The Thesis and Slides

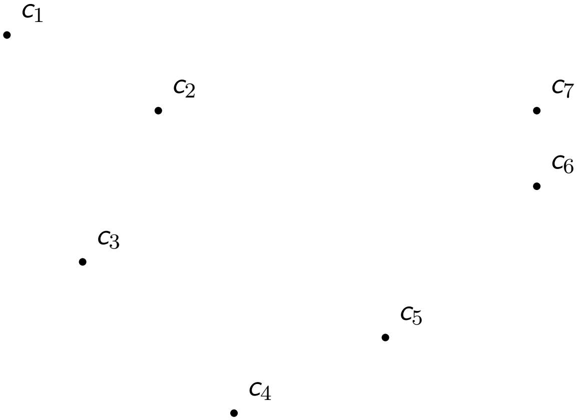
You can find my thesis and the slides under this link:

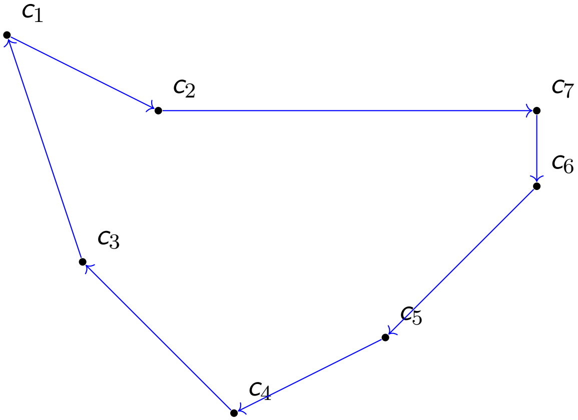
Comparing Algorithms Approximating the Maximum Travelling Salesman Problem

Bachelor Thesis Defence

Feliks

October 24, 2025


c_1
c_2
c_7
c_6
c_3
c_5
c_4

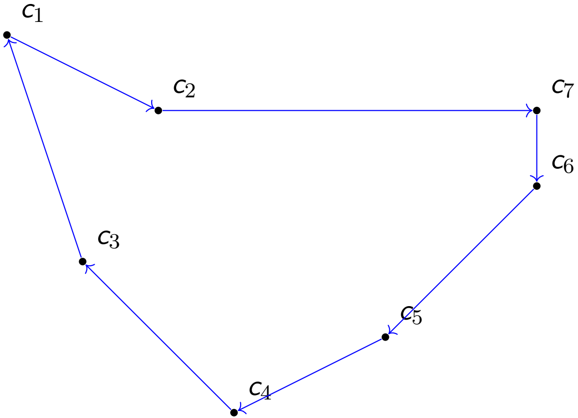

c_1
c_2
c_7
c_6
c_3
c_5
c_4

Travelling Salesman Problem

Different instances of TSP

- directed/undirected Graphs
- in metric space
- under different distances
- maximizing/minimizing tour

We focus on TSP maximizing tour length under euclidean distance in metric space


c_1
c_2
c_7
c_6
c_3
c_5
c_4

We know is NP-hard:

- Minimum TSP under any dimension d and any L_p norm
- Maximum TSP under Euclidean distance is NP-hard for $d \geq 3$

Solvable in polynomial time:

- Maximum TSP under any dimension d for some L_p norms
- specifically Manhattan and Supremum norm

- $L_p : ||x||_p = (\sum_i x_i^p)^{1/p}$
- $L_p : dist_{L_p}(x, y) = (\sum_i (x_i - y_i)^p)^{1/p}$
- well known norms L_1, L_2, L_{max}
- Manhattan, Euclidean, Supremum norm respectively

We know is NP-hard:

- Minimum TSP under any dimension d and any L_p norm
- Maximum TSP under Euclidean distance is NP-hard for $d \geq 3$

Solvable in polynomial time:

- Maximum TSP under any dimension d for some L_p norms
- specifically Manhattan and Supremum norm

- $L_p : ||x||_p = (\sum_i x_i^p)^{1/p}$
- $L_p : dist_{L_p}(x, y) = (\sum_i (x_i - y_i)^p)^{1/p}$
- well known norms L_1, L_2, L_{max}
- Manhattan, Euclidean, Supremum norm respectively

- Conclusion: max TSP is difficult, how do we approximate?

Approximation Algorithms

- find a good but not necessarily optimal solution
- $Approx = \rho(n) \times OPT$ (when maximizing)
- Approximation ratio $\rho(n) \geq 1$
- should provide "good" solution
- in "good" runtime

Structure

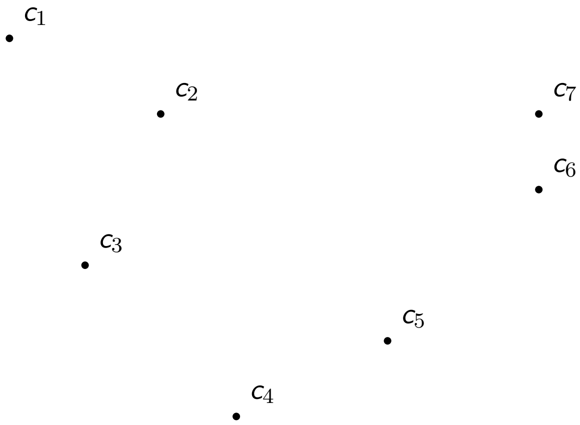
1. Greedy Patch Heuristic[3]
2. Tunnelling Algorithm[1][2]
   - Approximate Euclidean norm
   - Tunnelling Max TSP
   - Tunnelling Algorithm
3. Compare Results of GPH and TA

The Greedy Patching Heuristic[3]

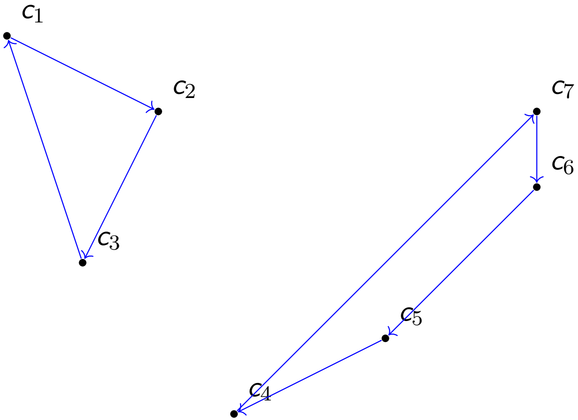
The idea:

1. create **maximum-weight cycle cover**
2. **patch** two cycles together with least loss
3. repeat step 2 till there's only 1 cycle ($\mathcal{O}(n)$)

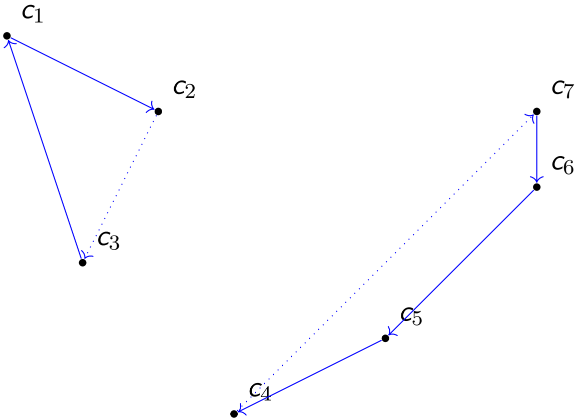
Maximum-Weight Cycle Cover



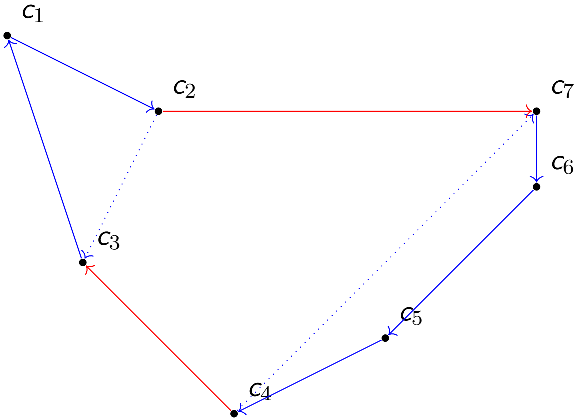
Maximum-Weight Cycle Cover



Patching step



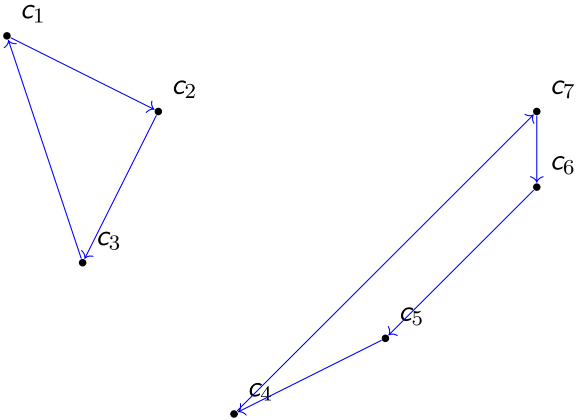
Patching step



Maximum-weight Cycle Cover

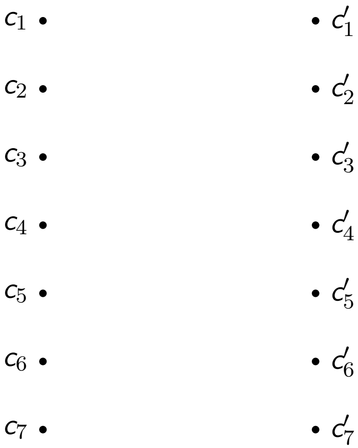
- solve through maximum-weight bipartite matching
- solvable with Simplex Algorithm, Hungarian Algorithm
- Hungarian Algorithm can be implemented in $\mathcal{O}(n^3)$

Maximum-weight Cycle Cover

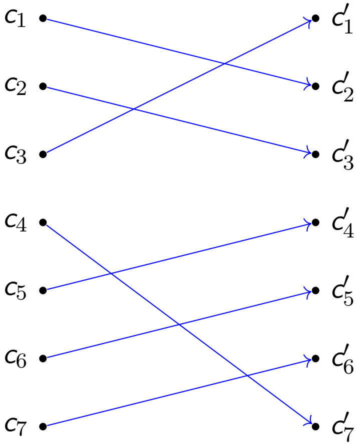


Maximum-Weight Bipartite Matching

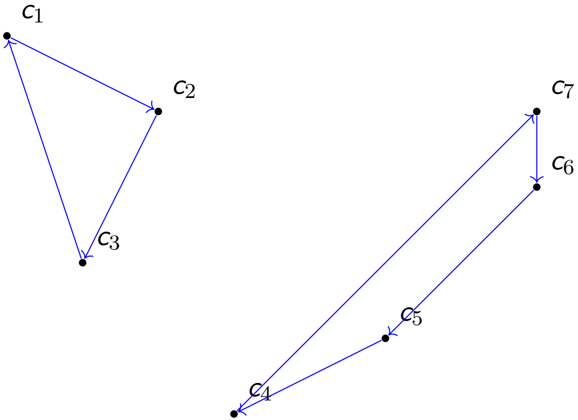
Maximum-Weight Bipartite Matching



Maximum-Weight Bipartite Matching



Maximum-weight Cycle Cover



Patching Step

$$\begin{tikzcd}
C \arrow[r] \arrow[dr, dotted] & ? \\
A \arrow[r] \arrow[ur, dotted] & ?
\end{tikzcd}$$

Patching Step

$$\begin{tikzcd}
C \arrow[r] \arrow[dr, dotted] & ? \\
A \arrow[r] \arrow[ur, dotted] & ?
\end{tikzcd}$$

$$\begin{array}{cc}
\begin{array}{ccccc} A & c_2 & \dots & & c_n \end{array} & \\
\left(\begin{array}{ccccc}
\dots & \dots & \dots & \dots & \\
\dots & & & & \\
Loss & & & & \\
\dots & & & & \\
\dots & & & & \\
\dots & & & & \\
\dots & & & & \\
\dots & & & &
\end{array}\right) & \begin{array}{c} A \\ \\ C \\ \\ \\ \\ \\ c_n \end{array}
\end{array}$$

Patching Step

$$\begin{tikzcd}
C \arrow[r] \arrow[dr, dotted] & ? \\
A \arrow[r] \arrow[ur, dotted] & ?
\end{tikzcd}$$

$$\begin{array}{cc}
\begin{array}{ccccc} A & c_2 & \cdots & & c_n \end{array} & \\
\begin{pmatrix}
\cdots & \cdots & \cdots & \cdots & \\
\cdots & & & & \\
Loss & & & & \\
\cdots & & & & \\
\cdots & & & & \\
\cdots & & & & \\
\cdots & & & & \\
\cdots & & & &
\end{pmatrix} & \begin{array}{c} A \\ \\ C \\ \\ \\ \\ \\ c_n \end{array}
\end{array}$$

$\mathcal{O}(n)$ patches with $\mathcal{O}(n^2)$ for each step

Greedy Patching Heuristic

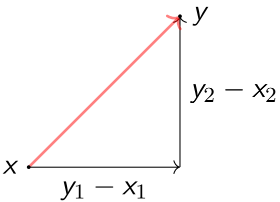
Results

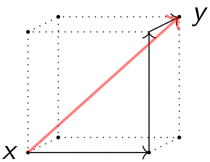
- Runtime: $\mathcal{O}(n^3) + \mathcal{O}(n \times n^2) = \mathcal{O}(n^3)$
- $\rho(n)_{GPH} \leq e^{1/3} \approx 1.3956$
- $\rho(n)_{GPH} \leq (1 - \frac{7/3}{n^{1/2dim+1}})^{-1}$
- (simplified $dim \in \theta(d)$)

- Remember: $Approx = \rho(n) \times OPT$

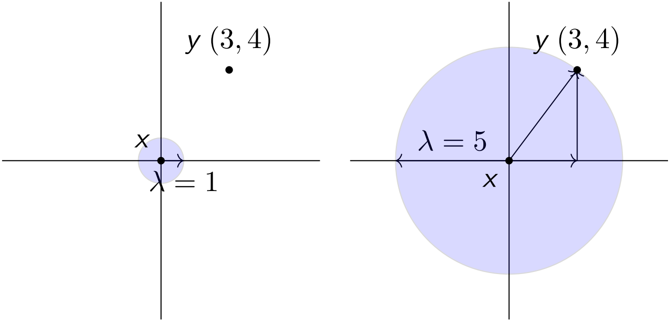
Tunnelling Algorithm[1][2]

- Goal: Approximate Max TSP under Euclidean distance
- Idea: Approximate the Euclidean distance
- Euclidean norm/distance: $||x - y||_2 = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$



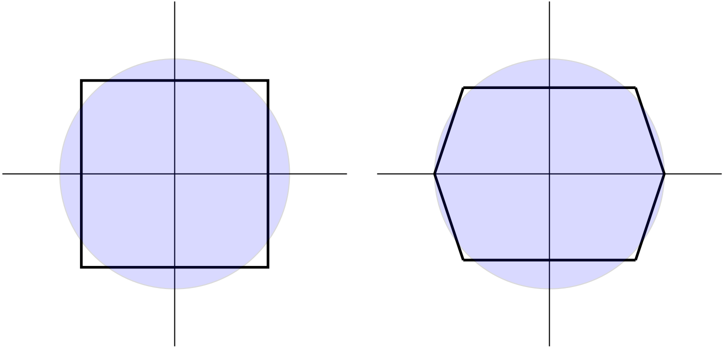


Euclidean Distance as Unit Ball

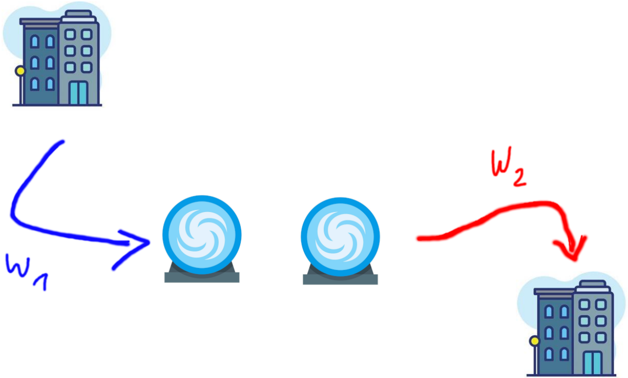


Polyhedral Norm

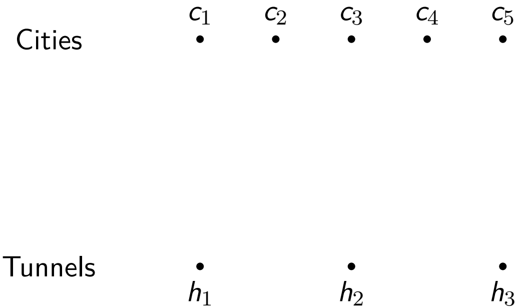
We want to approximate the shape of the Euclidean Metric



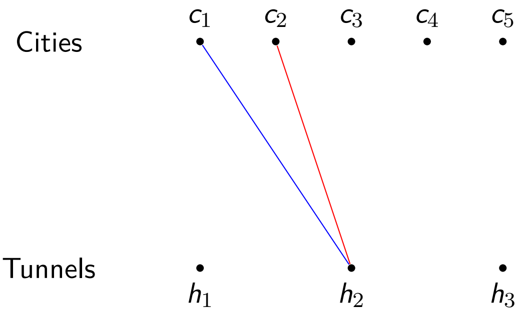
Tunnelling TSP



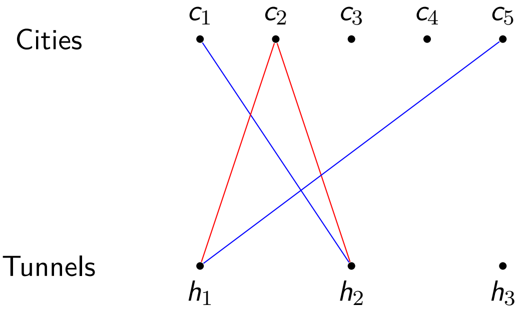
Tunnelling TSP



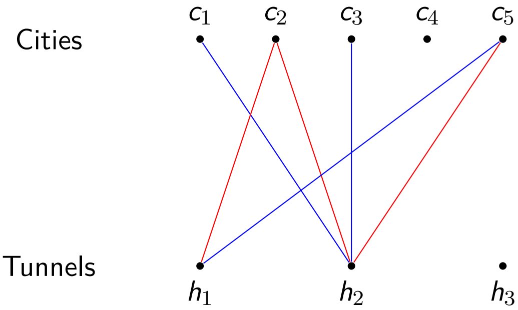
Tunnelling TSP



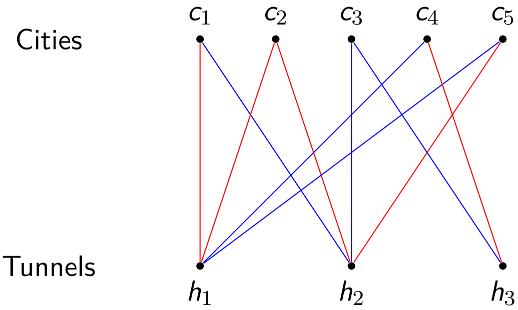
Tunnelling TSP



Tunnelling TSP



Tunnelling TSP



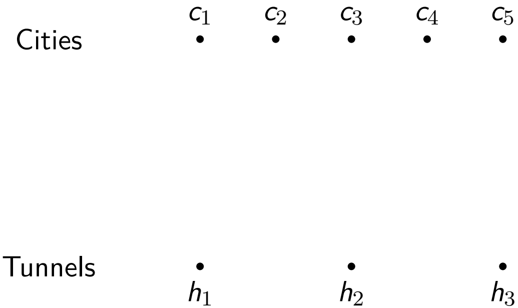
Why?

$$\begin{tikzcd}
\text{Euclidean Max TSP} & & \text{Polyhedral Max TSP} \arrow[ll, "\text{approximates}"'] \arrow[dd, leftrightarrow, "\text{equivalent}"] \\
& \{h_1, \ldots h_k\} \arrow[ur, "\text{polytope}"] \arrow[dr, "\text{tunnels}"'] & \\
\text{Tunnelling Algorithm} \arrow[rr, "\text{solves}"] & & \text{Tunnelling Max TSP}
\end{tikzcd}$$

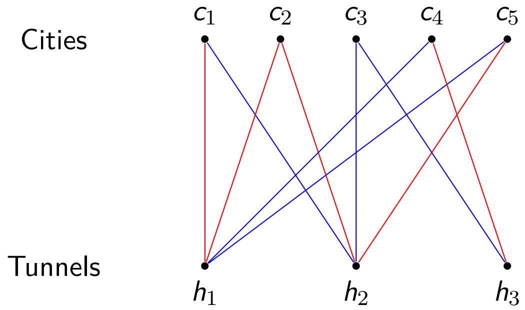
Tunnelling Algorithm

1. find a way to enumerate over outlines
2. for each guessed outline find optimal assignment
3. solution is best solution across all outlines

Tunnelling TSP



Tunnelling TSP



How to solve?

Idea: Guess the city order and solve for tunnels

$\Rightarrow n! \in \mathcal{O}(n^n)$ number of outlines

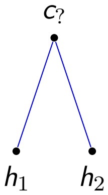
How to solve?

Idea: Guess the city order and solve for tunnels

$\Rightarrow n! \in \mathcal{O}(n^n)$ number of outlines

Instead: Guess tunnel structure and solve for cities!

Guess Outline



- $k \times k$ setting the tunnels
- 4 different colour options (red-red, blue-blue, red-blue, blue-red)
- n transitions
- together $\mathcal{O}(n^{4k^2})$ guessed outlines
- now assign the cities for each guess!

Maximum-Weight Bipartite Matching

$h_1 h_2, rb$ •

$h_2 h_2, bb$ •

$h_1 h_2, br$ •

$h_2 h_2, rr$ •

Maximum-Weight Bipartite Matching

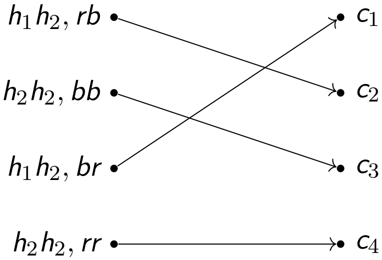
$h_1 h_2, rb$ • • c_1

$h_2 h_2, bb$ • • c_2

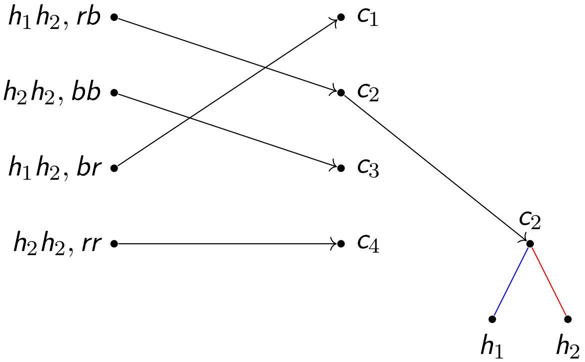
$h_1 h_2, br$ • • c_3

$h_2 h_2, rr$ • • c_4

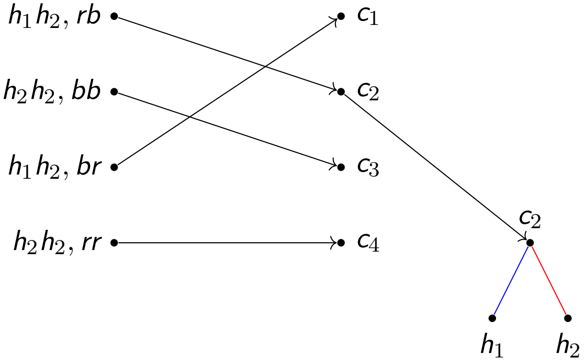
Maximum-Weight Bipartite Matching



Maximum-Weight Bipartite Matching

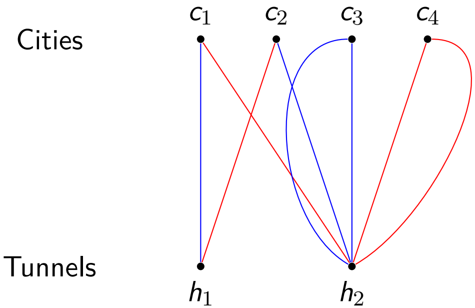


Maximum-weight bipartite matching



$\Rightarrow$ Hungarian Algorithm in $\mathcal{O}(n^3)$

Resulting Tour



Tunnelling Algorithm Results

- k number of tunnels/half the facets of polytope
- Runtime: Number outlines $\times$ bipartite matching effort
- Runtime: $\mathcal{O}(n^{4k^2} \times n^3) = \mathcal{O}(n^{4k^2+3})$ [1]
- better runtime Barvinok et al.'s Algorithm $\mathcal{O}(n^{2k-2} log(n))$ [2]
- Approximation ratio depending on k
- $\rho(n) =?$

Comparing the Algorithms

Idea:

- fixing the dimension to 2 dimensional space
- find exact $\rho(n)$ for Tunnelling Algorithm
- simplifying GPH
- find a way to compare runtime and $\rho(n)$ together

Simplify TA Results

Finally we get

- $\rho(n)_{TA} \leq \dfrac{1}{cos_D(90/k)}$
- assuming regular polygon with $2k$ sides
- assuming we're in the plane ($d = 2$)
- Runtime $\mathcal{O}(n^{2k-2} log(n))$

Simplify GPH Results

- Assume we're looking at 2 dimensional space
- $\rho(n)_{GPH} \leq e^{1/3}$
- $\rho(n)_{GPH} \leq (1 - \frac{7/3}{n^{1/(2dim+1)}})^{-1} = (1 - \frac{7/3}{n^{1/6.6}})^{-1}$
- (because $dim = 2.8$ for $d = 2$)
- Runtime $\mathcal{O}(n^3)$

Simplify GPH Results

- $e^{1/3} = (1 - \frac{7/3}{n^{1/6.6}})^{-1}$
- $n \approx 10^6$

- $\rho(n)_{GPH} \leq e^{1/3}$ (for $n < 10^6$)
- $\rho(n)_{GPH} \leq (1 - \frac{7/3}{n^{1/6.6}})^{-1}$ (for $n \geq 10^6$)

Compare the Algorithms

Greedy Patch Heuristic[3]

- Runtime $\mathcal{O}(n^3)$
- $\rho(n)_{GPH} \leq e^{1/3}$ (for $n < 10^6$)
- $\rho(n)_{GPH} \leq (1 - \frac{7/3}{n^{1/6.6}})^{-1}$(for $n \geq 10^6$)

Tunnelling Algorithm[1][2]

- Runtime $\mathcal{O}(n^{2k-2} log(n))$
- $\rho(n) \leq \frac{1}{cos_D(90/k)}$
- Idea: set $k = 2$

Compare the Algorithms

Greedy Patch Heuristic[3]

- Runtime $\mathcal{O}(n^3)$
- $\rho(n)_{GPH} \leq e^{1/3} \approx 1.39$ (for $n < 10^6$)
- $\rho(n)_{GPH} \leq (1 - \frac{7/3}{n^{1/6.6}})^{-1}$(for $n \geq 10^6$)

Tunnelling Algorithm[1][2]

- Runtime $\mathcal{O}(n^2 log(n))$
- $\rho(n) \leq \frac{1}{cos_D(45)} \approx 1.41$

Conclusion

- for roughly the same $\rho(n)$ Tunnelling Algorithm will perform better
- eventually GPH have a better $\rho(n)$ with $n \geq 10^6$
- we always have the option to increase k for TA

The Thesis and Slides

Thank you for your attention!

You can find my thesis and the slides under this link:

[1] G.J. Woeginger(2018). Some Easy and Some Not so Easy Geometric Optimization Problems. In: Epstein, L., Erlebach, T. (eds) Approximation and Online Algorithms. WAOA 2018. Lecture Notes in Computer Science(), vol 11312. Springer, Cham. DOI: 10.1007/978-3-030-04693-4_1.

[2] A.I. Barvinok, S.P. Fekete, D.S. Johnson, A. Tamir, G.J. Woeginger, R. Woodroofe(2003). The geometric maximum travelling salesman problem. J. ACM, Volume 50, Issue 5, 641-664. DOI: 10.1145/876638.876640. arXiv: cs/0204024.

[3] V.V. Shenmaier(2022). Asymptotic Optimality of the Greedy Patching Heuristic for Max TSP in Doubling Metrics. arXiv: 2201.03813 .

Questions for the Listeners

- Why does the Tunnelling Algorithm not work for Min TSP?
- How did we get the $\rho(n)$ of the Tunnelling Algorithm?
- How can we represent the polyhedral norm?
- Can't we just chose a better higher k in the comparison?
- What is *dim* in the GPH?

Extra: Polyhedral norm

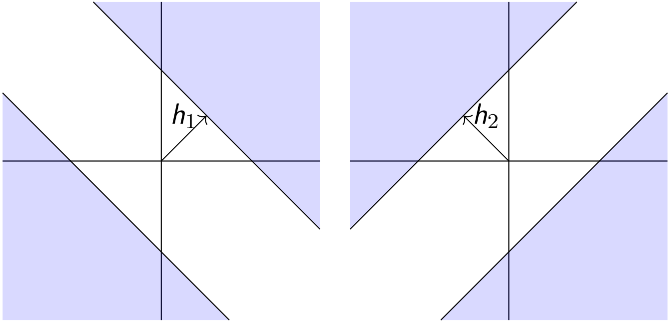
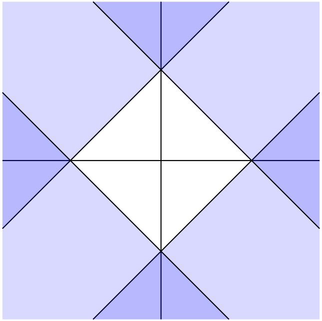
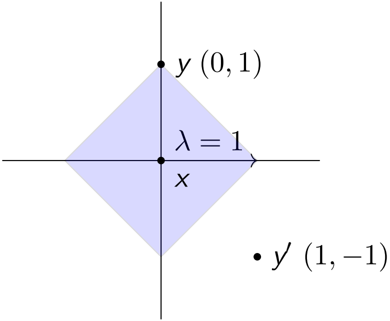


Figure: Left: two halfspaces created by the vector (1,1), Right: two halfspaces created by the vector (-1,1)

Extra: Polyhedral norm





- represent polytope with $2k$ sides through k vectors
- polyhedral distance: $max_i\{|h_i(x - y)|\}$

Extra: Why?

$$max_i\{|h_i(y-x)|\}$$

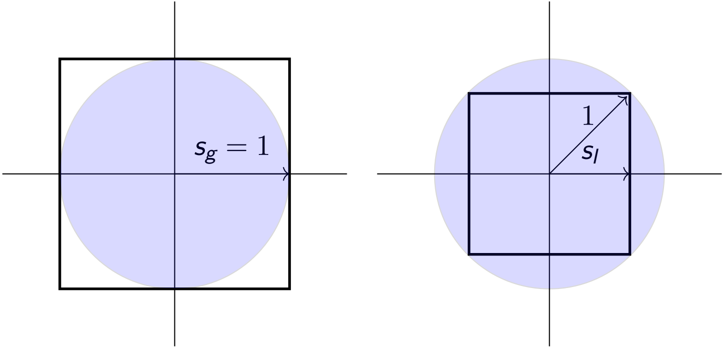
$$\begin{tikzcd}
\text{Euclidean Max TSP} & & \text{Polyhedral Max TSP} \arrow[ll, "\text{approximates}"'] \arrow[dd, leftrightarrow, "\text{equivalent}"] \\
& \{h_1, \dots h_k\} \arrow[ur, "\text{polytope}"] \arrow[dr, "\text{tunnels}"'] & \\
\text{Tunnelling Algorithm} \arrow[rr, "\text{solves}"] & & \text{Tunnelling Max TSP}
\end{tikzcd}$$

$$|h_i(y-x)|$$

Extra: Simplify TA Results

- $\mathcal{B}_1$ Euclidean unit ball
- $\mathcal{B}_2$ unit ball of polyhedral norm
- $(1-\epsilon)\mathcal{B}_2 \subseteq \mathcal{B}_1 \subseteq (1+\epsilon)\mathcal{B}_2$
- then $\rho(n) \leq (1+\epsilon)/(1-\epsilon)$

Extra: Simplifying TA



Extra: Simplifying TA

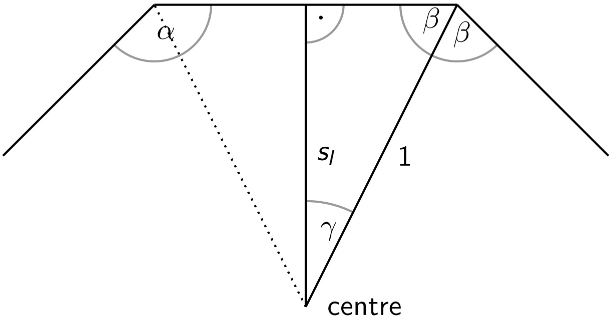


Figure: Triangle within the polygon

Extra: Conclusion

- set $(1 - \frac{7/3}{n'^{1/6.6}})^{-1} = \frac{1}{cos_D(90/k)}$
- $\Rightarrow n' \in \mathcal{O}(k^{14})$

- n' grow polynomially in k where GPH $>$ TA in case $n \geq n'$
- Runtime of TA grows exponentially in k

Extra: Conclusion

- TA is good for small n
- TA can get arbitrarily close independent of input size
- GPH eventually out scales TA with high input size
- GPH point of out scaling grows better to runtime of TA

Extra: Doubling Dimension

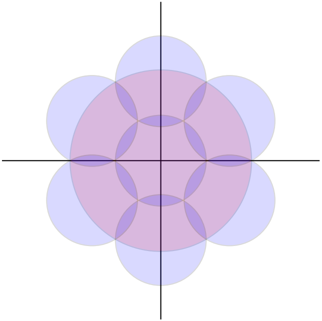


Figure: Covering a ball with 7 balls of half it's radius in the plane. Thus $7 = 2^{dim}$

Doubling dimension *dim*: Number of balls we need to cover a ball double their radius $= 2^{dim}$