n(\log(n)/\varepsilon)^{\mathcal{O}(1/\varepsilon)}$ time

Mitchell (SICOMP 1999) $n^{\mathcal{O}(1/\varepsilon)}$ time

2010 Gödel prize winners!

Rao and Smith (STOC 1998)
$(1/\varepsilon)^{\mathcal{O}(1/\varepsilon)} n \log n$ time

Bartal and Gottlieb (FOCS 2013) $2^{(1/\varepsilon)^{\mathcal{O}(1)}} n$ time

Complicated

Part 2: State-of-the-art

PTAS for Euclidean TSP: for any fixed $\varepsilon > 0$ find a tour that is at most $(1+\varepsilon)$ times longer than optimal in $n^{\mathcal{O}(1)}$ time.

History of PTASes for Euclidean TSP:

Arora (J.ACM 1998) $n(\log(n)/\varepsilon)^{\mathcal{O}(1/\varepsilon)}$ time

Mitchell (SICOMP 1999) $n^{\mathcal{O}(1/\varepsilon)}$ time

2010 Gödel prize winners!

Rao and Smith (STOC 1998)
$(1/\varepsilon)^{\mathcal{O}(1/\varepsilon)} n \log n$ time

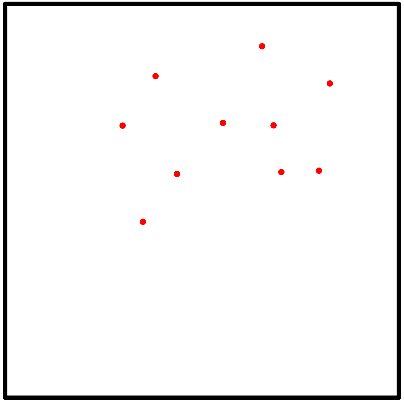
Bartal and Gottlieb (FOCS 2013) $2^{(1/\varepsilon)^{\mathcal{O}(1)}} n$ time

Complicated

Our result: Simple $2^{\mathcal{O}(1/\varepsilon)} n \log n$ time algorithm (joint work with Sándor Kisfaludi-Bak and Jesper Nederlof)

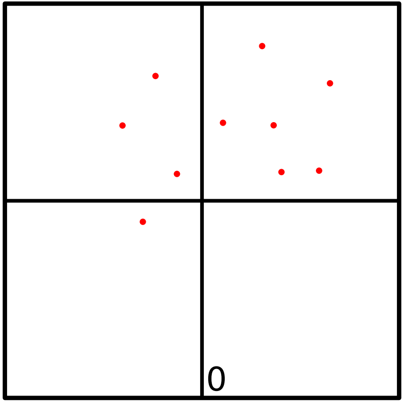
Part 2: State-of-the-art

Arora's algorithm



Part 2: State-of-the-art

Arora's algorithm

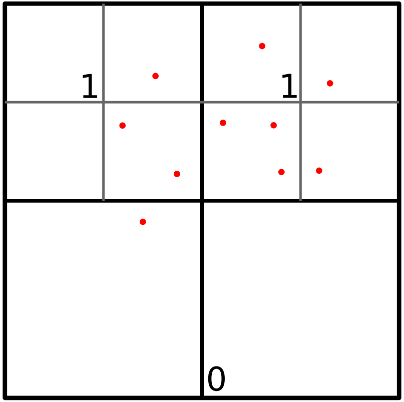


Algorithm:

1. Add randomly shifted quadtree

Part 2: State-of-the-art

Arora's algorithm

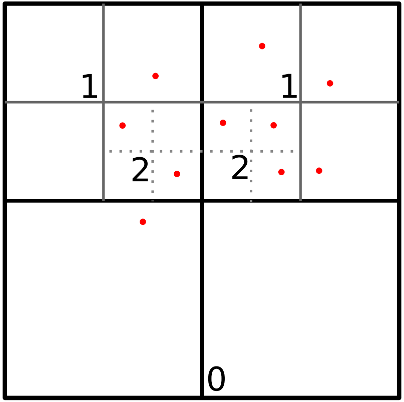


Algorithm:

1. Add randomly shifted quadtree

Part 2: State-of-the-art

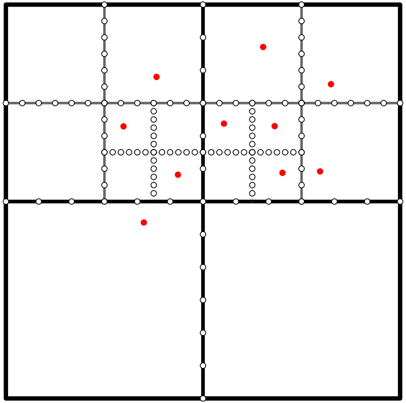
Arora's algorithm



Algorithm:
1. Add randomly shifted quadtree

Part 2: State-of-the-art

Arora's algorithm

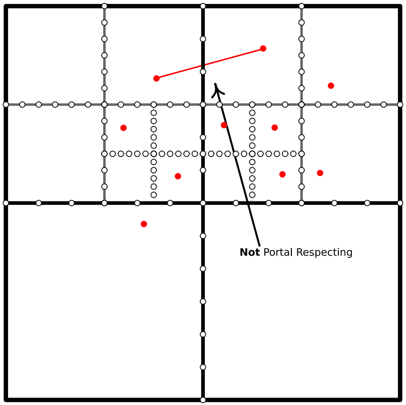


Algorithm:

1. Add randomly shifted quadtree
2. Add g equispaced portals for each cell

Part 2: State-of-the-art

Arora's algorithm

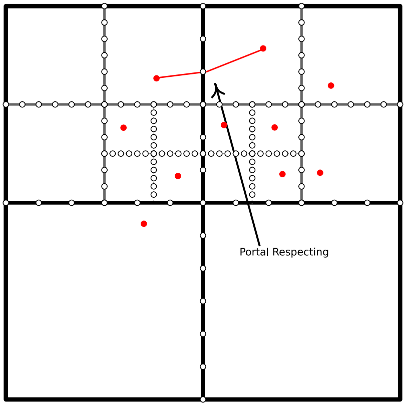


Algorithm:

1. Add randomly shifted quadtree
2. Add g equispaced portals for each cell

Part 2: State-of-the-art

Arora's algorithm

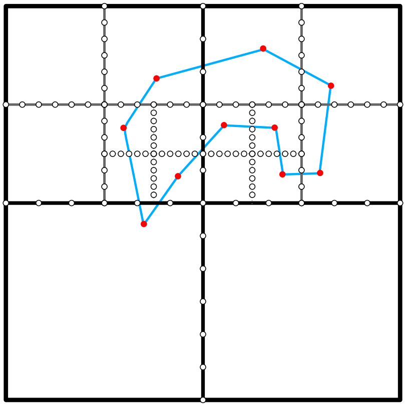


Algorithm:

1. Add randomly shifted quadtree
2. Add g equispaced portals for each cell

Part 2: State-of-the-art

Arora's algorithm



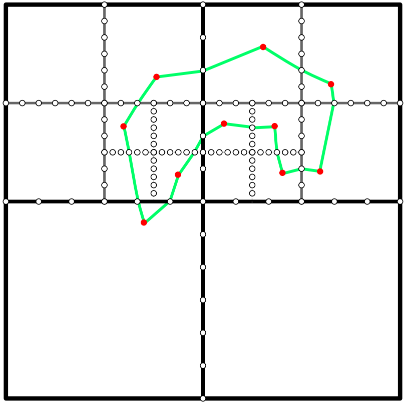
Algorithm:

1. Add randomly shifted quadtree
2. Add g equispaced portals for each cell

Structure Theorem There is a portal respecting tour of expected tour length $\leq (1+\varepsilon)$OPT

Part 2: State-of-the-art

Arora's algorithm

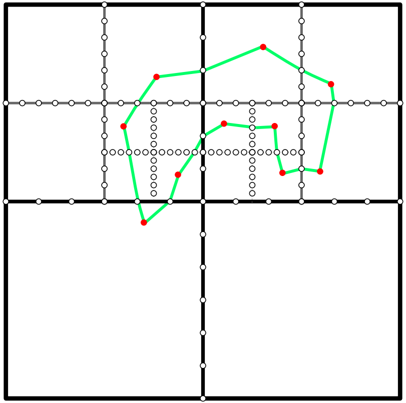


Algorithm:
1. Add randomly shifted quadtree
2. Add g equispaced portals for each cell

Structure Theorem There is a portal respecting tour of expected tour length $\leq (1+\varepsilon)$OPT

Part 2: State-of-the-art

Arora's algorithm

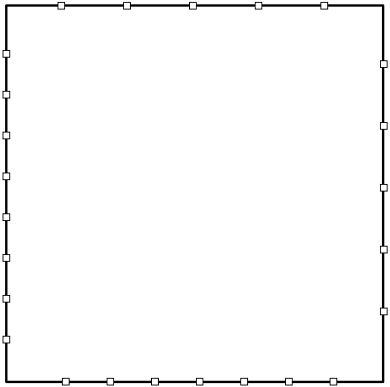


Algorithm:
1. Add randomly shifted quadtree
2. Add g equispaced portals for each cell
3. Find min. length portal respecting tour with DP

Structure Theorem There is a portal respecting tour of expected tour length $\leq (1+\varepsilon)$OPT

Part 2: State-of-the-art

Arora's algorithm



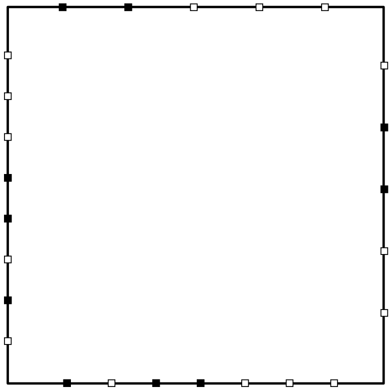
Algorithm:
1. Add randomly shifted quadtree
2. Add g equispaced portals for each cell
3. Find min. length portal respecting tour with DP

Structure Theorem There is a portal respecting tour of expected tour length $\leq (1+\varepsilon)$OPT

Dynamic Programming For each cell find shortest path cover for a given matching on portals

Part 2: State-of-the-art

Arora's algorithm



Guess selected portals out of $\mathcal{O}(\log(n)/\varepsilon)$

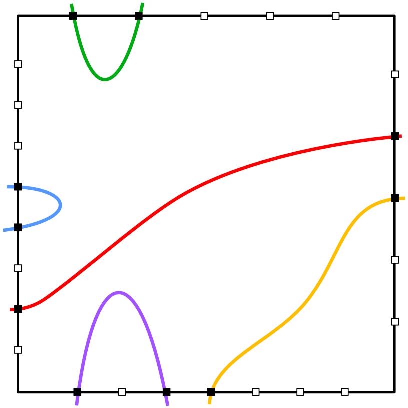
Algorithm:
1. Add randomly shifted quadtree
2. Add g equispaced portals for each cell
3. Find min. length portal respecting tour with DP

Structure Theorem There is a portal respecting tour of expected tour length $\leq (1+\varepsilon)$OPT

Dynamic Programming For each cell find shortest path cover for a given matching on portals

Part 2: State-of-the-art

Arora's algorithm



Guess noncrossing matching on selected portals

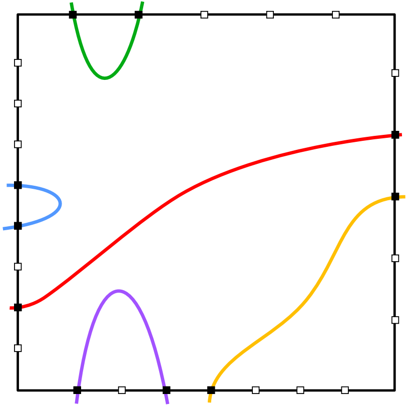
Algorithm:
1. Add randomly shifted quadtree
2. Add g equispaced portals for each cell
3. Find min. length portal respecting tour with DP

Structure Theorem There is a portal respecting tour of expected tour length $\leq (1+\varepsilon)\text{OPT}$

Dynamic Programming For each cell find shortest path cover for a given matching on portals

Part 2: State-of-the-art

Arora's algorithm



Guess noncrossing matching on selected portals

Algorithm:

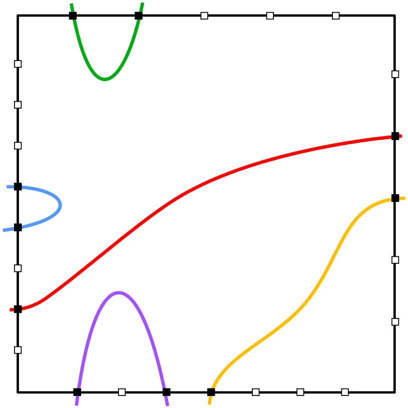
1. Add randomly shifted quadtree
2. Add g equispaced portals for each cell
3. Find min. length portal respecting tour with DP

Structure Theorem There is a portal respecting tour of expected tour length $\leq (1+\varepsilon)\text{OPT}$

Dynamic Programming For each cell find shortest path cover for a given matching on portals

Part 2: State-of-the-art

Arora's algorithm



Guess noncrossing matching on selected portals

Algorithm:
1. Add randomly shifted quadtree
2. Add g equispaced portals for each cell
3. Find min. length portal respecting tour with DP

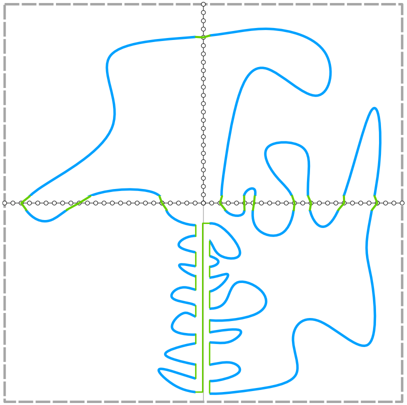
Structure Theorem There is a portal respecting tour of expected tour length $\leq (1+\varepsilon)\text{OPT}$

Dynamic Programming For each cell find shortest path cover for a given matching on portals

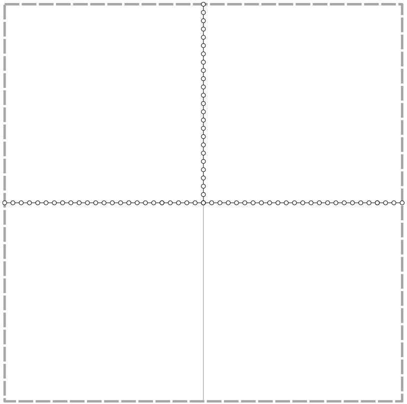
Total runtime: $n^{\mathcal{O}(1)} \cdot 2^{\mathcal{O}(g)} = n^{\mathcal{O}(1/\varepsilon)}$

Part 3: Our Improvement

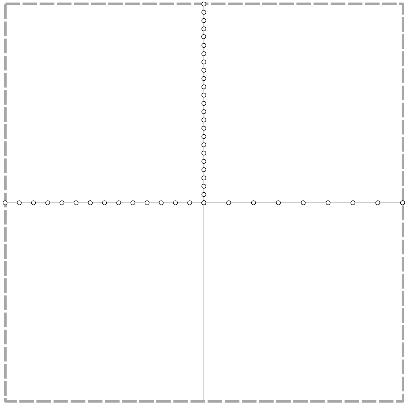
Part 3: Our Improvement



Part 3: Our Improvement

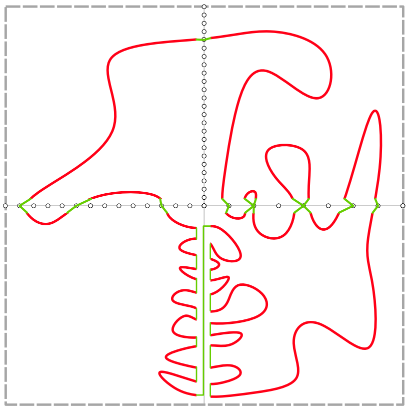


Part 3: Our Improvement



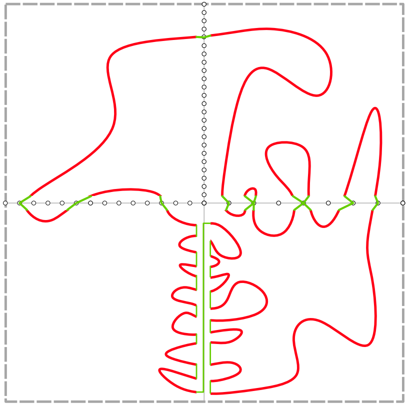
Main point: If tour uses $< 1/\varepsilon$ portals, then increase the granularity g

Part 3: Our Improvement



Main point: If tour uses $< 1/\varepsilon$ portals, then increase the granularity g

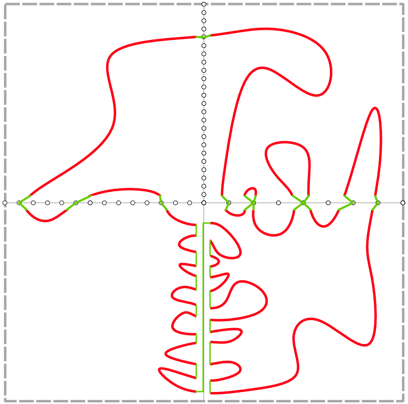
Part 3: Our Improvement



Main point: If tour uses $< 1/\varepsilon$ portals, then increase the granularity g

Our idea For every cell pick $k \leq 1/\varepsilon$ portals from the set of $g := \frac{1}{\varepsilon^2 k}$ equally spaced portals

Part 3: Our Improvement

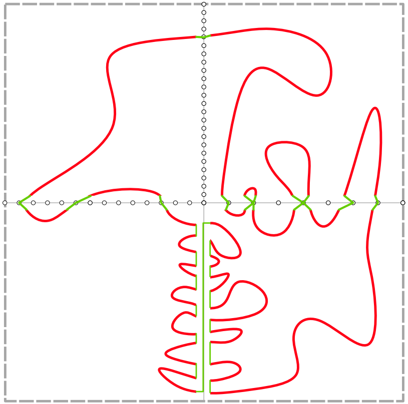


Main point: If tour uses $< 1/\varepsilon$ portals, then increase the granularity g

Our idea For every cell pick $k \leq 1/\varepsilon$ portals from the set of $g := \frac{1}{\varepsilon^2 k}$ equally spaced portals

In the Dynamic Programming consider states where k portals from $g(k)$ equally spaced portals are used (for every $k \in \{2, \ldots, \varepsilon^{-1}\}$).

Part 3: Our Improvement



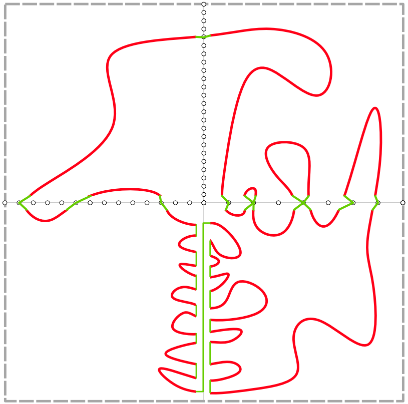
Main point: If tour uses $< 1/\varepsilon$ portals, then increase the granularity g

Our idea For every cell pick $k \leq 1/\varepsilon$ portals from the set of $g := \frac{1}{\varepsilon^2 k}$ equally spaced portals

In the Dynamic Programming consider states where k portals from $g(k)$ equally spaced portals are used (for every $k \in \{2, \ldots, \varepsilon^{-1}\}$).

Runtime: $\binom{g(k)}{k} n \log n \leq 2^{\mathcal{O}(1/\varepsilon)} n \log n$

Part 3: Our Improvement



Main point: If tour uses $< 1/\varepsilon$ portals, then increase the granularity g

Our idea For every cell pick $k \leq 1/\varepsilon$ portals from the set of $g := \frac{1}{\varepsilon^2 k}$ equally spaced portals

In the Dynamic Programming consider states where k portals from $g(k)$ equally spaced portals are used (for every $k \in \{2, \dots, \varepsilon^{-1}\}$).

Runtime: $\binom{g(k)}{k} n \log n \leq 2^{\mathcal{O}(1/\varepsilon)} n \log n$

Proof that this scheme works: Highly technical, check our paper

Part 4: Is it incremental improvement?

Part 4: Is it incremental?

Part 4: Is it incremental?

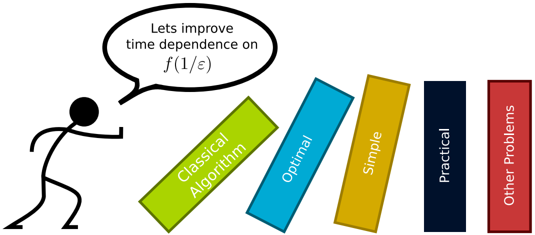
Maybe it is? Maybe it is not?

$$(1/\varepsilon)^{\mathcal{O}(1/\varepsilon)} n \log n \longrightarrow 2^{\mathcal{O}(1/\varepsilon)} n \log n$$

Part 4: Is it incremental?

Maybe it is? Maybe it is not?

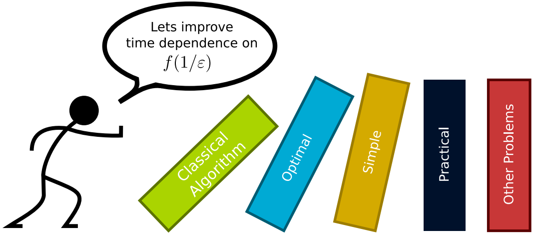
$$(1/\varepsilon)^{\mathcal{O}(1/\varepsilon)} n \log n \longrightarrow 2^{\mathcal{O}(1/\varepsilon)} n \log n$$



Part 4: Is it incremental?

Maybe it is? Maybe it is not?

$$(1/\varepsilon)^{\mathcal{O}(1/\varepsilon)} n \log n \longrightarrow 2^{\mathcal{O}(1/\varepsilon)} n \log n$$

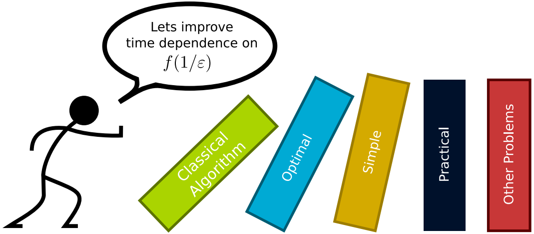


Theorem: No $2^{o(1/\varepsilon)} \cdot n^{100}$ time algorithm (assuming a widely believed hypothesis).

Part 4: Is it incremental?

Maybe it is? Maybe it is not?

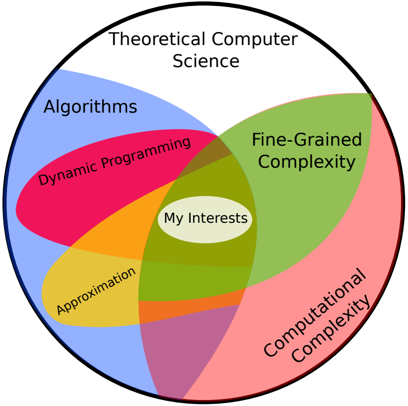
$$(1/\varepsilon)^{\mathcal{O}(1/\varepsilon)} n \log n \longrightarrow 2^{\mathcal{O}(1/\varepsilon)} n \log n$$



Theorem: No $2^{o(1/\varepsilon)} \cdot n^{100}$ time algorithm (assuming a widely believed hypothesis).

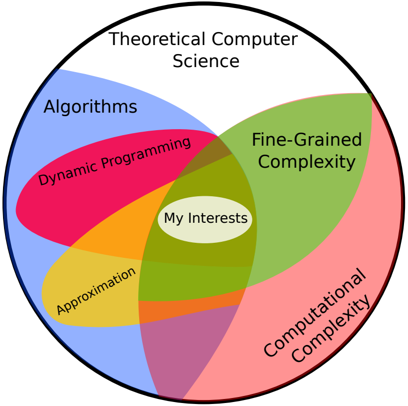
The point: This is the **final** improvement!

Conclusion



Theoretical Computer Science
Algorithms
Dynamic Programming
Fine-Grained Complexity
My Interests
Approximation
Computational Complexity

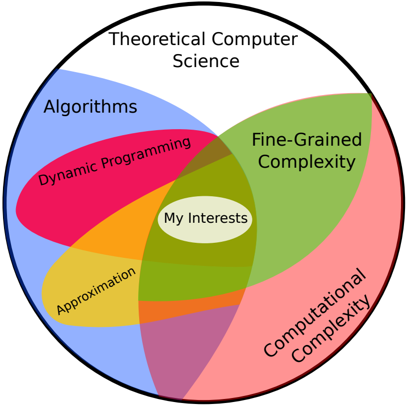
Conclusion



We saw a type of problem that I find interesting

My usual goal: settle the runtime complexity of fundamental problems

Conclusion



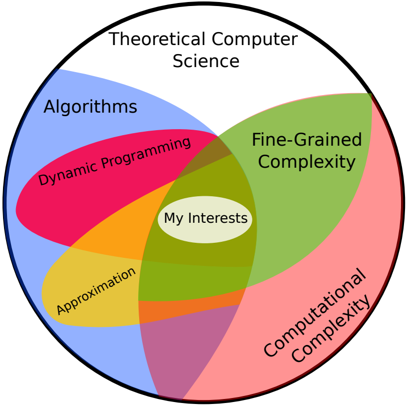
We saw a type of problem that I find interesting

My usual goal: settle the runtime complexity of fundamental problems

Message:

It is hard to tell which results are "incremental" and which are not.

Conclusion



We saw a type of problem that I find interesting

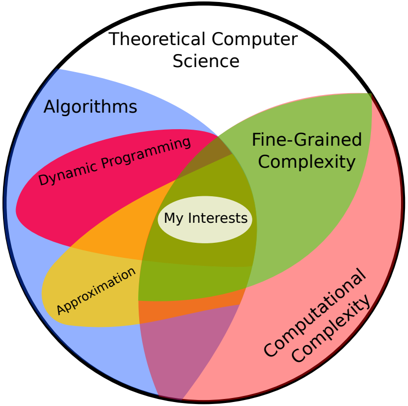
My usual goal: settle the runtime complexity of fundamental problems

Message:

It is hard to tell which results are "incremental" and which are not.

To make progress it is equally important to find lower bounds.

Conclusion



We saw a type of problem that I find interesting

My usual goal: settle the runtime complexity of fundamental problems

Message:

It is hard to tell which results are "incremental" and which are not.

To make progress it is equally important to find lower bounds.

Thank You!